Sviluppo di un SMA con tecnologia JADE su dispositivi mobili con Android OS

Alessandro Bulzacchi, Davide Galeotti

Anno Accademico 2010-2011

Indice

1 Architettura ad agenti basati sulla piattaforma JADE per dispositivi mobili

L'innovazione tecnologica, già da qualche anno, sta dirigendo i propri sforzi nello sviluppo di dispositivi mobili, detti smartphone, dotati di un'estrema facilità d'uso e di potenti capacità computazionali e comunicative. La loro esponenziale diffusione a livello globale unita alle loro capacità computazionali li rende soggetti interessanti per l'integrazione in sistemi multiagente. Gli agenti sono considerati, dalla comunità scientifica, uno dei più importanti paradigmi che possono migliorare gli attuali metodi di concettualizzazione, progettazione e implementazione dei sistemi software. Un agente è un software autonomo dotato di un proprio flusso di controllo le cui azioni sono volte a raggiungere un obiettivo o goal. Un sistema distribuito ad agenti ospita una collettività di agenti che possono comunicare tra loro sfruttando i mezzi che l'ambiente in cui sono immersi mette loro a disposizione. L'obiettivo del presente elaborato è quello mettere in evidenza gli aspetti del modello di agente JADE basati su dispositivi mobili e definire un'architettura generale che sfrutti le capacità della piattaforma e cerchi di superarne i limiti attuali.

1.1 Introduzione agli agenti software

Per la realizzazione del software oggetto di questo elaborato è stata utilizzata una piattaforma basata sugli agenti. Questi, come già anticipato, costituiscono un campo dell'informatica relativamente recente su cui è stata posta l'attenzione della comunità scientifica internazionale. Ne verranno presentate ora le nozioni e la terminologia di base, in modo da poter permettere al lettore la comprensione dei contenuti che verranno esposti successivamente. Il concetto di agente software è nato all'inizio degli anni '80, come punto d'incontro tra il campo di ricerca dell'intelligenza artificiale e quello dell'informatica. Il suo scopo era fornire una soluzione che permettesse di risolvere, in maniera automatizzata, i problemi che l'espansione delle reti di calcolatori (Internet in primis) iniziava a porre. Attualmente non esiste ancora una definizione universalmente accettata di cosa si intenda effettivamente con il termine "agente software". Volendone comunque dare una definizione intuitiva, esso può essere identificato come un software dotato di un certo grado di autonomia, intesa come capacità di prendere decisioni in modo indipendente e sulla base delle condizioni ambientali circostanti. Queste ultime possono includere ad esempio: informazioni ricevute o inviate verso altri agenti, output ricevuti da altri processi in esecuzione concorrente, risultati di query effettuate su un database, etc. . . Una delle caratteristiche peculiari di un agente software è la sua capacità di prendere autonomamente decisioni.

Per meglio caratterizzarne le abilità, le potenzialità e quindi i comportamenti che esso potrebbe esibire durante la sua esecuzione, la letteratura scientifica ha identificato alcune proprietà utili a fornirne una più accurata classificazione. Alcune tra le più importanti sono:

- Autonomia: caratterizza il grado di libertà che possiede l'agente nel prendere decisioni riguardo alle azioni da eseguire. Essa è anche sinonimo di intelligenza;
- Reattività: indica la capacità dell'agente di percepire l'ambiente ad esso circostante, e di conseguenza di reagire ai cambiamenti che si verificano in quest'ultimo;
- Proattività: capacità di reagire agli stimoli dell'ambiente circostante, di prendere autonomamente la decisione di intraprendere una determinata azione (comportamenti goal-oriented);
- Abilità sociali: capacità dell'agente di interagire con altri agenti, ed eventualmente anche con esseri umani; ovviamente questo deve avvenire in base a dei linguaggi e a delle semantiche concordate precedentemente;
- Persistenza: nel caso l'esecuzione dell'agente venisse sospesa, capacità di salvare un insieme di informazioni nell'ambiente in cui è situato in modo da poter riprendere il proprio flusso computazionale in un momento successivo;
- Mobilità: Capacità di muoversi tra i nodi che compongono la rete, nel caso in cui il calcolatore sul quale l'agente è in esecuzione sia connesso ad essa;
- Adattività: capacità di apprendere sia dalle proprie azioni che dall'ambiente circostante, migliorandone conseguentemente l'efficienza nel tempo;
- Fidatezza: garanzia agli utenti concui interagisce di non comunicare informazioni riservate a terze parti senza essere stato autorizzato precedentemente;
- Benevolenza: l'agente proverà sempre ad eseguire ciò che gli è stato richiesto, controllando comunque che gli obiettivi ad esso assegnati non siano in contrasto tra di loro;
- Cooperazione: l'agente esibisce logiche di collaborazione con altri agenti eventualmente presenti nel proprio ambiente. In questo modo esso può portare a termine i propri compiti in minor tempo od utilizzando meno risorse. Esiste anche il caso opposto: l'agente esibisce comportamenti attraverso i quali entra in competizione con gli altri agenti per l'utilizzo delle risorse presenti nell'ambiente; in questo caso si parla di competitività.

Con riferimento all'elenco sopra esposto, un agente viene classificato come debole se esibisce le seguenti proprietà: autonomia, reattività, proattività, abilità sociali; in caso contrario, esso viene definito forte.

