UNIVERSITÀ DEGLI STUDI DI TRENTO
Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Informatica

Elaborato Finale

Route Map turistica con pushing d'informazioni in real-time: MyBus

Relatore

Paolo Giorgini

Laureando

Devid Verra

Anno accademico 2007-2008

Ringrazio la mia famiglia per il costante
sostegno ed incoraggiamento.

Un ringraziamento particolare a
Michele, Fabrizio, Paolo, Fabrizio e Michele
con cui ho condiviso questi anni di studio.

Infine un grazie anche a Valentina e Michele
che mi hanno sopportato e supportato durante
la scrittura di questa Tesi e non solo.

Sommario

Il lavoro presentato prevede la realizzazione di un sistema per dispositivi mobili, quali telefoni cellulari o PDA, che permetta agli utenti di ricevere automaticamente informazioni turistiche in *real-time* sui punti d'interesse (POI) nelle vicinanze; inoltre sará possibile visualizzare su una mappa la posizione dei POI e richiedere al sistema come raggiungerli utilizzando la rete dei trasporti pubblici.
Il sistema sará formato da due applicazioni, una lato client ed una lato server, e verrá realizzato in Java; in particolare per l'implementazione della parte client sará utilizzata la piattaforma $J2ME^{\text{TM}}$, mentre per lo sviluppo del server sará utilizzata la piattaforma $J2EE^{\text{TM}}$

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Motivazione

Da sempre l'uso delle giude turistiche permette di scoprire, visitare e conoscere cittá, punti d'interesse artistico, storico e culturale (Points Of Interest). Il supporto cartaceo peró ne limita l'utilizzo in quanto risulta ingombrante, non aggiornabile e con costi di produzione relativamente alti. Inoltre potrebbe essere utile sapere come poter raggiungere tali POI utilizzando i mezzi di trasporto pubblico evitando i problemi del parcheggio, delle zone a traffico limitato e problemi di viabilitá di vario genere. Lo stato dell'arte attuale é diviso principalmete in due categorie: sono presenti guide turistiche in formato cartaceo oppure in formato digitale; ma entrabe non permettono di sapere, senza doverle sfogliare, informazioni su quello che si trova nelle vicinanze della propria posizione. Altri dispositivi, invece, permettono di conoscere la posizione dei luoghi di interesse nelle vicinanze. L'ulteriore sviluppo che si sta apportando deriva dal fatto che, grazie all'avvento di telefoni cellulari ad elevate prestazioni e con interfacce utente sempre piú sofisticate, sia possibile individuare la propria posizione attraverso dispositivi GPS (sia integrati nel telefono/palmare sia esterni) ovvero tramite l'utilizzo delle celle GSM, e fornire informazioni aggiornate e dettagliate in tempo reale su ció che ci circonda. Nasce quindi un sistema per ricevere informazioni turistiche georeferenziate, che permette, inoltre, di richiedere come raggiungere i punti d'interesse muovendosi attraverso la rete di mezzi pubblici. Ora, in qualun-

que momento ed in qualunque luogo l'utente si trovi (purché sia presente una copertura di rete da parte dei gestori di telefonia mobile) é possibile ricevere informazioni turistiche in real-time direttamente sul proprio telefono cellurale o PDA, senza dover acquistare in loco cartine turistiche e senza preoccuparsi di come arrivare a destinazione.

Applicazioni

Le caratteristiche dei nuovi telefoni cellulari presenti sul mercato consentono all'utente il download di filmati attraverso la rete GPRS ed UMTS, la possibilitá di effettuare video-chiamate e la possibilitá di vedere trasmissioni televisive in formato digitale. Si aggiunge ora a questi utilizzi, anche quello di poter ricevere informazioni turistiche, in qualunque posto ci troviamo. Si pensi, ad esempio, al turista che arriva a Trento, sprovvisto di cartina o guida turistica, ma che sicuramente avrá un telefonino; grazie al quale potrá avere tutte le informazioni su cosa visitare e come poter raggiungere tali luoghi. Oppure semplicemente passeggiando per la cittá, verranno visualizzate, automaticamente, sul display del proprio telefono cellulare, le informazioni sui POI nelle vicinanze.

