# Sommario

# 1. Introduzione

Da circa un decennio si sta assistendo ad uno sviluppo tecnologico profondo, caratterizzato da una forte diffusione a livello di massa che sta cambiando radicalmente il modo con cui le persone si rapportano tra di loro e con la realtà che le circonda.

Il settore nel quale questo progresso è stato più evidente è senza dubbio quello della comunicazione, che sta progredendo verso una situazione nella quale tutto dovrà poter comunicare. L'enorme diffusione di dispositivi di comunicazione mobile, cellulari su tutti, ha spinto la ricerca di nuovi metodi di comunicazione, non solo vocali ma anche digitali per sfruttare appieno le loro potenzialità. Oggi i cellulari non solo permettono la comunicazione tra persone via voce ma anche lo scambio di informazioni in formato elettronico con qualsiasi tipo di unità di calcolo. Questo permette anche ai device mobili di avere accesso a banche dati che dispongono di grandi quantità di informazioni, nonostante vi siano evidenti limitazioni legate al loro utilizzo diretto per problemi di visualizzazione e memorizzazione.

La gestione di queste risorse è stato un altro tema che ha suscitato l'interesse della ricerca in questi ultimi anni. Avere a disposizione grandi quantità di dati è certamente una cosa positiva, ma può anche risultare di scarso aiuto se non viene gestita in modo da facilitare le ricerche in merito a dei particolari argomenti e cioè "scartando" le cose che non sono rilevanti. Si sono resi necessari quindi dei meccanismi che permettessero di filtrare i risultati; l'approccio iniziale è stato quello di farlo in maniera oggettiva, sulla base di regole pre-configurate uguali per tutti gli utenti, poi successivamente si è cercato di arrivare ad avere dei "filtri soggettivi" che tengono conto del particolare utente con il quale si sta interagendo, attraverso una customizzazione sulle preferenze del singolo. La tendenza attuale è quella di non limitarsi più a considerare l'utente come entità a se stante, ma come parte integrante di un insieme di utenti che utilizzano lo stesso servizio e che quindi possono condividere le esperienze acquisite per aumentarne la qualità (collaborative filtering).

Da queste ultime considerazioni sulla comunicazione mobile e sulla gestione delle informazioni è nata l'idea per questo lavoro. L'obiettivo è duplice. Innanzitutto si vuole studiare e progettare un'architettura che permetta agli utenti di interfacciarsi e comunicare, con dispositivi di tipo mobile, con un sistema in grado di acquisire, gestire e condividere la conoscenza acquisita dagli utenti sotto forma di esperienza nell'uso delle risorse che il sistema stesso offre; la condivisione della conoscenza si può considerare realizzata sulla base di raccomandazioni che altri utenti (considerati simili per interessi o gusti) propongono ad un utente in seguito ad una ricerca su qualche argomento. Infine, sulla base dell'architettura progettata, ci siamo proposti di sviluppare, attraverso un prototipo software, una parte di questa ed in particolare quella legata alla comunicazione tra i device mobili ed il sistema.

Un concetto importante sul quale si è puntato è quello della "località" della conoscenza che sfrutta una caratteristica peculiare dei device mobili, ovvero quella di essere dove è fisicamente l'utente. Questo permette di realizzare un sistema in grado di fornire dei suggerimenti contestualizzati perché forniti da altri utenti che fisicamente frequentano quello stesso ambiente. In questo modo anche riferimenti a risorse "globali" assumono una rilevanza particolare legata all'uso che ne viene fatto in quello specifico ambito.

Per raggiungere gli obiettivi prefissati si è cercato dapprima di individuare i possibili scenari d'utilizzo e da questi si è iniziato a progettare l'architettura del sistema con un occhio di riguardo particolare alle problematiche che l'uso dei device mobili comportano. Alla fine di questa fase di progettazione come detto si è provveduto alla realizzazione di un prototipo che implementa la comunicazione tra i device mobili ed un singolo aceess-point del sistema.

Per quanto riguarda la comunicazione la tecnologia che è stata scelta è il Bluetooth. Le motivazioni possono essere così riassunte:

- è una tecnologia nuova, ma già ampiamente diffusa;
- è stata progettata per poter essere implementata anche su dispositivi con limitate capacità di calcolo e memorizzazione come i device mobili (cellulari, PDA);
- è wireless e non richiede quindi l'uso di cavi o adattatori;
- non richiede in caso di connessioni con altri dispositivi procedure di configurazione manuale da parte dell'utente, perché il tutto è automatizzato;
- ha un raggio di copertura dell'antenna che va da 10 a 100 metri e questo si addice all'utilizzo che ne abbiamo previsto;

In fase di progettazione, analizzando le caratteristiche del sistema, la soluzione trovata per la sua implementazione è stata quella di abbinare all'uso di un sistema multi-agente delle tecniche di collaborative-filtering. Questo perché una caratteristica fondamentale del sistema è quella dell'indipendenza e dell'autonomia nei confronti dell'utente nel raggiungimento dell'obiettivo principale. Questa indicazione infatti già ci indirizza verso un tipo di programmazione specifico che è quello della programmazione orientata agli agenti o AOP (Agent-Oriented Programming). In questo paradigma di programmazione gli oggetti vengono chiamati agenti e la loro caratteristica fondamentale è quella di essere entità autonome in grado di decidere quale azione eseguire senza l'intervento dell'utente, sulla base dell'osservazione dell'ambiente nel quale si trovano. Ovviamente il nostro sistema è composto da più utenti che debbono, attraverso gli agenti ad essi associati, collaborare, interagire, comunicare creando così un sistema detto multi-agente.
A questo punto occorre associare al sistema multi-agente un meccanismo che attraverso la collaborazione degli agenti riesca ad ottenere, dalle osservazioni fatte sul loro comportamento, i risultati che abbiamo precedentemente descritto, attraverso un filtraggio delle informazioni globali che tenga conto delle scelte e dell'esperienza degli utenti considerati "simili" per aree d'interesse. Vedremo che nell'implementazione del prototipo si è deciso di utilizzare a tale scopo un sistema chiamato System for Implicit Culture Support (SICS).

Il lavoro che vado a presentare è cosi articolato:

- nel capitolo 2 si cercheranno di individuare degli scenari specifici di utilizzo del sistema, ossia ne contestualizzeremo l'impiego in un caso reale. Verranno analizzati il concetto di località della conoscenza ed i possibili servizi offerti;
- nel capitolo 3 verrà descritta l'architettura progettata. Vedremo come un sistema multi-agente abbinato all'uso di tecniche di collaborative filtering può essere una soluzione al problema dell'acquisizione, gestione e condivisione della conoscenza. Infine verranno prese in considerazione alcune problematiche legate all'uso di device mobili e ne verranno proposte delle soluzioni architetturali;
- nel capitolo 4 descriveremo il prototipo realizzato, BTImplicitBoard. Ne analizzeremo le scelte implementative ed i protocolli utilizzati;
- nel capitolo 5 vedremo di tirare le somme del lavoro svolto, di trarne delle conclusioni e di immaginare eventuali sviluppi futuri.

# 2. Uno scenario collaborativo in un contesto localizzato

## 2.1 Premessa

Ricapitolando quanto detto finora, lo studio che si è deciso di affrontare è quello di un sistema software in grado di acquisire informazioni relative all'esperienza ottenuta dagli utenti, attraverso le loro operazioni, in modo tale da poterle utilizzare come conoscenza al servizio degli altri utenti. Abbiamo già detto inoltre che si è deciso di creare un sistema con il quale gli utenti si interfacciano attraverso dei device mobili via Bluetooth.
Un tema fondamentale da questo punto di vista è sicuramente il concetto di esperienza perché è proprio sulla base di questa che si riesce ad ottenere una forma di conoscenza implicita che rappresenta un bagaglio culturale di notevole spessore. Quante volte capita di ricorrere all'aiuto di altre persone quando si hanno difficoltà o quando si ha a che fare con una nuova realtà ancora poco conosciuta. L'aiuto che ci possono dare le altre persone è chiaramente fondamentale perchè semplifica ed accelera il nostro apprendimento, grazie a consigli e suggerimenti che sono frutto di loro esperienze passate.
Una dimostrazione di quanto sia importante questo tipo di collaborazione tra le persone è data per esempio dall'enorme successo che hanno avuto i forum di discussione su internet, nei quali le persone mettono a disposizione la propria esperienza e quindi le proprie conoscenze e si avvantaggiano di quelle di altre per approfondire tematiche specifiche.
Questo tipo di condivisione della conoscenza è tuttavia lo stesso di quello di cento anni fa e più perché richiede un coinvolgimento diretto da parte degli utenti; quello che è cambiato è solo il modo con il quale le persone possono interagire con le altre, che ha

permesso di azzerare il “fattore distanza” e quindi ha reso teoricamente possibile il confronto con qualsiasi persona della terra.
Vista l’importanza che può avere l’esperienza si è cominciato a pensare se questa potesse essere in qualche modo acquisita in modo autonomo dai sistemi informatici, in relazione alle operazioni che gli utenti svolgono su di essi, in modo tale poi da riuscire a condividere la conoscenza, che deriva dalle esperienze osservate, agli altri utenti in un modo trasparente all’utente stesso.
Sostanzialmente in questa nuova concezione di condivisione della conoscenza sono cambiati i ruoli rispetto all’approccio tradizionale. Mentre nei sistemi classici è l’utente che acquisisce conoscenza e decide di condividerla utilizzando come mezzo di divulgazione i sistemi informatici, nel nuovo scenario è lo stesso sistema informatico che acquisisce esperienza osservando le azioni svolte dai suoi utenti ed al tempo stesso le condivide agli altri.

Per approfondire lo studio si è reso necessario individuare degli scenari specifici di utilizzo ossia si è cercato di contestualizzare l’impiego di questo sistema in un caso reale anche ai fini della realizzazione di un successivo prototipo dimostrativo.
Per fare questo bisogna innanzitutto partire considerando i due elementi fondamentali e caratterizzanti che sono il modo di interazione degli utenti ed i possibili servizi offerti dal sistema.

## 2.2 Uno scenario d’utilizzo replicabile

Per quanto riguarda la comunicazione, la scelta di realizzare un sistema ad-hoc per device mobili già di per se comporta delle conseguenze rilevanti, come per esempio le problematiche relative allo spostamento degli utenti da un posto all’altro e conseguentemente il carattere al contempo locale e distribuito; locale perché i modi con i quali i device mobili ed in particolare i cellulari ed i palmari possono comunicare è limitato ad un raggio piuttosto ristretto; distribuito per rendere disponibili più punti d’accesso allo stesso sistema in modo da “seguire” l’utente nei suoi spostamenti nei punti in cui potrebbe tornare utile ai fini dell’applicazione.

Potremmo quindi cominciare ad immaginare come ambientazione dello studio quella di una struttura universitaria. Possiamo pensare per esempio di installare diversi punti di comunicazione con il sistema in diversi posti all'interno dell'edificio. Ognuna di queste unità (successivamente access-point) è in grado via Bluetooth di accettare richieste dagli utenti e fornire le risposte adeguate. Ovviamente per avere una situazione come quella appena descritta è necessario che gli access-point siano collegati tra loro attraverso una rete cablata o wireless che permetta di condividere le informazioni relative al sistema ed agli utenti (Figura 1). Le informazioni del sistema, sulla base delle quali gli utenti poi acquisiranno la loro conoscenza, si possono immaginare memorizzate in diverse sorgenti, alle quali tutti gli access-point possono accedere via Lan, mentre le informazioni che riguardano gli utenti e le loro preferenze vedremo più avanti come vengono gestite e condivise quando approfondiremo il discorso sui sistemi multi-agente ed Implicit.

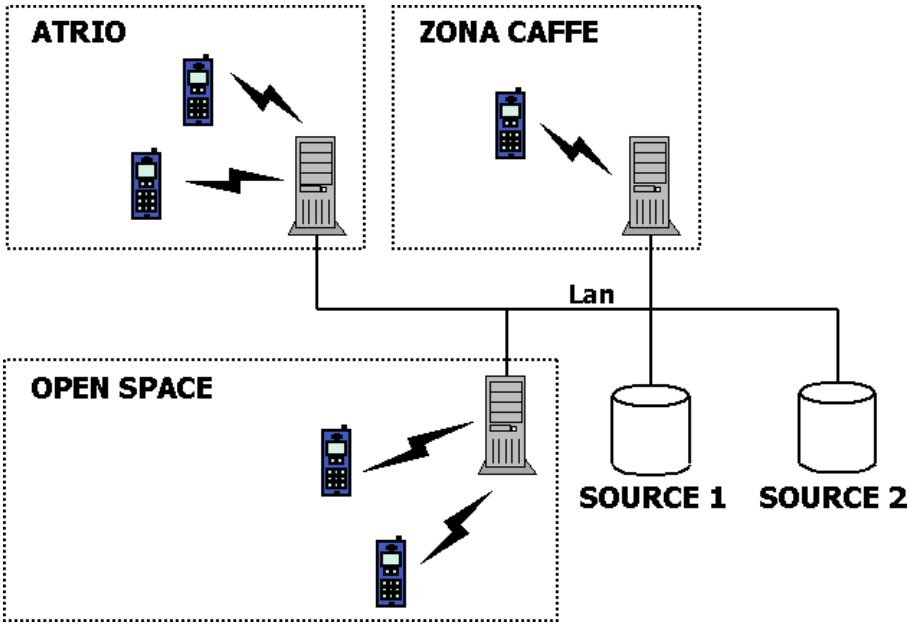


**Figura 1:** esempio di distribuzione degli access-point in un edificio universitario

Ampliando il discorso sull'ambientazione, va sottolineato che il sistema così come è stato presentato può essere replicato e contestualizzato in diversi ambiti. Per esempio possiamo immaginare di implementarlo in ogni facoltà dell'ateneo, nella biblioteca comunale, nelle agenzie di viaggio e più in generale in qualsiasi sistema informativo in

grado di fornire risorse utili ed acquisibili dagli utenti, formando così una scenario più esteso ed eterogeneo dal punto di vista dell'utilizzo che ne viene fatto (Figura 2).

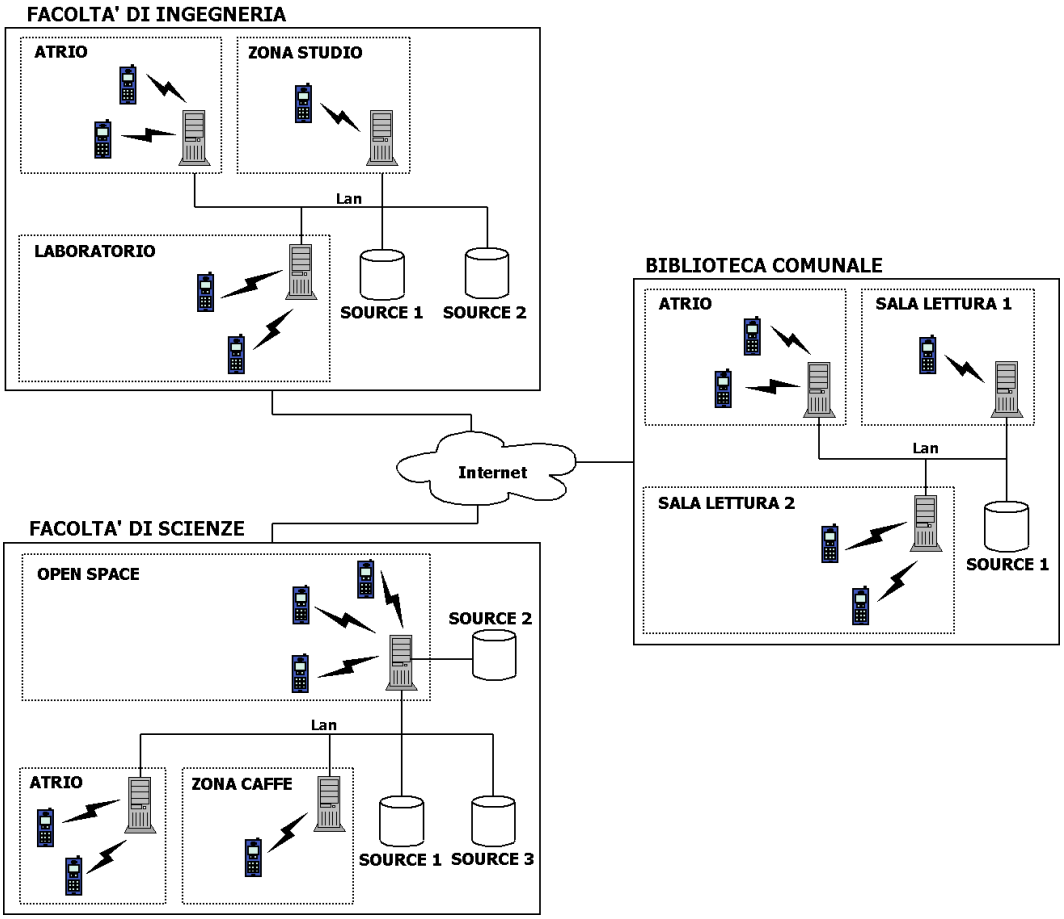


**Figura 2:** replicazione del sistema in contesti diversi

### 2.2.1 Esperienze contestualizzate per una conoscenza localizzata

In riferimento alla Figura 2 e' bene sottolineare fin d'ora (questo concetto verrà ripreso ed approfondito quando si descriverà l'architettura del sistema), che l'unione di tutti

questi diversi ambienti[1] non è finalizzata ad ampliare i contesti di interazione tra gli utenti o allargare la quantità di informazioni disponibili localmente ad ogni ambiente. Un concetto fondamentale che si vuole garantire in questo sistema infatti è quello di "località" della conoscenza. Questo aspetto va decisamente in controtendenza rispetto al trend generale degli ultimi anni che è quello di avere sistemi ultra distribuiti nei quali le informazioni sono il più possibile globalizzate. Questa scelta è stata fatta per sfruttare e ricalcare una caratteristica peculiare dei device mobili che è appunto quella della loro località, che può essere riassunta dicendo "dove c'è il device, c'è l'utente", ovvero le persone che interagiscono in un ambiente lo frequentano accumulando delle esperienze contestualizzate. Poiché il nostro sistema è basato sulla condivisione della conoscenza acquisita attraverso l'esperienza, questo approccio permette sia di specializzare il sistema in relazione al particolare contesto d'impiego, sia di assegnare allo stesso utente profili indipendenti e diversificati.

## 2.3 I possibili servizi offerti

L'altro punto da considerare come detto sono i tipi di servizio che possono essere offerti. Sappiamo che la funzione principale del nostro sistema è quella di filtrare i risultati disponibili nella loro totalità per renderli più significativi in relazione all'esperienza e competenza degli altri utenti. Questo significa che i risultati restituiti in seguito ad una ricerca tengono conto di una customizzazione da collaborative filtering che ritorna i riferimenti alle risorse che altri utenti considerati simili per gusti o interessi hanno precedentemente apprezzato.

