UNIVERSITÀ DEGLI STUDI DI TRENTO
FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea in INFORMATICA

Elaborato Finale

PROGETTAZIONE E SVILUPPO DI UN SISTEMA MULTI-AGENTE PER SERVIZI DI CARPOOLING ACCESSIBILI DA DISPOSITIVI MOBILI

Relatore: ing. Paolo Giorgini

Laureando: Sottini Francesco

ANNO ACCADEMICO 2005 - 2006

Ai miei genitori

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Ridurre il numero di auto circolanti in città a favore di mezzi di trasporto alternativi, migliorando così l'accessibilità dei centri urbani e diminuendo il grado di concentrazione di sostanze inquinanti è l'obiettivo principale della mobilità sostenibile.
La mobilità sostenibile ha infatti come obiettivo la gestione della domanda di mobilità soprattutto in rapporto ai spostamenti sistematici, sviluppando e implementando concetti e strategie orientati ad assicurare la mobilità delle persone e il trasporto delle merci in modo efficiente, con particolare attenzione a scopi ambientali, sociali e di risparmio energetico.
Tra gli interventi che è possibile adottare, uno è il carpooling, cioè uno scambio di passaggi automobilistici tra i cittadini in modo da favorire la condivisione delle loro automobili per gli spostamenti in città. I sistemi proposti ed applicati fino ad ora per offrire questo tipo di servizio si sono rivelati però inefficaci perchè soggetti a diversi problemi legati principalmente alla loro limitata accessibilità, alla mancanza di una infrastruttura di supporto per l'interazione, la collaborazione e/o la competizione tra gli utenti, senza riuscire a soddisfare la grande necessità di sistemi di supporto alla mobilità cittadina.

Gli utenti che potrebbero beneficiare di un servizio di carpooling solitamente sono quelli che presentano desideri e interessi simili e che quindi frequentano abitualmente gli stessi ambienti e luoghi, presentando un carattere localizzato e costituendo così una potenziale comunità. Ma una comunità di persone per esistere ha bisogno di poter interagire e comunicare al suo interno e quindi è necessario uno strumento che permetta alle persone di mettersi in contatto tra di loro, come ad esempio i cellulari. I dispositivi mobili, in particolare i cellulari, sono l'esempio ecclatante del boom che il settore delle comunicazioni ha avuto nell'ultimo decennio; oggi è quasi impensabile trovare infatti una persona che ne sia sprovvista. Di pari passo con la loro diffusione, sono aumentate anche le loro potenzialità, sia hardware che software, implementando nuove tecnologie che non limitano questi dispositivi alle semplici comunicazioni telefoniche ma permettono di avere in mano dei piccoli Personal Computer con tecnologie di comunicazione e funzionalità avanzate. Questa evoluzione tecnologica ha comportato anche la creazione di nuovi scenari di utilizzo dei dispositivi mobili come strumento per realizzare ad esempio comunità virtuali mobili dove gli utenti comunicano, si scambiano informazioni e interagiscono anche senza conoscersi personalmente; si presenta allora la necessità di un sistema che supporti queste comunità come i sistemi multiagente.
La programmazione ad agenti è un nuovo ambito di ricerca e sviluppo all'interno della Computer Science; essa permette all'utente di delegare ad una entità software, cioè ad un agente,

i propri desideri e interessi; l'agente opererà in seguito per soddisfare l'utente comportandosi effettivamente come farebbe una persona che deve interagire con altre persone.

In questo contesto è nata l'idea per questo lavoro, cioè la progettazione e lo sviluppo di un sistema multi-agente per servizi di carpooling accessibili da dispositivi mobili utilizzando tecnologie disponibili su larga scala, affidabili e gratuite. Il lavoro svolto ha come obiettivo principale il superamento dei problemi che affliggono le proposte viste fino ad ora, studiando e progettando un modello di interazione tra agenti ad hoc per il nostro caso e implementando un prototipo di sistema carpooling. Quindi ciò si traduce in ottenere una buona accessibilità fornendo la possibilità di accedere al servizio ovunque con il proprio cellulare attraverso la tecnologia di comunicazione Bluetooth e offrire un sistema di supporto all'utente per l'interazione con altri utenti attraverso una piattaforma multiagente dove gli agenti lavorano per l'utente in modo autonomo, proattivo e/o propositivo contrattando, collaborando o entrando in competizione con altri agenti.
Si è ottenuto così un sistema di carpooling basato sugli agenti, accessibile da dispositivi mobili dove la tecnologia Bluetooth è adottata per riflettere la località dell'utente. Ciascun utente è rappresentato da un agente personale che interagisce con gli agenti personali di altri utenti all'interno di comunità virtuali mobili per trovare oppure offrire un passaggio automobilistico da un luogo preciso in un preciso istante.

Il lavoro si articola come di seguito: il primo capitolo tratterà il contesto sociale in cui un sistema come quello presentato si inserisce, le comunità virtuali mobili e le tecnologie che le supportano; verranno inoltre discussi i tools esistenti e i loro limiti. Il secondo capitolo introdurrà il framework multiagente, parte principale del sistema realizzato, analizzando le varie fasi contenute nei processi di elaborazione interni. Il terzo capitolo descriverà invece ANDIAMO, il sistema sviluppato; verranno quindi descritte la sua architettura generale e le sue componenti, offrendo alcuni possibili scenari in cui il software prodotto potrebbe risultare utile agli utenti. Infine nell'ultimo capitolo verrà fatto il punto della situazione sul progetto ANDIAMO, traendone le conclusioni e proponendo possibili ulteriori sviluppi futuri del sistema.

Capitolo 1

Ambiente operativo, comunità virtuali e tecnologie di supporto

In questo capitolo verrà illustrato il contesto sociale e tecnologico in cui questo lavoro si inserisce e si introdurrà il concetto di comunità virtuale mobile. Verrà inoltre fatta una panoramica sugli agenti e sui sistemi multi-agente, introducendo la comunicazione Bluetooth con una rapida panoramica sui sistemi attualmente esistenti di supporto alle comunità virtuali che utilizzano queste tecnologie.

1.1 Mobilità sostenibile

La **mobilità sostenibile** è un'idea, un concetto, un movimento nato dalla reazione ad alcuni fattori che sono radicalmente e visibilmente peggiorati in seguito a politiche dei trasporti e di sviluppo sbagliate messe in atto dalla seconda metà del XX secolo in avanti (insostenibile richiesta di risorse, inquinamento atmosferico, livello dei servizi diminuito nonostante l'incremento degli investimenti). La mobilità sostenibile ha come obiettivo la gestione della domanda di mobilità soprattutto in rapporto agli spostamenti sistematici, sviluppando e implementando concetti e strategie orientati ad assicurare la mobilità delle persone e il trasporto delle merci in modo efficiente, con particolare attenzione a scopi ambientali, sociali e di risparmio energetico. L'obiettivo principale è quello di ridurre il numero di auto circolanti a favore di mezzi di trasporto alternativi, migliorando così l'accessibilità dei centri urbani e diminuendo il grado di concentrazione di sostanze inquinanti.

Le tecniche di gestione della mobilità iniziano ad affermarsi agli inizi degli anni '90 negli Stati Uniti ed in alcuni Paesi europei, quali il Belgio, la Gran Bretagna, l'Olanda e la Svizzera. In Italia la gestione della mobilità cittadina è stata ufficialmente introdotta attraverso il decreto del Ministero dell'ambiente del 27 marzo 1998 sulla *Mobilità sostenibile nelle aree urbane* (pubblicato sulla G.U. 3 agosto 1998), con la figura del mobility manager aziendale[14]. Questi ha il compito di individuare e gestire i piani degli spostamenti casa/lavoro del personale dipendente al fine di ridurre l'uso dei mezzi di trasporto privati e individuali e di limitare la

congestione del traffico urbano attraverso una razionale organizzazione degli orari di lavoro. È prevista l'obbligatorietà del piano per le strutture aziendali pubbliche e private con più di 300 dipendenti per unità locale e per le imprese con complessivamente oltre 800 dipendenti. Il decreto del Ministero dell'Ambiente, datato 20 dicembre 2000 (G.U. 5 aprile 2001)[15], incentiva l'implementazione del Mobility Management attraverso il finanziamento a Comuni e/o a forme consociate di Comuni, non solo per interventi relativi agli spostamenti casa-lavoro, ma anche in riferimento a piani per la gestione della domanda di mobilità riferiti ad aree industriali, artigianali, commerciali, di servizi, poli scolastici e sanitari o aree che presentano, in modo temporaneo o permanente, manifestazioni ad alta affluenza di pubblico.

I vantaggi derivanti dall'applicazione di politiche di mobilità sono molteplici, coinvolgono diversi attori e possono essere riassunte come di seguito:

Vantaggi per il dipendente:

- Minori costi di trasporto
- Riduzione dei tempi di spostamento
- Possibilità di premi economici
- Riduzione dei rischi di incidenti
- Maggiore regolarità nei tempi di spostamento
- Minore stress psicofisico da traffico
- Aumento delle facilitazioni e dei servizi per coloro che già utilizzano modi alternativi
- Socializzazione tra colleghi

Vantaggi per l'azienda:

- Una migliore accessibilità all'azienda rappresenta un valore aggiuntivo
- Riduzione dei costi dei problemi legati ai servizi di parcheggio
- Migliori rapporti con gli abitanti dell'area circostante l'azienda (più parcheggi e meno rumore)
- Riduzione dei costi per i rimborsi accordati sui trasporti
- Riduzione dello stress per i dipendenti e conseguente aumento della produttività
- Riduzione dei costi dei trasporti organizzati e pagati dall'azienda
- Conferimento di un'immagine aziendale aperta alle problematiche ambientali
- Promozione di una filosofia aziendale basata sulla cooperazione

Vantaggi per la collettività:

- Riduzione dell'inquinamento atmosferico
- Benefici in termini di sicurezza
- Riduzione della congestione stradale
- Riduzione dei tempi di trasporto

Una vincente politica di gestione della mobilità si basa su un buon piano degli spostamenti. Vi è una sequenza di fasi che ne permettono una corretta stesura ed attuazione, come ad esempio la fase informativa e di analisi, molto delicata ed importante, che permette di raccogliere tutte le informazioni neccessarie per inquadrare le esigenze di spostamento della comunità interessata; ma il punto più importante è la scelta di quali misure attuare per ridurre l'uso individuale dei veicoli a motore. Tra i possibili interventi che è possibile adottare, uno è la promozione del **Carpooling**, meglio conosciuto in ambito internazionale e di ricerca con il termine **Rideshare** (da qui in avanti adotteremo questo termine per identificare il servizio descritto in questo lavoro).

Il Rideshare è un metodo per ridurre l'utilizzo di automobili in una specifica città o territorio con un miglioramento delle condizioni ambientali, la riduzione dell'inquinamento atmosferico e di una serie di problemi legati al traffico cittadino.
Il Rideshare si compone di due attori: un automobilista, detto **ride-offerer** e proprietario di un automobile, che usa il proprio mezzo per muoversi da un luogo ad un altro lungo un percorso ben specificato; un'altra persona, detta **ride-seeker**, interessata ad andare da qualche parte lungo tale percorso e disposta allo stesso tempo a condividere i costi del viaggio con il ride-offerer. Situazioni del genere danno luogo ad un'interazione tra offerta e domanda di passaggi automobilistici che costituisce attualmente un mezzo affidabile di trasporto per numerose persone in un numero crescente di nazioni.

Il traffico automobilistico è una delle principali emergenze ambientali per le aree urbane al giorno d'oggi ed è un problema a cui ogni amministrazione pubblica cerca di porre soluzione. Purtroppo il numero di automobili in circolazione aumenta inesorabilmente di giorno in giorno e così anche i problemi di congestione stradale, di parcheggio e inquinamento atmosferico che rendono la qualità della vita peggiore.
Nella maggior parte delle aree urbane degli Stati Uniti, per esempio, la maggioranza dei viaggi automobilistici privati è compiuta da veicoli con un singolo occupante (Single Occupant Vehicle - SOV). Negli anni '90 si è stimato che approssimativamente il 90% degli spostamenti di lavoro e il 58% degli altri tragitti erano compiuti da SOV[23].

Le misure adottate negli ultimi anni ad esempio qui in Trentino sono orientate in particolare all'abbattimento delle polveri sottili all'interno dei centri urbani e promuovono il rideshare come forma spontanea e disarticolata di trasporto indicando, ad esempio, che le auto con più di 2 persone a bordo possono comunque circolare sulle vie cittadine anche se non rientrano tra quelle ammesse [7] ma i risultati ottenuti sono modesti ai fini della riduzione delle percorrenze complessive . É compito del cittadino trovarsi infatti alcuni partner con cui condividere il viaggio ma non è affatto una cosa semplice; si possono trovare infatti diverse lamentele sui forum locali a riguardo mentre si nota, tra le discussioni osservate, una grande necessità di un sistema che faccia incontrare domanda e offerta.
Il Trentino, con la sua caratteristica conformazione morfologica e territoriale, presenta una situazione dove un sistema di rideshare potrebbe essere molto di aiuto. Infatti il numero di pendolari che ogni giorno si portano, ad esempio, nel capoluogo di provincia da ciascuna val-

lata è nella maggior parte delle volte costituito da SOV; basta infatti trovarsi in coda la mattina sulle principali arterie locali per accorgersi di come moltissimi lavoratori o automobilisti viaggiano da soli con la propria automobile.
La Provincia di Trento si è accorta di questo fenomeno già da tempo ed aveva promosso in passato un sistema di rideshare basato sul web [1] che è naufragato però rapidamente perchè poco conosciuto e scomodo da usare. Ora ci riprova il Comune di Trento, promuovendo un nuovo sistema su web mirato ai dipendenti pubblici [17].
La condivisione delle automobili potrebbe essere incentivata anche premiando i cittadini che utilizzano spesso questo mezzo di trasporto alternativo. Ad esempio, supponiamo che ogni volta che un cittadino usufruisce del servizio di rideshare riceva una valutazione e quindi un punteggio; in seguito, verrà stilata una classifica e i cittadini più meritevoli oppure più fedeli potrebbero godere di alcuni vantaggi economici, come ad esempio biglietti gratuiti da utilizzare sui mezzi pubblici, parcheggio gratuito nelle aree a pagamento e così via.

La soluzione tecnica adottata dai due sistemi pubblici enunciati sopra è la medesima utilizzata dalla maggioranza dei sistemi di rideshare esistenti in altre realtà italiane ed internazionali; essi sono solitamente organizzati e gestiti attraverso soluzioni tecnologicamente basate sul web o addirittura utilizzando un operatore come terza parte che si occupa di far combaciare le richieste degli utenti.
Queste soluzioni soffrono però di alcuni problemi congeniti che ne impediscono un utilizzo esteso da parte della popolazione come la necessità di essere sempre collegati ad internet per poter usufruire del servizio ed altri ancora, ma principalmente legati all'accessibilità dei sistemi(es. privacy, sistemi prettamente informativi, corretto abbinamento delle richieste, ecc).
Perciò la soluzione più idonea al momento è parsa essere l'utilizzo di applicazioni mobili per implementare un sistema di rideshare, cioè si sfrutta l'enorme diffusione dei dispositivi mobili(cellulari e PDAs) tra la popolazione per offrire un servizio a cui si può accedere ovunque ed in ogni momento.
In letteratura si trovano già soluzioni basate su cellulare per offrire il servizio di Rideshare. Nel lavoro di Walbridge [27] , ad esempio, la distribuzione delle celle che formano la rete telefonica mobile sul territorio è utilizzata per riconoscere dove si trova un utente e raggruppare le persone in base alla cella in cui si trovano; la località di destinazione sarà ovviamente contraddistinta da un'altra cella.

Le motivazioni principali che spingono ad offrire il servizio di rideshare sono solitamente orientate ad ottenere la riduzione dell'inquinamento atmosferico prodotto dalle automobili, il risparmio sui costi di trasporto, ad evitare la formazione di congestioni stradali e la conseguente perdita di tempo; gli evidenti vantaggi che ne derivano hanno convinto i governi nazionali di numerosi paesi ad incoraggiare l'introduzione e l'uso di sistemi di supporto al rideshare. Di conseguenza, differenti attività di ricerca vengono portate avanti in differenti parti del mondo per incoraggiare la mobilità sostenibile in generale e i servizi di rideshare in particolare.

In conclusione, bisogna evidenziare il fatto che il termine *carpooling*, spesso, è associato erroneamente al termine *carsharing*: i servizi offerti corrispondenti a questi due termini sono

difatti completamente differenti. Con il termine carsharing si identifica un servizio che permette agli utenti di condividere l'utilizzo di una o più automobili pagando una piccola quota fissa annuale oppure in base al tempo di utilizzo del servizio; il parco auto può essere composto di auto in multiproprietà oppure messo a disposizione da un ente o azienda.

1.2 Comunità virtuali mobili

Con lo sviluppo delle tecnologie per la comunicazione mobile, intesa non solo come comunicazione vocale ma anche come scambio di informazioni (sms, mms, mp3, foto e in generale ogni tipo di file) è nata l'idea di formare dei gruppi entro i quali le persone potessero interagire, confrontarsi, comunicare. A questi gruppi di persone è stato dato il nome di *comunità virtuali* poichè i membri che le compongono possono anche non conoscersi e non avere rapporti di alcun tipo al di là di quelli virtuali appunto. La particolarità di queste comunità sta nel fatto che hanno un carattere mobile perchè utilizzano dispositivi come cellulari, PDAs, notebook, smartphone, ecc. per comunicare e non sono legate al luogo in cui operano, come l'ufficio o il luogo di lavoro: esse vengono quindi chiamate *comunità virtuali mobili*[21], per distinguerle da quelle normali.
Le comunità virtuali in genere (Virtual Community - VC) hanno la caratteristica di privilegiare, nel rapporto tra due o più individui che vivono anche in paesi diversi, la condivisione di interessi comuni; ciò avviene in virtù dell'abbattimento delle barriere socio culturali che nelle comunità reali possono creare distanze comunicative. La comunicazione virtuale permette inoltre il gioco di identità e lo scambio dei ruoli; come ciò possa influire sulla psicologia di un individuo rimane una riflessione ancora poco esplorata in campo psico-sociologico; comunque, l'abilità di mascherare la propria identità può produrre effetti diversi, negativi e positivi ed è una cosa da tenere in considerazione quando si vuole offrire un servizio come il rideshare.

Le comunità virtuali forniscono informazioni a cui è possibile accedere ovunque; tali informazioni tuttavia, per essere accessibili ed usabili in ogni luogo e in qualunque momento, necessitano di una tecnologia di comunicazione appropriata che diviene l'elemento cruciale del sistema. Dalla combinazione delle tradizionali tecnologie internet con le nuove capacità delle reti mobili, si ottiene un approccio per soddifare la richiesta di accesso e connessione in ogni momento e ovunque ci si trovi. Le comunità virtuali mobili hanno così l'enorme potenziale di poter servire in qualunque momento i desideri degli utenti.

Howard Rheingold, conosciuto nella comunità scientifica per aver inventato il termine *comunità virtuale*, ha recentemente ideato anche l'espressione *Smart Mobs*[22]: con tale termine ha voluto riferirsi ad un gruppo di persone fecente parte delle più moderne comunità high-tech intelligenti in cui le persone utilizzano le molteplici tecnologie oggi disponibili per organizzarsi e coordinarsi in azioni collettive di vario genere anche se non si conoscono tra di loro. Una comunità virtuale può tuttavia essere costituita in un qualsiasi ambito e con qualsiasi tecnologia mobile: basti pensare ad un'università, in cui gli studenti con interessi sostanzialmente comuni potrebbero formare dei gruppi di discussione per argomenti specifici, per lo scambio di infor-

mazioni e appunti, per l'organizzazione di eventi culturali e sportivi. Utilizzando i dispositivi mobili, oltre all'uso tradizionale di internet, e-mail, blog, forum, potrebbero rimanere sempre in contatto e dialogare: tutto questo senza l'obbligo di conoscersi personalmente ed incontrarsi. Le prospettive aperte per il futuro dalle comunità virtuali mobili porteranno ad avere nuovi modi di comunicare, agire, fare affari come si sta gia verificando oggi con i cellulari che usano la rete di terza generazione. Diviene quindi chiara la necessità di supportare queste comunità al fine di agevolare la comunicazione interna dei membri che la compongono.

Esistono numerose proposte in letteratura di sistemi a supporto di comunità virtuali mobili in molteplici contesti; il cuore di questi sistemi, al di là della tecnologia di comunicazione adottata, solitamente si trova nella piattaforma che gestisce gli utenti e che mira al soddisfacimento dei loro desideri, sia che essi ricerchino semplici informazioni statiche, sia che necessitino di complicate interazioni e contrattazioni per ottenere il risultato finale. Un esempio di progetto di ricerca nel campo del supporto informativo ai turisti e di utenti mobili è dato da PortableCicero [2]: esso utilizza dei raggi IR posti all'entrata di ogni sala di un museo per individuare il posto esatto in cui si trova l'utente per poi poter fornire ad esso tutte le informazioni di cui necessita. L'obiettivo di questo lavoro è fornire uno strumento che faccia da cicerone al turista all'interno del museo. Infatti grazie a delle mappe precaricate sul PDA in possesso dell'utente, il sistema può mostrare una piantina della stanza con le opere presenti; a questo punto il turista può selezionare l'opera a cui è interessato e ottenere informazioni su di essa tramite un'interfaccia grafica. PortableCicero presenta un solo grande inconveniente: richiede una buona dimestichezza con i computer palmari e ciò purtroppo limita il numero di utenti che possono usufruire del servizio.

In talune proposte l'utente si aspetta che il proprio software esegua del lavoro per lui in modo autonomo, proattivo e/o propositivo contrattando, collaborando o entrando in competizione con software di altri utenti.
A situazioni come la precedente bene si addice un sistema ottenuto con la *programmazione ad agenti*; questo paradigma prevede infatti che sulla piattaforma multi-agente vivano ed operino delle entità software, dette appunto agenti, che agiscono in maniera del tutto trasparente per il bene dell'utente.