2 Piattaforma JADE

2.1 Introduzione a JADE

JADE (Java Agent DEvelopment Framework) è una piattaforma ad agenti mobili sviluppata nei laboratori di ricerca Telecom di Torino. Il progetto, che ha avuto inizio alla fine del 1998, ha reso disponibile la prima versione per il download al pubblico all'inizio del 2000. Poco dopo, il team di sviluppo ha deciso di aderire allo standard FIPA e ha reso il software open-source, rilasciandolo sotto licenza LGPL (Lesser General Public License). Da quel momento, attorno al progetto, si è formata una comunità di sviluppatori che ha contribuito allo sviluppo di JADE e creato numerose estensioni allo scopo di renderlo adatto all'utilizzo nelle più svariate applicazioni. Il rilascio sotto licenza LGPL non ha comunque comportato un abbandono del progetto da parte dei laboratori Telecom che, anzi, ancora oggi, contribuiscono attivamente al suo sviluppo. Come risultato dell'interesse di alcune aziende coinvolte nel progetto, nel 2003 è stata fondata la JADE Board, un'organizzazione no-profit che si pone come obiettivo la promozione dell'utilizzo di JADE come middleware per lo sviluppo di applicazioni basate su agenti software. È da segnalare che tali aziende hanno contribuito attivamente allo sviluppo della piattaforma, in particolare attraverso la creazione di nuovi add-on. Questi, in virtù della licenza LPGL sotto cui è rilasciato JADE, sono stati resi disponibili al pubblico. Lo sviluppo di questo software è poi proseguito fino ad oggi: la versione più aggiornata, nel momento in cui viene scritto questo testo, è la 4.1. Per capire come è organizzata la piattaforma su cui è stato sviluppato il software per questo elaborato, è opportuno analizzarne l'architettura.

2.2 Elementi di base dell'architettura

JADE aderisce allo standard FIPA: implementa, quindi, la struttura e gli elementi fondamentali che tale standard definisce. È opportuno notare che i componenti descritti sono di natura logica: sono da intendersi, cioè, come insieme di servizi e non è quindi necessario che ad ognuno di essi corrisponda un componente software distinto.

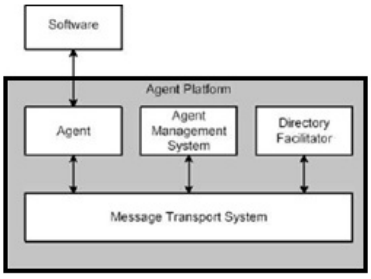

Figura 1: Elementi fondamentali di una piattaforma FIPA

Con riferimento alla figura 1, si può osservare che gli elementi fondamentali di una piattaforma, secondo lo standard FIPA, sono:

- l'agente: rappresenta l'entità principale operante all'interno della piattaforma. È in grado di svolgervi quindi un determinato insieme di azioni tra cui l'interazione con altri agenti, l'interazione con la piattaforma stessa o con software esterno ad essa. Ad ogni agente deve essere associato un proprietario (owner) che può essere un utente umano o un'entità software. Ad ogni agente è inoltre assegnato un AID (Agent IDentifier), identificatore univoco che permette di individure il primo con certezza all'interno della piattaforma in cui opera. Esso è costituito, in genere, da una stringa, da un numero o da una combinazione di questi due elementi. Inoltre, dato che un agente può utilizzare vari servizi di trasporto per spostarsi da un nodo ad un altro, ad esso viene anche associato un indirizzo di trasporto che permette di identificarlo univocamente, questa volta all'interno della rete in cui opera;

- l'Agent Management System (AMS): è un componente il cui compito consiste nel supervisionare gli accessi alla piattaforma da parte di entità esterne, oltre che nel monitorare le azioni che gli agenti compiono. È fondamentale, secondo le specifiche FIPA, che vi sia un solo AMS all'interno di una piattaforma. Se, ad esempio, un agente (esterno o appena creato) volesse operare all'interno della piattaforma sarebbe obbligato ad effettuare una registrazione presso l'AMS: dovrebbe cioè segnalare la propria presenza e attendere che quest'ultimo gli conceda l'autorizzazione ad operare. In questo caso l'AMS provvederebbe ad assegnargli un AID opportuno. Per poter svolgere questo tipo di compito, l'AMS mantiene al suo interno un elenco di tutti gli AID assegnati e, per ognuno di essi, anche altre informazioni accessorie utili alla gestione degli agenti (come ad esempio il loro indirizzo). Un'ulteriore ed utile funzione svolta da questo componente è il servizio di pagine bianche (white page service): ogni agente può conoscere, in un determinato istante, quali altri agenti sono presenti all'interno della piattaforma e dove si trovano;

- il Directory Facilitator (DF): come per l'AMS, la presenza di questo componente è obbligatoria all'interno della piattaforma. In questo caso, tuttavia, possono esservene molteplici. Il compito principale del DF è quello di fornire il servizio di pagine gialle (yellow page service): mantiene cioè un elenco dei servizi offerti da ogni agente all'interno della piattaforma. In questo modo, un agente che offra un determinato servizio può comunicarlo al DF il quale provvederà a memorizzare tale informazione. Simmetricamente, un altro agente che necessiti di tale servizio può effettuare una query al DF per conoscerne il nome e il luogo di chi lo renda disponibile;

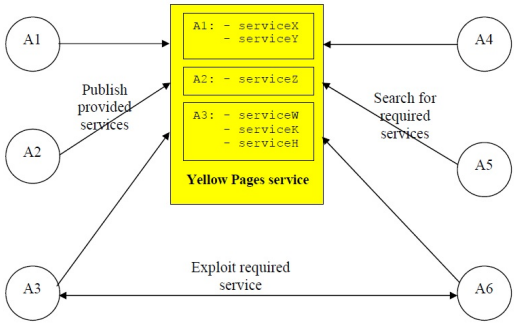

Figura 2: Il servizio di pagine gialle

- il Message Transport System (MTS): ha il compito di gestire e coordinare lo scambio di informazioni tra agenti. Questi ultimi, in particolare, possono essere situati nella stessa piattaforma o in piattaforme diverse;