La categorizzazione degli utenti in base alla loro esperienza su specifiche aree di interesse permette di non limitarsi a ritornare solamente le risorse consigliate ma può essere utilizzata anche per restituire dei contatti agli utenti stessi, che potrebbero essere utilizzati per mettersi in contatto con persone che hanno dimostrato di avere interessi comuni e frequentano fisicamente quel posto.

La conoscenza dei gusti e delle preferenze degli utilizzatori del sistema permette anche di arrivare ad un grado di indipendenza tra le richieste dell'utente e le risposte

[1] per ambienti intendiamo o singoli access-point o gruppi di access-point localizzati in uno spazio considerato unico ai fini dell'applicazione.

ritornate, che consente di dissociare in parte l'attività dell'utente e quella del sistema, cercando di rendere quest' ultimo propositivo o proattivo. Per capire meglio ritorniamo all'analogia tra le relazioni che esistono fra persone, come per esempio due amici. Se sono amico di una persona significa che la conosco abbastanza bene, so cosa le piace, quali sono i suoi interessi. Supponiamo che il mio amico sia un grosso fan di un certo gruppo musicale. Se un giorno, navigando per la rete, scopro casualmente che quel gruppo farà un concerto a breve, la prima cosa che faccio è farlo sapere al mio amico, ovviamente senza aspettare che lui mi chieda qualcosa a riguardo; questo perché so che la cosa potrebbe farli piacere (magari no, perché nel frattempo ha cambiato gusti musicali ma in ogni caso la cosa non li darà fastidio e verrà apprezzata). Dall'esempio appena proposto possiamo capire meglio cosa si stava intendendo pocanzi quando si parlava di indipendenza tra richieste e risposte. Se dall'osservazione delle precedenti richieste effettuate e dalle azioni compiute da un utente, il sistema riesce ad ipotizzare e delineare degli argomenti ritenuti rilevanti per l'utente, allora si può pensare di inviare degli eventuali aggiornamenti delle risorse disponibili in materia anche senza delle esplicite richieste. Allo stesso modo se ci sono delle richieste rimaste incompiute per assenza di risorse disponibili, nel momento in cui queste vengono inserite è sicuramente utile farlo sapere all'interessato anche senza una sua nuova ricerca.

In riferimento agli ultimi esempi è bene evidenziare e rimarcare ancora una volta il concetto fondamentale di "autonomia" che deve essere garantita al sistema e grazie alla quale è possibile ottenere dei servizi come quelli appena descritti in modo trasparente all'utente. Nello specifico, non è l'utente che definisce i suoi campi d'interesse in modo esplicito, come potrebbe fare per esempio nel caso in cui si iscriva ad una mailing list su un certo argomento, ma è l'osservazione del sistema sulle sue azioni compiute che permette di tracciarne un "profilo" e di raggrupparlo in insiemi di utenti con caratteristiche simili così da poter sfruttare le potenzialità del collaborative filtering.

Parlando dei possibili servizi offerti è necessario prendere in considerazione i tipi di risorse che il sistema può gestire. Partiamo dal presupposto di avere a disposizione, memorizzate in formato elettronico in diverse sorgenti, molte informazioni legate a risorse disponibili su diversi argomenti. Queste informazioni possono essere pensate come dei riferimenti a delle risorse che possono essere di vario tipo: elettroniche (on-line o meno), cartacee o altro (film, convegni, musica, viaggi).

Le caratteristiche dei device mobili però fanno sì che molte di queste informazioni non siano ad essi direttamente consultabili, per i soliti problemi di visualizzazione e memorizzazione, anche se in queste direzioni ultimamente si stanno facendo dei grossi progressi. Un riferimento ad un film o ad un libro per esempio può essere acquisito interamente, mentre nel caso di siti web, e-book e tutorial elettronici ci si deve limitare a memorizzare semplicemente il link. In quest'ultimo caso però l'informazione "incompleta" memorizzata sul cellulare può venire successivamente trasferita, sempre via Bluetooth in modo automatico, al computer di casa semplicemente avvicinando il cellulare al computer (ovviamente il Pc deve essere abilitato alla comunicazione Bluetooth). Questo è possibile grazie ad una funzionalità messa a disposizione dal Bluetooth che è quella di permettere ai dispositivi dotati di questa tecnologia di comunicare tra loro in modo automatico attraverso la creazione e riconfigurazione dinamica di reti ad-hoc. In questo modo l'utente ha la possibilità (se dispone di un collegamento Internet) di completare l'informazione ricevuta, visitando il sito o il forum consigliato oppure scaricando il file suggerito. I riferimenti a risorse on-line inoltre potrebbero per esempio essere inseriti automaticamente all'atto del trasferimento su Pc in qualche cartella apposita nei bookmarks del browser, così come i downloads potrebbero essere fatti partire in modo automatico alla successiva connessione internet.

### 2.3.1 Customizzazione collaborativa

La customizzazione da collaborative filtering rispetto a quella più classica content-base, che si basa sulle correlazioni tra il contenuto delle risorse e le preferenze dell'utente, tiene maggiormente conto delle implicazioni sociali del procedimento di raccomandazione [ 1 ], ossia invece di raccomandare elementi simili a quelli che l'utente ha dimostrato di gradire in passato, raccomanda elementi ai quali altri utenti simili hanno dato un feedback positivo. Questo modo di operare rispecchia quello che avviene quotidianamente nei rapporti interpersonali, quando per esempio ci si affida al consiglio degli amici in merito a delle azioni da fare come per esempio l'acquisto di un cd oppure la visione di un film, etc.

I computer permettono di automatizzare questo processo e di applicarlo ad un più grande numero di utenti, senza necessità di specificare quali sono le persone che hanno gusti simili ai nostri perché esse vengono individuate dal sistema in maniera automatica, e ovviamente possono essere persone mai conosciute ma con gusti simili ai nostri.

Il filtraggio delle informazioni collaborativo permette di superare alcune limitazioni dell'approccio content-base, come per esempio l'estrazione e la codifica delle caratteristiche delle risorse disponibili. Questo perché mentre quest'ultimo raccomanda elementi in base al contenuto, il collaborative filtering lo fa sulla base della storia del loro utilizzo e quindi le raccomandazioni dipendono dalla qualità degli oggetti piuttosto che dalle loro proprietà oggettive. In questo modo non è necessario che gli oggetti filtrati siano analizzabili (automaticamente o meno) per estrarne le caratteristiche descrittive e così facendo si amplia il range dei possibili elementi raccomandati a film, musica, arte, viaggi, etc… Altra importante caratteristica dei sistemi collaborativi è che il sistema può raccomandare all'utente oggetti che sono molto diversi tra loro dal punto di vista del contenuto da quelli che magari erano stati graditi precedentemente poiché la valutazione si basa sulle opinioni di altri utenti [ 1 ].

Anche il collaborative filtering ha però delle problematiche. Una risorsa appena inserita nel sistema non è stata "valutata" da nessuno e per questo non potrà essere consigliata in nessun modo, anche se potrebbe rappresentare una risorsa di notevole interesse, così come un utente "novello" che non ha compiuto azioni non può essere confrontato con gli altri sulla base delle sue preferenze (azioni) e quindi non è soggetto a raccomandazioni [ 1 ].

#### 2.3.1.1 Un esempio

Vediamo ora un esempio più concreto di come il sistema potrebbe consigliare all'utente delle risorse attraverso una customizzazione collaborativa.

Partiamo con il presupposto che il sistema sia già avviato e che quindi gli utenti abbiano già compiuto diverse azioni. Come detto il sistema tiene in qualche modo traccia delle azioni eseguite ed in particolare dei feedback positivi che riceve dagli utenti. Per feedback positivo intendiamo un'azione che possa essere interpretata come

sinonimo di gradimento della risorsa associata all'azione compiuta. Ce da tenere presente a questo proposito che queste preferenze possono essere di due tipi, esplicite o implicite. Esplicite se l'utente esprime coscientemente la sua preferenza ad una risorsa, per esempio assegnadole un voto in relazione al gradimento, implicita se questa valutazione viene fatta direttamente dal sistema e quindi all'insaputa dell'utente [ 1 ], basandosi per esempio sulla storia degli acquisti, sulle prenotazioni eseguite in precedenza oppure semplicemente sul fatto che una risorsa viene memorizzata piuttosto che scartata. Poiché noi vogliamo un sistema di acquisizione, gestione e condivisione della conoscenza autonomo, faremo riferimento per il nostro sistema ai feedback impliciti.

Supponiamo che l'utente A interroghi con il suo cellulare il sistema in merito all'argomento "musica". Il sistema per prima cosa deve ricercare altri utenti che possano essere considerati simili sulla base delle scelte passate e vedere quali sono state le loro scelte in relazione alla ricerca in corso. Prendiamo come esempio la tabella sottostante, che in modo semplicistico riassume le preferenze degli utenti A, B, C e D in materia di musica.

| RISULTATI | UTENTI | | | |
|---|---|---|---|---|
| | **A** | **B** | **C** | **D** |
| **Cd** | | | | |
| cd_1 | **x** | x | x | |
| cd_2 | **x** | | x | |
| cd_3 | | | | |
| cd_4 | **x** | | x | |
| cd_5 | | x | x | |
| **Siti web** | | | | |
| web_1 | **x** | | | |
| web_2 | | x | | |
| web_3 | **x** | | x | |
| web_4 | | | | |
| **Libri** | | | | |
| libro_1 | | | x | |
| **Mp3** | | | | |
| mp3_1 | **x** | | x | |
| mp3_2 | | x | | x |
| mp3_3 | | x | x | x |
| mp3_4 | **x** | | x | x |

**Tabella 1:** preferenze degli utenti in materia di musica. In azzurro vengono evidenziate le scelte fatte dell'utente A in modo da riuscire ad identificare altri utenti con gusti simili. In verde sono riportate le scelte dell'utente C non ancora valutate da A e ritornate come suggerimenti.

Come si può vedere le scelte dell'utente C rispecchiano in maniera significativa quelle dell'utente A e per questo il sistema considera in termini di preferenze ed interessi i due utenti simili. Il passo finale è quello di selezionare le risorse di C non ancora considerate da A e proporle a quest'ultimo come dei suggerimenti. Nel nostro caso le risorse raccomandate ad A sono cd_5, libro_1 e mp3_3. E' bene tenere presente

come questi suggerimenti possono essere teoricamente molto distanti dalle preferenze dimostrate in termini di contenuto. Per esempio supponiamo che le preferenze di A, ovvero cd_1, cd_3 e cd_5, siano tutti album di musica country; non è detto che cd_6 sia altrettanto e potrebbe essere per esempio un disco di rock'n'roll o qualsiasi altra cosa che non ha importanza. Il fatto è che un utente con gusti simili ha fatto quella scelta e quindi ci sono buone probabilità che possa interessare anche ad A. Questa imprevedibilità dei risultati si manifesta anche sul tipo di risorse restituite; per esempio ad A viene ritornato anche un libro, nonostante A non ne abbia mai scelti prima.

Concludendo è importante far notare come tutto questo procedimento deve essere trasparente all'utente, il quale, semplicemente utilizzando il sistema, accetta di condividere la propria conoscenza ed i propri interressi con gli altri utenti.

# 3. Architettura del sistema

## 3.1 Introduzione

In questo capitolo vedremo di descrivere l'architettura che è stata progettata per realizzare il sistema appena descritto.
Parlando dell'ambientazione del sistema abbiamo gia visto che la sua architettura può essere sintetizzata dicendo che è determinata dall'unione di access-point o gruppi di essi; abbiamo anche osservato come questa si adatta ad essere replicata in diversi contesti.
In questo capitolo prenderemo inizialmente in considerazione l'utilizzo abbinato di un sistema multi-agente a tecniche di collaborative filtering come soluzione al problema dell'acquisizione, gestione e condivisione della conoscenza. Con l'introduzione ai sistemi multi-agente sarà poi possibile dare una definizione più formale del concetto di "gruppi di access-point" ai quali si faceva riferimento qualche riga sopra, localizzandone il dominio operativo e di ricerca a gruppi ristretti di "access-point" definiti piattaforme.
Infine verranno analizzate alcune problematiche specifiche dei device mobili, legate soprattutto alla mobilità degli utenti e vedremo qual è l'impatto delle soluzioni proposte in termini di scelte architetturali.

## 3.2 Un sistema multi-agente per acquisire, gestire e condividere la conoscenza

L'uso del sistema prevede che gli utenti interagiscano con esso attraverso i singoli access-point secondo un modello request-response, ossia inviando ad un access-point una richiesta e ricevendo da questo una risposta. Consideriamo come risposta i suggerimenti che gli altri utenti considerati simili per interessi, sulla base della loro conoscenza, ritornano; questo significa che i risultati globali vengono filtrati con tecniche di collaborative filtering.

Il procedimento attraverso il quale il sistema acquisisce, gestisce e condivide la conoscenza degli utenti deve essere del tutto trasparente a questi ultimi; la conoscenza viene assimilata analizzando le azioni precedentemente osservate. Come riuscire ad ottenere tutto questo?

Iniziamo considerando una caratteristica fondamentale del sistema che è quella dell'indipendenza e dell'autonomia nei confronti dell'utente nel raggiungimento dell'obiettivo principale. Questa indicazione infatti già ci indirizza verso un tipo di programmazione specifico che è quello della programmazione orientata agli agenti o AOP (Agent-Oriented Programming). In questo paradigma di programmazione gli oggetti vengono chiamati agenti e sono programmati per raggiungere, attraverso l'interazione con gli altri agenti, un particolare obiettivo individuale, sono cioè "goal-directed". La caratteristica fondamentale degli agenti è quella di essere entità autonome in grado di decidere quale azione eseguire senza l'intervento dell'utente, sulla base dell'osservazione dell'ambiente nel quale si trovano.

Altre proprietà che caratterizzano un agente e si adattano ai nostri scopi sono [RIF 3]:

- la capacità di comunicare: non solo con l'utente ma anche con altri agenti;
- l'abilità sociale: ossia la capacità non solo di comunicare con altri agenti ma anche di cooperare con essi nel perseguimento di obiettivi comuni attraverso uno scambio di informazioni e conoscenza;

- la reattività: significa essere in grado di rispondere agli stimoli esterni in modo da adattare la propria attività ai cambiamenti che avvengono nell'ambiente;
- le nozioni mentali: ossia la capacità di avere le proprie conoscenze, di ricordare esperienze e sulla base di queste apprendere, avere visioni proprie dell'ambiente e degli altri gruppi con i quali collabora in base alle sue abilità sociali;
- la persistenza: significa essere in grado di permanere nel tempo; la durata di vita dell'agente è superiore alla durata dei compiti di base che esso deve eseguire. Un agente continua ad esistere con un suo stato interno, in modo da poter eseguire interazioni successive;
- la proattività: un agente può in maniera autonoma generare eventi nell'ambiente circostante, iniziare delle interazioni con altri agenti e coordinarne le attività stimolandoli a produrre dei risultati;

Ovviamente il nostro sistema è composto da più utenti che debbono, attraverso gli agenti ad essi associati, collaborare, interagire, comunicare creando così un sistema detto multi-agente o MAS (Multi-Agent System). L'idea è quindi quella di associare ad ogni utente uno specifico agente (Personal Agent) che in modo autonomo ne osservi le azioni compiute, ne tracci sulla base di queste "un profilo" ed interagisca con gli altri agenti per condividere la conoscenza acquisita secondo qualche meccanismo di collaborative filtering (Figura 3).

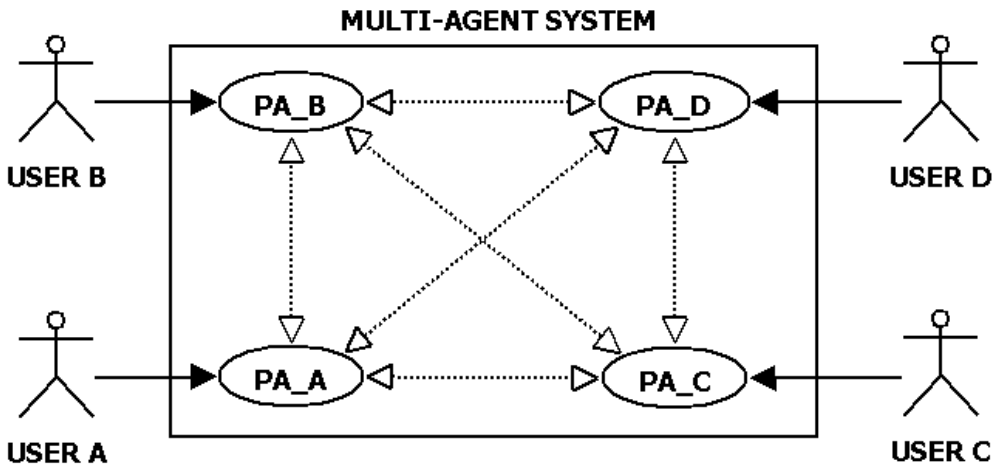


**Figura 3:** associazione tra gli utenti ed i Personal Agents di un sistema multi-agente

## 3.3 Il dominio operativo di un agente: la piattaforma

Ora che abbiamo trovato nel sistema multi-agente dotato di tecniche di collaborative filtering la soluzione per riuscire ad acquisire, gestire e condividere la conoscenza, vediamo come questa deve essere adattata all'architettura del sistema che abbiamo pensato.

E' gia stato introdotto precedentemente il concetto di "località" che il sistema deve avere per riuscire a sfruttare una proprietà specifica ed esclusiva dei device mobili che è quella di stare dove sta fisicamente l'utente, che permette di acquisire esperienza e quindi conoscenza contestualizzata al particolare luogo di interazione; in altri termini questo significa che si vuole puntare sulla condivisione della conoscenza acquisita da utenti che fisicamente frequentano lo stesso posto.

Questo si concretizza nella suddivisione degli access-point in base al dominio d'impiego; immaginando di avere nei diversi access-point gli agenti associati agli utenti che con esso hanno interagito, ai fini della mera attività di gestione collaborativa della conoscenza, solo gli access-point appartenenti allo stesso dominio d'utilizzo dovrebbero essere collegati tra loro e formare in seguito alle richieste di un utente un unico dominio di ricerca che d'ora in poi chiameremo per convenzione piattaforma. Una piattaforma quindi raccoglie tutti gli access-point che possono venire considerati identici per contesto d'utilizzo e nei quali c'è una corrispondenza strettamente univoca tra gli utenti e gli agenti. Un utente può accedere ad uno qualsiasi degli access-point di una piattaforma ed in ogni caso interagirà sempre con il medesimo agente a lui associato e le ricerche coinvolgeranno esclusivamente gli agenti presenti nei vari access-point di quella piattaforma. A piattaforme diverse invece corrispondono agenti diversi per il medesimo utente e questa soluzione permette di avere:

- piattaforme specializzate in relazione al contesto d'impiego;
- profili utente indipendenti e diversificati da una piattaforma all'altra;
- tecniche di collaborative filtering specifiche per ogni piattaforma;

Possiamo considerare quindi una piattaforma come il dominio operativo di un agente.