1.3 Sistemi multi-agente

La delegazione dei compiti e l'intelligenza implicano il bisogno, o meglio la necessità, di costruire sistemi di computer che possano agire effettivamente in base ai nostri comportamenti: questo signica che i computer devono essere in grado di agire indipendentemente e nella maniera che meglio interseca i nostri desideri e i nostri interessi, interagendo con altri umani o con altri sistemi. Poichè l'interconnessione e la distribuzione sono diventati il tema dominante della Computer Science, essi, con la necessità dei sistemi di soddisfare al meglio i nostri desideri, hanno portato allo sviluppo di software che possono cooperare e accordarsi (e perchè no, competere) con altri sistemi che hanno differenti interessi (perchè differenti sono gli in-

teressi delle persone per cui lavorano). Questo problema non è stato affrontato che di recente e ha portato alla creazione di un nuovo campo della Computer Science: i **sistemi multi-agente**.

Esistono varie denizioni (controverse) di **agente**; le più comuni e utilizzate sono tuttavia le seguenti:

1. Un agente è un sistema capace di azioni indipendenti nell'interesse del proprio proprietario (estrapolando i desideri che devono essere soddisfatti per raggiungere gli obiettivi piuttosto che chiederli ripetutamente) [28].

2. Un agente è un'entità software che funziona continuamente ed autonomamente in un particolare ambiente, spesso abitato da altri agenti e processi [24].

Altre denizioni concernono invece la conoscenza, l'apprendimento, la capacità deduttiva, che sono equamente importanti:

- Un agente è un'entità software che può possedere alcuni dei seguenti attributi:
  - Reattività (intuizione ed azione selettiva)
  - Autonomia (orientamento all'obiettivo, proattività e comportamenti auto eseguiti)
  - Collaborazione (lavoro con altri agenti ed entità per raggiungere un obiettivo comune)
  - Abilità di comunicazione in rapporto al livello di conoscenza (comunicazione con altre entità utilizzando un linguaggio comune)
  - Capacità deduttiva (esecuzione di task astratti utilizzando propri modelli, apprendendo situazioni o comportamenti di altri agenti)
  - Continuità temporale (persistenza dello stato e della personalità)
  - Personalità (manifestazione di attributi di un agente credibile)
  - Adattività (apprendimento e miglioramento con l'esperienza)
  - Mobilità (migrazione da un host ad un altro in modo autonomo)
- Un agente è un'entità software che esibisce (un sottoinsieme del) le seguenti caratteristiche:
  - Informazione (capacità cognitive, interazione, cooperazione, coordinazione e competizione con altre persone, agenti e processi)
  - Apprendimento (capacità di imparare autonomamente)
  - Autonomia (proattività)
  - Comunicazione
  - Mobilità (mobilità fisica o software)

Un sistema multi-agente (Multi-Agent System, MAS) è un sistema composto da un certo numero di agenti, capaci di mutua interazione. L'interazione può essere in forma di passaggio di messaggi o produzione di cambiamenti di stato nel loro comune ambiente.
L'esatta natura degli agenti è tuttavia argomento di qualche controversia. Essi sono solitamente detti essere autonomi ma in pratica, tutti gli agenti alla fine sono sotto la supervisione umana.

Inoltre, più importante è l'attività dell'agente per gli umani, più supervisione essi devono ricevere. Infatti l'autonomia completa è raramente desiderata. Si ha invece bisogno di sistemi interdipendenti.
Ai sistemi multi-agente può essere richiesta l'inclusione di agenti umani; le organizzazioni umane e le società in genere possono essere considerate come un esempio di sistema multi-agente.
Alcuni sistemi richiedono un livello di autonomia e dinamicità che, riteniamo, solo un sistema multi-agente possa offrire. Infatti, i MAS possono manifestare auto-organizzazioni e comportamenti complessi anche quando le strategie individuali di tutti gli agenti sono semplici.

Nel nostro caso, cioè l'offerta del servizio di rideshare, un sistema multiagente è fondamentale come supporto all'utente perchè il lavoro da svolgere lato server per abbinare due utenti per un viaggio non è una semplice verifica di compatibilità tra i due confrontando i loro desideri, ma un vero e proprio processo decisionale dove solo uno strumento evoluto come gli agenti può essere d'aiuto. Ciò che una piattaforma multiagente permette di ottenere è simulare effettivamente quello che avviene nella realtà tra esseri umani attraverso gli agenti a cui l'utente ha delegato le proprie necessità. Infatti tra due agenti si avvia una contrattazione con l'obiettivo di raggungere un accordo e dove è necessario fare proposte, valutare le proposte altrui e prendere decisioni, azioni che una applicazione normale non riesce a fare. I parametri che caratterizzano un viaggio sono infatti negoziabili e permettono ad un agente di cercare il partner più soddisfacente al suo caso.

In letteratura si possono già trovare alcuni tool basati su piattafome multi- agente e comunicazione wireless; è il caso di sistemi come, per esempio, MobiAgent [11] dove viene proposta una struttura basata sugli agenti che permette all'utente di accedere a diversi servizi (da ricerche nel web ad applicazioni di controllo remote) direttamente dal loro cellulare o PDA. Non appena l'utente invia una richiesta per uno specifico servizio, un agente personale è creato in un server centrale. A questo punto l'utente si può disconnettere dal sistema mentre l'agente continua a lavorare per il bene dell'utente. In seguito l'utente, avvisato tramite sms, può decidere di riconnettersi ed eventualmente scaricare i risultati prodotti.
I sistemi multi-agente non sono una novità neanche nell'ambito dei trasporti e sistemi di trasporto intelligente (Intelligent Transport Systems - ITS); come illustrato in [18], i sistemi multi-agente hanno una forte influenza nell'analisi e descrizione di sistemi per la regolazione del traffico, aumentando l'autonomia dei componenti che lo compongono e integrando strutture come, per esempio, un centro per la gestione delle emergenze collegato ai moduli deputati agli incidenti, inquinamento e supporto decisionale.
Nel 1995, ancora Edward Walbridge [27] aveva calcolato che il risparmio energetico in seguito alla promozione e diffusione del ridesharing poteva ammontare a 48 millioni di barili di petrolio all'anno. La conseguente riduzione delle congestioni stradali è stato stimato permetteva di recuperare tempo lavorativo degli automobilisti per un importo pari a 6.2 miliardi di dollari ogni anno. Walbridge cercò di esplorare la possibilià di poter controllare i posti a sedere liberi all'interno delle auto degli automobilisti, la realizzazione di strumenti per la condivisione di passaggi automobilistici e il movimento delle persone in una particolare area; tutto questo

attraverso l'uso di computers posizionati in ciascuna area urbana che si occupano di abbinare passeggeri e automobilisti in tempo reale e accessibili attraverso dispositivi mobili come già in precedenza descritto.
Nel 1999, D.J. Dailey e D. Meyers [16] hanno presentato il Seattle Smart Traveler, applicazione basata su tecnologia web per testare il concetto di matching dinamico e automatico di passaggi automobilistici. Diversi studenti hanno poi dimostrato attraverso questo modello che il processo di rideshare è una funzione quadratica del numero di utenti partecipanti. Recentemente, Claudio Cubillos, Claudio Demartini e Franco Guidi-Polanco [12] hanno utilizzato un approccio al problema differente: basato su una Contract-Net Protocol, processi di filtraggio fissati e l'uso di una struttura ad agenti, hanno presentato un modello di progettazione della mediazione per la programmazione di richieste di viaggio sotto un sistema di trasporto passeggeri. Xi SHI, Jianming HU, Yangsheng XU and Jingyan SONG [29] hanno invece presentato un sistema dinamico di navigazione stradale basato su una rete di rilevatori posti sulle vie cittadine che guida l'automobilista attraverso comunicazioni Bluetooth. Si evidenziano infine, MADARP [13] che è un esempio di architettura basata sugli agenti che implementa un sistema di carsharing e GENGHIS [20] invece che è un classico sistema basato su web che fornisce il servizio rideshare dove ogni utente è rappresentato da un agente. In quest'ultimo sistema vi è un super agente che è responsabile dell'abbinamento delle richieste degli utenti ma la decisione finale è lasciata comunque all'utente.
Quello che realmente ancora manca ai sistemi presentati fino ad ora relativi al rideshare è però la possibilità di supportare servizi che richiedono l'interazione, la collaborazione e la competizione fra gli agenti membri della comunità virtuale.

Con l'utilizzo del paradigma ad agenti, come del resto anche utilizzando altri approcci, l'utente ha la necessità di osservare il lavoro per lui prodotto dal suo agente personale (d'ora in poi *personal agent*). Questo lavoro può chiaramente essere osservato solo attraverso dei messaggi inviati dall'agente (o da una qualche altra entità facente parte del sistema) all'utente. Quando si lavora a livello di simulazione o comunque lavorando sul proprio PC, le informazioni possono essere tranquillamente osservate a video; quando invece si parla di Smart Mobs e di utenti mobili, risulta chiara la necessità di far giungere le informazioni a coloro che le desiderano in altro modo. Ed ecco quindi l'utilizzo di altre tecnologie per le comunicazioni mobili come il WAP, il GPRS, l'UMTS e il Bluetooth. Proprio su quest'ultima si focalizza la nostra attenzione: il Bluetooth infatti, per la sua portata limitata, consente lo scambio di messaggi solo a quegli utenti che si trovano nelle vicinanze di un punto di accesso (access point) attribuendo al sistema un carattere prettamente localizzato e permettendo quindi di fornire servizi ad hoc, specifici per ogni luogo e per ogni tipologia di utente.

1.4 Comunicazioni Bluetooth

I dispositivi mobili supportano diversi modi con cui scambiare dati e informazioni fra di essi con costi di utilizzo, prestazioni e tecnologie differenti. Uno di questi modi è l'utilizzo di

una tecnologia wireless chiamata Bluetooth. Essa si distingue dalle altre tecnologie wireless utilizzabili (IR, WAP, GPRS, UMTS, Wi-Fi) per vari fattori:

- carattere tipicamente localizzato, essendo il raggio di azione limitato
- costi di connessione nulli a differenza di altre ,WAP-GPRS-UMTS, per cui si deve pagare un prezzo che varia a seconda della compagnia telefonica e della quantità di dati trasportati
- grande diffusione (a bassi costi) che questa tecnologia sta avendo nel settore dei dispositivi mobili

In questa sezione verrà data una breve introduzione alla tecnologia Bluetooth; per i protocolli di comunicazione vedere l'appendice A di questo lavoro.

Cenni storici

Le prime ricerche sulla tecnologia Bluetooth partirono nel 1994, quando la Ericsson Mobile Communication intraprese una serie di studi allo scopo di trovare delle alternative wireless per il collegamento tra telefoni e accessori (es. auricolari). Lo studio riguardò un collegamento radio che, non essendo direzionale, non necessitava della cosiddetta line of sight, cioè della visibilità diretta ed aveva quindi degli evidenti vantaggi rispetto alle tecnologie infrarossi precedentemente utilizzate per collegare tra loro cellulari e dispositivi vari.
Il nome è stato ispirato da Harald Blåtand, ovvero re Aroldo I di Danimarca, abile diplomatico. Gli inventori della tecnologia devono aver ritenuto che fosse un nome adatto per un protocollo capace di mettere in comunicazione dispositivi diversi.

Nel febbraio 1998 nasce il SIG (Special Interest Group), un gruppo di compagnie leader nel settore delle telecomunicazioni ed elettronica, tra cui Ericsson, Nokia, Intel, IBM e Toshiba, che lavorano assieme per promuovere e definire le specifiche Bluetooth. In seguito entrano a far parte altre aziende importanti fino a raggiungere ad oggi un numero superiore a 2000 compagnie. Nel luglio 1999 venne pubblicata la prima versione delle specifiche Bluetooth, denominata 1.0, seguita poi da numerosi aggiornamenti e nuove releases fino alla versione 2.0 attualmente in uso.

Informazioni generali

Lo standard Bluetooth è stato sviluppato con l'obiettivo primario di ottenere bassi consumi, un corto raggio di azione (da 10 a 100 metri) e un basso costo di produzione per i dispositivi compatibili. Lo standard doveva infatti consentire il collegamento wireless tra periferiche come stampanti, tastiere, telefoni, microfoni, ecc, a computer o PDA o tra gli stessi dispositivi.

Oggi il settore trainante per la tecnologia Bluetooth è quello dei cellulari; dispositivi che integrano chip Bluetooth sono venduti in milioni di esemplari e sono abilitati a riconoscere e utilizzare periferiche Bluetooth in modo da svincolarsi dei cavi.

Questa tecnologia viene utilizzata poi nei più disparati campi; ad esempio, BMW è stato il primo produttore di autoveicoli a integrare la tecnologia Bluetooth nelle sue automobili in modo da consentire ai guidatori di rispondere al proprio telefono cellulare senza dover staccare le mani dal volante. Attualmente molti altri produttori di autoveicoli forniscono di serie o in opzione vivavoce Bluetooth che integrati con l'autoradio dell'automobile permettono di utilizzare il cellulare mantenendo le mani sul volante a quindi aumentando la sicurezza della guida. Comunque lo standard include anche comunicazioni a lunga distanza tra dispositivi per realizzare delle LAN wireless.

Ogni dispositivo Bluetooth è in grado di gestire simultaneamente la comunicazione con altri 7 dispositivi sebbene essendo un collegamento di tipo master slave solo un dispositivo per volta può comunicare con il server. Questa rete minimale viene chiamata piconet. Le specifiche Bluetooth consentono di collegare due piconet in modo da espandere la rete. Dispositivi in grado di gestire due piconet e quindi in grado di fare da ponte tra le due reti dovrebbero apparire nei prossimi due anni. Ogni dispositivo Bluetooth è configurabile per cercare costantemente altri dispositivi e per collegarsi a questi. Può essere impostata una password per motivi di sicurezza se lo si ritiene necessario. Inoltre è possibile avere fino a 200 dispositivi in modalità park, cioè che non possono essere attivi nello scambio di dati con il server, ma che restano sincronizzati con esso.
Caratteristica importante dei dispositivi Bluetooth è la capacità di creare e riconfigurare dinamicamente queste piconet senza l'intervento umano e di poter essere presenti in più di una piconet contemporaneamente facendo addirittura da master in una piconet e da slave in un'altra. Un gruppo di piconet collegate fra loro è detta scatternet

Il protocollo Bluetooth lavora nelle frequenze libere di 2,45 Ghz. Per ridurre le interferenze il protocollo divide la banda in 79 canali e provvede a commutare tra i vari canali 1600 volte al secondo. La versione 1.1 e 1.2 del Bluetooth gestisce velocità di trasferimento fino a 723,1 kbit/s. La versione 2.0 gestisce una modalità ad alta velocità che consente fino a 2,1 Mbit/s. Questa modalità aumenta la potenza assorbita ma la nuova versione ha risolto il problema utilizzando segnali più brevi e quindi dimezzando la potenza richiesta rispetto al Bluetooth 1.2 (a parità di traffico inviato).
Bluetooth non è uno standard comparabile con il Wi-Fi dato che questo è un protocollo nato per fornire elevate velocità di trasmissione con un raggio maggiore, a costo di una maggior potenza dissipata e di un hardware molto più costoso. Infatti il Bluetooth crea una personal area network (PAN) mentre il Wi-FI crea una local area network (LAN). Il Bluetooth può essere paragonato al bus USB mentre il Wi-Fi può essere paragonato allo standard ethernet.

Due sono le aree di maggiore interesse per il futuro: Voice over IP (VoIP) e Ultra WideBand (UWB).
La tecnologia Bluetooth costituisce parte fondamentale nello sviluppo del VoIP. Oggi viene già impiegata nei microfoni usati come estensioni wireless dei sistemi audio dei cellulari e dei PC. Dato l'incremento in popolarità e nell'uso del VoIP, il Bluetooth potrebbe essere utilizzato nei telefoni cordless e cellulari per la connessione a Internet per effettuare una chiamata VoIP.

Tabella 1.1: Classi di potenza

Classe	Potenza	Distanza (max)
Classe 1	100 mW (20 dBm)	100 m
Classe 2	2,5 mW (4 dBm)	10 m
Classe 3	1 mW (0 dBm)	1 m

Nel Marzo 2006 il SIG ha annunciato l'intenzione di lavorare insieme ai produttori del UWB per sviluppare la futura generazione di Bluetooth che usi lo standard e la velocità della tecnologia UWB. Questo permetterà l'uso dei profili su UWB consentendo un trasferimento dati molto veloce ottimo per la sincronizzazione, il VoIP e le applicazioni video e audio, con un consumo di potenza allo stato estremamente ridotto quando il dispositivo è inattivo.

Classi di potenza

I dispositivi dotati di bluetooth si dividono in tre classi come rappresentato in Tabella 1.1.
I dispositivi di Classe 1 hanno d'obbligo il requisito di potenza, che è opzionale per le classi 2 e 3. Comunque, per un minimo consumo di potenza, è preferibile avere attivo questo controllo. Il controllo di potenza agisce tramite un ricevitore che esamina un segnale di monitorazione della potenza, l'RSSI(Receive Signal Strength Indication); in caso di segnale troppo debole viene mandato un segnale di controllo che richiede di aumentare la potenza per avere un canale efficiente, mentre in caso di segnale più forte del necessario c'è una richiesta di diminuire la potenza.

Caratteristiche suddivise per versione

Bluetooth 1.0 e 1.0B Le versione 1.0, rilasciata nel luglio 1999, e la versione 1.0B, uscita subito dopo a dicembre, sono afflitte da molti problemi e spesso i prodotti di un costruttore hanno grosse difficoltà nel comunicare con il prodotto di un'altra società. Tra lo standard 1.0 e 1.0B vi sono state delle modifiche nel processo di verifica dell'indirizzo fisico associato a ogni dispositivo Bluetooth. Il vecchio metodo rendeva impossibile rimanere anonimi durante la comunicazione e quindi un utente malevole dotato di uno scanner di frequenze poteva intercettare eventuali informazioni confidenziali. La versione B apportò anche delle modifiche alla gestione dell'ambiente Bluetooth in modo da migliorare l'interoperabilità.

Bluetooth 1.1 Nel febbraio del 2001 è stata rilasciata la versione 1.1 delle specifiche Bluetooth che risolse errori introdotti nella versione 1.0B e permise la comunicazione su canali non cifrati.

Bluetooth 1.2 Questa versione, rilascita a novembre 2003, è compatibile con la precedente 1.1 e aggiunge le seguenti novità: Adaptive Frequency Hopping (AFH), nuova tecnica nell'ambito delle comunciazioni wireless, pensata per ridurre al minimo le interferenze fra le

varie tecnologie wireless che condividono la gamma di frequenze dei 2,4 GHz. L'AFH si preoccupa di analizzare dinamicamente lo spettro alla ricerca di frequenze già occupate da altri dispositivi, come i telefoni cordless e i device Wi-Fi(802.11a/b/g) e utilizzare solo quelle disponibili: questo si traduce in un sensibile incremento delle prestazioni in ambienti dove vi possono essere numerose interferenze.

Le specifiche 1.2 hanno introdotto inoltre miglioramenti anche a riguardo della velocità di instaurazione della connessione fra periferiche. Vengono introdotti diversi nuovi metodi per la ricerca dei devices che, attraverso l'adozione di algoritmi ottimizzati, riducono i tempi necessari affinchè due o più dispositivi si vedano e siano pronti a scambiarsi dati.

Bluetooth 2.0 La nuova versione, uscita nel 2004, è retrocompatibile con tutte le precedenti versioni e offre i seguenti miglioramenti:

- Evita di saltare tra i canali per ragioni di sicurezza. Commutare tra i canali per aumentare la sicurezza non è una buona strategia, risulta relativamente semplice controllare tutte le frequenze simultaneamente, la nuova versione del Bluetooth utilizza la crittografia per garantire l'anonimato.
- Supporta le trasmissioni multicast/broadcast, consente la trasmissione di elevati flussi di dati senza controllo degli errori a più dispositivi simultaneamente.
- Enhanced Data Rate (EDR) porta la velocità di trasmissione a 2,1 Mbit/s.
- Include una gestione della qualità del servizio.
- Protocollo per l'accesso a dispositivi condivisi.
- Tempi di risposta notevolmente ridotti
- Dimezzamento della potenza utilizzata grazie all'utilizzo di segnali radio di minore potenza.

Il futuro del Bluetooth

La versione successiva del Bluetooth, chiamata in codice Lisbon, prevede l'arricchimento di caratteristiche per aumentare la sicurezza e l'usabilità del Bluetooth. Sono stati definiti i seguenti punti di sviluppo:

- Atomic Encryption Change permette un cambio di password periodico per collegamenti criptati, incrementando così la sicurezza.
- Extended Inquiry Response prevede più informazioni durante la procedura di richiesta in modo da permettere un miglior filtraggio dei dispositivi prima di effettuare la connessione. Queste informazioni includono il nome del dispositivo, una lista di servizi e altro.

- Sniff Subrating riduce il consumo di potenza quando i dispositivi sono nello stato di sniff, particolarmente sui collegamenti con flussi di dati asimmetrici. Le Human Interface Devices (HID) beneficeranno di questo profilo potendo incrementare la durata della batteria; ad esempio con mouse e tastiera si avranno miglioramenti da 3 a 10 volte rispetto l'attuale standard.
- Miglioramenti nel QoS permetterà a dati audio e video di essere trasmessi con qualità superiore.
- Simple Pairing miglioramento nel controllo dei bit attraverso parità per motivi di sicurezza.

Capitolo 2

Il framework Rideshare

L'architettura generale del sistema ANDIAMO consiste in un numero di piattaforme multi-agente accessibili da dispositivi mobili abilitati alle comunicazioni Bluetooth. L'utente può accedere al sistema anche attraverso una rete di access-points Bluetooth direttamente collegati con le piattaforme multi-agente. Per ognuna di esse è stato implementato un sistema multi-agente dove i personal agents (PA) dei ride-offerer e ride-seeker interagiscono e contrattano eventuali passaggi automobilistici. La struttura del sistema di rideshare è concepita con un approccio a livelli ed ha permesso l'implementazione del prototipo descritto nel capitolo successivo. Come mostrato in Fig. 2.1, il modello è composto da tre livelli: partendo dal più basso (1) una piattaforma ad agenti (JADE), (2) un'architettura multiagente con il modulo di Cultura Implicita e (3) un sistema vero e proprio di rideshare al livello più alto. I vari elementi che compongo ciascun livello sono descritti in dettaglio di seguito.