- l'Agent Platform: è un componente che fornisce un ambiente uniforme in cui gli agenti possono svolgere i propri compiti, comunicare e spostarsi. È da notare come, secondo lo standard FIPA, ne facciano parte anche il sistema operativo e il calcolatore fisico su cui l'AP è in esecuzione;

- il software esterno: software non basato sulla tecnologia ad agenti che offre servizi a cui gli agenti della piattaforma possono accedere;

Introdotti gli elementi fondamentali definiti dallo standard FIPA, è ora interessante analizzare sinteticamente come essi siano stati implementati, in pratica, in JADE. Una piattaforma JADE è costituita da uno o più contenitori (container), chiamati così proprio perché ospitano agenti al proprio interno. Tra di essi, ve ne è uno "speciale", chiamato contenitore principale (main container). Questo controlla e coordina il funzionamento degli altri contenitori, detti anche contenitori periferici, e funge da punto di riferimento per essi. È importante sottolineare come esista una corrispondenza univoca tra un contenitore principale e una piattaforma JADE: se, infatti, venisse avviato un altro main container, esso darebbe origine ad un'altra piattaforma JADE a sé stante, indipendente ed esterna alla prima. Ciò che rende il contenitore principale fondamentale

per il funzionamento della piattaforma è la presenza obbligatoria al suo interno dell'AMS e di (almeno) un DF. Questi sono stati implementati in JADE come agenti software, e vengono istanziati automaticamente all'avvio della piattaforma stessa; in questo modo si garantisce la conformità con le specifiche FIPA. Sempre all'interno del contenitore principale vengono mantenute alcune informazioni essenziali per il funzionamento della piattaforma. Tra queste troviamo l'Agent Container Table (ACT), una tabella che mantiene al suo interno una sorta di mappa di tutti i contenitori della piattaforma, e l'Agent Global Descriptor Table (AGDT), che mette in correlazione le informazioni presenti nell'AMS riguardanti un agente ed il suo AID con altri dati utili alla sua gestione. È infine da segnalare che, per facilitare la gestione della piattaforma, è stata sviluppata una GUI (Graphical User Interface), un'interfaccia grafica che permette di monitorare lo stato degli agenti e dei container in esecuzione in un dato istante. Questa prende il nome di RMA (Remote Monitoring Agent) e, come si può dedurre, è stata implementata come un agente. Viene avviato con la piattaforma (insieme quindi all'AMS e al primo DF), ed ovviamente può essere chiuso e riavviato in qualsiasi momento, visto che il suo funzionamento non interferisce con quello della piattaforma.

JADE è inoltre fornito nativamente di alcuni agenti che mettono a disposizione altre funzioni; esse sono:

- Sniffer Agent: si tratta di un agente che permette di monitorare, ed eventualmente ispezionare, i messaggi scambiati tra gli agenti che popolano la piattaforma;
- Dummy Agent: è un agente di test, il cui scopo è quello di testare il funzionamento dello scambio di messaggi tra esso ed un altro agente: in questo caso può memorizzare i messaggi scambiati con gli altri agenti e comporre messaggi ad-hoc per testare la loro risposta;
- Introspector Agent: consente di visualizzare lo stato di ogni agente, le azioni di cui esso è capace e visualizzare quale azione stia attualmente eseguendo;
- Log Manager Agent: è un agente che, selezionato un dato container, permette di eseguirne un logging approfondito; memorizza quindi tutti gli eventi che accadono in tale container e permette di analizzarne i dettagli successivamente;

2.2.1 Scambio di messaggi

Un'abilità fondamentale di cui sono dotati gli agenti è quella di poter comunicare tra di loro: in JADE questo avviene tramite unità informative discrete chiamate messaggi. In particolare, JADE ha implementato l'Agent Communication Language (ACL) definito dalle specifiche FIPA, e supporta quindi tutti i protocolli di trasporto da esso definiti. Quale tra questi ultimi venga utilizzato per la trasmissione dei messaggi dipende da dove sono situati i due agenti che vogliono comunicare. Si hanno infatti tre casi distinti, a seconda del fatto che i due agenti si trovino nello stesso contenitore, in due contenitori diversi all'interno della stessa piattaforma, o in due piattaforme diverse.

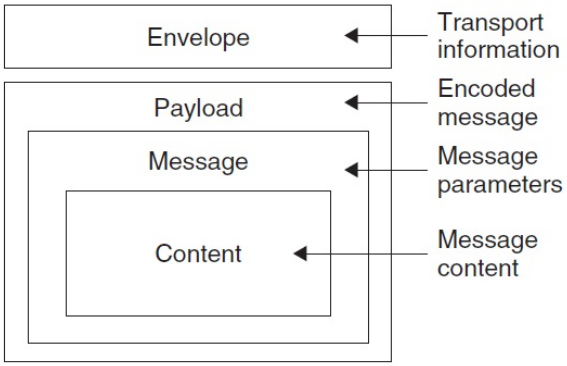

Figura 3: Struttura generale di un messaggio ACL

Con riferimento alla figura 3, si può notare come un messaggio ACL sia essenzialmente diviso in due parti: la prima, chiamata envelope, contiene informazioni utili per la consegna del messaggio, come ad esempio mittente, destinatario, codifica utilizzata, etc. . . La seconda, chiamata payload, contiene il contenuto vero e proprio del messaggio che si vuole inviare al destinatario. Nel caso di comunicazioni interne ad un contenitore, si utilizza un protocollo denominato IMTP (Internal Message Transport Protocol) che non deve essere necessariamente standard, dato che non coinvolge altri contenitori. Esso viene infatti utilizzato anche per il trasporto di comandi interni utili alla gestione della piattaforma, e per il monitoraggio dello stato dei container periferici. Nel caso invece di comunicazioni tra due agenti situati in due contenitori diversi, ma sempre all'interno della stessa piattaforma, viene utilizzato il meccanismo di invocazione remota di metodi fornito da Java (RMI). Grazie ad esso è possibile ottenere un riferimento ad un oggetto in modo trasparente rispetto al fatto che si trovi in un altro contenitore: in questo caso, il contenitore che contiene l'agente destinatario del messaggio agisce da server, mentre quello che contiene il mittente svolge il ruolo di client.