Rifacendoci all'ambientazione illustrata nel capitolo precedente, potremmo immaginare una suddivisione molto banale delle piattaforme che si limita a riunire sotto un'unica piattaforma tutti gli access-point di un edificio (Figura 4). Va fatto notare che questa non è l'unica soluzione possibile; per esempio anche all'interno di un edificio si potrebbero avere benissimo piattaforme diverse, specializzate in ambiti diversi.

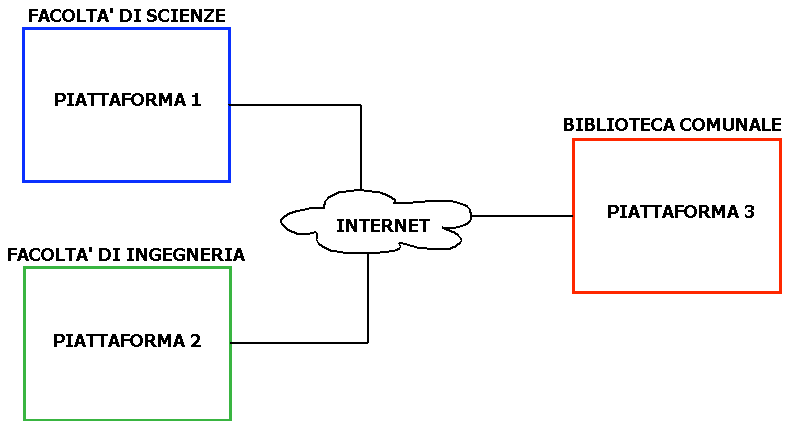


**Figura 4:** suddivisione degli access-point in piattaforme "indipendenti"

Una domanda che può sorgere spontanea osservando la figura è: "perché se le piattaforme rappresentano domini d'utilizzo indipendenti, vengono tra loro collegate attraverso una rete pubblica?". Risponderemo a questa risposta a breve.

## 3.4 Il problema dello spostamento dei device mobili

Finora abbiamo sempre parlato del Bluetooth come mezzo di comunicazione fra l'utente e la piattaforma, senza considerare le problematiche che questo comporta. Ora in fase di analisi architetturale è ora di farlo.

Già alcune limitazioni proprie dei device mobili erano state considerate in precedenza, come le limitate capacità di visualizzazione e memorizzazione; in un contesto molto generale come quello che stiamo descrivendo, le risorse ritornate come proposte dal

sistema potrebbero essere riferimenti a cose molte diverse: libri, risorse elettroniche su web, risorse elettroniche locali, etc. Non tutti questi riferimenti sono consultabili direttamente dal cellulare, pensiamo per esempio ad un file in formato pdf locale, o ad una risorsa on-line. La soluzione proposta era stata quella di considerare le risposte ritornate non direttamente acquisibili come incomplete e quindi di acquisirle successivamente, trasferendo il riferimento alla risorsa ad un computer collegato ad Internet come quello di casa o d'ufficio, per esempio sfruttando l'auto-configurazione di reti ad-hoc fornita dal Bluetooth (che permette di far comunicare in maniera automatica dei dispositivi nella stessa area di copertura dell'antenna). Esiste però un'altra problematica specifica che deve essere considerata e che è legata all'uso congiunto dei device mobili e della tecnologia Blueooth; il problema dello spostamento, determinato dal limitato range di copertura dell'antenna Bluetooth (10 – 100 metri) e dall'imprevedibilità dello spostamento degli utenti. Quest'ultimo problema è accentuato da una caratteristica dei dispositivi dotati di Bluetooth che abbiamo accennato pocanzi, ovvero la capacità di stabilire connessioni tra device in un modo del tutto trasparente all'utente. In questo modo un utente potrebbe anche non rendersi conto di essere nelle vicinanze di un access-point, ma comunque il device può autonomamente stabilire una connessione e comunicare con esso. Possiamo arrivare a pensare allora ad un utilizzo di questo tipo: l'utente mentre è sull'autobus per andare a lezione o al lavoro o a farsi un giro per la città memorizza le ricerche a cui è interessato, indipendentemente dalla vicinanza ad un access-point. Le ricerche memorizzate fungono poi da semplice promemoria per il device. A questo punto l'attvità dell'utente è terminata; l'inoltro effettivo delle richieste alle piattaforme incontrate durante il tragitto ed il recupero dei risultati è affidato esclusivamente al device. L'utente torna a casa la sera e controlla quali risultati durante la giornata è riuscito a recuperare sulle diverse piattaforme con le quali ha interagito il device. Ovviamente esistono anche interazioni più partecipi da parte dell'utente, che per esempio potrebbe inoltrare manualmente una richiesta ad un'access-point mentre è in pausa nella zona caffè all'access-point li vicino e aspettare il ritorno dei risultati.

Rimaniamo però al caso "limite" visto prima. Il problema di tutta questa automazione è che l'utente, essendo "all'oscuro di tutto", potrebbe non accorgersi che è stata inoltrata una richiesta e perciò uscire dalla raggio di copertura della comunicazione Bluetooth prima che i risultati gli siano ritornati, o semplicemente potrebbe avere fretta e non avere tempo di aspettarli. In questi casi ed in casi anologhi si vengono a creare dei

risultati mancanti, in seguito a delle richieste, per mancata consegna degli stessi, non per mancanza effettiva di risultati. Parleremo in questo caso di risultati pendenti (Figura 5).

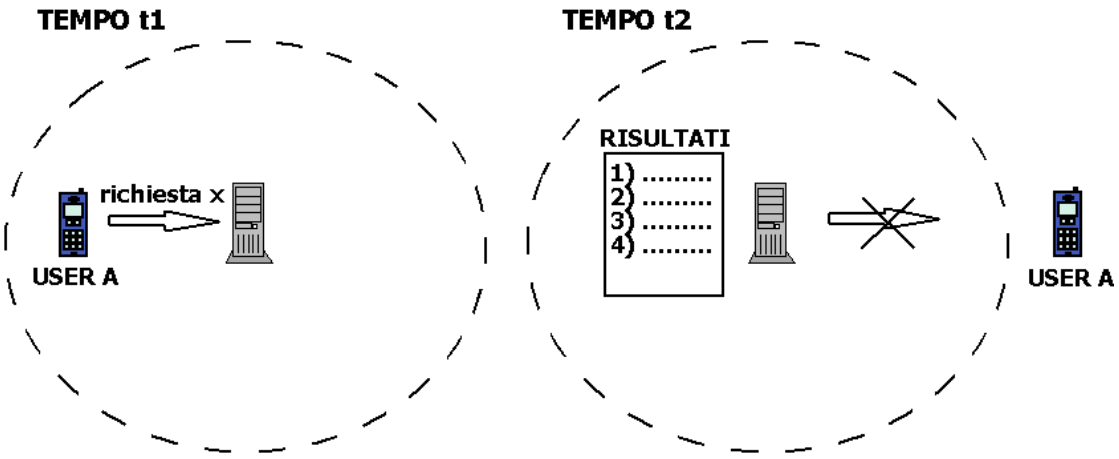


**Figura 5:** formazione di risultati pendenti

### 3.4.1 Il recupero dei risultati pendenti

Vediamo ora di proporre una soluzione al problema dei risultati pendenti.
Un utente può interagire nel corso di una giornata con diverse piattaforme e su ognuna di queste può generare dei risultati pendenti. Noi vogliamo fare in modo che l'utente riesca a recuperarli e per far questo è indispensabile che il device mobile tenga traccia di tutte le piattaforme con le quali interagisce e presso le quali ha delle ricerche in sospeso, in modo da poterle ricontattare successivamente.
Alla semplice ricerca all'interno della piattaforma con la quale sta interagendo, aggiungiamo una nuova funzionalità all'utente che è quella del recupero dei risultati pendenti su altre piattaforme. Questa scelta obbliga come soluzione architetturale il collegamento fisico tra le varie piattaforme attraverso una rete, formando cosi un'architettura distribuita (Figura 6) come quella dello scenario illustrato nel capitolo precedente.

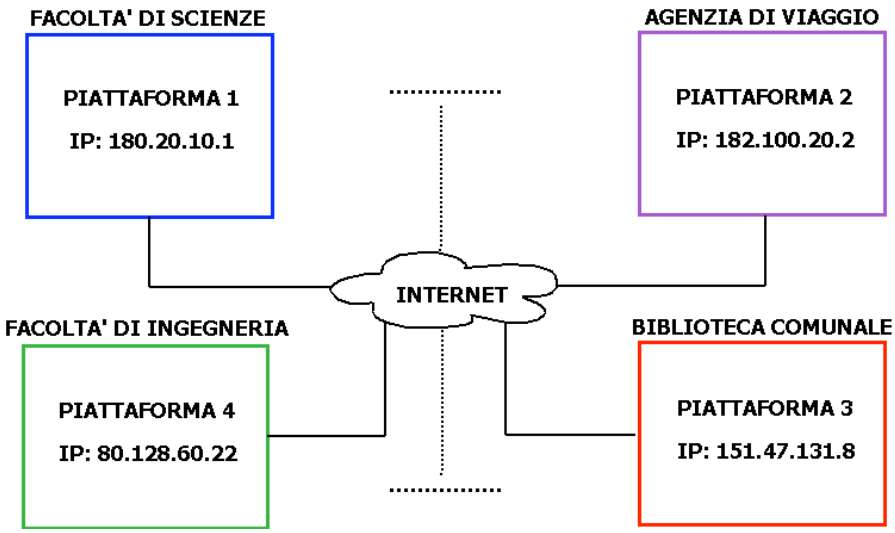


**Figura 6:** architettura distribuita per il recupero dei risultati pendenti

Ci sono due modalità, che possono coesistere, per il recupero dei risultati sospesi nelle piattaforme:

- attraverso altre piattaforme e quindi utilizzando ancora il device mobile come destinazione finale dei risultati;
- utilizzando l'accesso ad internet ed interagendo senza l'intermediazione di altre piattaforme in modo da ricevere direttamente i risultati sul proprio computer.

Prima di vedere in modo più approfondito questi due approcci, vediamo di descrivere il modo con il quale il device tiene traccia delle piattaforme con risultati pendenti, che è indipendente da quale soluzione di recupero viene successivamente utilizzata.
Sappiamo che un computer è rintracciabile in una rete attraverso il suo indirizzo IP. Abbinando all'IP un numero di porta si riesce ad accedere ad un determinato servizio su quell'host. Possiamo quindi implementare su ogni macchina che fa da gateway tra una piattaforma ed internet, un'applicazione "demone" in ascolto su una porta, con lo specifico compito di accettare connessioni di richiesta per risultati pendenti. Ovviamente non si deve limitare solamente ad accettare connessioni ma anche a processarle, ovvero recuperare i risultati pendenti relativi all'utente per il quale è stata ricevuta la richiesta e restituirli. Le informazioni indispensabili e sufficienti per rintracciare le risposte collezionate dagli utenti sono quindi:

- l'indirizzo IP pubblico della piattaforma;

- il numero di porta sul quale è in ascolto l'applicazione per la gestione delle connessioni;
- un numero identificativo, univoco a livello di indirizzo IP, della piattaforma; serve nei casi in cui ad un unico indirizzo IP corrispondono più piattaforme.

Queste tre informazioni sono quindi quelle che ogni device mobile deve memorizzare quando, in seguito ad una ricerca, non riceve risposte in seguito a risultati pendenti e possiamo immaginarle memorizzate in una semplice forma tabellare come segue:

| INDIRIZZO IP | PORTA | ID PIATTAFORMA |
|---|---|---|
| 180.10.20.1 | 5022 | 1 |
| 80.128.60.22 | 5017 | 2 |
| 182.100.20.2 | 5100 | 1 |

**Tabella 2**: informazioni memorizzate dal device mobile per tenere traccia delle piattaforme con risultati pendenti

Per spiegare il modo con il quale queste informazioni vengono acquisite, aiutiamoci con la Figura 7 . Innanzitutto il device non appena stabilisce una connessione con una piattaforma, invia la keyword di ricerca. La piattaforma dopo aver autenticato il device, risponde inviando le informazioni necessarie alla sua localizzazione via rete in una stringa del seguente formato:

```
indirizzoIP:porta,idPiattaforma
```

Il device riceve queste informazioni e le memorizza in una tabella come quella vista pocanzi. A questo punto ci sono due possibilità: i risultati della ricerca arrivano al device, oppure rimangono pendenti. Nel primo caso, una volta che i risultati arrivano a destinazione, il device elimina dalla tabella delle ricerche pendenti il riferimento alla piattaforma con la quale ha appena comunicato, mentre se non arrivano il device non fa nulla e quindi la piattaforma corrente rimane "tracciata" per successive riconnessioni.

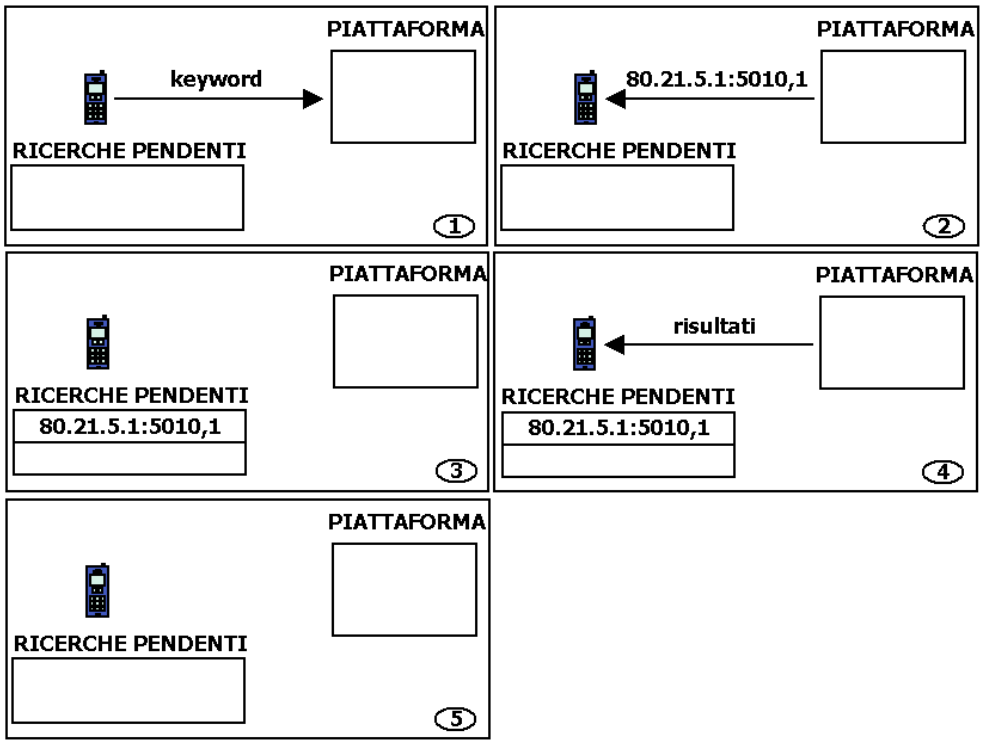


**Figura 7:** procedimento di "tracking" delle piattaforme con risultati pendenti

#### 3.4.1.1 Recupero da piattaforme diverse

Un primo modo di recuperare eventuali risultati pendenti è quello di farlo direttamente dal device mobile, per mezzo di altre piattaforme. Supponiamo che l'utente abbia inoltrato una richiesta su una piattaforma e da questa non abbia ricevuto risultati perché l'utente stava camminando ed è uscito dal range di copertura dell'antenna Bluetooth prima della loro consegna. Qualche ora dopo incontra un'altra piattaforma, l'utente non ha fretta e quindi invia manualmente una richiesta; all'atto dell'invio, il device notifica all'utente (dopo aver constatato che la tabella con i riferimenti a piattaforme da ricontattare non è vuota) che ci sono dei risultati pendenti da un'altra piattaforma e propone come funzionalità aggiuntiva il loro recupero. Se l'utente accetta, il device invia alla piattaforma con la quale ha instaurato la connessione Bluetooth, le informazioni memorizzate nella tabella, attraverso le quali la piattaforma riesce a contattare quella precedentemente visitata, richiedere e ricevere i risultati

pendenti per lo specifico utente ed infine inviarli al device mobile. (Figura 8). Se invece l'utente non vuole recuperare i risultati, le informazioni rimangono memorizzate nella tabella e la richiesta di recupero verrà proposta successivamente.

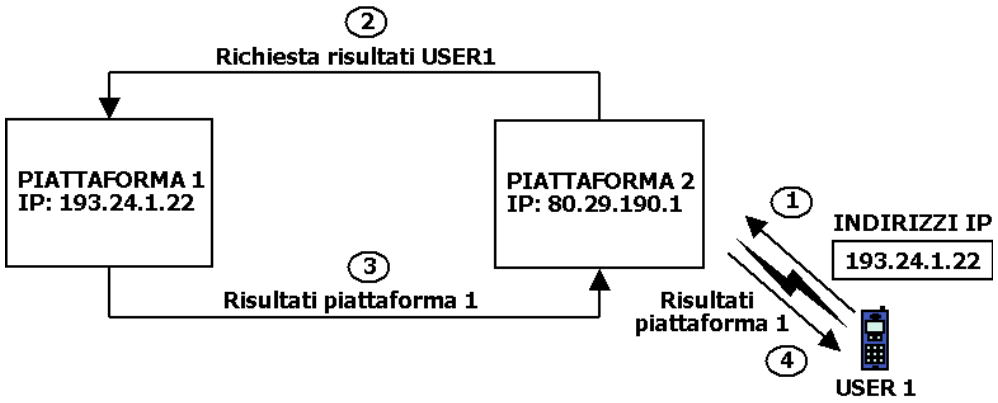


**Figura 8:** recupero dei risultati pendenti da altre piattaforme. Per semplicità viene considerato solamente l'indirizzo IP. Il numero di porta e l'ID della piattaforma sono tralasciati

La Figura 9 mostra in dettaglio il protocollo utilizzato per questo tipo di recupero dati, supponendo che USER1 accetti di recuperare i risultati pendenti nella piattaforma P2 attraverso la piattaforma P1.