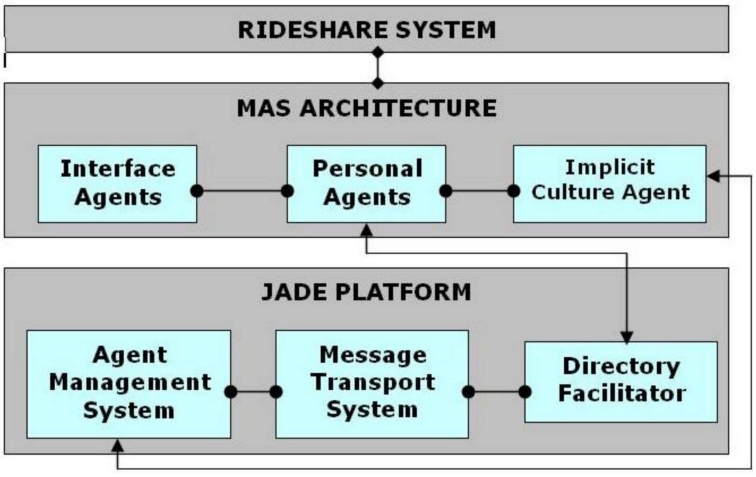

Figura 2.1: Il modello a tre livelli

2.1 La piattaforma ad agenti

La piattaforma ad agenti è implementata in JADE (Java Agent Development Framework) [3], conforme alle specifiche FIPA [4] per lo sviluppo di sistemi multiagente. JADE fornisce un supporto completo all'interazione e alla contrattazione tra i personal agents; essa implementa: (1) un sistema di gestione degli agenti (AMS- Agent Management System) che permette di avere contenitori di agenti in differenti hosts (una piattaforma distribuita), (2) un Directory Facilitator (DF) che fornisce il servizio di pagine gialle e (3) un sistema di trasporto dei messaggi (MTS-Message Transport System) che supporta la comunicazione tra gli agenti.

Ogni dispositivo mobile rappresenta nel sistema il proprio utente e ricordiamo vi è un solo personal agent per ogni utente all'interno di una piattaforma. Quindi ogni agente è identificato univocamente tramite l'indirizzo Bluetooth del corrispondente dispositivo mobile. Lo stesso dispositivo può comunque avere altri agenti su differenti piattaforme. Ogni personal agents comunica e interagisce con altri agenti in modo da trovare dei partners, per soddisfare i desideri del proprio utente, e resta attivo finchè rimane almeno una richiesta da soddisfare.

2.2 Il modello di Cultura Implicita

Tutte le piattaforme multiagente sono basate sul framework di Cultura Implicita [8].
La Cultura Implicita è basata su tecniche di Data Mining e estende tecniche di Collaborative Filtering; il suo scopo, in breve, permettere ai nuovi membri della comunità di comportarsi in accordo con la cultura della comunità stessa e dell'ambiente in cui si trovano.
Per esempio, lo studente X potrebbe voler conoscere i punti di ritrovo più utilizzati nei paraggi dell'università e il modulo della Cultura Implicita può essere adottato in una situazione del genere per supportare l'utente nel compiere delle scelte. Infatti l'idea di fondo è fare in modo che il sistema suggerisca i punti di incontro che sono stati utilizzati più di frequente in passato da altri studenti universitari (cioè, dai membri della comunità). In questo caso, il sistema potrebbe suggerire allo studente X di muoversi verso il parcheggio delle automobili vicino all'università supponendo sia il luogo più utilizzato fino a quel momento. Si intuisce quindi che era necessario integrare un modulo per il supporto della Cultura Implicita all'interno del nostro sistema dato che le situazioni in cui può rendersi utile un aiuto di questo genere sono molteplici. Ad ogni piattaforma multiagente corrisponde una comunità di utenti e quindi è fondamentale che un nuovo utente che accede al sistema possa essere guidato in maniera da comportarsi secondo la cultura della comunità stessa e ricavarne i massimi benefici. Maggiori informazioni a riguardo presso Appendice C.

2.3 Una situazione ideale

In questa sezione viene illustrato uno scenario che brevemente evidenzia il potenziale di un servizio di rideshare con utenti equipaggiati con dispositivi mobili.
Assumiamo che in Fig. 2.2 A, B e C siano tre differenti luoghi (o punti di incontro) che sono

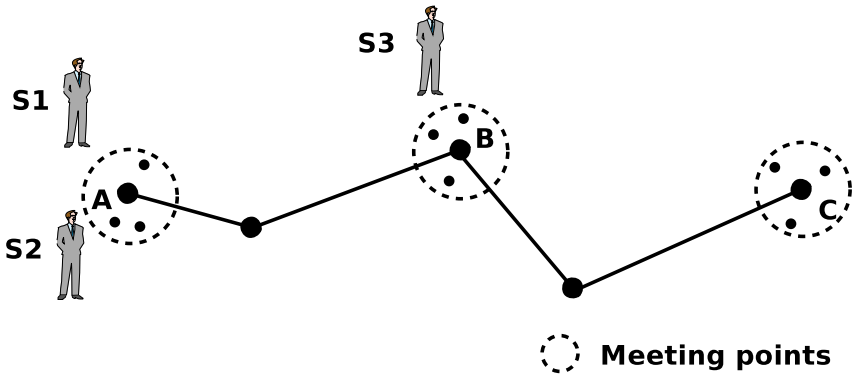

Figura 2.2: Scenario ideale di Rideshare

connessi e facilmente raggiungibili dalle automobili. I passeggeri e i proprietari d'auto sono distribuiti casualmente tra queste località e supponiamo che entrambe le tipologie di utente condividano i medesimi interessi e necessità (es. destinazione desiderata, momento e punto di partenza). Consideriamo il punto A come la stazione dei treni, B la fermata del bus e C l'università. Inoltre, S1 e S2 sono due differenti persone che si trovano in A mentre S3 è un'altra persona che si trova presso B. Queste persone hanno un unico obiettivo finale che consiste nella stessa destinazione, che assumiamo possa essere in questo caso C.
Supponiamo che S1 sia un ride-offerer, mentre S2 e S3 due ride-seeker; appare evidente che se queste tre persone si mettessero d'accordo potrebbero percorrere assieme il viaggio fino a C e l'utilizzo dell'automobile di S1 potrebbe costituire il mezzo di trasporto più vantaggioso per tutti e tre. Ovviamente B deve trovarsi sul percorso migliore tra A e C.
Tutto ciò è possibile soltanto se vi è un sistema che fa incontrare queste persone (non fisicamente, ma virtualmente) e magari facilita loro il raggiungimento di un accordo, come il sistema di rideshare basato su dispositivi mobili descritto in questo lavoro.
Alcuni aspetti dell'accordo tra gli utenti che prendono parte ad un viaggio sono chiari fin dall'inizio, come il fatto che S1 possa indicare alcune preferenze sugli utenti che desidera avere a bordo del suo mezzo e che S2 e S3 condivideranno con S1 le spese del viaggio, ma vi sono altri aspetti che rimangono in sospeso, come il modo per determinare la destinazione e quale percorso seguire per raggiungerla, la distanza e il tempo per raggiungerla, il tempo di attesa prima di ottenere un passaggio e per quanto tempo si usufruirà di questo passaggio. Tutti questi punti sono lasciati decidere al processo di negoziazione tra gli agenti, i quali prendono parte al sistema multi-agente che supporta l'applicazione presente sui dispositivi mobili.

2.4 Interface Agents

Gli agenti d'interfaccia sono programmi dedicati all'interazione con il mondo esterno alla piattaforma ad agenti; essi raccolgono informazioni da passare poi a ben determinati componenti all'interno dell'architettura multiagente. Ciascun agente d'interfaccia (IA - Interface Agent) che è implementato nel nostro framework ha tre caratteristiche principali: acquisizione di co-

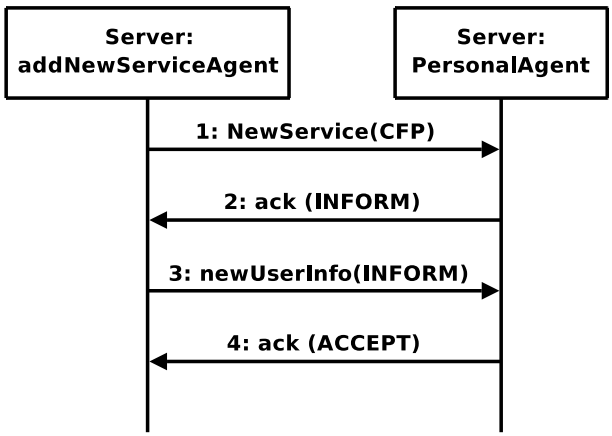

Figura 2.3: Protocollo di trasmissione dei parametri di un servizio

noscenza, autonomia e collaborazione. Questo fa si che gli IA si occupino della creazione di nuovi servizi e di dialogare con i dispositivi mobili. Questi agenti sono chiamati *AddNewServiceAgent* all'interno del nostro sistema perchè il loro compito principale è quello di ricevere le richieste di servizio dai dispositivi mobili, estrarne i parametri relativi al viaggio richiesto (indirizzo Bluetooth, informazioni sull'utente, parametri della richiesta di servizio), valutare se il servizio richiesto è fattibile e trasferire queste informazioni al corrispondente PA in modo che attivi il servizio. Bisogna evidenziare il fatto che quando gli IA ricevono una nuova richiesta di servizio da un dispositivo mobile, verificano se esiste un corrispondente PA nella piattaforma multiagente; se ciò non si verifica, un nuovo PA per quell'utente è creato. Gli agenti d'interfaccia rimangono attivi nel sistema soltanto il tempo necessario per svolgere queste operazioni e poi muoiono; ogni volta che viene ricevuta una richiesta di servizio viene creato un nuovo IA.
Il protocollo di interazione tra personal agent e interface agent è illustrato in Fig.2.3.

2.5 Personal Agent (PA)

Questo tipo di agente è considerato il cuore del servizio di rideshare; è infatti l'entità che si occupa effettivamente di cercare un partner valido, con gli stessi desideri del proprio utente. Questo agente rappresenta l'utente del sistema e lavora nel suo interesse; si comporta in pratica come farebbe un essere umano che deve interagire con altre persone per trovarsi un compagno di viaggio.
I PA su di una piattaforma ad agenti danno vita ad una comunità virtuale dove l'interazione tra questi agenti include due fasi principali: (1) l'elaborazione delle richieste dell'utente e (2) la negoziazione tra personal agents. D'ora in avanti, un PA che elabora una richiesta per un passaggio sarà chiamato **Passenger** mentre un PA che elabora una offerta di passaggio sarà chiamato **Driver**.

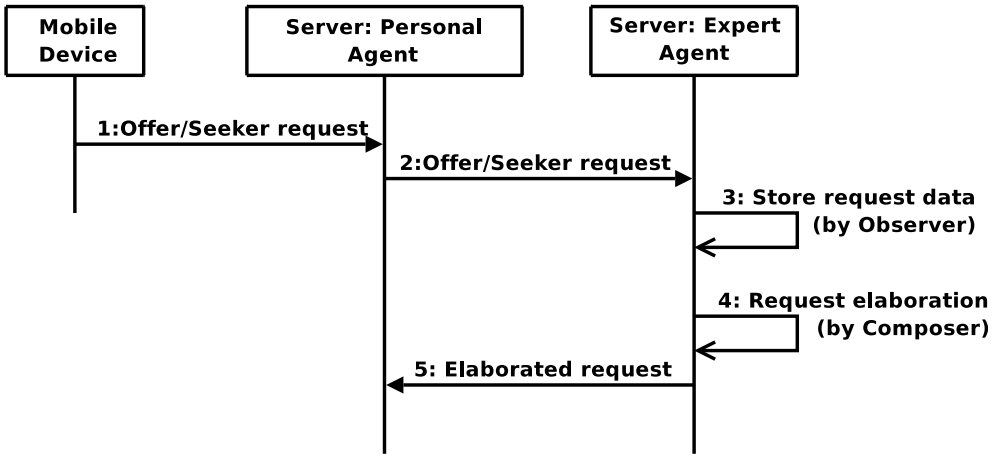

Figura 2.4: Processo di elaborazione richiesta dell'utente

Il primo passo in questa fase dell'architettura consiste nella elaborazione e correzione fin nei minimi particolari dei valori dei parametri della richiesta di servizio. Ritornando al nostro scenario ideale proposto in precedenza (Fig. 2.2), l'utente S2 potrebbe formulare la richiesta *Io chiedo per un passaggio dal luogo A al luogo B* senza specificare esattamente però i punti di ritrovo presso il luogo A, la mancanza dei quali produce una richiesta incompleta. A questo punto il PA rende la richiesta più chiara usando le informazioni riferite ai punti di incontro che altri utenti hanno recentemente e in passato indicato per la medesima località.
Un altro scenario interessante che potrebbe presentarsi è il seguente: l'utente S1 potrebbe richiedere *Io voglio sulla mia auto solo passeggeri con un valore di feedback minimo pari a 100* e allo stesso tempo non vi è un utente nel sistema che abbia un valore di feedback uguale a 100. In questo caso, l'azione che il PA compirà è ridurre il valore di feedback indicato dall'utente ma solamente dopo aver ottenuto il nullaosta dall'utente contattato tramite l'invio di un messaggio al suo dispositivo mobile.

In Fig. 2.4 viene presentato il protocollo di interazione usato dagli agenti durante la fase di elaborazione delle richieste di servizio; si evidenzia il fatto che è stato tralasciato il ruolo svolto dagli IA, dato che fanno solo da intermediari, per rendere più chiaro il protocollo.
Su ciascuna piattaforma si trova un agente dedicato, chiamato Expert Agent (EA), il quale contiene il sistema per il supporto della Cultura Implicita (SICS - System for Implicit Culture Support). Dopo che un PA ha ricevuto le richieste dal proprio utente (passo 1), egli le invia all'EA (passo 2) dove un componente, detto Observer del sistema SICS, estrae i dati dalla richiesta e li memorizza in un database dedicato ai comportamenti osservati degli utenti (passo 3). Un altro componente, detto Composer, calcola il valore corretto per i vari parametri se l'informazione ricevuta è insufficiente o sbagliata (passo 4); per fare questo, egli si basa sulle informazioni lagate alle azioni compiute dagli utenti nel passato, ricavate dall'Observer ed analizzate tramite l'Inductive Module. Infine il PA dell'utente riceve indietro la richiesta corretta (passo 5), la quale sarà elaborata durante la seconda fase.

Tabella 2.1: Parametri viaggio

Tipo richiesta	Orario partenza	Data partenza
Località di partenza	Tipo passeggero	Tipo di sistema
Località di arrivo	Offset	Punti ritrovo partenza
Feedback utente	Numero di persone	Punti ritrovo arrivo

Il sistema SICS necessita di raccogliere informazioni sul comportamento degli utenti. Nel prototipo realizzato per osservare il comportamento degli utenti, l'Expert Agent estrae i dati dalle richieste che gli pervengono dai PA. Il modulo di supporto alla Cultura Implicita può ottenere ulteriori informazioni anche da altre due fonti. La prima consiste nel database dove vengono memorizzati i risultati prodotti dall'interazione tra gli agenti; infatti ogni volta che due agenti raggiungono un'intesa per condividere un viaggio, le informazioni relative all'accordo finale sono inviate al database. Queste informazioni sono filtrate dall'EA, il quale ne estrae le più rilevanti (es, luoghi di partenza e arrivo, punti di ritrovo) e le memorizza nel proprio database interno.
La seconda sorgente di osservazione può essere anche il valore di feedback degli utenti e le valutazioni assegnate ad essi dopo aver usufruito del servizio. Il sistema, dopo che due agenti hanno stipulato un accordo, presuppone che tutto andrà a buon fine con i due partners interessati che si metteranno in contatto e condivideranno un viaggio, ottenendo ciascuno una valutazione positiva; ma ciò può non essere sempre vero. Il sistema monitorizza così come si concludono tutti gli accordi di viaggio, inviando i valori di feedback degli utenti e le valutazioni dopo un viaggio all'EA. Questo permette di avere una conoscenza sulla reputazione globale degli utenti e poter quindi suggerire correttamente in futuro per quanto concerne questi aspetti all'interno di una richiesta di servizio.

Il secondo passo compiuto in questa fase riguarda il meccanismo di interazione e negoziazione tra i personal agents; per il servizio di rideshare questo meccanismo si basa sui seguenti parametri di un viaggio (Tab. 2.1), i quali sono validi sia per un Driver che per un Passenger. I prossimi paragrafi descriveranno le varie fasi che compongono il processo di interazione tra gli agenti per raggiungere un accordo a riguardo di un viaggio; bisogna tenere in considerazione il fatto che non esiste un unico e lineare processo decisionale ma una sequenza di fasi di negoziazione, ciascuna contraddistinta da un sottoinsieme di parametri.
La possibilità di richiedere un servizio semi-automatico comporta che l'interazione tra due agenti può essere interrotta per chiedere un'opinione all'utente inviandogli una domanda.
Di seguito verranno descritte le fasi di negoziazione più significative considerando un modello con servizio automatico ma verranno comunque evidenziati i punti in cui potrebbe essere richiesto l'intervento dell'utente se il servizio fosse semi-automatico. In Fig. 2.5 viene mostrato il protocollo di interazione tipico utilizzato tra due agenti durante le varie fasi di negoziazione nell'architettura MAS.

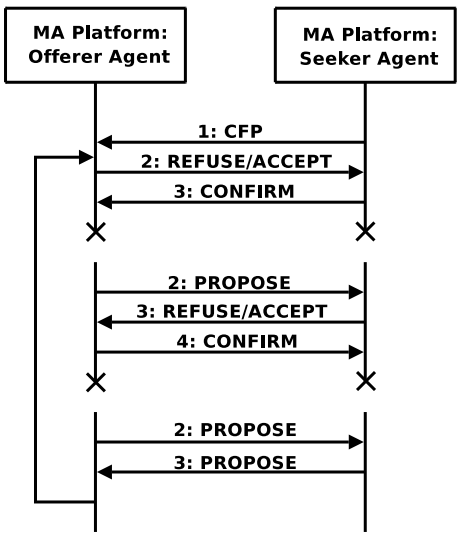

Figura 2.5: Il tipico protocollo di interazione tra PA in ANDIAMO

2.5.1 Pubblicazione del servizio

Ogni Personal agents pubblica nel Directory Facilitator di JADE la propria richiesta di servizio. Se questa richiesta è stata verificata dal sistema e corrisponde ad un'offerta di passaggi automobilistici, implica che il PA dovrà fare la parte del Driver; allora il servizio sarà registrato come di seguito :

Type =	Offer-ride
Name =	Driver-departurePlace-arrivalPlace-dayOfMonth
Property =	name = Feedback
	value = valueOfFeedback

mentre se si verifica il contrario il PA sarà un Passenger e sarà registrato nella seguente maniera:

Type =	Request-ride
Name =	Passenger-departurePlace-arrivalPlace-dayOfMonth

I parametri che contraddistinguono un servizio all'interno del Directory Facilitator sono: *Type* che indica di che tipo di servizio si tratta, offerta oppure richiesta di un passaggio; *Name* che corrisponde al nome del servizio ed è caratterizzato da una stringa composta inizialmente dal ruolo che l'utente vuol svolgere all'interno del sistema (Driver o Passenger), dal luogo di partenza, dal luogo di arrivo e dal giorno del mese in cui si vuol usufruire del servizio. Oltre a ciò, se si tratta di un Driver, viene aggiunta anche una proprietà al servizio chiamata *Feedback*, il cui valore è il feedback dell'utente fino a quel momento, non quello richiesto per il partner.

Un PA può essere attivo sulla piattaforma contemporaneamente come un Driver e un Passenger, ovviamente per due viaggi diversi.
La fase di interazione iniziale comincia con il Passenger che ricerca, all'iterno del DF, un Driver con lo stesso percorso nello stesso giorno e con un valore di feedback maggiore o uguale a quanto desiderato. Dopo ciò, il Passenger contatterà ogni Driver trovato dal sistema e gli invierà come prima cosa il proprio feedback.
In particolare, se i valori di feedback trovati dal Passenger sono inferiori a quanto desiderato, è possibile contattare l'utente per chiedergli se acconsente ad abbassare il valore di feedback richiesto in modo da aumentare le possibilità di trovare un Driver.

2.5.2 Negoziazione per l'orario di partenza

Questa fase di interazione cerca di trovare un orario comune di partenza tra un Driver ed un Passenger.
L'algoritmo 1, visualizzato di seguito, è usato per ottenere tale risultato dal lato Driver; l'algoritmo per il Passenger si differenzia soltanto nelle posizione in cui si trovano le operazioni send e receive.
Supponiamo di trovarci sempre nella situazione illustrata all'inizio di questo lavoro, cioè S1 un Driver e S2 un Passenger. Alla riga 8, S1 verifica se S2 ha già raggiunto il proprio limite temporale, cioè è arrivato ad uno dei due estremi dell'intervallo valido entro cui accettare un passaggio; ciò che permette di riconoscere questa situazione, è il fatto che S1 riceve per due volte consecutive lo stesso valore da S2. Se S2 ha effettivamente raggiunto il proprio orario limite e S1 verifica, alla riga 9, che questo orario rientra nel proprio intervallo temporale allora S1 accetta questo orario come valido altrimenti interrompe la negoziazione perchè l'agente S2 si è rivelato incompatibile. Se invece S2 non ha raggiunto il proprio orario limite, alla riga 16 S1 verifica se la sua ultima proposta è stata o meno accettata da S2.
Una caratteristica di questa fase è che se i due agenti che stanno interagendo hanno due orari di partenza inizialmente diversi, allora l'algoritmo cerca di trovare un orario comune facendo avvicinare passo dopo passo i loro orari. Questo avviene dalla riga19, dove S1 verifica se il suo orario è maggiore o minore dell'orario proposto da S2: se è maggiore, allora egli incrementa il proprio valore (righe 20 - 24) di una quantità arbitraria (all'interno dell'algoritmo chiamata *step*) oppure lo decrementa (righe 30 - 34). Alla riga 25, come alla 35, se l'orario proposto da S2 è in un periodo di *step* minuti vicino al valore di S1, allora la proposta può essere accettata altrimenti egli crea una nuova proposta da inviare a S2. Nel caso peggiore, cioè quando gli intervalli temporali dei due agenti non combaciano, l'algoritmo termina sempre quando ha inviato o ricevuto per due volte consecutive lo stesso valore.