2.2.2 Behaviour

Per permettere l'esecuzione di compiti (o più propriamente, di task) da parte dei propri agenti, e quindi il raggiungimento degli obiettivi ad essi assegnati, JADE implementa un particolare paradigma, basato sul concetto di behaviour (comportamento). Un behaviour è implementato concretamente dalla classe Behaviour contenuta nel package jade.core.behaviour; questo corrisponde ad un particolare task che l'agente dovrà eseguire durante il suo ciclo di vita all'interno della piattaforma. Le specifiche azioni che l'agente dovrà compiere per eseguire tale task andranno poi inserite al suo interno grazie ad appositi metodi che tale classe fornisce.

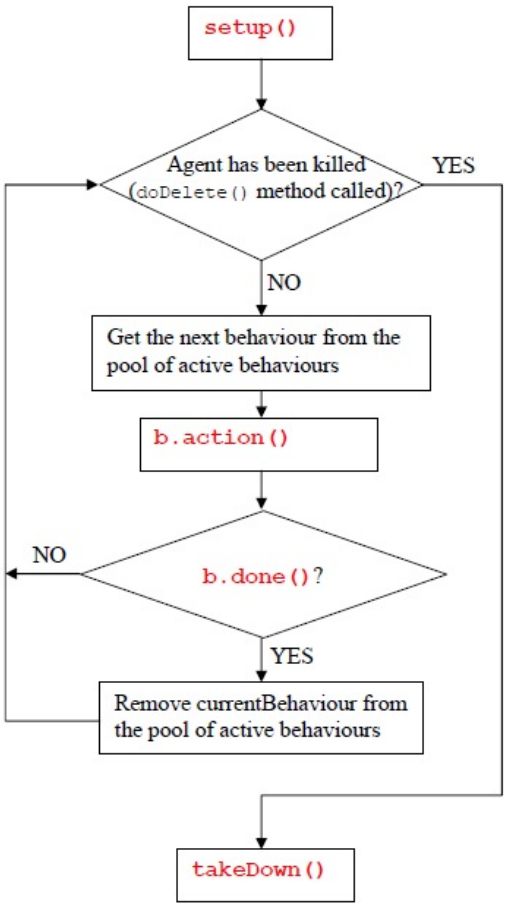

Figura 4: Schema generale di esecuzione di un behaviour

Lo sviluppatore di un agente può notificargli la presenza di un behaviour da eseguire grazie al metodo addBehaviour() della classe jade.core.Agent, messo a

disposizione per implementare l'agente stesso. Ogni agente mantiene al proprio interno una lista dei behaviour da eseguire: l'ordine in cui questo avverrà sarà lo stesso secondo il quale behaviour sono stati notificati all'agente tramite il metodo sopra citato.

Una volta iniziata l'esecuzione di un behaviour (che avviene invocando il suo metodo action()), essa verrà portata avanti fino alla sua terminazione. A questo punto l'agente controllerà, tramite il metodo done(), se il task associato al behaviour è stato portato a termine con successo. In caso affermativo, l'agente inizierà ad eseguire il behaviour successivo presente nell'elenco; in caso contrario, ricomincerà l'esecuzione del behaviour non terminato. Le API di JADE forniscono nativamente, oltre alla classe base Behaviour, anche numerose altre classi che la estendono e permettono di creare logiche di comportamento anche complesse. Dalle seguenti classi possono essere implementati vari tipi di Behaviour:

- OneShotBehaviour: si tratta di una sottoclasse della classe Behaviour il cui metodo done() restituisce sempre il valore true. Essa permette quindi di eseguire behaviour di tipo one-shot: il loro metodo action() verrà eseguito una ed una sola volta;

- CyclicBehaviour: è una classe simmetrica alla precedente, il cui metodo done() ritorna sempre il valore false. Questo tipo di behaviour verrà quindi eseguito ciclicamente fino alla terminazione dell'agente stesso;

- SequentialBehaviour: questa classe permette di creare un behaviour composto da uno o più sotto-behaviour, i quali possono essere aggiunti tramite il metodo addSubBehaviour(). Questi ultimi verranno eseguiti in sequenza nell'ordine specificato, ed il behaviour terminerà solo quando saranno terminati tutti i sotto-behaviour;

- FSMBehaviour: costituisce un'ulteriore estensione della classe precedente. Tramite questa classe è possibile specificare dei veri e propri automi a stati finiti deterministici, in cui ogni stato corrisponde ad un sotto-behaviour. Ad ognuno di questi viene inoltre data la possibilità di restituire un valore intero, a seconda dell'esito che ha avuto la sua esecuzione: tale valore decide quale sarà il prossimo sotto-behaviour che verrà eseguito. In questo modo si emulano le transizioni presenti in un automa a stati finiti. L'esecuzione del behaviour inizia sempre dal sotto-behaviour che corrisponde allo stato iniziale dell'automa (ve ne deve essere uno e uno solo), e termina quando è terminata l'esecuzione del metodo action() di uno dei sotto-behaviour marcati come stati finali dell'automa (gli stati finali di un'automa possono essere uno o più di uno);

- ParallelBehaviour: è una classe che permette di eseguire i sotto-behaviour scelti in modo concorrente. Il behaviour è da considerarsi terminato quando tutti i suoi sotto-behaviour sono terminati, oppure quando il primo sotto-behaviour ha ultimato la sua esecuzione (in quest'ultimo caso, l'esecuzione degli altri sotto-behaviour verrà interrotta);