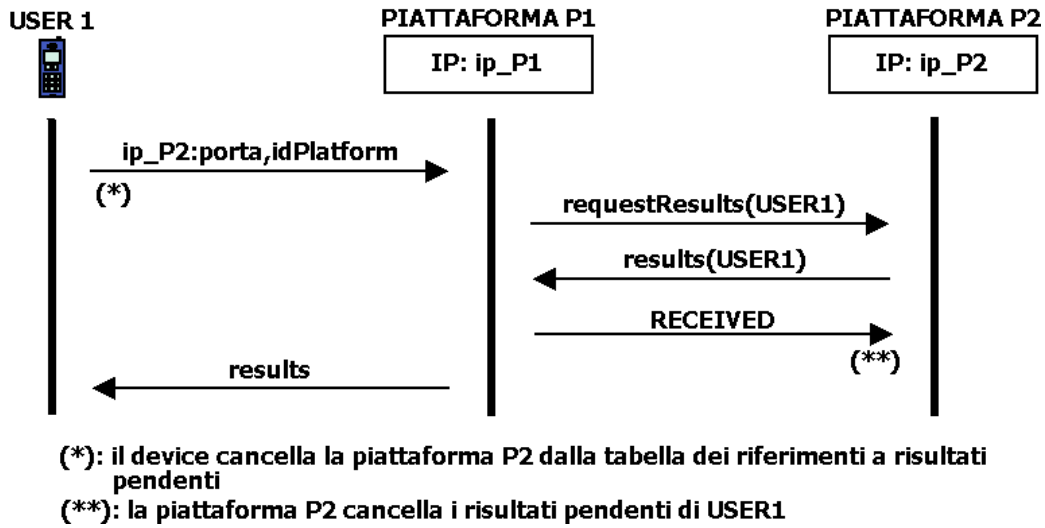


**Figura 9:** protocollo per il recupero dei dati da piattaforme diverse

Per qualche motivo, può succedere che la piattaforma P1 una volta recuperati i dati da P2 non riesca più a comunicare con il device. Ovviamente quando P2 riceve il messaggio di RECEIVED, cancella i risultati pendenti appena consegnati a P1 per evitare inutili ridondanze. Per questi motivi, è necessario che il device cancelli dalla sua tabella, il riferimento a P2 non appena invia le sue informazioni e non al momento della ricezione dei risultati, perché come detto potrebbe non avvenire. In questo modo se i risultati non dovessero arrivare al device questo comunque manterrebbe nella sua tabella il riferimento a P1, dal quale successivamente può ottenere i risultati originariamente prodotti da P2 ed in esso trasferiti.

#### 3.4.1.2 Recupero da PC via Internet

L'altro modo con il quale l'utente può recuperare i risultati collezionati e non ancora ricevuti, è attraverso un computer collegato ad internet, senza quindi l'intermediazione di altre piattaforme. Se un utente non dispone di un collegamento ad internet può comunque sempre utilizzare l'altro metodo di recupero.

Il procedimento è sostanzialmente identico a quello precedente, cambiano solamente gli attori; mentre prima l'intero lavoro di recupero era svolto da un'altra piattaforma ed il destinatario finale dei risultati era il device mobile, ora i due ruoli convergono in un unico attore che è il computer di casa per esempio (Figura 10). Ovviamente affinché il computer possa ricontattare le piattaforme visitate deve disporre come al solito della tabella memorizzata dal device e come già sappiamo questa può essere trasferita in modo automatico senza alcun intervento da parte dell'utente, semplicemente avvicinando il device mobile al computer, via Bluetooth.

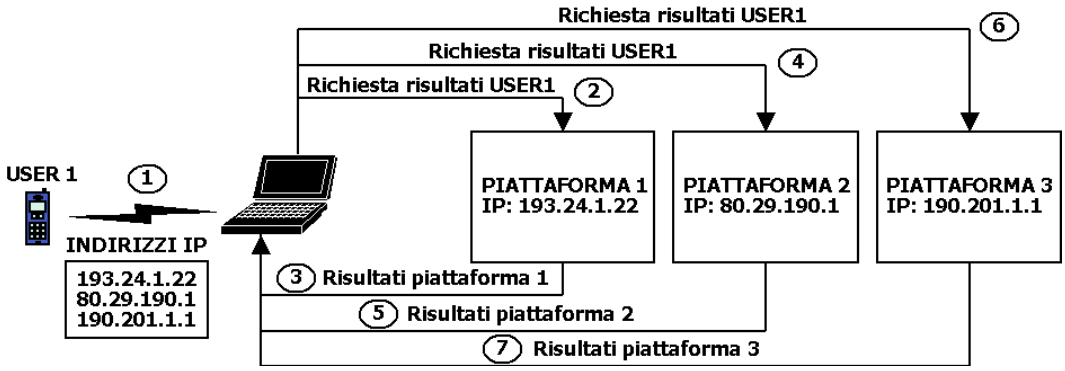


**Figura 10:** recupero dei risultati pendenti da Pc via Internet. Per semplicità viene considerato solamente l'indirizzo IP. Il numero di porta e l'ID della piattaforma sono tralasciati

Anche per il protocollo di interazione tra il computer e le piattaforme ci si può attenere a quello visto nella modalità precedente.

# 4. Un esempio pratico: il prototipo BTImplicitBoard

L'ultimo step di questo lavoro prevedeva la realizzazione di un prototipo dimostrativo finalizzato all'implementazione di una parte del sistema descritto nell'architettura. Lo sforzo implementativo si è concentrato sull'interazione dei device mobili via Bluetooth con una piattaforma o meglio con un singolo access-point. Un access-point dal punto di vista software può venire considerato come un sistema multi-agente fornito di qualche sistema per la customizzazione collaborativa dei risultati; deve inoltre essere dotato di un modulo Bluetooth in grado di accettare e processare le connessioni che riceve dai device mobili e fungere da intermediario tra il device mobile e lo specifico Personal Agent ad esso associato (Figura 11).

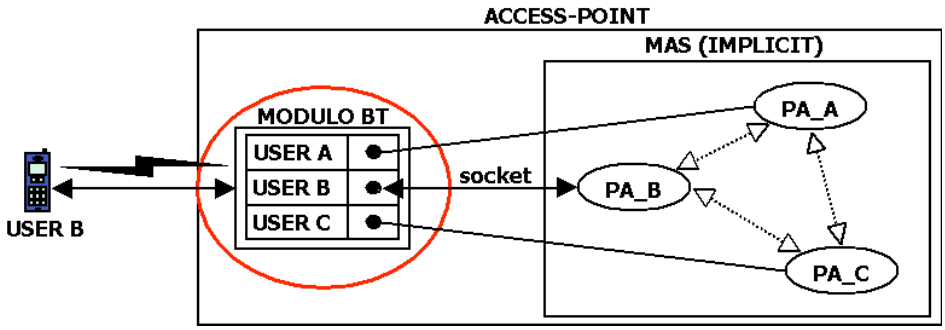


**Figura 11:** architettura software di BTImplicitBoard

Il focus di questo prototipo è stato quindi sommariamente quello dello sviluppo del modulo Bluetooth, come interfaccia tra le connessioni dei device e quelle con i rispettivi Personal Agent, e lo sviluppo dell'applicazione per la connessione e l'interazione con l'access-point da parte dei device mobili. Per questo motivo si è deciso di utilizzare una piattaforma multi-agente già sviluppata ed implementata in diversi contesti, Implicit.

Implicit è un sistema multi-agente sviluppato in Jade[1] il cui obiettivo è quello di risolvere il problema del trasferimento di conoscenza "implicita" attraverso la condivisione di esperienze acquisite nell'uso di pezzi d'informazione [ 3 ]. Ogni Personal Agent è dotato di un SICS, ovvero di un sistema per il supporto alla Cultura Implicita (System for Implicit Culture Support) il cui scopo è quello di far accettare all'utente le informazioni che li vengono suggerite dal sistema [ 3 ] attraverso la collaborazione e la condivisione dell'esperienza acquisita dagli altri utenti. Rimandiamo all'Appendice A per maggiori informazioni sulla Cultura Implicita.
L'architettura propria del prototipo può essere considerata di tipo client-server. Il software sviluppato quindi è composto da un'applicazione client che deve essere eseguita sui device mobili ed una server che deve essere eseguita dagli access-point. Naturalmente sviluppi successivi di questo prototipo saranno mirati ad un uso di più access-point coordinati in un'unica piattaforma secondo un'architettura più distribuita ed a tal scopo il sistema multi-agente utilizzato essendo sviluppato in Jade ne facilita l'implementazione. Jade è infatti orientato verso architetture peer to peer e quindi la collaborazione tra gli agenti in un unico access-point (di per sé già peer to peer) può essere facilmente ampliata a più access-point (utilizzando la terminologia Jade potremmo definirli Container) di una medesima piattaforma.

## 4.1 Il funzionamento

Il prototipo sviluppato prevede un utilizzo di questo tipo: un utente ha la possibilità attraverso il suo device mobile di comunicare ed interagire con un access-point. La comunicazione tra il device e l'access-point avviene attraverso la tecnologia Bluetooth.
Possiamo immaginare, in un contesto universitario, l'access-point come una bacheca elettronica nella quale sono memorizzate una serie di informazioni di vario tipo: links a siti web o a forum di discussione, riferimenti a libri, tutorial, convegni, risorse in formato elettronico locali. L'interazione tra le parti ha come fine quello di ottenere in seguito a

[1] Jade è un middleware sviluppato interamente in Java dalla TILab (Telocom Italia Lab) per lo sviluppo di applicazioni multi-agente distribuite basato su un'architettura di comunicazione peer to peer [ 2 ] o meglio agent to agent, visto che i peers in Jade vengono chiamati agents.

delle specifiche richieste su un argomento, dei risultati che si riferiscono a delle risorse riguardanti l'argomento in oggetto. Le risorse ritornate, possono sì essere considerate globali, ma nel contesto specifico assumono una rilevanza particolare; per esempio le risorse che riguardano una ricerca per la keyword "java", possono contenere riferimenti a risorse globali come un sito web, ma in ogni caso la cosa importante è che il suggerimento proviene da persone che le hanno precedentemente utilizzate e gradite e che hanno direttamente interagito con il sistema in quello specifico posto; questo significa che molto probabilmente gli utenti "suggeritori" sono altri studenti della stessa facoltà e che magari hanno già seguito il corso di programmazione ad oggetti che l'utente si appresta a seguire ed in riferimento al quale ha eseguito la ricerca per java. In questo modo i risultati dei colleghi possono essere più specifici ed indirizzare l'utente verso delle tematiche rilevanti e caratterizzanti per quel particolare corso di studio.
Il sistema quindi non si limita a restituire solamente tutte le risorse presenti nel sistema in modo "indiscriminato", ma i risultati vengono filtrati secondo le funzionalità di customizzazione collaborativa messe a disposizione dal SICS, tenendo quindi conto delle preferenze passate fatte dagli altri utenti, secondo la logica già menzionata della condivisione della conoscenza. All'atto pratico questo significa che l'utente riceverà come risposta da parte di Implicit attraverso l'intermediazione del modulo Bluetooth tre tipi di risultato:

- risultati oggettivi: vengono estratte le ultime n risorse inserite;
- risultati personali: risorse verso le quali l'utente ha già espresso una preferenza in passato. Possiamo immaginare questi risultati come una sorta di bookmark;
- risultati suggeriti da altri utenti considerati simili in quanto ad interessi.

Ovviamente i risultati che ci interessano maggiormente ai fini dell'applicazione vera e propria sono solamente quelli derivanti dall'azione collaborativa degli utenti. E' utile in ogni caso riportare anche i risultati non customizzati in alcun modo per garantire una certa indipendenza e visione globale delle risorse disponibili. In questo modo inoltre viene data la possibilità all'utente di scegliere delle risorse appena inserite nel sistema e quindi non ancora valutate da nessuno e per questo non soggette ad alcun tipo possibile di raccomandazione (abbiamo già visto che questo è un limite proprio di tutti i sistemi di collaborative-filtering). I risultati personali invece non presentano vantaggi rilevanti nella loro acquisizione da parte dell'utente poiché, con l'implementazione attuale del SICS, rappresentano solamente delle risorse precedentemente apprezzate.

Vengono comunque ritornati all'utente anche questi, semplicemente per mettere in evidenza possibili utilizzi futuri più significativi relativi alla customizzazione personale, che si possono ottenere dall'uso congiunto al SICS di altre tecniche di filtering più "single Personal Agent directed".

Possiamo quindi riassumere l' intero funzionamento dal punto di vista implementativo del prototipo in questi passaggi:

- connessione con l'access-point;
- protocollo di comunicazione tra il device mobile ed il modulo Bluetooth (richiesta nuova ricerca, autenticazione, invio dei risultati): ExternalCommunicationProtocol;
- protocollo di comunicazione tra il modulo Bluetooth ed il Personal Agent associato all'utente con il quale è in corso la connessione Bluetooth (inoltro della richiesta ed invio dei risultati): InternalCommunicationProtocol;
- interazione tra i Personal Agent della piattaforma per la produzione dei risultati basati sulla condivisione collaborativa della conoscenza.

Analizzeremo questi steps uno ad uno e vedremo in che modo sono stati implementati nel prototipo nei paragrafi seguenti.

## 4.2 Connessione con l'access-point

La prima cosa che i device mobili e l'access-point devono fare per poter iniziare la comunicazione è stabilire una connessione. Abbiamo già ripetuto più volte che il Bluetooth permette di creare e riconfigurare delle reti ad-hoc (successivamente piconets) in modo automatico; questo perché i device dotati di questa tecnologia hanno la possibilità di trovare gli altri device presenti nella stessa area di copertura dell'antenna e stabilire con essi delle connessioni senza l'intervento dell'utente. Esiste anche un'altra funzionalità offerta da questa tecnologia che permette di recuperare, non appena i device si riconoscono, le funzionalità offerte da ciascuno di essi secondo un concetto ampiamente diffuso nei contesti di rete che è quello dei services. Il tutto avviene secondo una logica di tipo master-slave in cui chi inizia per primo la comunicazione funge da master. Nel nostro caso gli access-point assumono una

funzionalità server-like, ovvero rimangono in attesa di connessioni da parte dei device mobili mettendo a loro disposizione il service per accedere ed utilizzare il sistema. In modo analogo al ruolo dei client, i device mobili invece cercano eventuali altri dispositivi ed una volta trovati richiedono i servizi offerti. Se tra questi c'è il service che permette la comunicazione con l'access-point, il device mobile può iniziare ad utilizzarlo secondo un protocollo di comunicazione stabilito. E' compito quindi dei device mobili prendere l'iniziativa e ritrovare gli access-point incontrati durante i vari spostamenti.

Tutta questa procedura (ritrovamento dei device, richiesta dei services e utilizzo degli stessi) può essere automatizzata.

### 4.2.1 Auto-configurazione di connessioni Bluetooth

Le reti Bluetooth differiscono molto dalle tradizionali reti cablate e aggiungono delle complessità che richiedono delle nuove operazioni per poter essere risolte.

Alla staticità delle reti cablate infatti si contrappone l'alta dinamicità delle piconets, nelle quali i dispositivi possono apparire e sparire in qualsiasi momento. Nelle prime inoltre esistono delle procedure di configurazione che devono essere fatte dall'utente, come per esempio la connessione fisica tramite cavo dei dispositivi e l'inserimento manuale degli indirizzi di rete, che nel caso delle piconets devono essere mascherate all'utente ed automatizzate.

Questa automazione delle connessioni tra device può essere del tutto nascosta all'utente finale e si realizza sequenzialmente attraverso i seguenti steps (Figura 12):

- ricerca dei device presenti nel range del dispositivo (inuqiry);
- connessione con questi dispositivi (paging);
- ricerca dei services disponibili presso i device in precedenza trovati.


**Figura 12:** procedimento di connessione tra device Bluetooth

#### 4.2.1.1 Inquiry

La prima cosa da fare è l'inquiry. Con questa procedura i dispositivi Bluetooth possono trovare altri dispositivi Bluetooth nel range di copertura della loro antenna. In questa fase il device che ha iniziato la ricerca memorizza le informazioni ricevute come risposta che contengono le informazioni necessarie (indirizzo e clock) per una successiva connessione con le unità che hanno risposto, attraverso una procedura che prende il nome di paging e che verrà illustrata nella sezione successiva.

Il messaggio di inquiry che viene inviato in modalità broadcast dalla sorgente può specificare anche la classe dei dispositivi che devono rispondere. Esiste un Generic Inquiry Access Code (GIAC) per "interrogare" qualsiasi tipo di dispositivo ed una serie di Dedicated Inquiry Access Code (DIAC) che "interrogano" solo alcuni tipi di device [ 5 ].

Durante il processo di inquiry l'unità che inizia la ricerca deve essere in modalità inquiry mentre chi risponde deve essere in modalità inquiry scan [ 4 ]; in questo modo un device può essere impostato in modo tale da non essere riconosciuto dagli altri device durante questa procedura, semplicemente non entrando nella modalità inquiry scan.

Il device in modalità inquiry comincia trasmettendo rapidamente su una sequenza di frequenze diverse dei pacchetti di dimensione molto ridotta (inquiry packets). La frequenza utilizzata cambia per la precisione 3200 volte al secondo, il doppio rispetto a quanto avviene durante una normale connessione, in maniera tale da massimizzare il numero di frequenze che vengono coperte. Viceversa i dispositivi che sono in uno stato di inquiry scan cambiano frequenza molto lentamente, una volta ogni 1.28 secondi [ 4 ]. Questo diverso comportamento è indispensabile perché in questa fase non è ancora avvenuta nessuna sincronizzazione tra i dispositivi relativa alla sequenza di frequency hopping[1] e quindi questa tecnica permette di agevolare sensibilmente "l'incontro" dei device sulla stessa frequenza, visto che non è realizzabile l'idea di utilizzare una frequenza fissa per problemi di interferenze [ 4 ].

---

[1] Il Frequency Hopping (salto di frequenza) è una tecnica utilizzata dall'unità master per gestire la comunicazione e la sincronizzazione con le unità slaves, attraverso la condivisione di una sequenza di frequency hopping unica per l'intera piconets.

Infine quando un device in modalità inquiry scan riceve un inquiry packets, attende un breve tempo random ed infine risponde con un pacchetto FHS (Frequency Hopping Synchronisation) che contiene tutte le informazioni rilevanti per permettere al device che ha spedito l'inquiry di stabilire connessioni con esso [ 4 ].

#### 4.2.1.2 Paging

Con le informazioni recuperate durante la fase di inquiry è possibile stabilire delle connessioni Bluetooth con i device trovati. Il concetto di connessione cambia a seconda del livello dello stack Bluetooth al quale si fa riferimento. Per i livelli più bassi questo significa che tutti i device nella piconet sono sincronizzati per quello che riguarda la sequenza di frequency hopping ed il clock dell'unità master, mentre per i livelli più alti vuol dire poter stabilire dei collegamenti (ACL links[1]) sui quali possono essere trasportati dei dati che verranno poi utilizzati e gestiti dai livelli superiori dello stack. Facendo riferimento a quest'ultima visione di connessioni, il protocollo che viene utilizzato per realizzare questi links prende il nome di paging..

In modo del tutto analogo a quanto avviene per l'inquiry, anche nel paging è previsto che nel caso in cui due device vogliono stabilire una connessione, chi inizia deve essere in uno stato di page (master) e l'altro in page scan (slave) [ 4 ].

Durante questa procedura le due unità si scambiano una serie di parametri in modo tale che al termine di essa entrambe condividano le informazioni indispensabili per mantenere e gestire la connessione.

#### 4.2.1.3 Ricerca services

Il termine service discovery è usato per indicare il modo con il quale un dispositivo di rete ricerca i servizi disponibili su quella rete [ 4 ].

---

[1] ACL (Asynchronous Connectionless Link): sono links destinati al trasferimento dati; forniscono un tipo di connessioni a commutazione di pacchetto tra il master e tutti i device slave attivi presenti nella piconet.