Per esempio, se S1 indicasse come orario di partenza le ore 12.00 e S2 indicasse invece le 12.25 ed entrambi hanno fissato a 15 minuti l'offsset massimo consentito, il sistema in questo caso troverebbe un orario comune di partenza pari alle 12.10 con step di 5 minuti.
Nel sistema realizzato, l'offset temporale massimo è pari a 30 minuti; inoltre potrebbbe sembrare a prima vista che venga preso il valore medio compreso tra i due orari, ma non è così perchè il sistema deve trovare un accordo anche in un caso dove S1 ha come orario le 18.00

con offset pari a 0 e S2 ha orario 18.30 con offset uguale a 30; in questo ultimo caso l'orario comune trovato saranno le 18.00.

```
1: temp ←proposal {departure time proposed by the offerer}
2: response ←0 {departure time responded by the seeker}
3: find ←false
4: repeat
5:     send_proposal(temp)
6:     old_resp ←response
7:     response ←receive_response()
8:     if old_resp = response then {check if the seeker has already reached the limit}
9:         if (proposal − offset) ≤ response ≤ (proposal + offset) then
10:            accept(response)
11:            find ←true
12:        else
13:            return NULL
14:        end if
15:    else
16:        if temp = response then
17:            accept(response)
18:            find ←true
19:        else if response > temp then
20:            if (temp + step) ≥ (proposal + offset) then
21:                temp ←proposal + offset
22:            else
23:                temp ←temp + step
24:            end if
25:            if (temp − step) ≤ response ≤ temp then
26:                accept(response)
27:                find ←true
28:            end if
29:        else {response < temp}
30:            if (temp − step) ≤ (proposal − offset) then
31:                temp ←proposal - offset
32:            else
33:                temp ←temp - step
34:            end if
35:            if temp ≤ response ≤ (temp + step) then
36:                accept(response)
37:                find ←true
38:            end if
39:        end if
40:    end if
41: until find
42: return response;
```

Algorithm 1: Departure Time Negotiation

2.5.3 Negoziazione per la tipologia di utente

Ogni utente specifica all'interno di una richiesta di servizio una lista di tipi di partenrs preferiti con cui vuole aver a che fare; questa fase di negoziazione verifica se il partner con cui si sta

interagendo rientra tra quelli desiderati oppure no.
Di seguito possiamo vedere l'algorimo 2 d'interazione, usato sempre dal lato Driver; il lato Passenger invece può essere considerato il rovescio di quello del Driver perchè le parti dalla riga 14 alla 24 e dalla riga 4 alla 10 sono invertite.
Nell'algoritmo, con *preference* si identifica un elenco con i tipi di persone richieste dall'utente, mentre con *profile* si considera un elenco che contiene i tipi che rappresentano l'utente. L'algoritmo può essere suddiviso in due parti: la prima parte, che va dalla riga 4 alla riga 10, verifica che il partner rientri nell'elenco preference. Se questa operazione ha esito positivo, riga 11, allora inizia la seconda parte dove il Driver controlla se ha i requisiti desiderati dal partner, cioè la sua tipologia di utente è ciò che il Passenger vuole.

Per esempio, se S1 è uno studente e acccetta sulla propria auto solo professori e S3 è un professore universitario e è disposto a ricevere un passaggio anche da uno studente, si intuisce subito che questi due utenti sono compatibili, almeno per questa fase.
Infine, anche in questo caso è possibile richiedere l'intervento dell'utente inviandoli un messaggio formulato per chiedere il permesso di negoziare anche con categorie di utenti non presenti nella lista predefinita.

Require: preference {array of preference of the offerer}
profile {array of profile of the offerer}
1: OK-preference ←false
2: match ←true
3: agreement ←false
4: **for** $i = 0$ to $preference.length - 1$ **do**
5: $send\_pref(preference[i])$
6: OK-preference ←$rec\_pref()$
7: **if** $OK - preference$ **then**
8: break
9: **end if**
10: **end for**
11: **if** $\neg OK - preference$ **then** {the seeker profile is not in the preference}
12: $send\_pref(\mathsf{NOT} - \mathsf{MATCH})$
13: **else**
14: **repeat**
15: str ←$rec\_profile()$
16: **if** $str \neq \mathsf{NOT} - \mathsf{MATCH}$ **then** {check whether the seeker informs that the offerer profile is not in its preference array}
17: **if** $is\_member(str, profile)$ **then**
18: $send\_profile(true)$
19: agreement ←true
20: **else**
21: $send\_profile(false)$
22: **end if**
23: **end if**
24: **until** $(str = \mathsf{NOT} - \mathsf{MATCH}) \vee agreement$
25: **end if**
26: **return** agreement

Algorithm 2: Preference-Profile Negotiation

2.5.4 Negoziazione per i punti di ritrovo

L'interazione tra Driver e Passenger in questa fase concerne a trovare punti di ritrovo in comune per la partenza e per l'arrivo. Il messaggio che viene scambiato tra i due soggetti durante questa fase è la coppia di valori data dal nome del punto di ritrovo presso la partenza e quello presso l'arrivo:

departure - place_name — arrival - place_name

Questi valori non saranno sempre gli stessi ma varieranno durante l'esecuzione di questa fase; infatti ad ogni passaggio ciascun agente prova a proporre al partner con cui sta contrattando un nuovo punto di ritrovo in modo da veder se ve ne è almeno uno in comune. Quando i due agenti trovano un punto di ritrovo in comune sia per la partenza che per l'arrivo allora concludono questa fase di contrattazione altrimenti proseguono nell'interazione fino a che non esauriscono le proposte e quindi si rendono conto che le loro richieste di servizio sono incompatibili. Nell'eventualità non vi fossero punti in comune, solo il Passenger in questo caso può rivolgersi all'utente chiedendo se può rivedere la precedente lista grazie all'aiuto della Cultura Implicita.

Per esempio, assumiamo che S1 fornisca i seguenti punti di ritrovo per la partenza (in ordine di preferenza): *car parking - university hall - time square*. Dall'altro lato, utente S2 fornisce i seguenti punti: *research centre - library hall - university hall*.
L'interazione tra i due PA produrrà come punto di ritrovo comune per la partenza *university hall* dato che è l'unico e comunque il primo che soddisfa entrambi con la preferenza più alta.

2.5.5 Accordo finale per il viaggio

Raggiungere questa fase del processo decisionale significa che entrambi il Driver e il Passenger sono sicuri della loro compatibilità di preferenze e le loro richieste combaciano a riguardo di uno specifico viaggio. I due PA non stipulano immediatamente un accordo tra di loro, ma invece aspettano fino ad un momento arbitrario, deciso dall'utente in fase di configurazione del servizio, prima dell'orario prefissato per la partenza. Questo permette di aumentare le possibilità per ciascun PA di interagire con altri agenti per trovare le soluzioni migliori.

L'obiettivo principale in questa fase è ottenere la soluzione migliore in base alle preferenze dell'utente; infatti, più i parametri della negoziazione ottenuti saranno simili a quelli passati dall'utente nella richiesta di servizio, più la richiesta sarà soddisfatta meglio. Così al termine di questa fase, il PA partner riceve un punteggio calcolato sui valori ottenuti dalla sequenza di negoziazioni e il suo nome, con punteggio e valore dei parametri, memorizzato in una lista locale ordinata in base al punteggio; il punteggio assegnato al partner non ha nulla a che fare con il suo feedback, ma è relativo alla richiesta di servizio in questione e serve per stilare una classifica dei partner compatibili in modo da scegliere poi quello o quelli che soddisfano di più l'utente. Infatti a punteggio più alto corrisponde partner con parametri della richiesta più simili a quelli forniti dall'utente.

Tabella 2.2: Score computing

Parameter	**Weight**
offset(request_departure_time - negotiated_time)	25%
departure_meeting_point	25%
arrival_meeting_point	20%
value_of_feedback	30%

In tab. 2.2 viene mostrato come è calcolato il punteggio da assegnare ad un partner e il peso in percentuale assegnato a ciascun parametro considerato; il calcolo si basa sui risultati prodotti dalla contrattazione tra i due personal agents: più i desideri degli utenti sono simili, maggiore sarà il punteggio che verrà assegnato. Al momento solo quattro parametri sono considerati per il calcolo del punteggio: l'orario di partenza comune trovato, il cui peso per il calcolo del punteggio è proporzionale alla differenza in minuti tra l'orario proposto dall'utente e quella ottenuto dalla contrattazione; il punto di incontro per la partenza e per l'arrivo concordati, il cui peso è calcolato in base alla posizione che ricoprono negli elenchi passati dall'utente attraverso la richiesta di servizio; il feedback del partner trovato, il cui valore è utilizzato per calcolare il peso del parametro nel punteggio.

Quando arriva l'orario prefissato prima della partenza per chiudere, diciamo in termine economico, le contrattazioni il PA contatterà ogni partner compatibile che è stato trovato, dal migliore al peggiore finchè non avrà raggiunto il numero di persone indicato nella richiesta di servizio iniziale. Se due PA hanno lo stesso punteggio all'interno della lista, il sistema sceglierà il primo arrivato. in seguito, il PA produrrà una risposta con le proprie informazioni personali e la invia ad ogni PA con cui ha raggiunto un accordo, oltre ad assegnare a quest ultimo un feedback positivo. L'utente avrà poi 24 ore per confermare o meno il feedback positivo per il suo partner. Il feedback è calcolato moltiplicando il numero di volte che si è ricevuto un feedback positivo per 100 e dividendo il tutto per il numero totale di volte che si è usufruito del servizio ottenendo un accordo per un viaggio. Ad esempio, se l'utente ha ottenuto 10 volte un feedback positivo e 2 volte un feedback negativo il suo valore di feedback finale sarà *83,33%*. Supponiamo ora che partecipi ad un altro viaggio e riceva un feedback positivo; il suo nuovo valore sarà dunque *84,61%*, mentre se fosse stato negativo sarebbe stato di *76,92%*.

Capitolo 3

ANDIAMO: Architettura di sistema

La volontà di realizzare un sistema accessibile, semplice da usare e con supporto MAS alle comunità mobili ci ha spinto a sviluppare ANDIAMO, un sistema in grado di offrire il servizio di rideshare ovunque grazie alla possibilità di realizzare reti Bluetooth a cui accedere con dispositivi mobili e dove gioca un ruolo essenziale la competizione e la collaborazione tra gli agenti.

In questo capitolo verrà illustrata l'architettura generale usata per ANDIAMO cominciando dai requisiti di sistema per arrivare poi ai vari sottocomponenti e alle interazioni che avvengono tra di loro. L'architettura è ottenuta estendendo e personalizzando il sistema ToothAgent [26] con servizio di compravendita di libri usati, il quale offre un sistema in grado di comunicare con utenti mobili tramite la tecnologia Bluetooth e scambiare informazioni corrispondenti agli interessi degli studenti presenti in un luogo preciso.

3.1 Perchè ANDIAMO?

I continui insuccessi delle soluzioni basate su web da un lato e la comunque grande necessità di sistemi che supportano i servizi di rideshare dall'altro ci hanno convinto e motivato per creare ANDIAMO.

ANDIAMO nasce per colmare quel vuoto lasciato dopo che le soluzioni basate sul web si sono rivelate inefficaci e le varie attività di ricerca hanno prodotto modelli poco attuabili in pratica. Il nostro è un sistema per la comunicazione, coordinazione, collaborazione e competizione fra utenti; i quali però, pur essendo persone reali fisicamente, giocano nel sistema un ruolo totalmente virtuale e sono rappresentati da agenti che si comportano, nel nostro caso, come effettivamente si comporterebbero gli esseri umani. L'idea di base è infatti quella di permettere la collaborazione e la competizione tra agenti e la costituzione di una comunità virtuale attiva e dinamica utile alle persone.
L'architettura studiata è legata all'offerta del servizio di rideshare e permette di ottenere una grande comunità virtuale sul territorio, all'interno della quale esistono varie sotto-comunità autonome legate al luogo preciso dove è dislocato il server che offre il servizio.

Si immagini di distribuire infatti sul territorio una serie di server, tutti collegati fra loro via internet. Ognuno di questi server supporta il servizio rideshare e può comuncare con altri dispositivi mobili (cellulari o PDAs) attraverso una connessione wireless Bluetooth. Il raggio di azione è ristretto dunque e la comunicazione può avvenire solo se il dispositivo mobile (e quindi la persona) si trova nelle immediate vicinanze del server. Per aumentare l'area coperta da ciascun server è inoltre possibile realizzare una rete Bluetooth attraverso access points Bluetooth collegati via ethernet o Wi-Fi al server.
Ogni utente, interfacciandosi al sistema con il proprio dispositivo, può inviare ai server una sorta di richiesta, con le sue preferenze e i suoi desideri relativi al servizio, dall'interpretazione della quale viene creato un agente che lavorerà in modo attivo ed autonomo sulla piattaforma multi-agente installata sul server. Il lavoro di questo agente prevede il dialogo, la competizione, la collaborazione e la negoziazione con gli altri agenti presenti sulla piattaforma, sulla base delle preferenze indicate nella richiesta fatta in precedenza, per il raggiungimento di un solo obiettivo: trovare un passaggio (ride-seeker) oppure un passeggero (ride-offerer) per il proprio utente. I risultati prodotti sono poi trasferiti dall'agente al dispositivo mobile dell'utente.
Il sistema realizzato ha un carattere principalmente localizzato che non permette all'utente di essere sempre ed ovunque collegato ad esso; ciò comporta che gli agenti devono continuare a vivere e lavorare anche quando l'utente non è co-localizzato con il server. Infine all'utente deve essere garantito il fatto che potrà interfacciarsi al sistema in modo rapido e automatizzato ogni volta si trovi nel raggio d'azione di un punto di accesso del sistema con il proprio dispositivo mobile.

3.2 Requisiti di sistema

3.2.1 Requisiti Principali

- permettere all'utente di esprimere i propri desideri all'interno di una richiesta di servizio
- fornire l'accesso al servizio di rideshare quando il dispositivo mobile e il server o l'accespoint Bluetooth sono nella stessa area geografica, cioè co-localizzati.
- permettere all'utente di recuperare i risultati pendenti; i risultati devono essere accessibili non appena l'utente entra nel raggio d'azione di un punto d'accesso del sistema.

3.2.2 Requisiti Funzionali

- L'utente, per usufruire del servizio reso disponibile dal sistema, come prima operazione dovrà registrarsi attraverso un sito internet fornendo i propri dati personali e quelli relativi al dispositivo mobile che utilizzerà.(Fig. 3.1) Una volta terminata la procedura di registrazione, l'utente potrà scaricare il software necessario per l'utilizzo del sistema; si tratta di una applicazione che verrà installata sul dispositivo mobile. I dati personali forniti in fase di registrazione saranno salvati in un database centrale e potranno essere utilizzati dal servizio Rideshare per reperire informazioni riguardanti l'utente, come per esempio il suo nome, la sua età, il recapito telefonico o l'indirizzo e-mail. Un nuovo

utente registrato al sistema riceve un feedback inizializzato a 100 come reputazione di partenza.

Title: Registrazione Utente

Summary:
Questo use-case descrive come l'utente effettua la registrazione
al sistema per poter usufruire del servizio RideShare..

Description
Step1: L'utente effettua la registrazione attraverso un sito internet fornendo i propri dati personali e quelli relativi al dispositivo mobile che utilizzerà. [Exception 1]

Step2: L'utente fornisce la password con cui intende accedere in
seguito al sistema.

Step3: L'utente effettua il download del software.

Exceptions
Exception 1: se nel sistema compare già un altro utente con lo stesso indirizzo BlueTooth per il dispositivo mobile viene emesso un messaggio d'errore

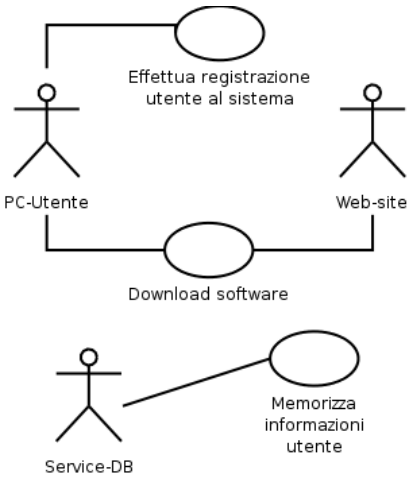

Figura 3.1: Registrazione utente

- Il livello di sicurezza offerto dal sistema dovrà essere all'inizio minimo e molto semplice; ogni utente è contraddistinto da indirizzo Bluetooth, del dispositivo mobile usato per accedere al servizio, e password. All'utente verrà richiesto di identificarsi fornendo soltanto la password in modo da evitare l'accesso al sistema a persone non registrate al servizio. Non è necessario al momento alcun sistema di criptazione delle informazioni.
- Il sistema dovrà permettere all'utente di modificare i propri dati personali forniti in fase di registrazione in qualsiasi momento. Ad esempio l'utente potrebbe voler modificare il proprio numero di telefono o aggiungere un nuovo dispositivo mobile a quelli già esistenti (Fig. 3.2).
- Il sistema dovrà permettere agli utenti di accedere al servizio presso qualsiasi server di ANDIAMO.
- Il sistema dovrà permettere agli utenti di esprimere i loro interessi nella formulazione delle richieste di servizio. Tramite l'applicazione mobile l'utente compilerà le richieste di servizio rideshare fornendo diverse informazioni a seconda che egli sia una offerer o un seeker; se egli offre un passaggio, avrà la possibilità di specificare, oltre al luogo di partenza e di arrivo, anche una serie di località che attraverserà per compiere il tragitto in modo da poter permettere di usufruire del passaggio anche a utenti che si trovano in un punto intermedio sul persorso specificato. I servizi configurati saranno visibili sull'interfaccia grafica dell'applicazione e saranno modificabili e eliminabili.

Title: Aggiornamento informazioni

Summary:
Questo use-case descrive come l'utente può aggiornare i propri dati personali memorizzati nel sistema

Description
Step1: L'utente accede al sito internet del sistema
Step2: L'utente fornisce password ed indirizzo Bluetooth per l'autenticazione [exception 1]
Step3: L'utente può procedere ad aggiornare i propri dati personali se lo desidera

Exceptions
Exception 1:se l'utente non compare tra quelli autorizzati all'utilizzo del servizio, viene emesso un messaggio d'errore

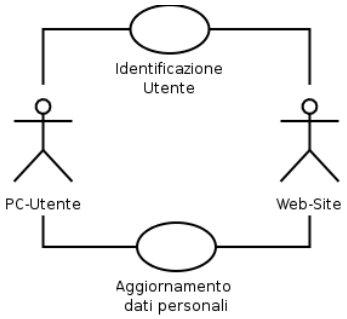

Figura 3.2: Aggiornamento informazioni

- Il sistema dovrà aiutare l'utente nella configurazione dei servizi, cioè le piattaforme dovranno permettere ai nuovi membri di una comunità di comportarsi in accordo con la cultura (ovvero le usanze) della comunità stessa. Questo requisito nasce dal fatto che l'utente potrebbe trovarsi in difficoltà difronte a scelte da compiere per configurare in maniera corretta e ottimale le richieste di un servizio; l'utente potrebbe allora delegare al software il compito di completare le proprie preferenze, in base a politiche che si basano sul comportamento degli altri utenti che popolano la piattaforma.

- Il sistema dovrà grantire l'accesso ai servizi richiesti quando il dispositivo mobile si trova nelle immediate vicinanze di una rete wireless Bluetooth collegata ad un server che fornisce il servizio rideshare, cioè quando si trova in una Rideshare Zones. Ovviamente tale distanza dipenderà molto dalle caratteristiche dell'apparecchio mobile dell'utente, ma lo scambio di informazioni fra i dispositivi dovrà essere automatico e istantaneo. L'utente deve avere in esecuzione quindi l'applicazione sul proprio cellulare o computer palmare per poter accedere al servizio. Il sistema dovrà essere in grado di supportare la ricerca continua di server e dovrà garantire che le richieste dell'utente vengano trasferite sul server con cui il dispositivo sta comunicando. Per contro il server, ricevuto l'indirizzo Bluetooth dell'apparecchio e la password scelta dall'utente procederà a verificare, prima di accettare qualsiasi altra informazione, la validità dei dati ottenuti, ovvero controllare che l'utente sia autorizzato ad utilizzare il servizio.