2.3 Lo standard FIPA

Lo standard de facto attuale per l'interoperabilità tra agenti è FIPA (Foundation for Intelligent and Physical Agents). Esso è stato elaborato dall'omonimo consorzio, con sede in Svizzera, di cui fanno parte numerose multinazionali operanti nel campo dell'informatica e dell'elettronica. Dal 2005 è entrato a far parte dell'IEEE come Standards Committee. Come FIPA stessa dichiara, lo scopo di tale standard è solo quello di definire un'interfaccia comune per i diversi oggetti che popolano l'ambiente in cui gli agenti software si troveranno a operare, in modo da permettere la comunicazione tra agenti di diverse piattaforme,ed anche tra agenti software ed altri tipi di tecnologie non basate sugli agenti. In particolare, lo standard FIPA coinvolge i seguenti cinque ambiti:

- Applications: è costituito da un insieme di esempi di applicazioni, per ognuna delle quali FIPA propone un insieme minimo di servizi che l'agente dovrebbe essere in grado di fornire;
- Abstract Architecture: definisce, a livello astratto, gli elementi essenziali per la realizzazione di una piattaforma per agenti software e le relazioni tra di essi ed indica alcune linee guida per la loro implementazione;
- Agent Communication: riguarda un insieme di specifiche per la comunicazione tra agenti. Con riguardo a questa tematica FIPA ha elaborato un linguaggio, chiamato ACL (Agent Communication Language), costituito da un formato comune per lo scambio di informazioni tra agenti. Questo, inoltre, prevede numerosi protocolli e schemi di interazioni;
- Agent Management: è formato da una serie di linee guida per la gestione ed il controllo degli agenti e del loro comportamento, sia quando si trovano all'interno di una piattaforma, sia durante il transito da una piattaforma all'altra;
- Agent Message Transport: definisce modalità standard per il trasferimento e la rappresentazione delle informazioni attraverso reti disomogeneeper;

Per poter aderire allo standard FIPA, una piattaforma ad agenti software deve quindi implementare le specifiche contenute nei cinque ambiti sopra elencati.

2.4 JADE-LEAP

Fra i numerosi add-on disponibili, che rendono estremamente versatile questa piattaforma ad agenti, ve ne è uno in particolare che merita attenzione. Esso permette, infatti, lo sviluppo di sistemi che possono funzionare non solo su hardware complessi ma, parte di essi, anche su dispositivi a risorse limitate, quali palmari o cellulari.

In base a questi presupposti è stato sviluppato Jade-Leap: questo, se combinato con Jade, rimpiazza alcune parti del kernel JADE formando un ambiente runtime modificato (che prende il nome di "JADE powered by LEAP") e che può essere impiegato in un ampia gamma di dispositivi che vanno dai server fino ai cellulari dotati di ambiente J2ME. Dal punto di vista del programmatore, un ambiente Jade-Leap perfettamente coincidente con uno Jade, in quanto si utilizzano le stesse API e gli stessi metodi di amministrazione. È da sottolineare, tuttavia, che non è possibile mescolare nella stessa piattaforma container appartenenti ai due framework (possono ovviamente comunicare attraverso MTP se sono su piattaforme separate). In generale, un ambiente Jade-Leap può essere eseguito in due modalità distinte:

- modalità stand-alone : quando viene avviato un container completo su un dispositivo dove è presente un runtime JADE;

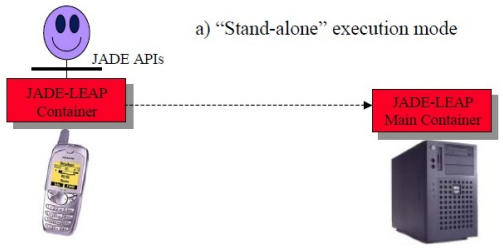

Figura 5: Esecuzione in modalità "Stand-alone"

- modalità split-container: quando il container viene diviso in due parti chiamate rispettivamente "FrontEnd" (eseguito sul dispositivo dove è presente un runtime Jade) e "BackEnd" (eseguito su un server remoto, in cui le parti sono collegate tra loro in modo permanente -in senso logico-).

La modalità split è utilizzata quando si ha a che fare con dispositivi mobili. Visto che il FrontEnd è molto più leggero di un container completo, la fase di bootstrap è molto più veloce, perchè tutte le comunicazioni per accedere alla piattaforma vengono eseguite dal BackEnd, cosicchè infine anche il collegamento wireless è ottimizzato. È da sottolineare, comunque, che questi processi sono del tutto trasparenti allo sviluppatore in quanto il set di API cui può accedere è esattamente lo stesso.

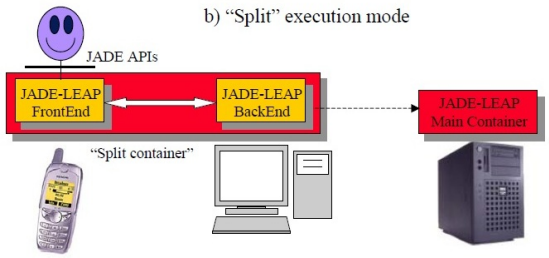

Figura 6: Esecuzione in modalità "Split"

2.5 JADE for Android

Android è un sistema operativo, per dispositivi mobili, sviluppato da Google, in collaborazione con la Open Handset Alliance. L'obiettivo principale di Android è quello di fornire tutto ciò di cui un operatore, un vendor di dispositivi o uno sviluppatore, ha bisogno. Inoltre, la sua caratteristica principale è di essere open. Infatti:

- Android è open in quanto utilizza tecnologie open, prima fra tutte il kernel di Linux nella versione 2.6.
- Android è open in quanto le librerie API che sono state usate per la sua realizzazione sono esattamente le stesse disponibili ad un programmatore. Questo comporta che non ci sia quasi nessun limite alla personalizzazione dell'ambiente.
- Android è open in quanto il suo codice è open source, consultabile da chiunque possa contribuire a migliorarlo, voglia documentarlo o semplicemente voglia scoprirne il funzionamento.