Nel caso di reti ad-hoc come quelle generate da dispositivi Bluetooth questa procedura, oltre che ricoprire un ruolo essenziale, si arricchisce di problematiche dovute alla natura della rete stessa perché per esempio non ci possono essere infrastrutture centralizzate disponibili che fungono da directory service e perché il set di services disponibili cambia dinamicamente in base alla distanza dei device che possono essere anche in movimento.

Il Bluetooth SIG[1] vista l'importanza che ricopre questa procedura ha deciso di sviluppare un protocollo apposito che prende il nome di Service Discovery Protocol. Questo protocollo definisce le regole rispetto alle quali si devono attenere chi ricerca nuovi servizi (client) e chi fornisce tali informazioni (server) con l'obiettivo di minimizzare le operazioni di configurazione richieste per massimizzare la flessibilità del sistema.

Il Service Discovery Protocol permette a qualsiasi dispositivo Bluetooth di operare sia come server che come client; come server quando viene interrogato da altri dispositivi e risponde con i propri services, da client quando interroga a sua volta gli altri device della piconet.

Per fornire le informazioni ai clients SDP, il server SDP mantiene una lista di service records che descrivono le caratteristiche dei services ad esso associati. Un service può essere considerato un'entità in grado di fornire informazioni, eseguire azioni o gestire delle risorse e la cui implementazione può essere software, hardware, o mista.

Tutte le informazioni che riguardano un service sono mantenute presso l'SDP server in un unico service record che è costituito da una lista di service attributes (id, valore) che descrivono delle singole caratteristiche del service.

Per utilizzare questo protocollo è necessario che in precedenza sia già stata settata una connessione di base con un qualche device e per questo motivo è evidente che questa operazione segue sequenzialmente la procedura di inquiry e di paging. E' proprio sulla connessione che viene creata dal paging infatti che le richieste e le risposte dettate dal SDP vengono scambiate. Poichè la ricerca dei services si può estendere a più device, per questioni di risparmio di risorse (memoria, processore, batteria), una volta che le informazioni da uno specifico device sono state ottenute, la

[1] Il Bluetooth SIG (Bluetooth Special Interest Group) è uno speciale gruppo di lavoro formato da diverse società leader nel settore delle telecomunicazioni e del wireless che ha prodotto lo sviluppo di uno standard Bluetooth basato su specifiche aperte e senza royalty

connessione viene rilasciata e nel caso in cui successivamente si voglia utilizzare un service su qualche dispositivo è necessario ristabilire una nuova connessione.

### 4.2.2 Implementazione in BTImplicitBoard

Alla luce di quanto abbiamo appena detto, vediamo come sono state adattate ed implementate le procedure di ricerca di altri device e di service discovery nel nostro prototipo. Per quanto riguarda la parte che funge da server, ossia l'access-point, ricordiamo che è stato implementato un modulo apposito per la gestione delle connessioni e della comunicazione Bluetooth e che fornisce una duplice interfaccia per le comunicazioni tra il device mobile ed il Personal Agent ad esso associato nella piattaforma multi-agente: BTConnection.java. Faremo riferimento a questa classe quando ci riferiremo al modulo Bluetooth. La classe che implementa l'applicazione da eseguire sui device mobili invece si chiama BTBoardConnection.java.
Per sviluppare questa parte di codice e quella successiva relativa alla comunicazione Bluetooth sono state utilizzate un set di API standardizzate Java che sono indipendenti dalla piattaforma utilizzata. Queste librerie prendono il nome di JSR-82 (Java Specification Request 82) o JABWT (Java Api for Bluetooth Wireless Technology); esse definiscono le basi per lo sviluppo di applicazioni Bluetooth [ 6 ] e sono state pensate e progettate per poter essere utilizzate anche da sistemi di tipo CLDC (Connected Limited Device Configuration) ovvero da device con limitate capacità di calcolo, di memoria e batteria. Questo permette il loro utilizzo anche sulla parte client della nostra applicazione ossia sui cellulari.
Rimandiamo all'appendice B per maggiori informazioni sullo sviluppo di software Bluetooth e l'uso dI JSR-82.

Ritornando al prototipo l'idea è quella di fare in modo che il modulo Bluetooth crei un particolare service (BTImplicitBoard Service) la cui funzionalità è quella di processare le connessioni ricevute dai device mobili secondo un protocollo di comunicazione ben definito che verrà illustrato successivamente. Brevemente questo significa autenticare il device, ricevere la keyword di ricerca, inoltrare la richiesta al Personal Agent corretto e restituire i risultati da questo ritornati e prodotti da Implicit.

Descriviamo ora in dettaglio quali sono le operazioni che il modulo Bluetooth e l'applicazione eseguita sui device mobili eseguono per riuscire ad instaurare una connessione.

Il modulo Bluetooth come detto crea innanzitutto un service record per "BTImplicitBoard Service" e lo identifica univocamente attribuendoli un identificatore UUID (Universally Unique IDentifier) a 128 bit. Grazie a questo identificativo i device mobili riusciranno a cercare questo service nei device con i quali entreranno in comunicazione. Al service record creato vengono poi aggiunti due attributi che specificano il nome del service e l'autore dello stesso.

Una funzionalità importante che il modulo Bluetooth deve fornire è quella di gestire più connessioni simultanee e per permettere questo è stata creata un'apposita classe (ConnProcessor). Prima di attivare il service viene quindi creata un'istanza di questa classe. A questo punto il service record viene inserito nel SDDB[1] locale e da quel momento il service è visibile agli altri device ed il modulo Bluetooth è in grado di accettare connessioni ed eseguirne le funzionalità offerte.

Ogni volta che si riceve una connessione, questa viene inserita in un apposito vettore (`queue`) e viene processata secondo il protocollo stabilito o rimane in attesa di esserlo se il Thread che gestisce questa coda è già occupato con altre connessioni. Il funzionamento del modulo Bluetooth è schematizzato in Figura 13.

Il device mobile dal canto suo esegue inizialmente le tipiche operazioni di ricerca dei device presenti nel proprio range di copertura dell'antenna e successivamente, su ognuno dei device trovati (memorizzati nel vettore `devices`), ricerca il service "BTImplicitBoard Service" specificando nei parametri di ricerca lo UUID associato dal server al service e gli ID degli attributi che gli sono stati aggiunti. Non appena qualche ricerca ritorna un service record, le altre ricerche vengono cancellate e le informazioni contenute nel service record vengono utilizzate per stabilire una connessione con l'access-point. Quando la connessione è stabilita inizia il protocollo di comunicazione previsto dal service. Il funzionamento dell'applicazione del device mobile è riepilogato in Figura 14.

---

[1] Il SDDB (Service Discovery DataBase) rappresenta un database "astratto" di service records nel quale un device memorizza i propri service records relativi ai services che rende disponibili.

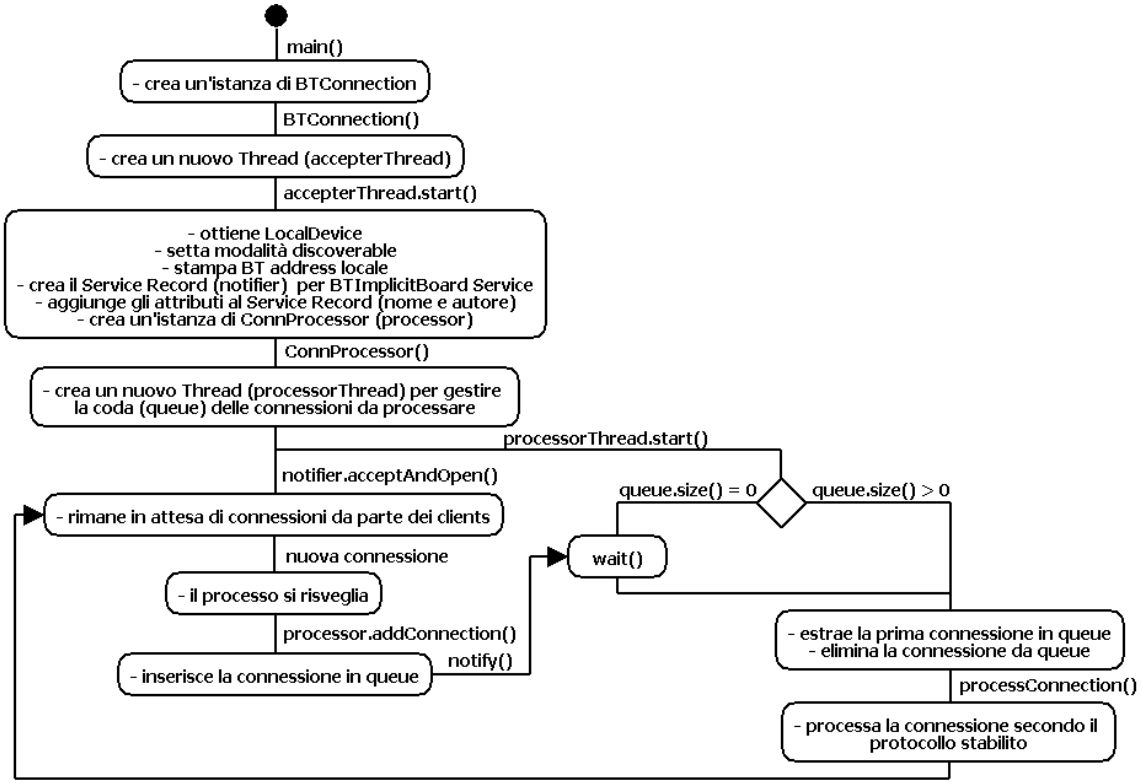


**Figura 13:** inizializzazione e gestione delle connessioni del modulo Bluetooth

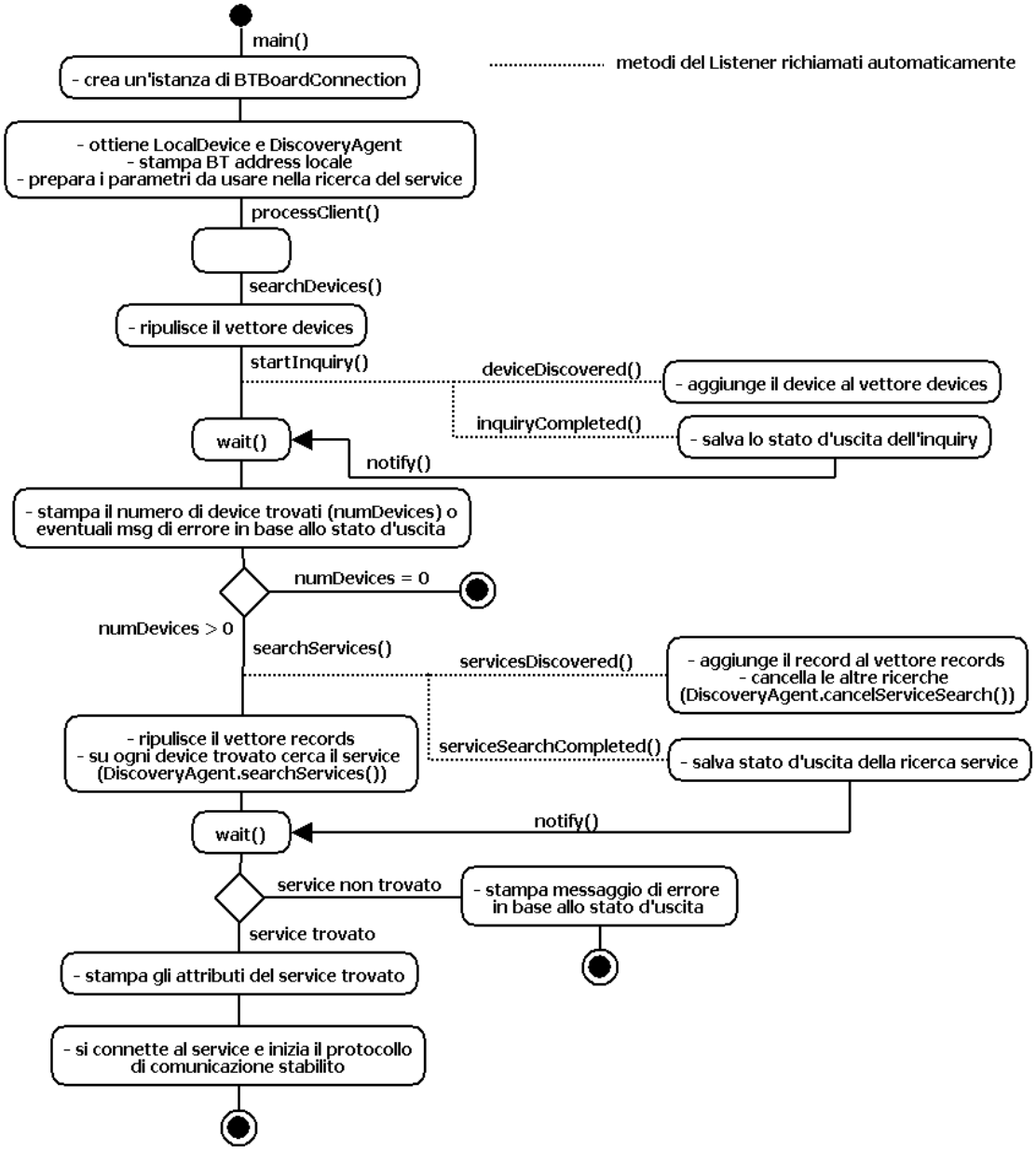


**Figura 14:** procedimento di connessione all'access-point del device mobile

## 4.3 Comunicazione tra i device mobili e l'access-point

Fino ad ora abbiamo spiegato come i device mobili riescono ad identificare gli access-point e stabilire una connessione con essi. Se riescono a fare questo, il passo successivo è quello di iniziare ad interagire con l'access-point, inviando delle richieste di ricerca e aspettando i risultati prodotti dal sistema.
La comunicazione come in tutti i contesti nei quali più parti devono interagire tra di loro deve essere regolata attraverso dei protocolli definiti a priori e conosciuti da entrambe le parti. E' consuetudine assegnare ad ogni protocollo un nome, e cosi è stato fatto scegliendo ExternalCommunicationProtocol, per distinguerlo dal protocollo di comunicazione usato all'interno dell'access-point per la comunicazione tra il modulo Bluetooth e la piattaforma multi-agente.
Dal punto di vista implementativo, la comunicazione via Bluetooth è stata realizzata utilizzando RFCOMM, un semplice protocollo di trasporto che emula una linea seriale RS-232 e riesce a gestire fino a 60 connessioni simultanee tra dispositivi Bluetooth [ 5 ]. Rimandiamo all'appendice B per maggiori informazioni su RFCOMM e sul suo utilizzo.

### 4.3.1 ExternalCommunicationProtocol

Vediamo ora di descrivere in dettaglio l'interazione che avviene in termini di scambio di messaggi tra i device mobili ed il modulo Bluetooth secondo quanto definito nell'apposito protocollo ExternalCommunicationProtocol.
Le operazioni che coinvolgono questo scambio di messaggi sono l'invio della keyword di ricerca, l'autenticazione del device, l'invio dei risultati prodotti dal sistema multi-agente (Imlpicit) e l'invio del feedback da parte del device mobile in relazione alle risorse gradite dall'utente. ExternalCommunicationProtocol è illustrato in Figura 15.

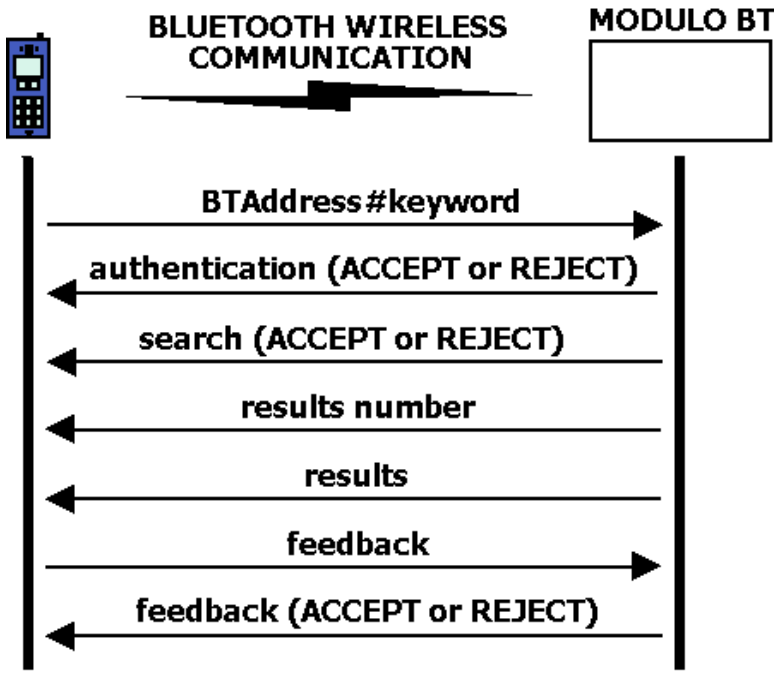


**Figura 15:** ExternalCommunicationProtocol

Il device mobile inizia la comunicazione inviando la keyword inserita dall'utente concatenata al suo indirizzo Bluetooth  Il modulo Bluetooth servendosi dell'indirizzo del device esegue la procedura di autenticazione (descritta più avanti) comunicando al device il suo esito attraverso un messaggio di ACCEPT or REJECT. A questo punto il modulo Bluetooth inoltra la keyword ricevuta al Personal Agent associato al device attivando in questo modo la procedura con la quale il sistema multi-agente produce i risultati. Se l'inoltro della richiesta va a buon fine viene inviato al device un messaggio di ACCEPT, relativo all'attivazione della ricerca , altrimenti REJECT. Quando il modulo riceve dal Personal Agent i risultati li spedisce al device mobile inviando prima il loro numero  e successivamente i risultati veri e propri. L'ultimo passaggio consiste nella spedizione al modulo Bluetooth del feedback relativo alle preferenze dimostrate dall'utente sulle risorse ricevute e del solito messaggio di conferma (ACCEPT or REJECT). L'utilizzo del feedback verrà chiarito nelle prossime sezioni.

#### 4.3.1.1 Autenticazione

La prima cosa che il modulo Bluetooth deve fare quando riceve il primo messaggio da parte del device mobile è autenticarlo. Nella concezione comune del termine, l'autenticazione mira al riconoscimento delle parti in gioco finalizzata a scopi di sicurezza per l'accesso a informazioni o sistemi di rete, attraverso diversi sistemi di crittografia o più comunemente utilizzando la canonica procedura di login ovvero username e password

Nel nostro prototipo sappiamo che ad ogni utente è associato un agente nella piattaforma. Lo scopo di questa procedura è quindi quello di permettere il riconoscimento univoco di un device mobile per poterlo poi associare al suo Personal Agent; a tal proposito è importante sottolineare come questa procedura deve essere fatta in modo automatico per non perdere tutti i benefici già discussi legati all'automazione completa dell'interazione tra i device e gli access-point.

Deve quindi innanzitutto esistere una struttura dati nella piattaforma (condivisa quindi da tutti gli access-point che ad essa appartengono) che memorizzi l'associazione tra i device ed i rispettivi Personal Agents. Vediamo come questa è stata realizzata.