- Le richieste dell'utente dovranno essere trasferite al server in maniera automatica, così come in maniera automatica dovrà avvenire la creazione dell'agente. Per ogni utente dovrà essere creato un solo agente per ogni server; ciò vuol dire che se l'utente vuole at-

Title: Accesso al servizio

Summary:
L'utente accede ad una Carpooling Zones con il proprio dispositivo mobile e la sua richiesta è elaborata automaticamente

Description
Step 1: Il dispositivo mobile entra in una CarPooling Zones
Step 2: Viene attivata in automatico una connessione tra server ed apparecchio
Step 3: Il server verifica le informazioni relative all'utente [exceptio1]
Step 4: Vengono inviate al server le informazioni relative alle richieste dell'utente
Step 5: Il server attiva il personal agents dell'utente [extension1]
Step 6: il personal agents salva la richiesta di servizio nel db
Step 7: il personal agents comincia l'eleaborazione della richiesta di servizio

Exceptions:
Exception1: Se l'utente non è registrato al sistema, viene emesso un messaggio d'errore
Extensions
Extension 1: se il personal agents non esiste, ne viene creato uno nuovo

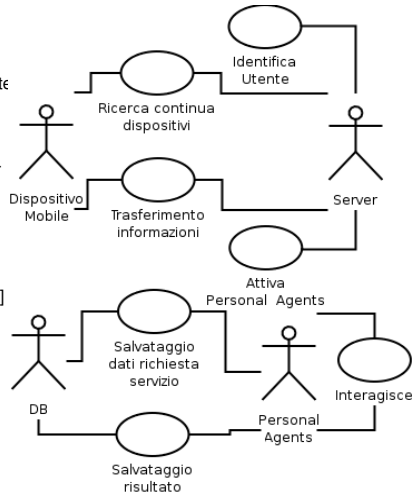

Figura 3.3: Accesso al servizio

tivare più richieste contemporaneamente sullo stesso server, dovrà esserci un solo agente che si occuperà di esse. Il personal agent dell'utente si occuperà dell'interazione con gli altri agenti e della memorizzazione delle informazioni sul database interno del server su cui sta operando.(Fig. 3.3)

- Il sistema dovrà permettere all'utente di scaricare le proprie risposte pendenti (risposte pendenti = l'esito del lavoro dei personal agents, memorizzato sui database ma non ancora scaricato) quando nel raggio d'azione di un server, che sia quello dove ha creato l'agente oppure un altro non ha importanza. L'utente avrà quindi la possibilità di accedere al database di qualsiasi server visitato per mezzo di un qualsiasi altro server facente parte della rete ANDIAMO. Contattando via Bluetooth il server che si trova nelle immediate vicinanze, l'utente potrà recuperare le informazioni presenti sugli altri server; tutto questo avverrà ancora una volta in maniera completamente automatica.(Fig. 3.4)

- Il sistema dovrà implementare un meccanismo che permetta di avere un giudizio sui singoli utenti e che possa garantire l'affidabilità di una persona, ad esempio prevedendo che ogni utente registrato al sistema sia legato ad un valore percentuale chiamato feedback; più questo valore si avvicina al 100%, più l'utente è affidabile. L'utente, quando configura il servizio per richiederne una prestazione, avrà la possibilità quindi di indicare una preferenza a riguardo del feedback minimo desiderato per le persone con cui andrà a interagire; quindi, nei risultati forniti dal sistema, verrà visualizzato il feedback di ogni utente che prende parte all'interazione. Dopo aver usufruito del servizio, cioè quando l'utente ha ricevuto oppure dato un passaggio a qualcuno, l'utente ha 24 ore per assegnare al partner un punteggio positivo o negativo tramite l'applicazione mobile. Se

Title: Recupero risultati pendenti

Summary:
L'utente recupera i risultati pendenti accedendo ad una CarPooling Zones con il suo apparecchio

Description
Step 1: Il dispositivo mobile entra in una CarPooling Zones
Step 2: Viene attivata in automatico una connessione tra server ed apparecchio
Step 3: Il server verifica le informazioni relative all'utente [exceptio2]
Step 4: Viene avviata la ricerca dei risultati su tutti i server visitati
Step 5: Vengono scaricati dal server i risultati relativi all'utente

Exceptions:
Exception2: Se l'utente non è registrato al sistema, viene emesso un messaggio d'errore

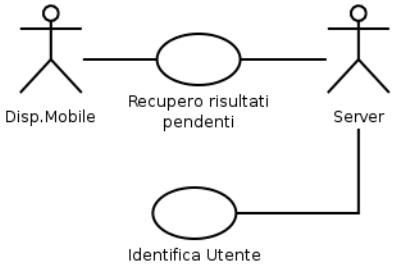

Figura 3.4: Recupero risultati pendenti

ciò non avviene, il sistema assegnerà un feedback positivo di default. L'utente potrà poi verificare il proprio punteggio di feedback tramite il sito web.

3.2.3 Requisiti Non Funzionali

- Il sistema dovrà essere semplice da utilizzare e non dovrà richiedere particolare dimestichezza o esperienza con i dispositivi mobili; per questo il sistema dovrà essere aperto a tutti ed essere il meno invasivo possibile. L'utente interagirà minimamente con il sistema in quanto ogni azione e scambio di messaggi dovranno essere completamente automatizzati. Inoltre non dovranno essere richieste particolari abilità all'utente per la configurazione e l'attivazione dei servizi.
- Il sistema dovrà essere utilizzabile su ogni dispositivo dotato di tecnologia di comunicazione Bluetooth e che esegue una piattaforma Java (J2ME).

3.3 Componenti di sistema

L'architettura di sistema prevede cinque componenti principali:

- **Il dispositivo mobile** permette all'utente di comporre le richieste, inviarle al server e recuperare i risultati prodotti dal personal agent.
- **I server distribuiti** nel nostro sistema sono i principali responsabili del servizio di rideshare. Ciascun server contiene: 1) una piattaforma multi-agente con i personal agents,

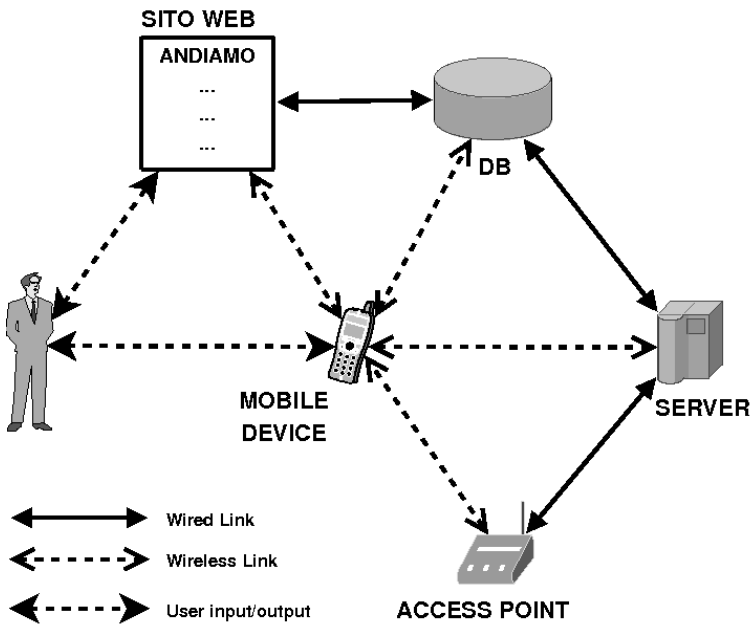

Figura 3.5: Interazione tra i componenti del Sistema

ciascuno dei quali rappresenta un singolo utente, 2) un database dove i risultati vengono archiviati, 3)una interfaccia responsabile di stabilire le comunicazioni con i dispositivi mobili e dirigere le richieste dell'utente verso il corrispondente personal agent.

- **Il database dei servizi centrale**, accessibile via web, contiene le informazioni a riguardo di tutti i server e le loro proprietà, quali nome, locazione, ecc. Il database memorizza anche le informazioni relative agli utenti registrati al sistema.

- **Le Rideshare Zones** sono aree dove è possibile accedere al servizio Rideshare; possono essere costituite direttamente da un server oppure da access-points collegati ad un server ma fanno riferimento sempre e solo ad un server.

- **Il sito web** fornisce informazioni a riguardo del sistema ANDIAMO, offre una procedura di registrazione per i nuovi utenti e la possibilità di modificare i dati per quelli vecchi

Fig. 3.5 illustra l'architettura generale di sistema e l'interazione tra i vari componenti. La connessione tra il dispositivo mobile e il server o access-point è stabilita attraverso la tecnologia di comunicazione wireless Bluetooth.

In particolare, il telefono cellulare dell'utente invia al server tutte le richieste di servizio e poi ne ottiene di ritorno i risultati. Il cellulare può anche ricevere talvolta richieste relative a possibili modifiche da apportare alle preferenze indicate dagli utenti nelle richieste di servizio, oppure al processo decisionale attuato dal personal agent e permettere di inviare quindi una

risposta al server. Inoltre, il cellulare è usato per inviare al sistema una valutazione a riguardo del/dei partner/s di viaggio, il quale si rifletterà sul futuro valore del feedback di questi utenti.

Il server comunica con il DB centrale dei servizi per controllare le informazioni dell'utente (età, feedback, password, ecc) e dopo che l'utente ha usufruito del servizio, prendendo parte ad un viaggio, il server aggiornerà anche i valori relativi alla reputazione degli utenti che hanno preso parte all'accordo.

Il DB centrale è responsabile della memorizzazione di tutte le informazioni relative ad una specifico accordo per un viaggio e delle interazioni avvenute tra i personal agents coinvolti.

Infine il sito web permette di raccogliere le informazioni relative ad ogni utente e memorizzarle nel database centrale, di modificare o consultare questi dati e di scaricare l'applicazione mobile da installare poi sul proprio dispositivo personale.

3.4 Scenari d'utilizzo

ANDIAMO può essere applicato a qualsiasi realtà, pubblica o privata indifferentemente, dove vi sia la necessità di offrire il servizio di rideshare; si può pensare di installarlo, per esempio, negli edifici universitari come le facoltà, nella biblioteca comunale, in un ospedale ma anche in fabbriche o industrie.

I dispositivi mobili (cellulari e PDAs) sono una delle componenti principali di questo sistema; sono contraddistinti da problematiche relative alla mobilità dei dispositivi, cioè rapidità e semplicità di spostamento da un luogo ad un altro, e dal carattere al contempo locale e distribuito che li caratterizza: locale per la ridotta copertura data dalla tecnologia Bluetooth con cui i dispositivi comunicano; distribuito perchè dovrebbe seguire l'utente nei suoi spostamenti nei punti in cui potrebbe tornare utile ai fini dell'applicazione. Quindi fondamentale è una disposizione dei punti d'accesso al sistema in modo da compensare le problematiche sopra enunciate.

Un possibile scenario d'utilizzo del sistema potrebbe essere una facoltà universitaria dove il servizio di rideshare potrebbe risultare molto utile dato che la comunità che anima questo luogo è contraddistinta da persone, quali studenti e docenti, che quotidianamente si muovono per raggiungere la facoltà e che quindi potrebbero condividere il viaggio di andata o ritorno. In una situazione come questa, dove magari esiste già una rete Wi-Fi che copre l'intero edificio, si potrebbe pensare di predisporre un unico server centrale e costruire una rete tramite il Wi-Fi di access point Bluetooth ad esso collegati con un unico database centrale. In Fig. 3.6 è riportato un esempio di come potrebbe essere offerto il servizio all'interno della facoltà.
Altro scenario potrebbe essere rappresentato da un edificio pubblico in una grande città come quello del comune, ad esempio, dove lavorano un numero di dipendenti molto alto e sopratutto transitano parecchie persone. Anche qui, grazie al fatto che le reti ethernet o Wi-Fi sono presenti oramai ovunque, è possibile proporre ANDIAMO attraverso una architettura come

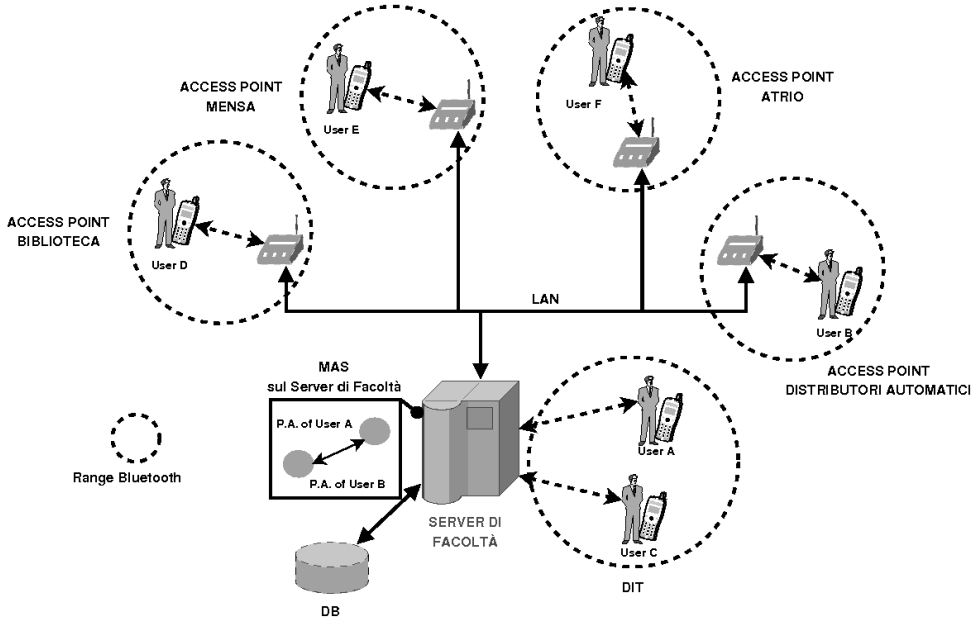

Figura 3.6: ANDIAMO all'interno di una facoltà

quella descritta precedentemente.

Quando su un territorio si trovano diverse implementazioni di ANDIAMO in diversi ambiti si potrebbe prevedere di avere un unico sistema per tutte le realtà. Lo scenario globale che si presenterebbe potrebbe assomigliare a quello proposto in Fig. 3.7 e sarebbe caratterizzaato da tante piccole comunità virtuali eterogenee dal punto di vista degli utenti che le animano ma omogenee invece per quanto rigurda l'utilizzo del sistema che ne viene fatto. Ogni singola realtà è collegata alle altre attraverso la rete Internet e fa riferimento ad un unico database centrale dei servizi in maniera di condividere informazioni relative al sistema e agli utenti.
I punti di accesso al sistema sono composti, come ben sappiamo, o da un access point o da direttamente un server. Questi due componenti hanno però finalità differenti: i server, infatti, sono in grado, via Bluetooth, di accettare richieste di servizio dagli utenti, elaborarle e fornire le risposte adeguate mentre gli access point si limitano a fare da tramite tra il dispositivo mobile dell'utente e il server inoltrando nelle opportune direzioni i messaggi che questi due componenti si scambiano.

Come già detto in precedenza, gli utenti e i loro dispositivi sono mobili, cioè si spostano da un luogo ad un altro rapidamente; questo può essere visto anche come un passaggio da una Rideshare ad un altra e quindi da un server ad un altro. Questi spostamenti necessitano di essere monitorati perchè altrimenti sarebbe impossibile in seguito recuperare i risultati prodotti in ogni singola piattaforma del sistema dove è stato attivato un personal agent. Quindi il dispositivo mobile ogni volta che si collega ad un server direttamente oppure attraverso gli access points,

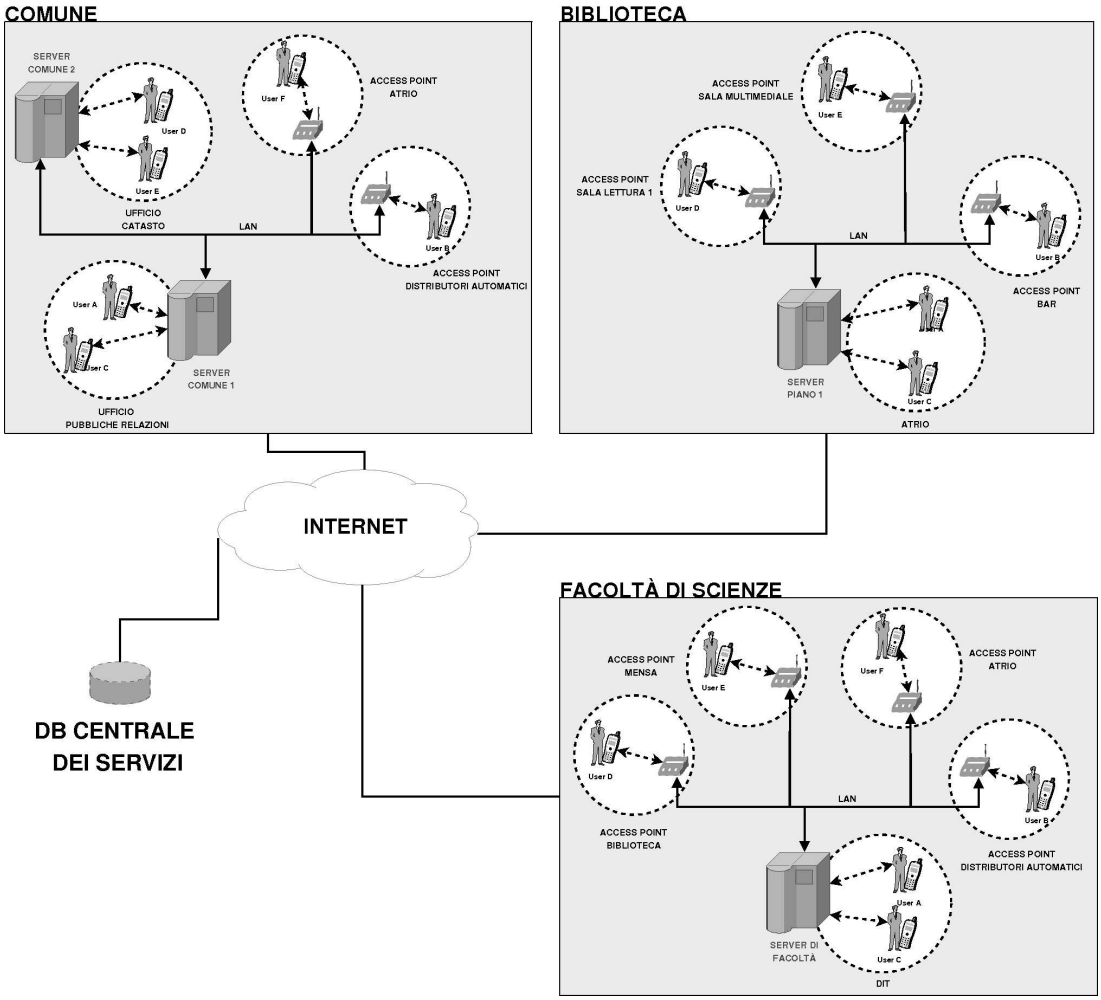

Figura 3.7: ANDIAMO replicato in diversi ambiti

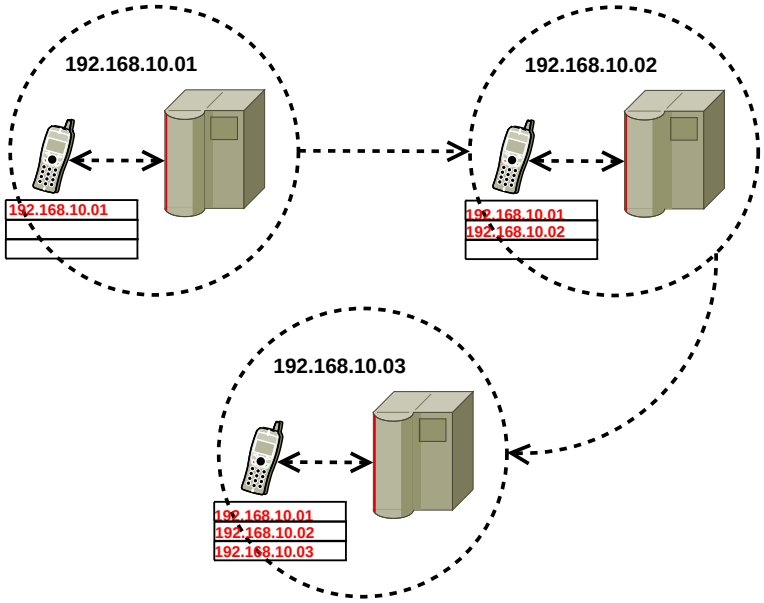

Figura 3.8: Sequenza di spostamenti attraverso Rideshare Zones e memorizzazione IPs

memorizza l'indirizzo IP, come è mostrato anche in Fig. 3.8, in modo da poter avere un elenco dei server visitati e dei luoghi dove si è usufruito del servizio.

3.5 Registrazione online

Per poter accedere al sistema, l'utente deve prima registrarsi. Per fare ciò, egli deve effettuare la registrazione online compilando un modulo con informazioni personali quali nominativo, età, indirizzo e-mail, numero di telefono, indirizzo Bluetooth del dispositivo mobile che utilizzerà, password, anno in cui ha ricevuto la patente se vuole essere un ride-offerer, parametri che ne caratterizzano la tipologia di utente e altri ancora. La registrazione, fondamentalmente, permette al sistema di identificare l'utente e il dispositivo mobile che verrà utilizzato.

3.6 Accedere al servizio

L'accesso al servizio avviene come in Fig. 3.9.

Un utente accede al servizio attraverso una sequenza di tre passi: (1) compilare una richiesta di servizio sull'applicazione mobile, (2) eseguire l'applicazione sul proprio dispositivo mobile con il Bluetooth attivo e (3) compiere alcune operazioni per attivare il servizio richiesto. L'applicazione mobile è scritta in Java e usa la libreria JSR-82 [5], cioè le API Bluetooth per Java, per implementare un sistema di comunicazione. Più avanti, nella apposita sezione, verranno mostrate le varie interfacce per la gestione dell'applicazione mobile e l'acquisizione

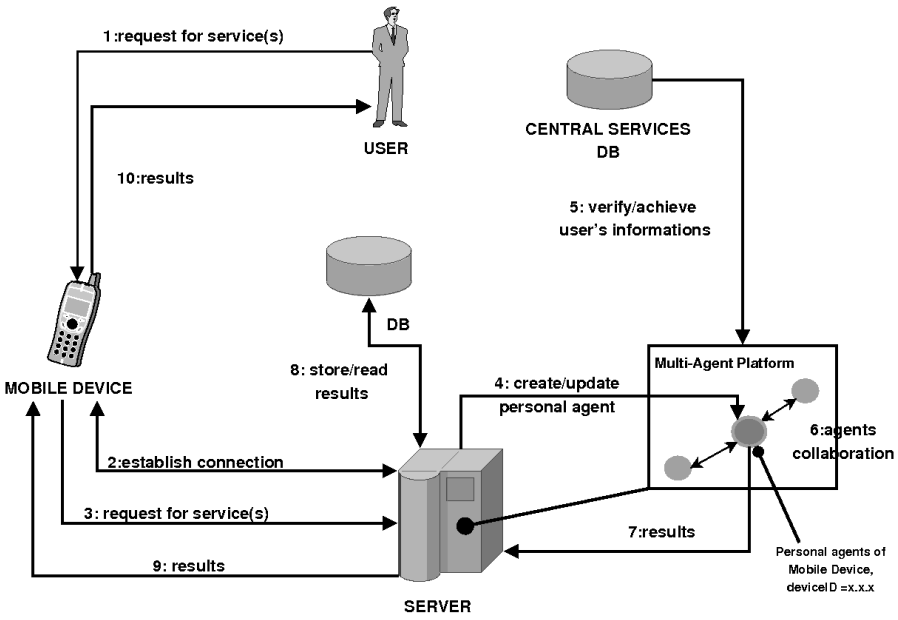

Figura 3.9: Accedere al servizio

delle informazioni relative alle richieste di servizio.
In dettaglio, l'applicazione avvia una ricerca continua di dispositivi Bluetooth-abilitati nei dintorni e quando trova un server Rideshare, stabilisce una connessione con esso (passo 2) e gli invia le proprie richieste (passo 3) sottoforma di una stringa XML. Le richieste sono poi elaborate dal server e i risultati rispediti all'utente (passi 4 - 10). Il dispositivo mobile memorizza sempre l'indirizzo Ip del server in modo da ricordarsi dove ha avuto accesso al servizio in modo da poter poi recuperare i risultati finali. Un esempio di richiesta è visibile in Fig. 3.10.