Android, inoltre, è completamente sviluppato in linguaggio di programmazione Java, e permette di interfacciarsi con il sistema operativo, di controllare le risorse hardware del dispositivo mobile e di sviluppare delle Android GUI (Graphic User Interface). La possibilità di combinare l'espressività della comunicazione FIPA, supportata dagli agenti JADE, con la potenza della piattaforma Android, apporta un notevole contributo allo sviluppo di applicazioni innovative basate sui modelli sociali. L'add-on JADE-Android è stato progettato per supportare questo tipo di applicazioni. In questo senso un'applicazione Android può essere facilmente incapsulata in un agente JADE e diventare parte di un più ampio sistema distribuito il quale può includere anche altri dispositivi mobili, non necessariamente Android. Entrando nel dettaglio, l'add-on fornisce un'interfaccia che permette all'applicazione di avviare un agente locale, innescare behaviour e, più in generale, scambiare oggetti con esso. E' possibile individuare, quindi, peer remoti, effettuare complesse conversazioni con essi, sfruttando il supporto dell'ontologia JADE per gestire messaggi strutturati, eseguire attività in background, in accordo con il modello della composizione di behaviour e,

in generale, utilizzando le capacità della piattaforma JADE. Al fine di rendere compatibile la Dalvik virtual machine di Android, e di far fronte adeguatamente alle limitazioni e ai vincoli di dispositivi mobili e reti wireless, l'add-on fa uso della versione JADE-Leap per Java CDC (o Personal Java) e della modalità di esecuzione split-container.

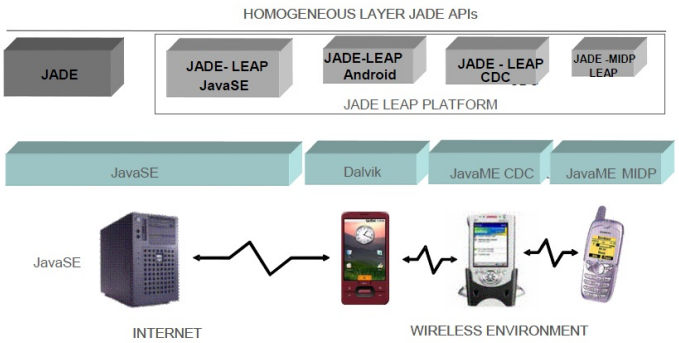

Figura 7: Livelli API JADE

2.6 Limiti degli agenti Jade sui dispositivi mobili

Quando si distribuiscono agenti su dispositivi mobili, come PDA o smartphone, bisogna tenere in considerazione diversi tipi di vincoli.

2.6.1 Limiti Hardware

Tipicamente, I dispositivi mobili hanno una ridotta potenza computazionale su processori a 16 bit e, in generale, la frequenza di clock è inferiore ai 200 MHz. Essi sono anche caratterizzati da limiti in termini di memoria, la quale può variare dai 64 Kb delle schede SIM ad approssimativamente 16 MB negli smartphone e 64 MB nei PDA (Dati relativi al 2007). Anche se le prossime generazioni di smartphone supereranno questi limiti, e il divario tra le capacità di laptop e desktop si assottiglierà, gli sviluppatori, per andare incontro alle aspettative degli utenti, dovranno trattare la memoria e la potenza computazionale come preziose risorse.

2.6.2 Limiti di Java

La Java Virtual Machine, disponibile sui dispositivi portatili, non fornisce le funzionalità complete presenti sui computer portatili e sui desktop. In particolare, la specifica MIDP (l'unica comunemente disponibile sulla maggior parte degli smartphone) non fornisce il supporto all'accettazione di connessioni in ingresso. Anche nei PDA, dove la JVM è conforme sia al profilo personale della configurazione CDC sia alle vecchie specifiche Personal Java, l'opportunità di

aprire connessioni verso un dispositivo non è garantita. Ciò è dovuto alla pratica degli operatori telefonici di utilizzare firewalls. Altri limiti degni di nota nelle specifiche MIDP riguardano la mancanza della Java Reflection, della Serializzazione e del Collection Framework. Inoltre, la Java Virtual Machine ha un supporto limitato per la grafica e le interfacce utente.

2.6.3 Limiti delle reti

JADE assume che ci sia piena e continua connettività fra i containers. Questo significa che ogni containers deve essere in grado di aprire e accettare in ogni momento connessioni di rete da e verso ogni altro container della piattaforma. Ma in JADE le interazioni container-to-container avvengono tramite il protocollo RMI, che non è particolarmente efficiente in termini di quantità bytes trasmesse sulla rete. Sfortunatamente le reti wireless quali GPRS e UMTS non rispettano queste assunzioni e sono caratterizzate da:

- Connettività intermittente dovuta al fatto che alcune zone non sono ben coperte dal segnale, aree protette, e dal bisogno di spegnere il dispositivo per conservare la durata della batteria.
- Indirizzo IP variabile, dovuto al fatto che gli indirizzi IP dei dispositivi mobili non sempre rimangono gli stessi dopo una disconnessione temporanea.
- Alta latenza della rete e bassa larghezza di banda, che varia da 9,6 kbps (GSM) a 128kbps(GPRS) e 1Mbps(HSDPA) e oltre.