Per prima cosa bisogna individuare dei parametri che permettano l'identificazione del device e l'associazione dello stesso con il suo Personal Agent. Per l'identificazione è stato utilizzato l'indirizzo Bluetooth[1] essendo questo universalmente unico ed infatti rifacendoci alla Figura 15 possiamo vedere che questo viene passato come primo messaggio insieme alla keyword. Per quanto riguarda l'associazione con il Personal Agent dobbiamo analizzare meglio la struttura software dell'access-point. Sappiamo che quando il modulo Bluetooth riceve una richiesta, deve poi inoltrarla nella piattaforma multi-agente all'agente corretto. Questo inoltro avviene attraverso socket. Ogni Personal Agent quindi si interfaccia al modulo Bluetooth attraverso un socket che è univocamente associato ad un numero di porta; questo numero è stato scelto per identificare l'agente. L'unica cosa che rimane da fare è quindi creare una associazione tra questi due identificativi.

---

[1] Ogni device Bluetooth ha un proprio indirizzo IEEE MAC Bluetooth a 48 bit univoco. E' sulla base di questo indirizzo dell'unità master che in una piconet viene definita la sequenza di hopping.

A tale scopo è stata creata un'apposita classe (AgentsTable.java) che implementa una semplice Hashtable nella quale le chiavi ed i valori ad esse associati sono rispettivamente gli indirizzi Bluetooth ed i numeri di porta dei socket.
Quando un'agente viene creato, tra le operazioni di inizializzazione deve quindi aprire un socket per interfacciarsi con il modulo Bluetooth su una porta non ancora utilizzata e registrarsi in AgentsTable passando come parametri l'indirizzo Bluetooth del device ad esso associato (che viene passato al costruttore dell'agente) ed il numero di porta del socket appena aperto.
La procedura di autenticazione si limita quindi ad un'operazione di get() su una Hashtable. Un esempio è illustrato in Figura 16.

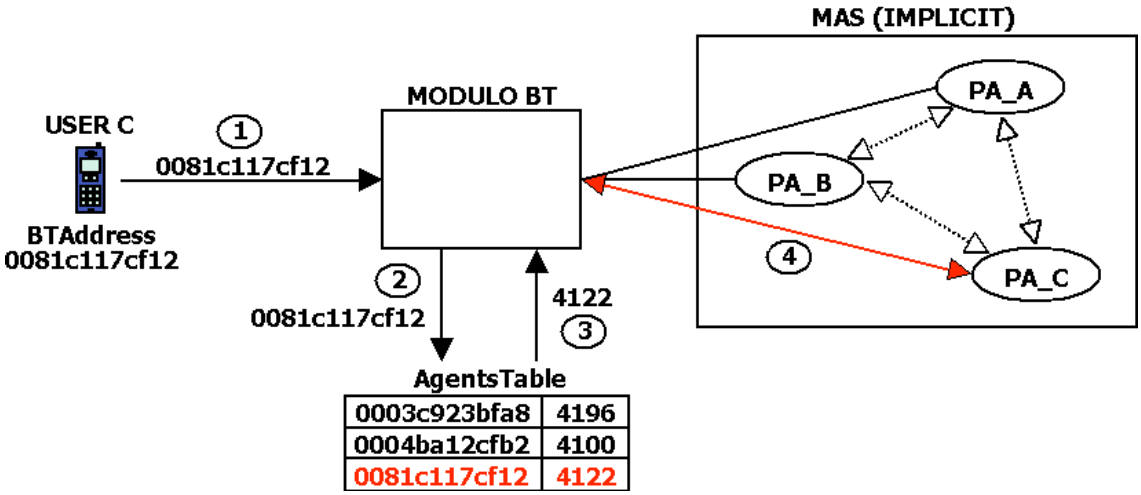


**Figura 16:** processo di autenticazione

Se l'autenticazione va a buon fine, significa che nella piattaforma esiste già un agente associato al device. In caso contrario si può per esempio creare ex-novo un agente per il nuovo device (non implementato nel prototipo).

## 4.3.1.2 Considerazioni sul feedback

E' bene chiarire il concetto e l'utilità del feedback. Per feedback intendiamo un "report" sulle risorse che l'utente ha dimostrato in qualsiasi modo di gradire. Questa

informazione è indispensabile per la piattaforma multi-agente perché rappresenta l'esperienza e quindi la conoscenza acquisita dall'utente. E' attraverso l'osservazione delle azioni compiute in termini di ricerche effettuate e gradimento delle risorse ritornate in seguito che Implicit riesce ad acquisire e successivamente condividere la conoscenza tra gli utenti attraverso raccomandazioni a risorse riguardanti determinati argomenti. Un problema è quello di come recuperare questi feedback senza un esplicito intervento dell'utente, perché ricordiamo che la nostra proposta è quella di creare un sistema in grado di acquisire autonomamente la conoscenza. La forte eterogeneità delle risorse trattate e degli ambienti di utilizzo, unita al tipo di device utilizzati rende questo lavoro non semplice. Nella nostra applicazione, essendo questa solo un prototipo, non è stata approfondita questa problematica. L'approccio utilizzato è stato il seguente: quando ad un utente vengono visualizzati i risultati, questo ha la possibilità di richiedere maggiori informazioni sulla risorsa a cui è interessato. Questo comportamento viene considerato come un gradimento e quindi viene inviato al modulo Bluetooh un feedback per la risorsa in oggetto. Un feedback molto analogo a questo e direttamente acquisibile dal device può essere considerato il salvataggio di un riferimento ad una risorsa nella memoria del cellulare.

Delle deduzione più significative si potrebbero fare per esempio nel caso in cui delle piattaforme mettessero a disposizione tra le funzionalità anche l'acquisto o la prenotazione di certi articoli. Immaginiamo per esempio il caso in cui una piattaforma sia installata in una libreria universitaria che permette di prenotare dei libri (anche nei giorni festivi e fuori dagli orari d'apertura). Sicuramente in questo caso l'azione di prenotare un libro acquisisce una rilevanza maggiore in termini di valutazione del gradimento o dell'interesse ad una risorsa.

Un'altra considerazione va fatta per le risorse on-line che sappiamo non essere accessibili direttamente dal device; abbiamo già detto che una soluzione è quella di trasferire queste risorse ad un computer collegato ad internet. Possiamo allora immaginare che su questo computer sia in esecuzione un'apposita applicazione che tiene traccia della storia dei links che vengono scaricati dal cellulare; controlla se vengono consultati, per quanto, se vengono scaricati. A questo punto potremmo fissare dei parametri in base ai quali valutare se le azioni compiute sui links possono essere considerate come feedback "positivi". Per feedback positivi possiamo considerare per esempio un link che viene visitato almeno n volte, o per un tempo complessivo di n minuti o se ad esso è associato il download di qualche file. Questi links potrebbero poi

essere ritrasferiti sul cellulare nelle successive riconnessioni e trasferiti alla piattaforma dai quali erano stati prelevati come feedback. Una soluzione ancora migliore e meno laboriosa potrebbe essere quella di trasferire direttamente il feedback alla piattaforma da Pc via internet.

## 4.4 Comunicazione tra il modulo Bluetooth ed il Personal Agent

Con l'autenticazione abbiamo visto che riusciamo a creare un collegamento virtualmente diretto tra un device ed il suo Personal Agent. In realtà questo è possibile grazie alla mediazione del modulo Bluetooth che riceve messagi da ambo le parti e le inoltra dall'altra parte. Ci rimane da vedere in che modo la comunicazione avviene all'interno dell'access-point negli scambi di messaggi via socket tra il modulo e la piattaforma multi-agente secondo quanto stabilito dal protocollo InternalCommunicationProtocol.

### 4.4.1 InternalCommunicationProtocol

Fondamentalmente l'interazione tra le parti in gioco è finalizzata all'inoltro della keyword di ricerca ricevuta dal device e all'attesa dei risultati prodotti da Implicit in merito. Ovviamente in questo caso lo scambio di messaggi è stato dettato dall'implementazione di Implicit, dal modo e dal formato cioè in cui questa piattaforma si aspetta di ricevere le richieste e fornisce le riposte.
Il protocollo ha inizio quando il modulo Bluetooth riceve la keyword di ricerca. Per prima cosa il modulo "adatta" in termini di formato la richiesta e la spedisce via socket all'agente, il quale se la riceve risponde con un messaggio di INFORM. A questo punto Implicit prevede che si possano richiedere diversi tipi di risultato e quindi è necessario che il modulo invii un messaggio, che nel nostro caso è standard, per richiedere tutti i risultati (<ALLREPLY>). Se la ricerca viene attivata correttamente all'interno della

piattaforma, l'agente risponde con un messaggio di INFORM, REFUSE altrimenti, e si attendono i risultati. Quando questi sono pronti, l'agente spedisce uno di seguito all'altro un messaggio con il numero dei risultati trovati ed i risultati rispettivamente. L'ultima azione riguarda l'inoltro del feedback ricevuto dal device all'agente e la consueta risposta di avvenuta ricezione con un messaggio di INFORM. La Figura 17 riepiloga questa sequenza di comunicazione.

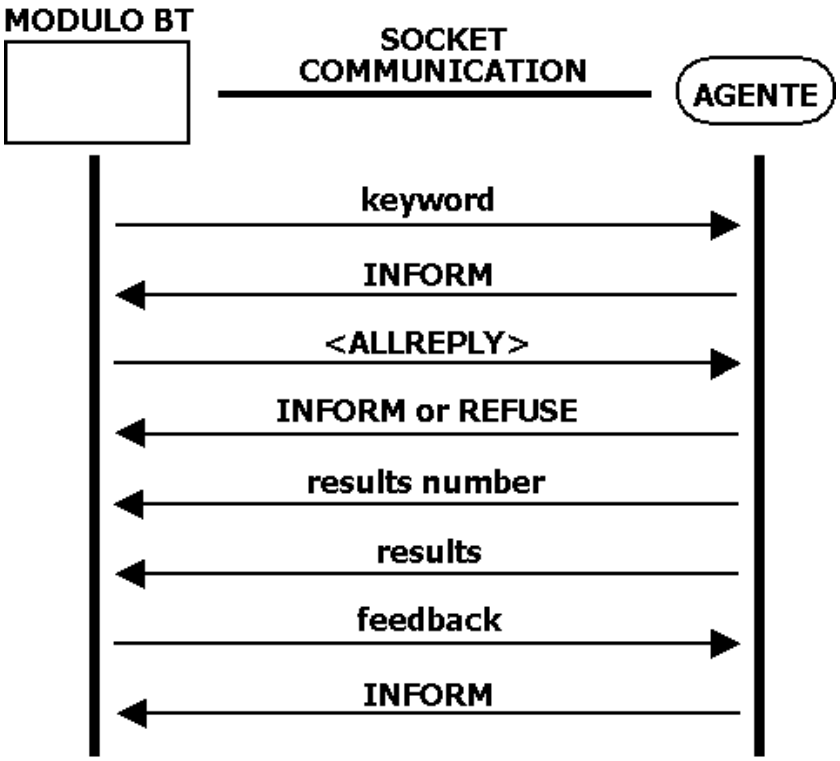


**Figura 17:** InternalCommunicationProtocol

# 4.5 CommunicationProtocol

Come si può facilmente immaginare, i due protocolli visti finora devono essere tra loro integrati ed amalgamati in modo tale da formare un unico protocollo di comunicazione: CommunicationProtocol (Figura 18).

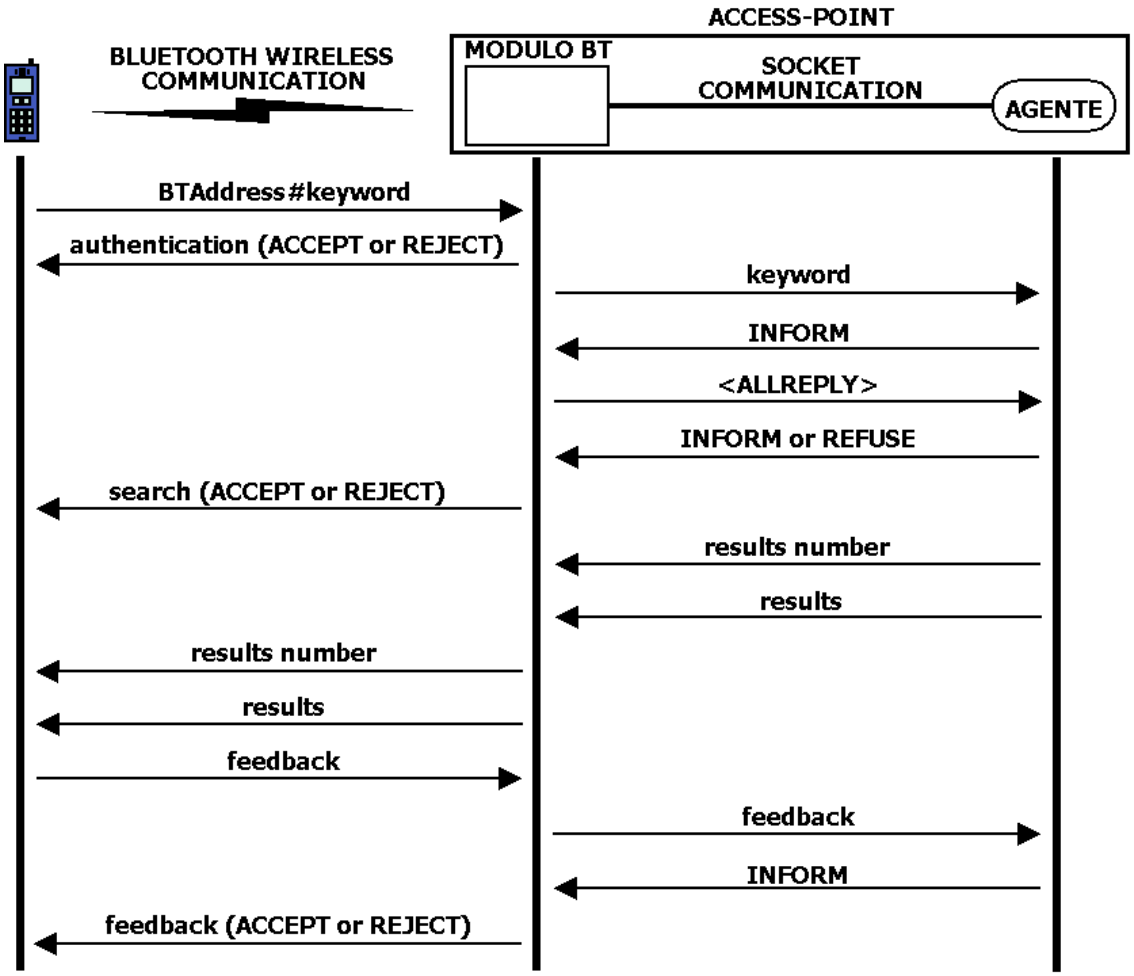


**Figura 18:** CommunicationProtocol

# 4.6 Implicit

Implicit è un sistema di raccomandazione multi-agente basato sul concetto di Cultura Implicita (Appendice A) e sviluppato in Jade, un framework Java per lo sviluppo di sistemi multi-agente FIPA[1]-compliant. Il contesto nel quale è già stato implementato è quello delle ricerche su web, con lo scopo di aumentare le funzionalità di ricerca di

[1] FIPA: Foundation for Intelligent Physical Agents; organizzazzione no-profit per la produzione di standards legati all'interazione di agenti software eterogenei

comunità di persone con interessi simili. Quando un membro di una comunità ha bisogno di informazioni su un certo argomento, l'idea è quella di verificare se altre persone in passato hanno avuto lo stesso problema di ricerca ed in tal caso proporre le soluzioni che questi hanno successivamente dimostrato di gradire [ 8 ].
Il sistema può essere visto come una collezione di Personal Agent che interagiscono tra di loro per soddisfare le richieste degli utenti. Ogni agente è dotato di un SICS (System for Implicit Culture Support) con il quale vengono forniti i suggerimenti sia all'utente associato, sia agli altri agenti, sulla base delle informazioni che ha collezionato osservando il comportamento del suo utente e degli agenti con i quali ha interagito.
Nell'implementazione classica, gli agenti eseguono tre tipi di ricerche [ 8 ]:

- ricerca su Google, se la richiesta proviene da utenti (potrebbe venire anche da altri agenti);
- ricerca interna al SICS: vengono ritornati dei links sulla base delle azioni passate eseguite dall'utente;
- ricerca esterna al SICS: viene ancora utilizzato il SICS, ma in questo caso l'obiettivo è quello di ritornare altri agenti da contattare.

Come possiamo capire, l'aspetto collaborativo di questo procedimento deriva dal fatto che un agente tenta di fornire dei suggerimenti all'utente attraverso le osservazioni fatte non solo sul comportamento del singolo utente ma anche sul comportamento degli utenti associati ai Personal Agents con i quali ha interagito. Questo permette all'agente di inoltrare la richiesta ad altri agenti i quali trattano la richiesta come se questa fosse fatta dai loro rispettivi utenti, con la differenza che i risultati vengono poi inviati non all'interfaccia utente ma ritornati all'agente che li aveva richiesti, il quale a sua volta li restituisce all'utente sotto forma di raccomandazioni. E' bene notare come i risultati restituiti dall'agente possono a loro volta contenere dei riferimenti ad altri agenti da contattare.
Gli agenti riescono ad interagire tra di loro usando il protocollo FIPA-Iterated-Contract-Net che inizia con una "call for proposal" per verificare la disponibilità di un agente nell'eseguire un'azione di ricerca. Gli utenti invece interagiscono con il loro agente usando il protocollo FIPA-Query (nel prototipo questo lavora "sopra" l'InternalCommunicationProtocol). Oltre a questi è stato implementato un terzo protocollo per la propagazione del feedback dell'utente. Questo protocollo prevede che

l'utente informi il suo Personal Agent riguardo l'accettazione o il rifiuto delle risorse suggerite e che il Personal Agent informi di questo anche gli altri agenti in precedenza interrogati [ 7 ].