In Fig. 3.11 invece viene mostrato il protocollo di interazione tra i diversi componenti del sistema per usufruire del servizio di rideshare. Sul server risiede un modulo specifico dedicato alla comunicazione che gestisce l'interazione con i dispositivi mobili.
Egli riceve l'indirizzo Bluetooth e la password dal dispositivo mobile (passi 1 and 3) e verifica se sulla piattaforma in esecuzione sul server vi sia già un PA assegnato a quel dispositivo mobile (passo 4). Questo modulo assume la coppia composta da indirizzo Bluetooth del dispositivo dell'utente e password come identificatore per mappare il dispositivo mobile con il specifico PA. Se però non vi è alcun PA attivo, precedentemente assegnato all'utente, il modulo di comunicazione si connette al database dei servizi centrale e verifica se l'utente è registrato al sistema controllando che all'indirizzo Bluetooth corrisponda effettivamente la password fornita (passi 5 - 6). In caso l'esito della verifica sia positivo, viene creato un nuovo agente e assegnato al dispositivo mobile dell'utente in questione (passo 8). Poi il dispositivo mobile invia la stringa XML di configurazione al modulo di comunicazione (passo 9), il quale provvede ad inoltrarla al personal agent appropriato (passo 10). A questo punto il personal agent avvia l'interazione con gli altri agenti sulla piattaforma cercando di soddisfare tutte le richieste dell'utente (passo

```
<REQUEST>
    <CATEGORY>CarPooling</CATEGORY>
    <PARAMETERS>
        <PARAM>
            <CONTENT>Rule</CONTENT>
            <PARAMVALUE>Offerer/Seeker</PARAMVALUE>
        </PARAM>
        <PARAM>
            <CONTENT>Date</CONTENT>
            <PARAMVALUE>20.11.2006</PARAMVALUE>
        </PARAM>
        <PARAM>
            <CONTENT>DepCity</CONTENT>
            <PARAMVALUE>Povo</PARAMVALUE>
        </PARAM>
        <PARAM>
            <CONTENT>DepPoints</CONTENT>
            <PARAMVALUE>IRST</PARAMVALUE>
            <PARAMVALUE>Piazza</PARAMVALUE>
            <PARAMVALUE>Circoscrizione</PARAMVALUE>
        </PARAM>
        <PARAM>
            <CONTENT>ArrCity</CONTENT>
            <PARAMVALUE>Rovereto</PARAMVALUE>
        </PARAM>
        <PARAM>
            <CONTENT>ArrPoints</CONTENT>
            <PARAMVALUE>Piazza Rosmini</PARAMVALUE>
            <PARAMVALUE>Stazione</PARAMVALUE>
            <PARAMVALUE>MART</PARAMVALUE>
        </PARAM>
        <PARAM>
            <CONTENT>Seats</CONTENT>
            <PARAMVALUE>1</PARAMVALUE>
        </PARAM>
        <PARAM>
            <CONTENT>Range</CONTENT>
            <PARAMVALUE>15</PARAMVALUE>
        </PARAM>
        <PARAM>
            <CONTENT>Time</CONTENT>
            <PARAMVALUE>20.30</PARAMVALUE>
        </PARAM>
        <PARAM>
            <CONTENT>Interactive</CONTENT>
            <PARAMVALUE>Yes/No</PARAMVALUE>
        </PARAM>
        <PARAM>
            <CONTENT>Feedback</CONTENT>
            <PARAMVALUE>80</PARAMVALUE>
        </PARAM>
        <PARAM>
            <CONTENT>Deadline</CONTENT>
            <PARAMVALUE>180</PARAMVALUE>
        </PARAM>
    </PARAMETERS>
</REQUEST>
```

Figura 3.10: Richiesta di servizio

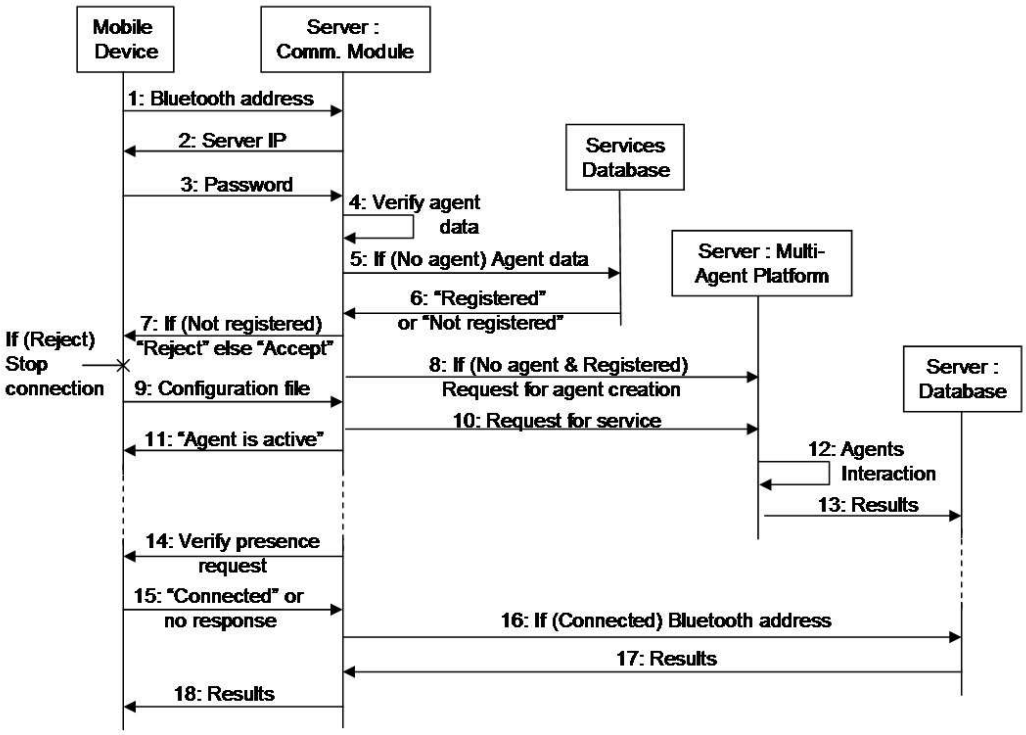

Figura 3.11: Accessibilità del servizio

12).
Nel sistema ANDIAMO qui descritto, ciascun PA può ricevere contemporaneamente una o più richieste da parte dell'utente, sia per offrire un passaggio, sia per riceverlo; se l'agente raggiunge un accordo con un altro agente a riguardo di una specifico viaggio, allora memorizza il risultato localmente nel database del server (passo 13). Più tardi i risultati potranno essere scaricati dall'utente attraverso il dispositivo mobile (passi 14-18).

3.7 Scaricare i risultati pendenti

Quando viene stabilita una connessione tra un dispositivo mobile ed un server, il modulo di comunicazione invia l'indirizzo IP del server al dispositivo mobile (step 2 in Fig. 3.11). Il dispositivo mobile memorizza infatti gli indirizzi IP di tutti i server visitati in una lista XML che sarà utilizzata in seguito per recuperare tutti i risultati pendenti.
Il contenuto delle risposte fornite dal personal agent sarà composto da un identificatore relativo alla richiesta, informazioni(es. numero telefonico) su come contattare il/i partner/s del viaggio, l'orario di partenza, punto di ritrovo e così via.
L'utente può ricevere i risultati sul proprio dispositivo mobile solo se si trova in una Rideshare Zone, cioè si trova nell'area Bluetooth coperta dal server o da un access-point. Il modulo di

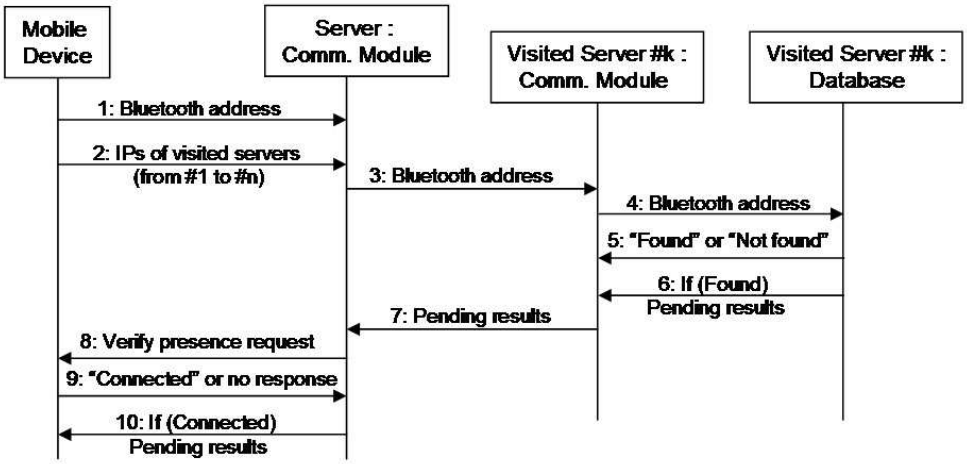

Figura 3.12: Scaricare i risultati pendenti dal dispositivo mobile

comunicazione verifica la presenza del dispositivo mobile e gli invia i risultati memorizzati nel database interno dal corrispondente PA. La Fig. 3.12 mostra il protocollo di interazione tra i componenti del sistema per il recupero dei risultati tramite il dispositivo mobile.
Prendiamo come esempio il caso di un utente alla stazione dei treni che si trova nella Rideshare Zone di un server. Dopo aver stabilito una connessione, il dispositivo invia al server della stazione la lista degli indirizzi IP di tutti i server precedentemente visitati (es. server dell'università, server presso il centro storico, ecc)(passo 2). Il modulo di comunicazione del server invia l'indirizzo Bluetooth del dispositivo mobile a tutti i server elencati nella lista(passo 3). A turno, il modulo di comunicazione di ciascun server estrae dal proprio database interno i risultati relativi a quel specifico utente-indirizzo Bluetooth e li invia al server che ne ha fatto richiesta, in questo caso il server della stazione (passi 4 - 7). Tutti i risultati sono raccolti dal modulo di comunicazione di quest ultimo server ed infine inviati al dispositivo mobile (passi 8 -10). Se il dispositivo non fosse più collegato al server (es. l'utente ha lasciato la stazione)il processo di recupero dei risultati fallisce e i risultati raccolti cancellati. Questi risultati saranno ancora comunque reperibili presso i server di origine.

Questa procedura è utilizzata anche per inviare delle domande all'utente come già descritto in precedenza. L'utente quando scarica i risultati può infatti ricevere anche dei particolari messaggi che contengono una domanda posta dal sistema, alla quale l'utente risponderà attraverso un'apposita interfaccia sul dispositivo mobile.

3.8 Applicazione Mobile

Il servizio di ridesharing è accessibile, come abbiamo già visto, solo tramite dispositivi mobili abilitati alla comunicazione Bluetooth e che supportano Java, su cui viene eseguita l'applicazione con cui l'utente accede al sistema ANDIAMO.

L'applicazione mobile si compone di varie interfacce, ognuna destinata a svolgere compiti diversi come la raccolta o la visualizzazione di informazioni e la comunicazione con i punti di accesso del sistema.

L'interfaccia principale dell'applicazione mobile è visibile in Fig. 3.13; sono riportati due stati in cui può presentarsi: il primo a sinistra è relativo al primo utilizzo dell'applicazione mentre a destra è possibile vedere la schermata che si presenta quando sono giunte una nuova risposta e una nuova domanda dal sistema (notare i contatori a lato degli elementi del menù *Messages* e *Question*).

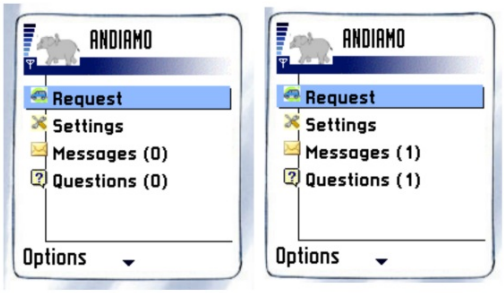

Figura 3.13: Interfaccia principale

In Fig. 3.14 è possibile vedere invece l'interfaccia della sezione dedicata alla gestione delle richieste di servizio e vengono riportati tre stati in cui si può trovare. Il significato degli indici del menù è il seguente: con *Active requests* si accede alla lista di richieste di servizio già attivate presso il sistema ANDIAMO; con *Pending requests* viene visualizzata la lista delle richieste di servizio che si è già deciso di attivare ma che non sono ancora state inviate al server; con *Archive* si accede all'archivio di tutte le richieste di servizio configurate con l'applicazione mobile; con *New requests* si avvia la configurazione di una nuova richiesta di servizio.
Lo stato mostrato in Fig.3.14 più a sinistra presenta l'interfaccia al primo utilizzo dell'applicazione con tutti i contatori settati a zero; quello in centro indica che sono state già configurate e archiviate due richieste di servizio, ma non ancora attivate e infine quello a destra mostra che le due richieste di servizio sono attivate e in attesa di essere inviate al sistema.

Figura 3.14: Gestione richieste di servizio

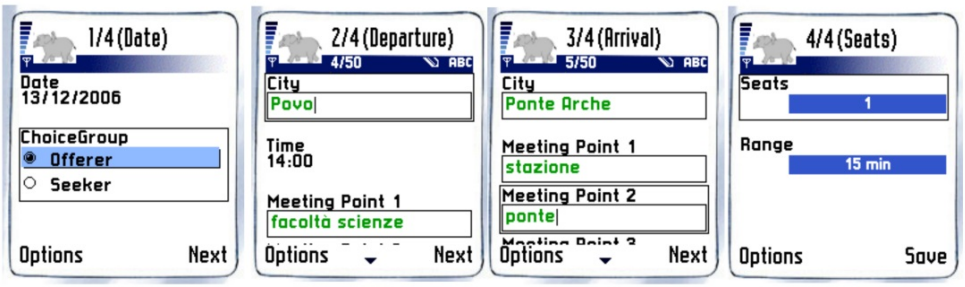

Figura 3.15: Passi di configurazione di una richiesta di servizio

Figura 3.16: Salvataggio di una richiesta

La sequenza di passi da compiere per configurare una richiesta di servizio è visibile da sinistra a destra in Fig. 3.15. Vengono richiesti la data in cui si desidera usufruire del servizio, se si offre o chiede un passaggio, la località di partenza, orario di partenza, tre punti di ritrovo per la partenza, località di destinazione, tre punti di arrivo per la destinazione, numero di posti a sedere richiesti o offerti e per concludere l'anticipo o il posticipo in minuti accettato rispetto all'orario indicato. In Fig. 3.16 sono mostrati i due modi con cui, al termine della sequenza di configurazione appena descritta, è possibile salvare la richiesta: il primo effettua il salvataggio nell'archivio e contemporaneamente attiva la richiesta mentre il secondo la salva solamente nell'archivio.

Il menù *Settings* dell'applicazione permette di configurare gli ulteriori parametri necessari per una richiesta di servizio. In Fig. 3.17 viene mostrato l'intero menù nella prima immagine a sinistra e poi, proseguendo anche in Fig. 3.18, le varie interfacce relative a ciascun parametro; in ordine, la prima richiede l'inserimento di *username* e *password* fondamentali per essere identificati dal sistema, la seconda chiede se si vuole usufruire di un servizio automatico o semi-automatico, la terza richiede di specificare, tramite diverse voci, la reputazione del partner che si desidera trovare, la quarta immagine chiede quanto tempo prima dell'orario di partenza si desidera ricevere una risposta, la successiva è dedicata alla tecnologia di comunicazione che si intende adottare per connettersi al sistema (al momento solo il Bluetooth è utilizzabile) ed infine tutti i parametri sono salvati quando si uscirà dal menù *settings* per passare a quello principale.

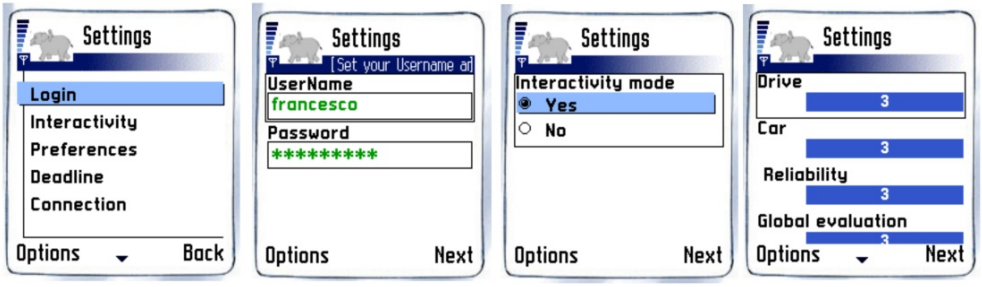

Figura 3.17: Configurazione parametri A

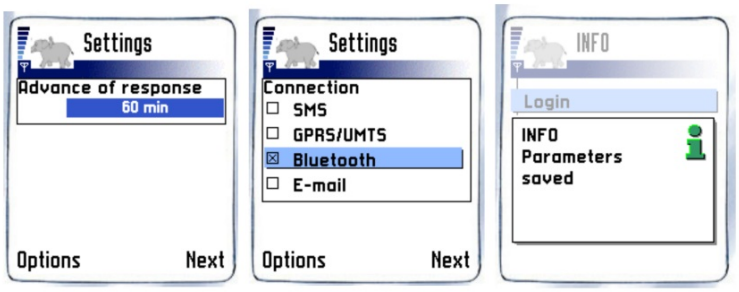

Figura 3.18: Configurazione parametri B

Infine l'ultima interfaccia che viene mostrata è quella relativa a come si presenta una domanda posta dal sistema e come viene acquisita la risposta dell'utente, la quale può essere soltanto *true* o *false*. In Fig. 3.19 è mostrato un esempio di domanda.

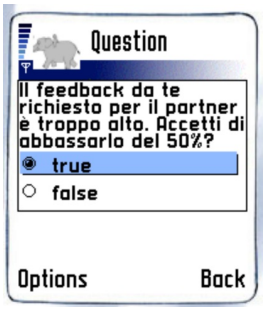

Figura 3.19: Interfaccia relativa ad una domanda posta dal sistema

3.9 Testing e risultati ottenuti

I problemi appartenenti all'usabilità di un sistema, ai telefoni cellulari e le interazioni che avvengono tra di loro sono molteplici; ciò che caratterizza queste interazioni sono una serie di

carratteristiche cha hanno la particolarità di essere tutte sensibili ai cambiamentei di scenario [19].

ANDIAMO è stato testato usando telefoni cellulari Nokia 6630 e PC/Server dotati con dispositivi Bluetooth generici di classe 2. L'applicativo ha dimostrato di funzionare correttamente senza evidenziare problemi sul modello di dispositivo da noi adoperato. In ambienti aperti come in quelli chiusi, cioè con pareti ed ostacoli, il raggio d'azione è pari a quello fissato dallo standard, cioè circa 10 metri.
Anche se la tecnologia wireless Bluetooth è molto diffusa per i suoi ridotti consumi energetici, alcuni telefoni potrebbero mostrare un consumo eccessivo della batteria quando l'applicazione è in esecuzione e il modulo Bluetooth abilitato. Si è pensato inizialmente di permettere all'utente di configurare l'applicazione mobile in modo da avere un risparmio energetico più o meno efficiente. Questo poteva avvenire permettendo all'utente di scegliere a piacimento la durata dell'intervallo che intercorre tra due ricerche di nuovi dispositivi e nuovi server, sui quali avviare nuovi agenti oppure scaricare i risultati pendenti. L'operazione continua di ricerca di dispositivi è infatti la causa principale dell'eccessivo consumo della batteria. Ma la possibilità di impostare il periodo di inattività tra due ricerche potrebbe però comportare dei problemi con un periodo di attesa piuttosto lungo; infatti aumenta la possibilità di attraversare una Rideshare Zones senza mettersi in contatto con il server che la gestisce e quindi non attivare così il personal agent oppure scaricare i risultati pendenti. L'utente così è costretto a valutare attentamente come configurare l'applicazione in base alle proprie abitudini o alla tipologia di spostamenti compiuti, cosa che può risultare difficile da comprendere e calcolare però.
Ad esempio, un utente che passa parecchio del suo tempo in una Rideshare Zones ed è in attesa di una risposta, potrebbe non essere interessato ad una continua ricerca di dispositivi ma invece dilatare il periodo di attesa fra una ricerca e la successiva fino al massimo consentito. Si è preferito comunque, nell'applicazione implementata, fissare il valore dell'intervallo che intercorre tra una ricerca e l'altra a 60 secondi; questo valore è risultato infatti, in seguito ai test effettuati, un buon compromesso tra corretto funzionamento del sistema e risparmio energetico del dispositivo.

Un punto incerto è tuttora l'utilizzo di access points Bluetooth; il loro funzionamento è stato testato fino ad ora solo in laboratorio e anche se i risultati ottenuti sono buoni, non si è ancora provato realmente a integrare questi dispositivi con le piattaforme multiagente.

Il sistema è stato testato in ambiente universitario ed ha prodotto risultati soddisfacenti su diversi scenari.
Purtroppo manca una vera e propria verifica del funzionamento dell'applicazione mobile su dispositivi non Nokia e con sistema operativo diverso da Symbian; si è notato comunque che per una compatibilità a largo spettro nel mondo delle piattaforme Java per dispositivi mobili (sono state rilasciate fino ad oggi già alcune versione di J2ME) è conveniente produrre l'applicazione mobile tenendo come riferimento le prime versioni di J2ME.
Non è stato tuttavia possibile, a causa del ridotto numero di dispositivi mobili in nostro possesso, un testing più completo e di più ampio raggio che includesse un gran numero di comu-

nicazioni wireless mentre dal lato server si è potuto comunque testare il sistema come se fosse utilizzato da un numero di utenti considerevole avviando manualmente i personal agents.

Come ultima considerazione si evidenzia il fatto che il tempo richiesto per raggiungere un accordo tra due agenti è lo stesso per ogni situazione e se si dovessero verificare dei problemi, questi sarebbero indipendenti dal numero di agenti contemporaneamente attivi sulla piattaforma.

3.10 Strumenti e tools

- Il sistema è stato sviluppato su sistema operativo Microsoft Windows XP Prof. SP2 poichè, programmando in linguaggio Java, non esiste possibilità di usare la tecnologia Bluetooth al momento su altri sistemi operativi causa la mancanza di adeguate librerie gratuite. Inoltre è necessario avere obbligatoriamente il Service Pack 2 perchè implementa il supporto al Bluetooth e lo stack Bluetooth Microsoft, senza la necessità di driver proprietari (vedremo in seguito perchè questa cosa è importante), altrimenti tutte le versioni precedenti del sistema non lo supportavano, a meno di utilizzare software ad hoc.