3 Caso di studio

Il caso di studio che abbiamo deciso di implementare per testare il framework JADE è lo sviluppo di un sistema per la prenotazione a distanza di un parcheggio. Nel nostro scenario saranno presenti tre parcheggi, ognuno dei quali metterà a disposizione un numero arbitrario di parcheggi, con una determinata tariffa oraria. L'utente, tramite smartphone con sistema operativo android, avrà la possibilità di prenotare un posto auto nei tre parcheggi. In particolare, gli sarà suggerito il posto migliore, in base alla tariffa oraria più bassa e al numero di posti disponibili. L'utente, infine, avrà la possibilità di accettare o meno il posto suggerito, e in caso di accettazione gli sarà fornito un servizio di navigazione verso il parcheggio scelto.

3.1 Architettura del sistema

Per implementare questo caso di studio dobbiamo tenere conto dei limiti tecnologici dell'estensione JADE-LEAP esaminati nel capitolo 2.6.

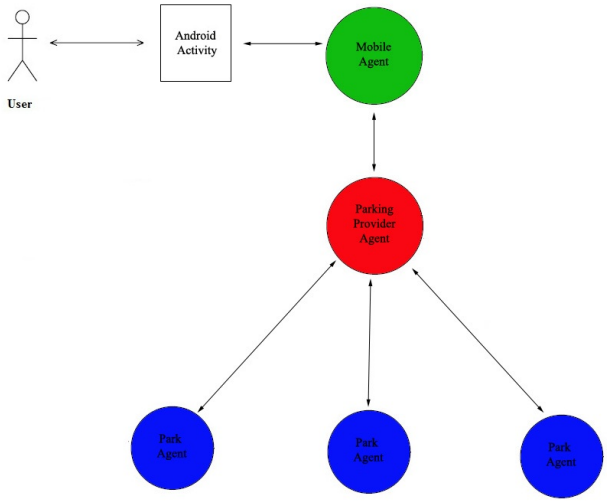

Figura 8: Architettura del sistema

Il sistema risulta essere composto da 3 tipi di agente e da 2 activity android.

Mobile Agent: agente avviato dallo smartphone Android dell'utente. Fa uso di activity Android per l'interfacciamento con l'utente le quali gli permettono di ricercare il servizio, di richiedere un parcheggio e di prenotarlo. Ogni comando impartito dall'utente verrà eseguito dal Mobile Agent che si occuperà dello scambio di messaggi con il ParkingProvider.

Parking Provider: agente che fornisce il servizio di suggerimento del miglior parcheggio all'utente. Rimane in attesa di richieste da servire da parte dei MobileAgent ed aggiorna continuamente le informazioni relative a ogni singolo parcheggio interloquendo con i ParkAgent. A fronte di una richiesta calcola il miglior parcheggio disponibile sulla base delle informazioni a disposizione in particolare rispetto al costo e alla disponibilità dei parcheggi. Il ParkingProvider fa anche da tramite per la comunicazione della prenotazione dell'utente al relativo ParkAgent.

Park Agent: c'è un agente di questo tipo per ogni parcheggio. Questi agenti rispondono alle richieste di prenotazione inoltrate dal ParkingProvider oppure alle richieste di informazioni generali. Nel primo caso viene indicato il numero del parcheggio prenotato mentre nel secondo il messaggio di risposta contiene il costo ed il numero di posti disponibili.

In seguito si entrerà con maggiore dettaglio nella descrizione delle componenti del sistema e delle interazioni tre esse.

3.2 Ricerca degli agenti e servizi offerti

Al fine di rendere reperibili i servizi offerti, i ParkAgent si registrano al servizio di pagine gialle implementato dal DFService disponibile nativamente in Jade. All'atto della registrazione vengono indicate le coordinate relative alla posizione del parcheggio ed altre informazioni, quali l'identificativo del servizio offerto ("book-park") e dell'agente che espone il servizio. All'avvio, il ParkingProvider ricerca tutti i servizi di tipo ("book-park") memorizzando per ciascuno di essi l'identificativo dell'agent in modo da poterli contattare in seguito; inoltre si registra anch'esso alle pagine gialle in modo da rendersi reperibile ai MobileAgent. A differenza dei ParkAgent il servizio offerto viene identificato dalla stringa ("book-service") e non ha bisogno di specificare coordinate relative alla posizione.

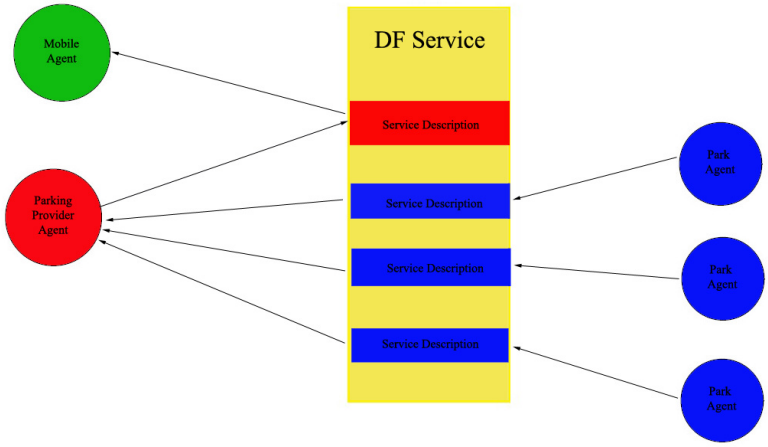

Figura 9: Uso del DFService nel sistema

Activity Android: Un activity è il componente principale di android per visualizzare le informazioni e ricevere input dall'utente.

La MobileAgentActivity offre la possibilità all'utente di accedere ai servizi del sistema, quindi di prenotare il miglior parcheggio in relazione al costo e alla disponibilità, tramite appositi pulsanti. Alla pressione dei vari pulsanti, vengono scatenati diversi eventi, gestiti da un handler (GuiHandler), il quale associa ad essi diversi tipi di comandi che verranno eseguiti dal Mobile Agent. Il comando tipico prevede l'esecuzione di un behaviour.