### 4.6.1 Esempio di interazione collaborativa tra gli agenti

E' interessante vedere in che modo avviene l'interazione tra i Personal Agents nel loro compito collaborativo, senza entrare troppo nei dettagli del funzionamento di Implicit che non è oggetto di questo lavoro.
Consideriamo un semplice esempio, il caso in cui l'utente A richieda al suo agente (agent-A) informazioni riguardo alla parola "java". Supponiamo che il SICS suggerisca i seguenti links:

- www.mokabyte.it/forum/index.jsp;
- www.java.net;

e ritorni il riferimento a agent-B. Il Personal Agent di A quindi inoltra la richiesta a agent-B, il quale risponde con un altro indirizzo web:

- www.javaalmanac.com

ed il riferimento ad un altro agente, agent-C. Come prima agent-A inoltra la richiesta anche a agent-C il quale ritorna l'indirizzo:

- www.j3d.org;

A questo punto agent-A raccoglie tutti i risultati collezionati e li spedisce all'utente A.
Infine se l'utente esegue qualche operazione che viene considerata come un atto di gradimento e di accettazione verso qualche riferimento ritornato, per esempio www.j3d.org viene informato agent-A di questa scelta. Al Personal Agent quindi non rimane che propagare il feedback ricevuto e quindi informa agent-C in quanto è stato lui a suggerire quella risorsa e anche agent-B perché è stato lui a suggerirre agent-C. L'intero procedimento è schematizzato in Figura 19.

```
1. request(userA, agentA, "java");
2. return(SICS-A, agentA,  "www.mokabyte.it/forum/index.jsp" +
                           "www.java.net"+
                           agentB);
3. request(agentA, agentB, "java");
4. return(SICS-B, agentA, "www.javaalmanac.com" +
                          agentC);
5. request(agentA, agentC, "java");
6. return(SICS-C, agentA, "www.j3d.org");
7. inform(agentA, userA, "www.mokabyte.it/forum/index.jsp" +
                         "www.java.net" +
                         "www.javaalmanac.com" +
                         "www.j3d.org");
8. inform(userA, agentA, "accept(www.j3d.org)");
9. inform(agentA, agentC, "accept(www.j3d.org)");
10. inform(agentA, agentB, "accept(www.j3d.org)");
```

**Figura 19:** esempio di interazione tra gli agenti. Il formato dei messaggi è: operazione(mittente, destinatario, contenuto)

### 4.6.2 Modifiche eseguite su Implicit

Oltre al passaggio dall'interfaccia web a quella Bluetooth fin qui descritta, altre modifiche marginali sono state apportate alla versione originale di Implicit. Queste modifiche riguardano:

- la sostituzione delle fonti di acquisizione delle risorse disponibili; nella versione originale le risorse vengono acquisite attraverso delle ricerche con Google; nel nostro prototipo invece è stato utilizzato un file come banca-dati delle informazioni disponibili in ogni piattaforma; ovviamente questa è una soluzione temporanea e in sviluppi successivi si renderà necessario una migrazione di queste risorse in database;
- l'integrazione di nuovi tipi di risorse; Implicit è focalizzato come detto sulle ricerche web e quindi le risorse che tratta sono esclusivamente links web; nel nostro caso invece vogliamo estendere i tipi di risorse che possono essere oggetto di utilizzo

del nostro sistema, anche alla luce di quanto abbiamo detto sulla forte eterogeneità delle ambientazioni nel quale potrebbe operare. Per questo i risultati ritornati possono essere oltre che links a siti web, anche riferimenti a risorse locali, piuttosto che libri o conferenze.

Nessuna modifica è invece stata fatta al SICS ed al modo con il quale osserva le azioni degli utenti e degli altri agenti, comunica con gli altri agenti e fornisce i risultati.

## 4.7 Note sull'hardware ed il software utilizzati

L'idea di partenza del prototipo era quella di testare la comunicazione tra i device mobili (cellulari) ed un'access-point di una piattaforma (computer). Inizialmente la parte del prototipo che riguardava strettamente la comunicazione Bluetooth è stata realizzata, attraverso un'applicazione di tipo client-server, interamente utilizzando come ambiente di lavoro e di testing quello fornito dal J2ME Wireless Toolkit 2.2 della Sun Microsystems che nelle sue ultime versioni fornisce anche il supporto per il Bluetooth. Una volta terminato il lavoro e testato su questo emulatore, il passo successivo è stato quello di integrare il lato server di questa applicazione nel sistema multi-agente (Implicit) e implementare il lato client su un device mobile reale. Purtroppo non è stato possibile disporre di nessun cellulare con supporto al Bluetooth e quindi si è deciso di realizzare una comunicazione tra due computer. Per permettere ai computer di comunicare sono state utilizzate delle chiavette USB Bluetooth e si è reso indispensabile anche per il lato client il passaggio dalla piattaforma J2ME alla J2SE. Va sottolineato come questa "migrazione" sia risultata del tutto indipendente dalla piattaforma per quanto riguarda l'uso delle librerie JSR-82.
Per poter utilizzare la piattaforma J2SE però occorre disporre dell'implementazione di uno stack Bluetooth che supporti i profiles ed i livelli dei quali l'applicazione necessita per il suo funzionamento. Nel nostro caso è sufficiente disporre del Serial Port Profile e avere accesso ai livelli dello stack L2CAP E RFCOMM.

Si è quindi avviata una ricerca per trovare un'implementazione dello stack che facesse al caso nostro e alla fine è emerso che gli stack disponibili per Microsoft Windows sono a pagamento (versione trial di 15 giorni escluse). Solo con il service-pack 2 di Windows XP Microsoft fornisce una propria implementazione di questo stack.
Per quanto riguarda Il mondo Linux invece fortunatamente le cose cambiano; è già da qualche anno infatti che sono disponibili diverse implementazioni di stack Bluetooth (gratuite ed open-source), tra le quali il progetto meglio avviato e stabile è Bluez che è già presente come modulo aggiuntivo nel kernel linux a partire dalle versioni del kernel 2.4.6.
I pacchetti necessari per poter utilizzare Bluez sono bluez-kernel, bluez-libraries e bluez-utils.

| PROTOCOLLI | PROFILES |
|---|---|
| HCI | General Access Profile |
| L2CAP | Service Discovery Profile |
| RFCOMM | Serial Port Profile |
| BNEP | DialUp Networking Profile |
| | LAN Access Profile |
| | OBEX Object Push Profile |
| | OBEX File Transfer Profile |
| | PAN Profile |

**Tabella 3:** protocolli e profiles supportati da Bluez [ 9 ]

Le librerie di Bluez hanno però "l'inconveniente" di essere scritte in C e questo si scontra con l'implementazione del sistema multi-agente che è stato sviluppato utilizzando un framework Java.
Fortunatamente esistono però delle librerie-wrapper che permettono attraverso Jini di scrivere delle applicazioni che utilizzano JSR-82 e "rimappano" le chiamate scritte in Java in C. Di queste librerie ne esistono diverse e generalmente non supportano tutte le funzionalità messe a disposizione dalle JSR-82. La scelta è ricaduta su delle librerie chiamate AvetanaBluetooth che sono disponibili a pagamento per Windows e MacOS e gratuitamente per Linux. A seconda del sistema operativo usato queste librerie richiedono stack diversi e supportano funzionalità diverse. Nel caso di Linux lo stack

richiesto è Bluez e le limitazioni sull'uso delle JABWT riguardano essenzialmente la sicurezza; la cifratura e l'autenticazione infatti non sono supportate [ 10 ].
La Figura 20 riassume l'implementazione hardware e software utilizzata per la comunicazione Bluetooth in termini di protocolli, livelli dello stack e librerie.

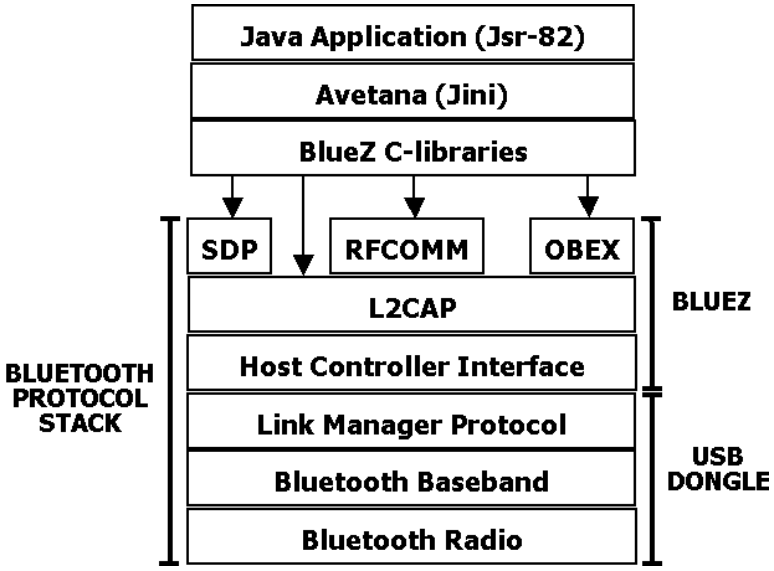


**Figura 20:** implementazione hardware e software utilizzata per la comunicazione Bluetooth

Nel dettaglio il prototipo è stato testato utilizzando i seguenti componenti software e hardware:

- kernel linux della famiglia 2.6;
- modulo bluez-kernel presente tra i moduli aggiuntivi del kernel linux di cui sopra;
- pacchetto bluez-libs e bluez-utils versione 2.11 scaricati dal sito ufficiale www.bluez.org;
- pacchetto bluez-firmware versione 1.0 che fornisce i driver bcm203x per utilizzare chiavette USB di tipo Broadcom;
- piattaforma Java 2 Standard Edition versione 1.4.2;
- librerie AvetanaBluetooth versione 1.0.23;
- Implicit versione aggiornata al 13/02/2004;
- Jade versione 2.5;
- Tecom Bluetooth USB dongle modello BT3030;
- D-Link Bluetooth USB dongle modello DBT-120 revisione B3;

La disponibilità di solo due chiavette Bluetooth ha impedito il testing delle funzionalità di gestione di più connessioni da parte del modulo Bluetooth. Tale funzionalità era stata comunque testata con esito positivo con la versione iniziale testata sull'emulatore software della Sun.

## 4.8 Screenshots

Vediamo ora due screenshots che si riferiscono ad un esempio di utilizzo del prototipo. La comunicazione avviene tra due notebook; in uno è in esecuzione il software che simula un access-point (modulo Bluetooth e sistema multi-agente), mentre nell'altro l'applicazione lato client, ovvero quella che dovrebbe essere eseguita dai device mobili. L'esempio illustrato è stato eseguito partendo dal presupposto di avere un sistema già "rodato", nel quale fossero già state compiute diverse azioni da parte degli utenti, per fare in modo che Implicit riesca a produrre qualche tipo di risultato.
La piattaforma multi-agente, quando ha finito di inizializzarsi lancia l'esecuzione del modulo Bluetooth che pubblica il service "BTImplicitBoard Service" e attende connessioni (Figura 21). L'applicazione del device mobile inizia quindi con le classiche operazioni di inquiry e ricerca services. Ogni volta che trova un device, richiede a questo il servizio "BTImplicitBoard Service" e se lo trova si connette al device, ossia all'access-point e inizia la comunicazione secondo quanto definito dal già discusso CommunicationProtocol (Figura 22).

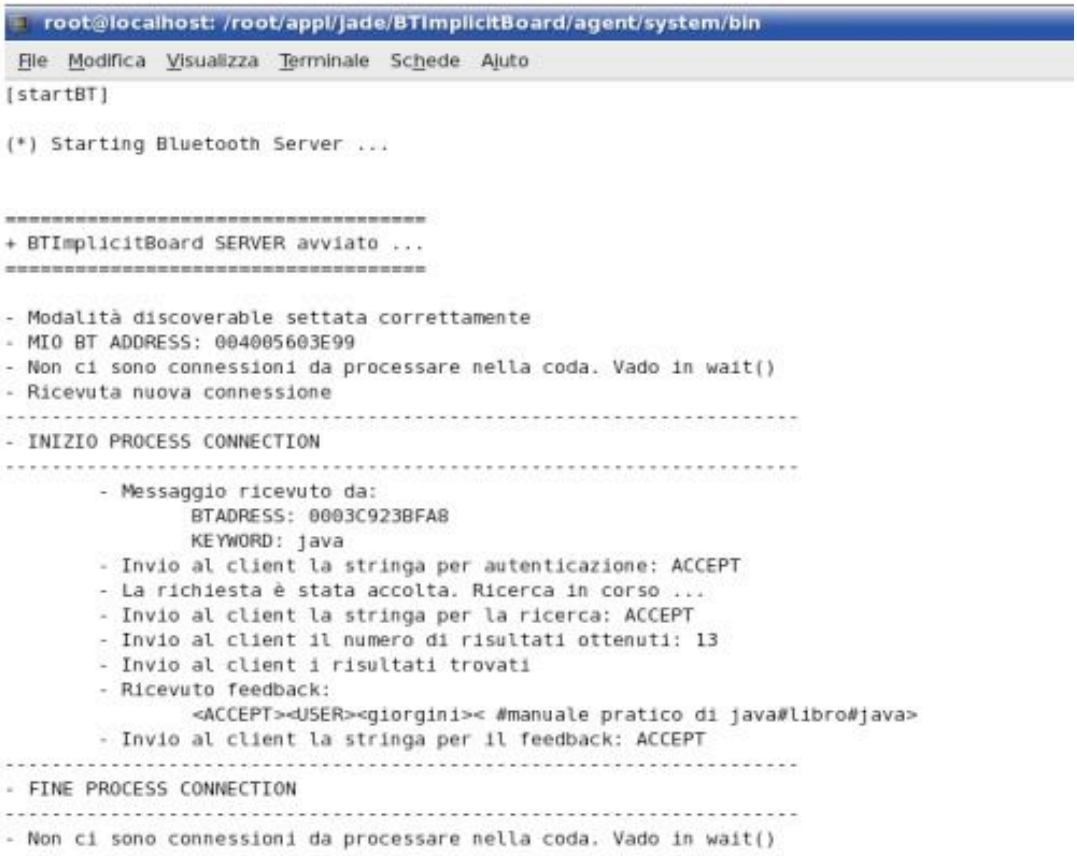


**Figura 21:** screenshot del modulo Bluetooth


**Figura 22:** screenshot dell'applicazione per il device mobile

# 5. Conclusioni

In questo lavoro abbiamo proposto un nuovo sistema per l'acquisizione, la gestione e la condivisione della conoscenza acquisita nell'uso di pezzi d'informazioni basato sul concetto di località. L'utilizzo di device mobili come mezzo di interazione con questo sistema ha permesso di sfruttarne una caratteristica peculiare che è quella di essere dove è fisicamente l'utente; in questo modo le esperienze raccolte possono essere considerate contestualizzate in base al posto fisico nel quale avviene l'interazione.
Il modo con il quale il sistema riesce ad acquisire la conoscenza è basato sull'osservazione delle azioni compiute dagli utenti (esperienza nell'uso delle risorse fornite). La condivisione della conoscenza invece si concretizza attraverso le raccomandazioni che altri utenti considerati simili per interessi (sulla base delle osservazioni fatte) forniscono.
Un'altra proprietà sulla quale si è puntato è quella dell'autonomia del sistema nel raggiungimento del suo obiettivo. Questo significa che il sistema deve in maniera autonoma riuscire ad acquisire, gestire e condividere la conoscenza, senza un intervento volontario dell'utente. L'utente semplicemente utilizzando il sistema, accetta di condividere la propria conoscenza ed i propri interressi con gli altri utenti.

Nella prima parte del lavoro è stata progettata e proposta un'architettura per realizzare il sistema appena descritto. In particolare abbiamo visto come il problema dell'acquisizione e condivisione della conoscenza può essere risolto utilizzando un sistema multi-agente abbinato all'uso di tecniche di collaborative-filtering. Abbiamo visto inoltre come le soluzioni proposte per risolvere alcune problematiche relative allo spostamento dei device mobili hanno influito in modo singificativo sull'architettura del sistema.
Dopo la progettazione si è passati allo sviluppo di una parte dell'intero sistema. Il prototipo sviluppato, BTImplicitBoard, ha come obiettivo quella della comunicazione via

Bluetooth tra i device mobili ed un sistema multi-agente attraverso un access-point. Per questi motivi è stato utilizzato Implicit, un sistema multi-agente già sviluppato e che di basa sul concetto di Cultura Implicita per il trasferimento della conoscenza. Alla luce di questo, i test effettuati si sono limitati a verificare innanzitutto la comunicazione via Bluetooth e successivamente la corretta interazione tra un device mobile ed il Personal Agent ad esso associato all'interno della piattaforma multi-agente. Il funzionamento ed i risultati forniti da Implicit invece non sono stati valutati perché già oggetto di altri studi e perché non rientrava tra gli obiettivi del prototipo.

In conclusione il prototipo realizzato rappresenta il punto di partenza per la realizzazione del sistema finale che abbiamo immaginato e descritto. Sviluppi futuri saranno mirati all'estensione del dominio operativo di un agente, ovvero si dovranno implementare piattaforme composte da più access-point per rendere disponibili più punti d'accesso dislocati in diversi punti. Un'altra funzionalità che deve essere sicuramente implementata se si vuole arrivare ad avere un sistema realmente funzionale è quella del recupero dei dati pendenti, secondo le modalità che già abbiamo previsto: il recupero attraverso altre piattaforme e quello via internet utilizzando un Pc.

# 6. Bibliografia

[ 1 ]
P. Massa,
*Dal Collaborative Filtering alla Cultura Implicita: analisi e sviluppo di un sistema di supporto (1999-2000).*
URL: http://sra.itc.it/people/massa/thesis/

[ 2 ]
F. Bellifemine, G. Caire, A. Poggi, G. Rimassa,
*Jade. A White Paper (Settembre 2003).*
URL: http://jade.tilab.com

[ 3 ]
E. Blanzieri, P. Giorgini, F. Giunchiglia, C. Zanoni
*A Multi-agent System for Knowledge Management based on the Implicit Culture Framework*
URL: http://dit.unitn.it/~implicit Sezione: Papers

[ 4 ]
David Kammer, Gordon McNutt, Brian Senese, Jennifer Bray,
*Bluetooth Application Developer's Guide: The Short Range Interconnect Solution*
Published by Syngress Publishing Inc. (2002)

[ 5 ]
Bluetooth SIG,
*Specification of the Bluetooth System Core (Version 1.1) (2001)*
URL: http://www.bluetooth.com

[ 6 ]
Ben Hui,
*Go Wild Wirelessly with Bluetooth and Java. Developing portable Bluetooth apps. (February 5, 2004)*
URL: http://jdj.sys-con.com/read/43551.htm

[ 7 ]
E. Blanzieri, P. Giorgini, F. Giunchiglia e C. Zanoni,
*Implicit Culture-based Personal Agents for Knowledge Management*
URL: http://dit.unitn.it/~implicit Sezione: Papers

[ 8 ]
A. Birukov, E. Blanzieri, P. Giorgini,
*Implicit: An Agent-Based Recommendation System for Web Search (2004)*
URL: http://eprints.biblio.unitn.it/archive/00000714/

[ 9 ]
Bluez,
URL: http://www.bluez.org/

[ 10 ]
AvetanaBluetooth,
URL: www.avetana-gmbh.de/avetana-gmbh/produkte/Readme.xml

[ 11 ]
E. Blanzieri and P. Giorgini,
*From collaborative filtering to implicit culture: a general agent-based framework.*
In "Proceedings of the Workshop on Agent-Based Recommender System (WARS in Autonomous agents (Agents2000, ACM)".