- Come dongle USB Bluetooth è stato utilizzato un dispositivo generico di classe 2, compatibile con le specifiche standard 1.2. Con questo dispositivo si è abilitato il computer alla comunicazione Bluetooth fino a distanze di circa 10 m.

- La comunicazione Bluetooth per applicazioni Java lato server è stata realizzata con l'utilizzo delle librerie Blue Cove, prodotto del progetto Blue Cove; esse sono un'implementazione Open Source delle APIs Bluetooth JSR-82 per Java di Sun ma prodotte per la piattaforma desktop (J2SE) e sviluppate inizialmente da Intel. Al momento solo un sottoinsieme delle APIs JSR-82 è implementato all'interno di Blue Cove e viene supportato soltanto lo stack Bluetooth Microsoft per Windows XP SP2; questo implica una serie di limitazioni e vincoli alla programmazione e utilizzo pesanti, in primo luogo a riguardo del sistema operativo che è possibile usare e poi relative ai dispositivi Bluetooth che si possono adoperare. Infatti lo stack Bluetooth Microsoft non riconosce tutti i dispositivi presenti sul mercato ma solo un piccolo sottoinsieme (esiste una lista on-line presso *http://support.microsoft.com/kb/841803*) e neanche così completo. La maggioranza dei dispositivi in commercio è legata a driver e software proprietari, come WIDCOMM o BlueSoleil per citare i due più conosciuti, e non si trovano informazioni a riguardo del supporto o meno dello stack Bluetooth Microsoft nelle loro specifiche. L'unica operazione che è possibile fare è provarli effettivamente su una macchina e sperare di non rimetterci troppo denaro.
  Sono state scelte le librerie Blue Cove per integrare il nostro sistema principalmente perchè sono gratuite e Open Source. La versione delle librerie è datata 14.05.2005. Per la traduzione dei comandi fra Java e C++ vengono usate le API Windows Socket e JNI (Java Native Interface).

- Per lo sviluppo di MIDP e applicazioni Personal Profile lato mobile Java è stato utilizzato il tool Nokia Carbide.j 1.0; esso può essere integrato nei più importanti tool di sviluppo Java moderni come NetBeans, JBuilder e Eclipse. Carbide.j permette di sviluppare applicazioni per ogni tipo di dispositivo che supporti Java ed esegua una J2ME; il tool è corredato da numerose funzionalità e da un emulatore per dispositivi mobili.

- La piattaforma Java utilizzata per lo sviluppo della piattaforma multi-agente lato server è la versione 1.5.0_06 della J2SE e il tool di sviluppo adottato è stato Eclipse SDK versione 3.1.2.
  Eclipse è un'applicazione Open Source, dietro cui esiste una vera e propria comunità di sviluppatori, destinata allo sviluppo di software in diversi linguaggi e su piattaforme diverse.

- La piattaforma ad agenti utilizzata nel nostro sistema è JADE (Java Agent DEvelopment framework) versione 3.4; essa richiede la presenza della versione 1.4.2 o più recente della Java J2SE. JADE, come meglio si vedrà nell'Appendice , attraverso un middleware compatibile con le specifiche FIPA e grazie ad un set di tool grafici che supportano il debugging e le fasi di sviluppo, permette di sviluppare sistemi multiagente in modo semplice e rapido. JADE è rilasciata sotto licenza LGPL (Lesser General Public License).

- Le basi di dati usate in ANDIAMO sono ottenute usando MySQL Database System versione Server 5.0. MySQL è il più popolare database SQL Open Source al mondo ed è sviluppato, distribuito e supportato da MySQL AB, compania commerciale fondata dagli sviluppatori di MySQL.
  Per lo sviluppo del database è stato usato MySQL Query Browser versione 1.1.20, prodotto Open Source anche questo di MySQL AB.

Capitolo 4

Conclusioni

La mobilità sostenibile è un movimento che ha come obiettivo primario la gestione della mobilità cittadina in modo da ridurre il numero di auto circolanti in città a favore di mezzi di trasporto alternativi; ciò viene raggiunto attraverso l'uso anche delle moderne tecnologie come computers e dispositivi mobili, grazie ai quali vengono realizzati sistemi di supporto alla mobilità cittadina. Uno degli interventi che è possibile adottare per ridurre le auto che circolano in città è il rideshare, cioè lo scambio di passaggi automobilistici tra i cittadini. Gli utenti ideali che possono usufruire di un servizio di questo tipo sono contraddistinti da caratteristiche particolari che permettono di raggrupparli in comunità, all'interno delle quali bisogna però dar la possibilità agli utenti di comunicare e scambiarsi informazioni tra di loro anche se non si conoscono personalmente. Quindi è necessario implementare un sistema di supporto agli utenti, con il quale essi dialogano attraverso dispositivi mobili, ad esempio, e che magari offre loro un servizio come potrebbe essere appunto il rideshare.

Gli obiettivi di questo lavoro erano quindi la progettazione e lo sviluppo di un sistema multi-agente per il servizio di rideshare accessibile da dispositivo mobile utilizzando tecnologie disponibili su larga scala, affidabili e gratuite e risolvere i problemi che affliggono le proposte viste fino ad ora, cioè offrire una buona accessibilità e supportare l'utente nell'interazione con altri utenti. Questo è stato ottenuto studiando e progettando un modello di interazione tra agenti ad hoc per il nostro caso e implementando un prototipo di sistema rideshare, basato sui dispositivi mobili e dove piattaforme multiagente e la tecnologia di comunicazione wireless Bluetooth sono state combinate per supportare comunità di utenti co-localizzati.

Il lavoro prodotto necessita ancora di ulteriori processi di verifica a riguardo della scalabilità di sistema, prima di poterlo applicare al mondo reale e delle sue prestazioni in ambienti con un numero di utenti piuttosto alto.
Il nostro lavoro aveva come prerogative quelle di essere facilmente accessibile e utilizzare tecnologie ampiamente diffuse; questi due obiettivi sono stati raggiunti puntando sui dispositivi mobili che supportano la tecnologia Bluetooth e programmabili in Java, come strumento che l'utente userà per accedere al servizio.
Ciò che introduce di veramente innovativo il modello studiato e progettato è però una piatta-

forma multiagente di supporto a comunità virtuali dove viene attuata una interazione, comunicazione, collaborazione e contrattazione tra gli agenti focalizzata all'offerta del servizio di rideshare.
Le prospettive di utilizzo di ANDIAMO nella pratica sono principalmente legate alla possibilità di creare le Rideshare Zones attraverso appositi access points e non tramite un PC; se si riuscirà veramente ad ottenere questo i costi di realizzazione per il sistema si ridurrebbero notevolmente dato che le uniche spese da sopportare sarebbero dunque composte dagli access points appunto e da un unico PC che fa da server rispetto al fatto di acquistare un PC per ogni Rideshare Zone che si vuole realizzare. Ovviamente si deve avere a disposizione una rete ethernet o Wi-Fi già esistente da sfruttare per collegare gli access points al server, cosa che al giorno d'oggi è data però per scontata. La Facoltà di Scienze potrebbe essere il primo ambito in cui testare il sistema; infatti la struttura è interamenta coperta dalla rete Wi-Fi e la comunità che la frequenta solitamente, composta da studenti, ricercatori e professori, sarebbe secondo me molto interessata a fruire di un servizio rideshare perchè è sempre stata molto attenta ai problemi legati ai trasporti e di natura ambientale.

I vantaggi derivanti dall'utilizzo del sistema ANDIAMO per offrire il servizio di rideshare riguardano principalmente la facilità con cui l'utente può usufruire di questo servizio tramite il proprio dispositivo mobile, oltre agli aspetti ambientali legati ad un servizio di questo tipo. L'utente, grazie ad ANDIAMO, spende del tempo solo per configurare la propria richiesta di servizio, operazione che dura pochi minuti, e poi non deve far altro che aspettare una risposta dal sistema; non succede come in altre proposte che egli stesso deve preoccuparsi di trovare un partner ma nel nostro sistema questa fase è delegata agli agenti.
Gli svantaggi al momento riguardano l'assenza di un sistema di gestione delle informazioni geografiche utilizzate nel sistema che quindi limita l'abbinamento degli utenti; infatti al momento sono ritenuti compatibili soltanto utenti che desiderano partire e arrivare nelle medesime località. Inoltre il concetto di *località* che si realizza attraverso l'uso della tecnologia Bluetooth, ad un servizio come questo va un pò stretto perchè l'utente può usufruire del servizio, quindi attivare richieste o ricevere risposte, soltanto presso una Rideshare Zone e quindi limita l'accessibilità al sistema che invece vogliamo sia massima.

Il futuro riserva per il nostro sistema molte nuove possibilità di sviluppo in modo da ottenere un sistema veramente flessibile, sicuro e potente. La prima che ci proponiamo di attuare è l'integrazione con database spaziali per la gestione delle informazioni geografiche usate nel servizio. Un'altra potenziale direzione in cui ci si potrà orientare è quella di integrare anche sistemi di web-services per sfruttare le potenzialità delle applicazioni su internet per raggiungere l'utente. Esiste poi la possibilità di aprirsi a e implementare ulteriori tecnologie di comunicazione come WAP, GPRS, UMTS e SMS; i cambiamenti che potrebbero introdurre queste tecnologie sono radicali dato che faranno cadere il concetto di località su cui fino ad ora ci siamo basati permettendo di accedere al sistema ovunque ma con un sacrificio pagato dall'utente: il servizio non sarà più gratuito; infatti l'utilizzo di queste tecnologie è a pagamento.
Infine l'ultimo passo da compiere sarà l'introduzione di un modulo dedicato alla sicurezza del sistema non solo orientato al buon funzionamento del servizio ma per ottenere un software

safe in gergo tecnico, cioè che non possa procurare danni all'utente; infatti si chiederanno informazioni sul documento d'identità e sulla patente per riconoscere gli utenti che utilizzeranno il sistema in modo da non permettere che il servizio venga utilizzato da pirati della strada o malintenzionati. Di quest'ultimo passo però non si è ancora verificata la effettiva fattibilità in quanto si basa su banche dati in mano a enti pubblici e quindi soggette a giuste e rigide limitazioni d'utilizzo.

Capitolo 5

Ringraziamenti

... e sono già arrivato alla laurea!
Il tempo vola, mi sembra ieri la prima volta che ho messo piede in questa facoltà. Sono riuscito a laurearmi in regola, superando difficoltà come cambi di piano e ordinamento e due anni di pendolarismo, ma il merito non è soltanto tutto mio.
La parte dedicata ai ringraziamenti può sembrare una formalità da esplicare ma in realtà i ringraziamenti non sono affatto sprecati perchè in questo mondo moderno, con tutta la buona forza di volontà che una persona può metterci, se non vi sono attorno delle persone che ci sostengono, incitano, motivano, aiutano e criticano è difficile raggiungere determinati traguardi.

Comincio quindi ringraziando veramente di cuore coloro che con me hanno frequentato fino ad ora gli studi, cioè i miei compagni di studio quotidiani, sia quelli con cui avevo già condiviso i banchi di scuola alle scuole superiori, *nonno* (Luca) e *dona* (Michele), sia quelli nuovi trovati lungo la strada in questi tre anni come Cristina, Valeria e Fabrizio. Grazie a voi per l'aiuto che mi avete dato, il sostegno e l'interesse mostrato per quanto facevo e per le risate regalate da sempre durante la pausa pranzo!!!
Ringrazio poi i miei cari amici Claudio, Mattia e Andrea per la loro compagnia nei week-end e nei periodi passati in valle e i miei compagni d'appartamento Cristian, Franco, Luca e Fabrizio con cui passo volentieri le serate qui in città.
Un ringraziamento va anche al mondo dello sport, in particolare al calcio, e del volontariato ai quali dedico la maggior parte del mio tempo libero e che mi danno sempre sensazioni ed emozioni stupende, nel bene e nel male. La possibilità di giocare a calcio in una società mi ha dato l'opportunità da sempre di potermi sfogare, scaricare la tensione, lo stress e le pressioni giocando a pallone con gli amici, mi ha permesso di fare nuove amicizie, mi ha motivato spingendomi a far sempre meglio in qualsiasi, usando il termine calcistico, campo scendessi ma in particolar modo è stata scuola di vita per me, educandomi e trasmettendomi valori che ritengo fondamentali per vivere nella società moderna. Grazie Calcio Bleggio.
Ora è arrivata l'ora di nominare una particolare associazione; talvolta bisogna ringraziare anche quanti magari non ti aiutano con uno scopo legato agli studi universitari che stai compiendo ma a riguardo di altri aspetti della vita normale. Così grazie GGQ (Gruppo Giovanile Quadra) per la straordinaria amicizia che mi ha dato, per il sostegno e la comprensione che solo dei veri

amici possono dare, per le grandi cose che con loro ho realizzato. Grazie *presidente* (Samuel), Chiara, Stefania D., *scaia* (Daniele), *ioraz* (Ivan), Licia, Stefania B., *il sommo rettore* (Nicola), *ingegnere* (Alberto), Francesca, Paolo, Alessia, *ceppa* (Mirco), Lara, Renzo, Fabiano, *jecky* (Michele), *strobo* (Davide), *ciacky* (Alessio), la mia cuginetta Sabrina che prova a distrarmi sempre in tutte le maniere ed infine il mio insuperabile amico e compagno di divertimento *tevez* (Valter).
Ringrazio la mia nonna Maria perchè con i suoi racconti, le sue osservazioni e il suo sostegno indiretto mi ha fatto capire quanto sono fortunato dato che ho l'opportunità di studiare.

Un ringraziamento doveroso e in particolare al mio relatore, il prof. Paolo Giorgini che ha accettato di guidarmi fino a qua attraverso un lungo percorso dove ho avuto anche la possibilità di poter provare a far *ricerca*, adeguata alle mie capacità ma che comunque ha prodotto buoni frutti e cosa per cui gli sono molto grato. Ringrazio poi i suoi collaboratori del BlueAgents Team, in particolare Stefano Fante e Sameh Abdel-Naby, per il grande aiuto che mi hanno dato e ArsLogica di Mezzolombardo che mi ha ospitato qualche pomeriggio.

Infine concludo con un grazie immenso alla mia famiglia la quale è stata fondamentala per raggiungere questo traguardo importante; ringrazio i miei genitori e fratelli perchè mi sono sempre stati vicini interessandosi a quanto facevo e studiavo anche se poi non avrebbero capito nulla delle mie spiegazioni, mi hanno sostenuto e incitato nei momenti di difficoltà, hanno condiviso con me i momenti di felicità e hanno sempre cercato di non farmi pesare gli immensi sacrifici a cui si sottoponevano per permettermi di studiare nelle migliori condizioni. Grazie!

Appendice A

Bluetooth e J2ME

In questa sezione verrà dapprima fornito un approfondimento relativo alla tecnologia Bluetooth con le principali fasi necessarie per instaurare una connessione. Poi verrà introdotta brevemente la piattaforma Java J2ME per dispositivi mobili e come essa supporta la tecnologia Bluetooth.

Connessione di dispositivi Bluetooth

Per comunicare tra loro due device devono stabilire una connessione. Se si conosce l'indirizzo Bluetooth dell'altro dispositivo e l'identificativo del servizio, è possibile inoltrare una richiesta di connessione. I dispositivi si connetteranno se la loro distanza non è superiore alla massima consentita dal modulo installato e se i parametri di sicurezza lo consentono; questo implica chiaramente che il dispositivo a cui ci si connette deve possedere il servizio richiesto. Le operazioni da svolgere sono tre: ricerca di dispositivi (inquiry), gestione dei servizi offerti(ricerca oppure pubblicazione) e connessione con dispositivi.

Inquiry

La prima cosa che deve fare un'applicazione di rete Bluetooth è ricercare gli altri dispositivi abilitati presenti nel proprio range di copertura dell'antenna. Per dispositivi abilitati si intendono tutti quei dispositivi Bluetooth che sono in modalità discoverable ossia in inquiry scan. Il dispositivo che inizia questa procedura spedisce in modalità broadcast un messaggio di inquiry e attende che i dispositivi presenti nella zona rispondano. Per stabilire quando un dispositivo deve rispondere sono state definite tre modalità di discoverable (general, limited e not discoverable) e due diversi tipi di inquiry (general e limited). Quando un dispositivo esegue una general inquiry tutti i dispositivi in modalità general e limited discoverable devono rispondere al messaggio. Nel caso in cui invece si esegua una limited inquiry solo i dispositivi in modalità a limited discoverable risponderanno. I dispositivi che sono in modalità not discoverable non rispondono in nessun caso a messaggi di inquiry ed è come se fossero invisibili agli altri. Per permettere di settare la modalità di discoverable le API Java JSR- 82 [5] definiscono il metodo setDiscoverable() della classe LocalDevice. L'argomento passato a setDiscoverable() rappresenta la modalità di discoverable che il dispositivo dovrà usare per rispondere a messaggi

di tipo inquiry. Per eseguire la ricerca vera e propria dei dispositivi si utilizzano i metodi della classe DiscoveryAgent che come vedremo si occupa anche della ricerca dei servizi. A tale scopo sono forniti due metodi: startInquiry() e retrieveDevices(). Come si intuisce facilmente il metodo startInquiry() da inizio al procedimento di ricerca e prende come argomento due parametri che rappresentano il tipo di inquiry da eseguire (general o limited) ed un DiscoveryListener. L'applicazione deve infatti creare una classe che implementi l'interfaccia DiscoveryListener e implementarne i metodi, che nel caso di inquiry sono deviceDiscovered() e inquiryCompleted(). Il metodo startInquiry() è una chiamata non bloccante e questo significa che ritorna prima del termine della procedura. È per questo motivo che si rende necessario il DiscoveryListener; al termine dell'inquiry infatti l'implementazione delle JABWT (Java APIs for Bluetooth Wireless Technology) prevede che venga richiamato il metodo inquiryCompleted() della classe DiscoveryListener per mettere a conoscenza l'applicazione della conclusione della ricerca. Il metodo deviceDiscovered() viene invece richiamato automaticamente ogni volta che viene scoperto un nuovo dispositivo e si riceve la sua risposta. A questo metodo viene passato come argomento un'istanza della classe RemoteDevice che appunto rappresenta il dispositivo remoto appena conosciuto e permette attraverso una serie di metodi di conoscerne alcune informazioni come l'indirizzo Bluetooth o il nome alfanumerico che lo identifica (user-friendly name). L'altro metodo messo a disposizione dalla classe DiscoveryAgent per l'inquiry è retrieveDevices() che in realtà non esegue un'inquiry vera e propria ma si limita a ritornare quei dispositivi che erano già stati scoperti da inquiry precedenti e quindi non fornisce nessun tipo di garanzia sulla loro reale presenza. Con questo metodo è possibile anche inserire nella lista dei device ritornati, una serie di dispositivi di default che possono tornare utili nel caso in cui la topologia della rete è piuttosto stabile e non soggetta a grossi cambiamenti.

Service Discovery

Una volta che si conoscono i dispositivi presenti nell'area di *operabilità*, il passo successivo è quello di determinare quali servizi questi mettono a disposizione; per fare questo si utilizza il **Service Discovery Protocol** definito dalle specifiche Bluetooth. Un'applicazione che vuole fornire dei servizi, e che per questo funge da server, deve permettere agli altri dispositivi di trovare questi servizi: li pubblica così tra i suoi service records i quali, attraverso una serie di attributi, li descrivono. Questi attributi specificano il nome del servizio, il modo con il quale connettersi ad esso, una breve descrizione del suo funzionamento ed altre informazioni utili al suo corretto utilizzo. La procedura con la quale un server rende visibili i propri service records ai potenziali client prende il nome di **Service Registration**. Questa inizia con la chiamata a Connector.open() che restituisce uno StreamConnectionNotifier e crea un apposito service record per il nuovo servizio, con una serie di attributi di default già settati in base ad una stringa che viene passata come argomento al metodo Connetor.open() e che verrà descritta in dettaglio nel paragrafo successivo. Per ora basta sapere che in questa stringa è contenuto l'identificatore univoco globale dei servizi (UUID - Universally Unique Identifier). Una volta creato il service record è possibile aggiungerci dei propri attributi con il metodo setAttributeValue() della classe ServiceRecord che prende come argomenti un identificatore e il valore dell'attributo sotto forma di istanza della classe DataElement. Arrivati a questo punto il servi-

zio non è ancora visibile agli altri dispositivi; per renderlo visibile occorre andare a modificare il contenuto del SDDB usando il metodo acceptAndOpen() sull'istanza della classe StreamConnectionNotifier ritornata da Connector.open(). Ora il servizio è correttamente registrato e il server è in attesa di client che si connettano. Vediamo ora quali sono le operazioni, definite dal **Service Discovery Application Profile**, che devono compiere i client per trovare e connettersi ai servizi. Supponiamo che sia già stata eseguita la procedura di inquiry e che abbia ritornato come risultato alcuni dispositivi. Si vuole ora verificare se qualcuno di questi dispone di un particolare servizio al quale si è interessati e del quale già si conosce il suo UUID. Per fare questo si deve richiamare su ognuno dei dispositivi trovati il metodo searchServices() della classe DiscoveryAgent che prende come argomento una lista degli UUID che si stanno cercando e una lista dei service attributes che devono essere ritornati per ogni service record trovato e identificato con uno degli UUID ricercati. Come nel caso della ricerca dei dispositivi anche in questo caso occorre implementare l'interfaccia DiscoveryListener (passando il Listener alla funzione searchServices()) e definirne i metodi servicesDiscovered() e serviceSearchCompleted(). La chiamata a searchServices() come startInquiry() non è bloccante e l'implementazione delle JSR-82 si preoccupa di richiamare il metodo servicesDiscovered() ogni volta che si ricevono dei service records di risposta. Generalmente quando riceve la prima risposta, il client sospende la ricerca invocando il metodo cancelServiceSearch(). Nulla impedisce comunque di portare a termine la ricerca e anzi può essere utile farlo perchè, nel caso in cui il servizio non sia più raggiungibile o funzionante, si potrebbero già avere delle altre possibili fonti sulle quali provare. In ogni caso al termine della ricerca viene automaticamente richiamato il metodo serviceSearchCompleted() che riporta lo stato d'uscita dell'operazione (ricerca completata correttamente, ricerca terminata dall'utente, errori, ecc.). Se la ricerca è andata a buon fine il client è ora in grado, con le informazioni contenute nel/i service record/s, di creare delle connessioni con altri dispositivi che mettono a disposizione il servizio ricercato.