La MapViewerActivity viene avviata alla pressione di un apposito pulsante e mostra all'utente il percorso più breve per raggiungere il parcheggio selezionato dal sistema, a partire dalla posizione attuale dello smartphone.

Mobile Agent: Il Mobile Agent è un Gateway Agent che permette alla MobileAgentActivity di comunicare con gli altri agenti del sistema. Viene creato all'avvio dell'applicazione quando quest'ultima si connette alla piattaforma JADE-LEAP. A seconda del pulsante premuto, l'activity impartisce un differente comando al MobileAgent tramite il metodo processCommand:

- Discover Service. Il behaviour eseguito è il "SearchingSearvicesBehaviour" che invia al DFService la descrizione del servizio "book service" da ricercare ed attende il messaggio con l'esito della ricerca. Non è stato adottato un tipo di ricerca bloccante perchè provocava il crash del MobileAgent stesso.

- Request. Il behaviour eseguito è il "ReceiveParkBehaviour", il quale invia un messaggio ACL di tipo REQUEST al ParkingProvider e si mette in

attesa del messaggio INFORM che contiene le informazione del parcheggio: costo, identificativo del parcheggio (l'AID del relativo agente), e coordinate geografiche.

- Accept. La pressione di questo pulsante comporta l'esecuzione del behaviour "ReceiveConfirmBehaviour", il quale prevede l'invio dell'accettazione della proposta tramite il messaggio ACCEPT-PROPOSAL e la ricezione della risposta, che può essere un messaggio REFUSE oppure CONFIRM. Nel primo caso il parcheggio suggerito non ha più posti disponibili, perciò l'utente si vede costretto ad effettuare nuovamente la richiesta. Nel secondo, al contrario, i posti disponibili ci sono ed il messaggio contiene il numero del posto che è stato riservato all'utente.

Parking Provider: Il Parking Provider ha un comportamento che dipende, in parte, dalle interazioni con il MobileAgent e, in parte, rimane sempre lo stesso. Infatti, quando il Parking Provider non sta servendo richieste, i behaviour attivi sono :

- "UpdateBehaviour". Ogni 500ms avvia l'aggiornamento delle informazioni locali che comporta l'esecuzione di "UpdateRequestBehaviour" e "WaitResponses".
- "UpdateRequestBehaviour". Invia un messaggio REQUEST a tutti gli agenti ParkAgent rilevati all'avvio.
- "WaitResponses". Prevede l'attesa dei messaggi INFORM di risposta da ciascun ParkAgent e l'aggiornamento delle informazioni locali con quelle appena ricevute.
- "ReceiveBehaviour". È il behavior che mette l'agente in condizione di ricevere richieste dai MobileAgent e di servirle.

Il parking provider, come già accennato, fa anche da tramite per la comunicazione tra utente e ParkAgent durante l'effettiva prenotazione del posto. Queste funzionalità vengono realizzate dai seguenti behavior:

- "WaitParkAcceptBehaviour". Attivato dopo aver servito una REQUEST, questo behaviour si mette in attesa di un messaggio ACCEPT-PROPOSAL che abbia come mittente lo stesso agente che ha fatto la richiesta. Una volta ricevuto il messaggio, esso viene inoltrato al ParkAgent inizialmente suggerito. Successivamente viene attivato il behaviour "WaitParkConfirmBehaviour".
- "WaitParkConfirmBehaviour". Attende la ricezione del messaggio CONFIRM e lo inoltra al MobileAgent

Park Agent: Il comportamento di questi agenti è molto semplice, perciò è stato scomposto in sole due parti:

- “ReceiveBehaviour”. Risponde alle richieste di aggiornamento inviate dal ParkingProvider con un messaggio INFORM contenente i posti disponibili e la tariffa oraria.
- “BookNotification”. Risponde alle richieste di prenotazione inoltrate dal ParkingProvider. A seconda della disponibilità di posti, invia un messaggio di CONFIRM che contiene il numero del posto riservato, oppure un messaggio REFUSE.

Entrambi sono attivi fin dall’avvio del ParkAgent

4 Conclusioni

L'idea, presente in Jade, di adottare un Gateway Agent per permettere ad un sistema multiagente di interagire con qualsiasi altro sistema non ad agenti, come Android, si è rivelata, contemporaneamente, semplice ed estremamente efficace. Le API del Gateway Agent sono molto limitate ma, allo stesso tempo, non precludono nessuna possibilità al programmatore, che sarà libero di implementare qualsiasi tipo di funzionalità di cui abbia bisogno. Queste caratteristiche sono fondamentali nell'ottica di un futura evoluzione delle applicazioni al paradigma ad agenti, e di una loro integrazione con i sistemi esistenti, in particolare con quello dei dispositivi mobili che si stanno, attualmente, diffondendo in maniera esponenziale. Nonostante le limitazioni tecnologiche attuali, JADE Leap si è rivelato un framework molto agile, che permette un rapido sviluppo di applicazioni ad agenti nell'universo mobile. L'innovazione a livello hardware che il settore mobile sta apportando, come l'introduzione di processori dual core, renderanno superate le limitazioni attuali di JADE Leap, rendendola una piattaforma appetibile anche per lo sviluppo di applicazioni molto più complesse di quella da noi sviluppata. Il caso di studio, inoltre, lascia aperte le porte a possibili miglioramenti dell'applicazione, come l'adozione di un algoritmo più complesso per l'individuazione del parcheggio migliore, che tenga conto non solo del costo e della disponibilità di parcheggi, ma anche della attuale posizione dell'utente.