[ 12 ]
E. Blanzieri, P. Giorgini, F. Giunchiglia e C. Zanoni,
*A Multi-agent System for Knowledge Management based on the Implicit Culture Framework*
URL: URL: http://dit.unitn.it/~implicit Sezione: Papers

[ 13 ]
*Developing Bluetooth Applications in Java: Part 1 (June 11, 2003).*
URL: http://www.commsdesign.com/ Sezione: Design Corner

[ 14 ]
Sun Microsystems,
*Java APIs for Bluetooth Wireless Technology (JSR-82). Specification Version 1.0, Java 2 Platform, Micro Edition*
URL: http://jcp.org/aboutJava/communityprocess/final/jsr082/index.html

[ 15 ]
C Bala Kumar, Paul J. Kline, and Timothy J. Thompson, Motorola, Inc.
*Developing Bluetooth Applications in Java: Part 2 (June 11, 2003)*
URL: http://www.commsdesign.com/ Sezione: Design Corner

# Appendice A

## Cultura Implicita

Il concetto di Cultura Implicita è stato introdotto per la prima volta da Blanzieri e Giorgini in [ 11 ]. Essi indicano come fenomeno di Cultura Implicita quello che si verifica quando un gruppo di agenti si comporta in maniera consistente con la “cultura” di un altro gruppo senza interazione diretta o sforzo addizionale da parte di alcuno.

In particolare, quando un agente comincia ad agire in un nuovo ambiente con conoscenze ed abilità limitate, il suo comportamento non può che essere inferiore a quello ottimale; questo perché non è in possesso di sufficiente conoscenza riguardo all'ambiente e agli altri agenti [ 1 ].

Un agente, per rendere più efficace il suo comportamento, dovrebbe poter agire in accordo con la “cultura” del gruppo. Le soluzioni immaginabili sono diverse ma tutte comportano il fatto che l'agente venga a conoscenza di quella che è la ``cultura'' con tutti i problemi di formalizzazione, comunicazione o apprendimento di essa [ 1 ].

Per ottenere questo, invece di agire sulle capacità dell'agente, è possibile modificare la visione che lo stesso ha dell'ambiente e, di conseguenza, le sue azioni. Ciò viene fatto per far in modo che esso agisca con un comportamento analogo a quello del gruppo. D'altra parte, non è necessario che ne l'agente, ne il gruppo sappiano nulla di questo procedimento, riuscendo così a condividere la stessa ``cultura'' in modo implicito [ 1 ].

L'assunzione di base è che il mondo in cui gli agenti agiscono sia composto di oggetti e altri agenti. Prima di eseguire un'azione, un agente ha di fronte una scena formata da una porzione del mondo (quindi agenti e oggetti) e dalle azioni che sono in esso possibili. Si dice quindi che l'agente compie azioni situate. Dopo aver eseguito un'azione, l'agente ha di fronte una nuova scena che dipende sia dall'ambiente che dall'azione situata appena compiuta. L'azione scelta dall'agente tra quelle possibili nella scena in cui è situato dipende dai suoi stati interni ed inaccessibili e, in generale,

non è predicibile in maniera deterministica. Tuttavia si assume che si possa caratterizzare in termini di probabilità e valori attesi [ 1 ].
Inoltre, dato un gruppo di agenti, si suppone che esista una teoria che predica sulle loro azioni situate attese. Se essa è consistente con le azioni eseguite dal gruppo, viene considerata validata. La teoria cattura le abilità e la conoscenza che gli agenti hanno del mondo [ 1 ].
Se un gruppo di nuovi agenti esegue azioni che soddisfano la teoria culturale validata del gruppo, il problema del loro comportamento subottimale rispetto al gruppo di riferimento è risolto. Il termine ``Cultura Implicita'' rappresenta proprio questo ovvero il fatto di avere un gruppo di agenti tali che le loro azioni situate soddisfano la teoria culturale validata di un altro gruppo senza la necessità che nessuno sappia di questo procedimento [ 1 ].

# Sistema di Supporto alla Cultura Implicita (SICS)

Lo scopo di un Sistema di Supporto alla Cultura Implicita è quello di creare un fenomeno di Cultura Implicita. Questo può essere fatto attraverso la creazione di teorie culturali validate a partire dalle osservazioni sulle azioni situate e la presentazione di scene agli agenti in modo che le loro azioni situate soddisfino la teoria [ 3 ].
Un SICS è costituito dai seguenti tre componenti (Figura 23):

- Observer: osserva e memorizza in un database le azioni eseguite da un gruppo di agenti G [ 7 ] : le richieste, le risposte, i risultati considerati rilevanti e quelli rifiutati. Le informazioni memorizzate sono del tipo:
  - richiedi (a,b,w,x): l'entità a invia una richiesta all'entità b per il servizio w al tempo x;
  - informa (a,b,w,y): in risposta ad una azione di richiesta, l'entità a informa l'entità b sul servizio w al tempo y.
- Modulo Induttivo: usa le azioni memorizzate dall'Observer per produrre una teoria culturale valicata $\sum$ per G [ 7 ]. Attualmente questo componente non è implementato e la teoria culturale utilizzata è fissata a priori. La regola utilizzata

per esprimere la teoria culturale è la seguente [ 7 ]: se al tempo $t_1$ x (utente o agente) chiede all'agente PA informazioni a riguardo della keyword k, e PA al tempo $t_2$ risponde informando x che y (link o agente) è per lui importante, con $t_1$ minore di $t_2$, allora x al tempo $t_3$ accetterà come rilevanti informazioni da y per la keyword k, con $t_2$ minore di $t_3$. Questo significa che in base alla teoria specificata gli agenti dovrebbero accettare le informazioni che vengono loro proposte e possiamo formalizzarlo così:

$$\text{richiesta}(x,PA,k,t_1) \wedge \text{informa}(PA,x,y,t_2) \wedge \text{minore}(t_1, t_2) \longrightarrow$$

$$\text{accetta}(x,y,k,t_3) \wedge \text{minore}(t_2, t_3)$$

- Composer: usando la teoria $\sum$ prodotta dal Modulo Induttivo e le azioni memorizzate dall'Observer, propone ad un gruppo di agenti G' un insieme di scene, in modo che le loro azioni situate attese soddisfino la teoria culturale validata per il gruppo G [ 7 ].

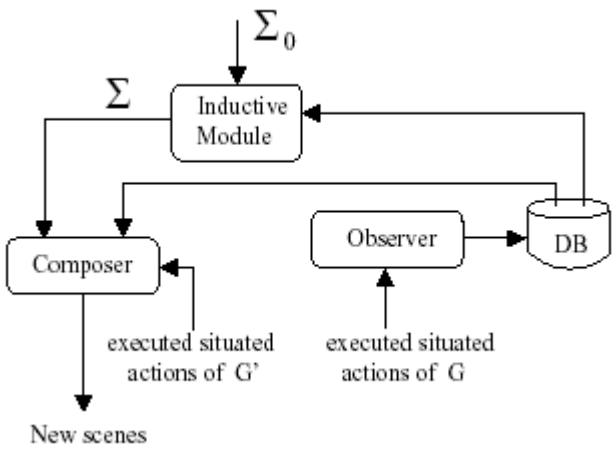


**Figura 23:** architettura schematica del SICS [ 12 ]

# Appendice B

## Sviluppare software Bluetooth: JSR-82

Per permettere l'implementazione del Bluetooth sul maggior numero di device possibili, agevolandone la diffusione, è stata progettata e rilasciata una specifica aperta e senza royalty dal Bluetooth SIG che ne definisce sia i livelli hardware, sia quelli software.

Un comportamento ben diverso è invece stato adottato dai produttori dei moduli Bluetooth che forniscono delle apposite SDK per interfacciarsi con i propri moduli, vincolando in questo modo gli sviluppatori all'uso di API proprietarie per i loro rispettivi chip-sets. 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 device Bluetooth si comportano e come interagiscono tra di loro attraverso i profiles, tralasciando il modo con cui questi devono essere programmati o come un'applicazione deve utilizzare le funzionalità di questi dispositivi ed i canali di comunicazione [ 6 ].

Per risolvere questi problemi di compatibilità, che ostacolavano la creazione di 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, la scelta del linguaggio di programmazione da utilizzare ricadde su Java. Oltre ai benefici della portabilità, che permettono alle API Java di essere eseguite su hardware, sistemi operativi e classi di device diversi, Java permette inoltre un rapido sviluppo delle applicazioni, grazie al suo paradigma di programmazione orientato agli oggetti che ne garantisce un'astrazione ad alto livello, ed una comunità di sviluppatori già molto estesa [ 13 ].

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 Technology) 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) che hanno limitate capacità di calcolo, di memoria e batteria [ 14 ].

Gli sviluppatori possono così creare delle applicazioni Bluetooth indipendenti dall'hardware utilizzato e che quindi possono essere implementate anche su device diversi purchè questi supportino JSR-82. Per poter utilizzare le JABWT un device deve disporre di uno stack che sia qualificato per almeno il Generic Access Profile, il Service Discovery Application Profile ed il Serial Port Profile e che permetta l'accesso al Service Discovery Protocol ed ai suoi livelli RFCOMM e L2CAP [ 14 ].

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) [ 14 ].

Proprio per 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 [ 14 ]:

- la comunicazione dati e non quella vocale;
- l'utilizzo dei protocolli;
- L2CAP (solo nella versione orientata alla connessione);
- RFCOMM;
- SDP;
- OBEX (OBject EXchange protocol).

ed i profiles:

- Generic Access Profile (GAP)
- Server Discovery Access Profile (SDAP)
- Serial Port Profile (SPP)
- Generic Object Exchange Profile (GOEP)

Le funzionalità che mettono quindi a disposizione le JABWT sono [ 14 ]:

- la ricerca di device e services
- la registrazione dei services

- la possibilità di stabilire connessioni di tipo RFCOMM, L2CAP e OBEX
- la possibilità di eseguire queste operazioni in una modalità “sicura”

# Utilizzare JSR-82

Vediamo ora in che modo e attraverso quali classi ed interfacce JSR-82 permette di eseguire alcune delle funzionalità base e standard dei dispositivi Bluetooth: la ricerca dei device, la ricerca dei services disponibili e la comunicazione attraverso un canale RFCOMM.

# Ricercare altri device Bluetooth

La prima cosa che deve fare un’applicazione di rete Bluetooth è ricercare gli altri device abilitati presenti nel proprio range di copertura dell’antenna. Per device abilitati si intendono tutti quei device Bluetooth che sono in modalità discoverable ossia in inquiry scan.
Il device che inizia questa procedura spedisce in modalità broadcast un messaggio di inquiry e attende che i device presenti nella zona rispondano. Per stabilire quando un device deve rispondere sono state definite tre modalità di discoverable (general, limited e not discoverable) e due diversi tipi di inquiry (general e limited) [ 13 ].
Quando un device esegue una general inquiry tutti i device in modalità general e limited discoverable devono rispondere al messaggio. Nel caso in cui invece si esegua una limited inquiry solo i device in modalità limited discoverable risponderanno. I device che sono in modalità not discoverable non rispondono in nessun caso a messaggi di inquiry ed è come se fossero invisibili agli altri dispositivi [ 13 ].
Per permettere di settare la modalità di discoverable le JSR-82 definiscono il metodo setDiscoverable() della classe LocalDevice. L’argomento passato a setDiscoverable() rappresenta la modalità di discoverable che il device dovrà usare per rispondere a messaggi di tipo inquiry [ 13, 14 ].

Per eseguire la ricerca vera e propria dei device si utilizzano i metodi della classe DiscoveryAgent che come vedremo si occupa anche della ricerca dei services. A tale scopo sono forniti due metodi: startInquiry() e retrieveDevices() [ 13, 14 ].
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() [ 13, 14 ].
Il metodo startInquiry() è una chiamata non bloccante e questo significa che ritorna prima della terminazione della procedura. E' per questo motivo che si rende necessario il DiscoveryListener; al termine dell'inquiry infatti l'implementazione delle JABWT prevede che venga richiamato il metodo inquiryCompleted() della classe DiscoveryListener per mettere a conoscenza l'applicazione della conclusione della ricerca [ 13, 14 ].
Il metodo deviceDiscovered() viene invece richiamato automaticamente ogni volta che viene scoperto un nuovo device 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) [ 13, 14 ].
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 device che erano già stati scoperti da inquiry precedenti e quindi non fornisce nessun tipo di garanzia sulla loro reale presenza [ 13, 14 ]. 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.

# Ricercare i services disponibili

Una volta che si conoscono i device presenti nella nostra area di “operabilità”, il passo successivo è quello di determinare quali services questi mettono a disposizione e per fare questo si utilizza il Service Discovery Protocol definito dalle specifiche Bluetooth. Un’applicazione che vuole fornire dei services, e che per questo funge da server, deve permettere agli altri device di trovare questi services, e per far questo li pubblica tra i suoi service records i quali, attraverso una serie di attributi, li descrivono (Figura 24). Questi attributi specificano il nome del service, il modo con il quale connettersi ad esso, una breve descrizione del suo funzionamento ed altre informazioni utili al suo corretto utilizzo [ 15 ].

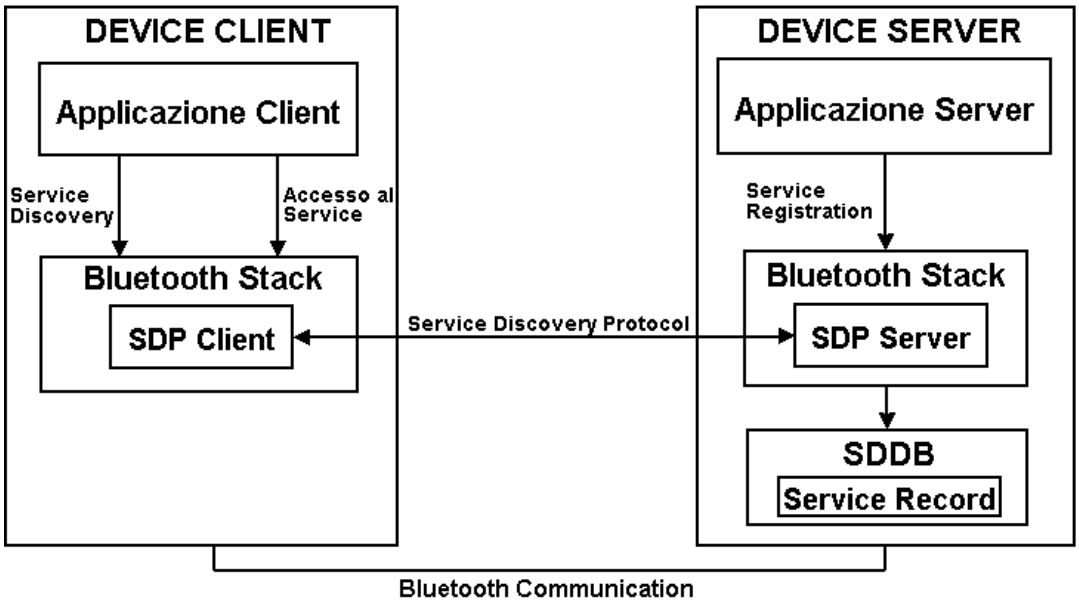


**Figura 24**: Il device server ha registrato nel SDDB un service record che il client può ottenere interrogando l’SDP server utilizzando il Service Discovery Protoxcol

La procedura con la quale un server rende visibili i propri service records ai potenziali clients 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 service, 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 anche l'identificatore globalmente univoco del service (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 [ 14, 15 ].

Arrivati a questo punto il service non è ancora visibile agli altri dispositivi. Per farlo occorre andare a modificare il contenuto del SDDB e per far questo si usa il metodo acceptAndOpen() sull'istanza della classe StreamConnectionNotifier ritornata da Connector.open(). Ora il service è correttamente registrato e il server è in attesa di clients che si connettano al service [ 14, 15 ].

Vediamo ora quali sono le operazioni, definite dal Service Discovery Application Profile, che devono fare i clients per trovare e connettersi ai services. Supponiamo che sia già stata eseguita la procedura di inquiry e che abbia ritornato come risultato alcuni device. Vogliamo ora verificare se qualcuno di questi dispone di un particolare service al quale siamo interessati e del quale già conosciamo il suo UUID.

Per fare questo dobbiamo richiamare su ognuno dei device 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 devices anche in questo caso occorre implementare l'interfaccia DiscoveryListener (passando il Listener alla funzione searchServices()) e definirne i metodi servicesDiscovered() e serviceSearchCompleted() [ 14, 15 ].

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 [ 14, 15 ].

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 perché, nel caso in cui il service 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 richiamano il

metodo serviceSearchCompleted() che riporta lo stato d'uscita dell'operazione (ricerca completata correttamente, ricerca terminata dall'utente, errori).
Se la ricerca è andata a buon fine ora il client è in grado, con le informazioni contenute nel/i service record/s, di creare delle connessioni con altri dispositivi che mettono a disposizione il service ricercato.

# Comunicare con RFCOMM

Il protocollo RFCOMM fornisce l'emulazione di una linea seriale di tipo RS-232 tra due dispositivi Bluetooth [ 5 ]. 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) [ 13 ]. Questa scelta permette di avere un'estrema flessibilità dal punto di vista dei protocolli di comunicazione utilizzati, perché come vedremo 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 come parametro riceve una stringa che identifica il protocollo da utilizzare e il device 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 service 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 [ 13 ].
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:efca5621975548568f27b437f6e5e6b2
btspp://localhost:93007CA747114F42b1dcce878a6531f;name=MySe
rvice
```

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 services disponibili sui device trovati in fase di inquiry richiamando il metodo getConnectionURL() dell'interfaccia ServiceRecord che ritorna la stringa di connessione completa per connettersi a quel service.
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 [ 13 ].
Se invece a Connector.open() viene passata una stringa di connessione del server (riconoscibile perché 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 clients. 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 [ 13 ].

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 volessimo 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

# Indice delle figure

# Ringraziamenti

Siamo alla fine della fine, ovvero alla fine del lavoro finale di questo periodo di studi e quindi all'inizio di un non so di che cosa.
Le persone che dovrei e vorrei ringraziare sono davvero tante fortunatamente.
Mi sembra giusto iniziare con chi ha condiviso direttamente questa esperienza e quindi tutti i compagni di corso; non mi sembra corretto elencarvi uno ad uno perché sicuramente dimenticherei più di una persona. Voi sapete chi siete.
Uno special thanx va anche a tutti gli amici "extra-universitari", alcuni pure extra-terrestri, ed in particolare:

- il fedele compagno di bevute e di avventure Omar (Cicio) ed il mio stilista personale Rodolfo Rodolfi (Lorenzo);
- Elvis, Andrea e Matteo ovvero la parte mancante dei Koroba Milk;
- Giacomo, inseparabile compagno di stanza;
- Francesco (Turco), Mario e Erika, grandi compagni di appartamento;

Un ringraziamento sentito anche al Prof. Giorgini per aver accettato di essere il mio relatore e per avermi seguito ed assistito in questo lavoro anche nei momenti in cui era dall'altra parte del globo. Grazie anche ad Alexander Birukov che mi ha aiutato nella comprensione del funzionamento della piattaforma Implicit.

Infine non posso dimenticare di ringraziare la mia famiglia per l'aiuto costante che mi è stato dato e soprattutto per aver dimostrato di saper rispettare le mie scelte e le mie idee: una cosa davvero rara.