Comunicare con RFCOMM

Il protocollo RFCOMM fornisce l'emulazione di una linea seriale di tipo RS- 232 tra due dispositivi Bluetooth. JSR-82 non definisce nessuna nuova classe o interfaccia per utilizzare RFCOMM ma si limita a riutilizzare delle classi e delle interfacce già esistenti e che fanno parte del **GCF (Generic Connection Framework)**. Questa scelta permette di avere un'estrema flessibilità dal punto di vista dei protocolli di comunicazione utilizzati, perchè come si vedrà permette di utilizzarne indifferentemente diversi senza quasi modificare il codice, e ne agevola il porting su piattaforme diverse. Tutti i tipi di connessione iniziano con Connector.open() che riceve come parametro una stringa che identifica il protocollo da utilizzare e il dispositivo al quale connettersi. Questa stringa segue lo standard definito dal GCF per il formato delle stringhe di connessione secondo il quale la stringa è divisa in tre parti separate nel modo seguente:

schema://target;parametri

Lo schema rappresenta il protocollo da utilizzare e nel caso di RFCOMM è btspp dove bt sta per Bluetooth e spp per Serial Port Profile che è il profilo che realizza RFCOMM. Il target nel

caso del client è l'indirizzo Bluetooth e l'identificatore del canale utilizzato dal service separati da *:*, mentre nella stringa di connessione del server il target è la parola chiave *localhost* separata dall'Universally Unique IDentifier (UUID) del servizio anche in questo caso da *:*. Infine la stringa si conclude con una serie di attributi opzionali tra i quali sono anche inclusi quelli relativi alle impostazioni di sicurezza. Un esempio di stringhe di connessione RFCOMM valide per il client sono:

btspp://00803d000001:1

btspp://008034AD2AA1:3;authenticate=true

mentre per il server:

btspp://localhost:EFAA5621975548568F27B437B6C5E6B2

btspp://localhost:93007CA747114F42B1DCCE878A6531FA;name=MyService

Di queste URL, mentre quelle lato server possono venire generate anche staticamente a livello di implementazione (basta conoscere solamente l'UUID del service ed impostare i parametri desiderati), quelle lato client vengono costruite dinamicamente in quanto bisogna conoscere le informazioni relative al server al quale ci si vuole connettere. Queste informazioni vengono recuperate dai service records che si sono ottenuti durante la ricerca dei servizi disponibili sui dispositivi trovati in fase di inquiry richiamando il metodo getConnectionURL() dell'interfaccia ServiceRecord che ritorna la stringa di connessione completa per connettersi a quel servizio. Quando la stringa di connessione lato client viene passata a Connector.open() questo ritorna un oggetto StreamConnection con il quale l'applicazione è in grado di crearsi gli stream di input ed output con i rispettivi metodi openInputStream() ed openOutputStream() per poter comunicare con il server. Se invece a Connector.open() viene passata una stringa di connessione del server (riconoscibile perchè contiene la keyword localhost) allora viene ritornato un oggetto di tipo StreamConnectionNotifier. Utilizzando il metodo bloccante acceptAndOpen() di questa classe l'applicazione si mette in attesa di connessioni da parte dei client. Solo quando qualche client si connette, il metodo risveglia il processo restituendo un oggetto StreamConnection. Come nel caso del client, anche il server utilizza questo oggetto per ottenere gli stream di input e output per leggere e scrivere dati.

In questo paragrafo abbiamo visto come si stabiliscono delle connessioni utilizzando RFCOMM. In realtà la scelta di riutilizzare le librerie del GCF rende questa procedura standardizzata a diversi protocolli di comunicazione. Per esempio nel caso in cui si volesse sostituire il protocollo RFCOMM con L2CAP l'unica cosa che dovrebbe essere modificata è la stringa di connessione che viene passata a Connector.open(), nella quale bisognerebbe semplicemente sostituire lo schema *btspp* usato per RFCOMM con *btl2cap* usato invece da L2CAP.

J2ME e Bluetooth

La tecnologia Bluetooth è applicata al giorno d'oggi a quasi ogni dispositivo mobile che venga messo in vendita e è implementata in contesti sempre più numerosi ed eterogenei; la sua diffusione è stata rapida e agevolata dal fatto che è stata progettata e rilasciata una specifica aperta e senza royalty dal Bluetooth SIG che ne definisce sia i livelli hardware, sia quelli software.

La stessa mossa non è stata però compiuta dai produttori dei moduli Bluetooth che si sono comportati in maniera ben diversa: invece di accordarsi e rilasciare generiche SDK che permettano di interfacciarsi con qualsiasi modulo, forniscono delle apposite SDK per interfacciarsi con i loro moduli, vincolando in questo modo gli sviluppatori all'uso di API proprietarie a seconda del chip-sets che utilizzano. Una diretta conseguenza è che le applicazioni Bluetooth costruite utilizzando queste API non sono portabili su device e piattaforme diverse, nonostante la tecnologia sia basata su delle specifiche standardizzate. In realtà però queste specifiche sono di tipo funzionale nel senso che descrivono come i dispositivi Bluetooth si comportano e come interagiscono tra di loro attraverso i profiles, lasciando liberi gli sviluppatori di decidere come programmare i dispositivi o come un'applicazione deve utilizzare le funzionalità di questi dispositivi e i canali di comunicazione.

Per risolvere questi problemi di compatibilità che ostacolavano l'utilizzo delle comunicazioni wireless Bluetooth nelle applicazioni, si è resa necessaria la creazione di un set di API standardizzate che permettessero lo sviluppo di software Bluetooth indipendente dalla piattaforma. Quando si decise di creare questo set di API standardizzate per la tecnologia wireless Bluetooth venne ovviamente scelto il linguaggio di programmazione Java. Oltre ai benefici della portabilità, che permettono alle API Java di essere eseguite su hardware, sistemi operativi e classi di dispositivi diversi, Java permette inoltre un rapido sviluppo delle applicazioni grazie al suo paradigma di programmazione orientato agli oggetti che ne garantisce una astrazione ad alto livello ed una comunità di sviluppatori già molto estesa.
Il Java Community Process introdusse le prime specifiche di API standardizzate per il Bluetooth nell'anno 2000. Queste specifiche conosciute come JSR-82 (Java Specification Request 82) o JABWT (Java Api for Bluetooth Wireless Tecnology) definiscono le basi per lo sviluppo di applicazioni Bluetooth e sono state pensate e progettate per poter essere utilizzate anche da sistemi di tipo CLDC (Connected Limited Device Configuration) cioè che hanno limitate capacità di calcolo, di memoria e batteria come i cellulari, ad esempio. Gli sviluppatori possono così creare delle applicazioni Bluetooth indipendenti dall'hardware utilizzato e che quindi possono essere implementate anche su dispositivi diversi purchè questi supportino JSR-82. Per poter utilizzare JABWT un dispositivo deve disporre di uno stack che sia qualificato per almeno il Generic Access Profile, il Service Discovery Application Profile e il Serial Port Profile e che permetta l'accesso al Service Discovery Protocol e ai suoi livelli RFCOMM E L2CAP.
Molta attenzione è stata posta anche alla scalabilità, in modo da permettere a questo tipo di applicazioni di poter essere eseguite su qualsiasi tipo di piattaforma Java 2 (J2ME, J2SE, J2EE) dotata del Generic Connection Framework (GCF). Proprio per questi motivi di scalabilità queste nuove API non implementano tutti i livelli ed i profiles indicati dalle specifiche Bluetooth ma si limitano a ricoprirne solo una parte. Per esempio sono supportati:

* la comunicazione dati e non quella vocale
* l'utilizzo dei protocolli
* L2CAP (solo nella versione orientata all'applicazione)
* RFCOMM

* SDP
* OBEX (OBiect Exchange protocol)

I profiles supportati sono:

* Generic Accesc Profile (GAP)
* Service Discovery Access Profile (SDAP)
* Serial Port Profile (SPP)
* Generic Object Exchange Profile (GOEP)

Le funzionalità che mettono quidi a disposizione le JABWT sono:

* la ricerca di dispositivi e servizi
* la registrazione dei servizi
* la possibilità di stabilire connessioni di tipo RFCOMM, L2CAP e OBEX
* la possibilità di eseguire queste operazioni in una modalità ïcura.

La neccessità di avere uno stack qualificato per poter utilizzare le JABWT con dispositivi diversi non va d'accordo con il fatto che i produttori di dispositivi seguono strade diverse ed offrono SDK propietari; il risultato di questo conflitto di interessi è una serie di limiti imposti agli sviluppatori come descritto in precedenza nell'enunciare i tools e le tecnologie adottate per lo sviluppo di questo progetto.

Appendice B

JADE

JADE è un progetto Open Source attorno al quale una comunità di utenti e collaboratori è cresciuta in passato e recentemente ha formato una International Governing Board.
I due aspetti concettuali attorno a cui è costruita sono le reti peer-to-peer e il paradigma ad agenti. Vediamo di capire ora cosa lega questi due concetti: in un modello peer-to-peer non ci sono distinzioni di ruolo e ciascun peer è in grado di compiere un insieme di azioni e iniziative, ciascun nodo può avviare la comunicazione, essere soggetto oppure oggetto di una ricerca, essere intraprendente e attivo e fornire capacità oppure passivo; la logica dell'applicazione non è centralizzata in un server ma distribuita tra tutti i nodi della rete. Ciascun nodo può scoprire gli altri e può entrare, condividere, lasciare una rete in ogni momento. In alcune reti peer-to-peer ogni nodo offre un servizio; è necessario quindi un sistema di indicizzazione centrale tipo pagine gialle per effettuare ricerche di peer in base ad un specifico servizio. Il paradigma ad agenti è già stato discusso in precedenza all'interno di questo lavoro (Capitolo 1, sezione 3); è basato sull'astrazione di agente, il quale è un'entità software che deve essere autonoma, intraprendente e socievole.
I due concetti sopra esposti bene si sposano e hanno evidenziato e propongono tuttora prospettive di impiego in diversi ambiti talmente valide che nel 1996 TILAB promosse la creazione di FIPA (Foundation for Intelligent Physical Agent), un'associazione non-profit internazionale di companie e organizzazioni che condividevano obiettivi e sforzi per ottenere specifiche standard per la tecnologia ad agenti. Il team di JADE da sempre ha seguito le direttive di FIPA superando con successo i test FIPA di interoperabilità nel 1999 e nel 2001 fino a che nel 2002 FIPA ha rilasciato la versione definitiva dello standard, a cui JADE ha da subito aderito. Lo standard è incentrato sull'ambiente esterno ai componenti di sistema lasciando campo libero ai programmatori per quanto riguarda le scelte implementative e l'architettura interna.

JADE è un middleware sviluppato da TILAB conforme alle specifiche FIPA, per lo sviluppo di applicazzioni distribuite multi-agente basate su architetture di comunicazione peer-to-peer e può operare in maniera invisibile in qualsiasi ambiente.
L'intelligenza, l'iniziativa, l'informazione, le risorse e i controlli possono essere completamente distribuite sia su dispositivi mobili sia su computer in rete. L'ambiente può evolvere dinamicamente con nodi, in JADE chiamati agenti, che possono apparire e scomparire nel si-

stema in accordo con le necessità e i requisiti dell'applicazione di uno specifico ambiente.
JADE è completamente sviluppata in Java e offre un set di tools che supportano il debugging e la fase di sviluppo.
JADE si basa sui seguenti cinque principi:

interoperabilità JADE è conforme alle specifiche FIPA, per cui può comunicare e collaborare con qualsiasi altro agente conforme allo stesso standard

uniformità e portabilità JADE fornisce un insieme di APIs omogenee indipendenti dalla rete e dalla piattaforma Java sottostanti.

facilità d'utilizzo la complessità del middleware è nascosta dietro un semplice e intuitivo insieme di APIs.

paghi per quanto usi i programmatori non è necessario che conoscano e sappiano come sono fatte tutte le funzionalità e caratteristiche del middleware JADE e quindi non sono obbligati ad utilizzarle tutte, senza alcun effetto collaterale o negativo sulla loro applicazione.

L'architettura di comunicazione offre messaggi flessibili ed efficienti; JADE crea e manipola una coda di messaggi di tipo ACL entranti, privati ad ogni agente. Gli agenti possono accedere alla loro coda attraverso una combinazione di vari modi basati su blocking, polling, timeout e pattern matching. È stato implementato un modello di comunicazione completamente FIPA e le sue componenti sono state chiaramente distinte e completamente integrate. protocolli di interazione, sviluppo, ACL, schemi di encoding, ontologie e protocolli di trsporto. Il meccanismo di trasporto, in particolare, è come un camaleonte poichè si adatta ad ogni situazione, scegliendo trasparentemente il miglior protocollo disponibile. Vengono attualmente usati la Java RMI, la notifica degli eventi e IIOP ma possono essere facilmente aggiunti numerosi altri protocolli. La maggior parte dei protocolli di interazione definiti da FIPA sono già disponibili e possono essere instanziati dopo la definizione del comportamento -dipendente dall'applicazione- di ogni stato del protocollo.

Appendice C

Cultura Implicita: SICS

Il concetto di Cultura Implicita è stato introdotto per la prima volta da Blanzieri e Giorgini in [9]. Essi indicarono come fenomeno di Cultura Implicita quello che si verifica quando un nuovo agente che entra in una comunità si comporta in maniera coerente con la *cultura* del gruppo di agenti di quella comunità senza interazione diretta o sforzo addizionale da parte di alcuno.
La Cultura Implicita permette ai nuovi membri di una comunità di comportarsi in accordo con la cultura della comunità stessa e dell'ambiente in cui si trovano sfruttando tecniche di Data Mining e Collaborative Filtering. Il supporto della Cultura Implicita diviene utile, se non indispensabile, sia per gli agenti artificiali che per quelli umani. Pensiamo infatti a tutte le volte che per potersi inserire ed amalgamare con un gruppo di persone si è tenuti a comportarsi in accordo con le regole e le usanze di questo gruppo. Questo discorso vale anche per gli agenti legati ad utenti mobili; infatti un Personal Agent deve conoscere il nuovo ambiente in cui opera per poter lavorare al meglio ed ottenere i migliori risultati possibili per il proprio utente.
Lo scenario a cui bene si applica la Cultura Implicita e i benefici che apporta il suo insermento in piattaforme multiagente può essere riassunto come di seguito: un agente che opera in un ambiente che conosce poco si comporta in maniera non ottimale; se un gruppo di agenti agisce nello stesso ambiente, è possibile monitorare e osservare le loro azioni, ricavandone informazioni da usare poi per migliorare la conoscenza e il comportamento di ciascun agente. Allora si può pensare di risolvere il problema fornendo la conoscenza oppure la capacità di imparare agli agenti, ma in alcuni dominii la descrizione appunto della conoscenza può essere veramente difficile da rappresentare e la capacità di apprendimento osservando il comportamento di altri richiedere agenti molto complicati. La soluzione è, come già enunciato in precedenza, fare in modo che gli agenti si comportino come i membri di un gruppo che **implicitamente** appartengono alla stessa **cultura** sfruttando il supporto di un modulo esterno: **SICS**.

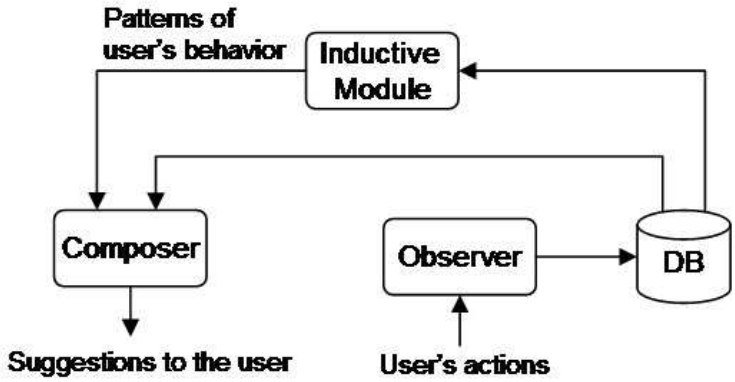

Figura C.1: Architettura generale di SICS

SICS

SICS (System for Implicit Culture Support) è un modello di sistema deputato al supporto della Cultura Implicita [10].
SICS va inserito all'interno delle applicazioni per raccogliere il comportamento degli agenti presenti nel sistema e registrare le azioni compiute in precedenza dagli agenti in modo da poter suggerire ad un nuovo agente che entra nella comunità le azioni coerenti da intraprendere; queste sono dette *azioni culturali* e sono rappresentative del comportamento dei membri del gruppo.
Il modello SICS è composto fondamentalmente da tre componenti e le loro interazioni sono visibili in Fig.C.1:

Observer memorizza in un database le informazioni a riguardo delle azioni compiute dagli utenti (osservazioni)

Inductive Module analizza le osservazioni prodotte dall'Observer e produce modelli comportamentali attraverso tecniche di Data Mining

Composer produce suggerimenti sulla base delle informazioni ottenute dai moduli Observer e Inductive Module

In un sistema multiagente un modulo SICS può essere sia una capacità generale del sistema, sia una capacità di un singolo agente. Nel primo caso, il SICS osserva tutti gli agenti che agiscono nel sistema e che manipolano l'ambiente. Nel secondo caso invece il SICS viene applicato a quello che l'agente singolarmente è capace di fare, ovvero a quello che è in grado di osservare e cambiare, cioè la parte di ambiente e gli agenti con cui interagisce. In conclusione il SICS, generale o specifico che sia, influenza l'intero sistema.
Maggiori dettagli a riguardo del framework di Cultura Implicita sono disponibili presso [6].

Bibliografia

[1] Web Site promosso dalla Provincia di Trento — http://www.abertech.it/carpooling/default.asp.

[2] Portable Cicero— http://giove.cnuce.cnr.it/Cicero.html.

[3] JADE: Java Agent DEvelopment Framework website — http://jade.tilab.com/.

[4] FIPA: Foundation for Intelligent Physical Agents — http://www.fipa.org/.

[5] JSR-82: Java APIs for Bluetooth — http://www.jcp.org/en/jsr/detailsid=82.

[6] Implicit Culture website — http://dit.unitn.it/implicit/.

[7] Deroghe per mezzi nuovi e car pooling. *L'Adige -17.11.2006*, p. 23, 2006.

[8] A. Birukov, E. Blanzieri e P. Giorgini. Implicit: An agent-based recommendation system for web search. In *Proceedings of the 4th International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 618–624. ACM Press, 2005.

[9] E. Blanzieri e P. Giorgini. From collaborative filtering to implicit culture: a general agent-based framework. In *WARS in Autonomous Agents*.

[10] E. Blanzieri, P. Giorgini, P. Massa e S. Recla. Implicit culture for multi-agent interaction support. In *CooplS '01: Proceedings of the 9th International Conference on Cooperative Information Systems*, pp. 27–39. Springer-Verlag, London, UK, 2001. ISBN 3-540-42524-1.

[11] C. Carabelea e M. Berger. Agent negotiation in ad-hoc networks. In *Proceedings of the Ambient Intelligence Workshop at AAMAS'05 Conference, Utrecht, The Netherlands*, pp. 5–16. 2005.

[12] C. D. Claudio Cubillos e F. Guidi-Polanco. Passengers trips planning using contract-net with filters. *8th International IEEE Conference on Intelligent Transportation Systems Vienna, Austria*, 13-15 September 2005.

[13] C. Cubillos e C. Demartini. Madarp: An agent-based passenger transport framework. In *Intelligent Transportation Systems*, pp. 437 – 442. 2005. Proceedings. 2005 IEEE, 2005.

[14] S. I. del Ministero dell'ambiente. Decreto 27 marzo: Mobilità sostenibile nelle aree urbane. In *Gazzetta Ufficiale della Repubblica Italiana n 179*. 3 Agosto 1998.

[15] S. I. del Ministero dell'ambiente. Decreto 20 dicembre: Promozione del car sharing. In *Gazzetta Ufficiale della Repubblica Italiana n 80*. 5 Aprile 2001.

[16] D.J.Dailey e D.Meyers. A statistical model for dynamic ridematching on the world wide web. *ITSC 99 Tokio, Japan*, 5-8 October, 1999.

[17] F.G. Il comune lancia il car pooling. *L'Adige -12.09.2006*, p. 23, 2006.

[18] M. G., B. B. e H. A. Applications of multi agent systems in traffic and transportation. In *IEE Transactions on Software Engineering*, p. 144(1):51 60. 1997.

[19] V. H.-h. C. Henry Been-Lirn Duh, Gerald C. B. Tan. Mobile usability: Usability evaluation for mobile device: a comparison of laboratory and field tests. In *8th conference on Human-computer interaction with mobile devices and services, MobileHCI'06*. September, 2006.

[20] A. B. Kothari. Genghis - a multiagent carpooling system. *B.Sc. Dissertation work, submitted to the University of Bath*, May 11, 2004.

[21] A. Rakotonirainy, S. W. Loke e A. Zaslavsky. Multi-agent support for open mobile virtual communities. In *Proceedings of the International Conference on Artificial Intelligence (IC-AI 2000) (Vol I), Las Vegas, Nevada, USA*, pp. 127–133. 2000.

[22] H. Rehingold. *Smart Mobs: The Next Social Revolution*. Basic Books; Reprint edition, October, 2003. ISBN 0738208612.

[23] S. S. e et al. Carsharing and partnership management an international perspective. *Paper N0.99-0826*, 1999.

[24] Y. Shoham. An overview of agent-oriented programming. In *In: J.M. Bradshaw, ed. Software Agents. AAAI Press/The MIT Press*, pp. 271 – 290. 1997.

[25] B. Volha, G. Paolo e F. Stefano. Toothagent: a multi-agent system for virtual communities support. In *Technical Report DIT-05-064*. Informatica e Telecomunicazioni, University of Trento, 2005.

[26] E. W. Walbridge. Real time ridesharing using wireless pocket phones to access the ride matching computer. In *Vehicle Navigation and Information Systems Conference Proceedings/6th International VNIS*, pp. 486 – 492. July, 1995.

[27] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley and Sons (Chichester, England), February, 2002.

[28] Y. X. Xi SHI, Jianming HU e J. SONG. A simulation study on agent-network based route guidance system. *8th International IEEE Conference on Intelligent Transportation Systems Vienna, Austria*, 13-16 September 2005.