Obiettivo della tesi

La presente Tesi si é occupata di sviluppare un'applicazione per dispositivi mobili che permetta agli utenti di ricevere informazioni in *real-time* relative a POI che si trovano nelle vicinanze della propria posizione, rilevata tramite GPS. Gli utenti possono inoltre richiedere al sistema come poter raggiungere un punto d'interesse specifico, scelto da una lista, utilizzando i mezzi pubblici, tale percorso verrá visualizzato sulla mappa oppure in formato testuale.
Trattandosi di un argomento innovativo e non ancora preso in considerazione, vi é la necessitá iniziale di orientarsi nello studio dello Stato dell'arte, per

poi passare ad un'implentazione che verrá via via sviluppata e performata in base alle problematiche riscontrate, cercando di renderla il piú possibile affidabile ed efficiente.

Struttura della tesi

La prima parte descrive la tecnologia utilizzata per lo sviluppo dell'applicazione ed i motivi della scelta (Capitolo 1). Segue un analisi delle caratteristiche che dovrá seguire il sistema, dei vincoli da rispettare e delle funzionalitá visibili dall'utente (Capitolo 2). Il capitolo sucessivo descrive l'implementazione del progetto focalizzando su due moduli, il primo relativo alla struttura dell'applicazione che dovra girare sul telefonino, il secondo riferito alla programmazione del lato server (Capitolo 3). Infine trovano spazio la presentazione dei risultati sperimentali ottenuti dai vari test effettuati (Capitolo 4). Concludono l'elaborato le considerazioni finali, insieme ad alcune indicazioni per futuri approfondimenti.

Capitolo 1

La tecnologia Java

1.1 Perché Java

Il linguaggio Java, inizialmente definito per applicazioni standard, sia client che server, é ormai largamente utilizzato anche per lo sviluppo di applicativi per dispositivi mobili, come telefoni cellulari e palmari. Questa tecnologia consente lo sviluppo e la realizzazione di applicazioni Java grazie alla programmazione di MIDLets: classi Java alla base della programmazione per dispositivi mobili, dai palmari ai telefoni cellulari di ultima generazione. Ed essendo il telefono cellulare un dispositivo che tutti portano sempre con se, é lo strumento perfetto per essere utilizzato come guida turistica. Semplicemente lanciando l'applicazione é possibile ricevere informazioni aggiornate sui punti d'interesse turistico che ci circondano.

1.2 La tecnologia J2ME

La *Java™ 2, Micro Edition* ($J2ME^{\text{TM}}$) é una piattaforma Java per dispositivi integrati come i telefoni cellulari ed i palmari. Tale piattaforma, proprio come la $J2EE^{\text{TM}}$ *EnterpriseEdition* e $J2SE^{\text{TM}}$ *Standard Edition*, é formata da un set di API [1] standard definite tramite il programma del

[1] Acronimo di Application Programming Interface, indica un insieme di procedure disponibili al programmatore.

Java Community Process e da un gruppo di esperti che comprende i maggiori produttori di dispositivi, di software e di servizi. La piattaforma J2ME fornisce i benefici della tecnologia Java adattata peró ai dispositivi integrati; includendo un'interfaccia utente flessibile, un'ampia gamma di protocolli di rete, un robusto modello di sicurezza ed un supporto per applicazioni standalone o di rete. Grazie alla J2ME le applicazioni sono scritte una sola volta per diversi tipi di dispositivi e scaricate dinamicamente, grazie alle capacitá native di ogni dispositivo. Si tratta, infatti, della piattaforma piú usata dai dispositivi *mobile*.

1.2.1 L'architettura

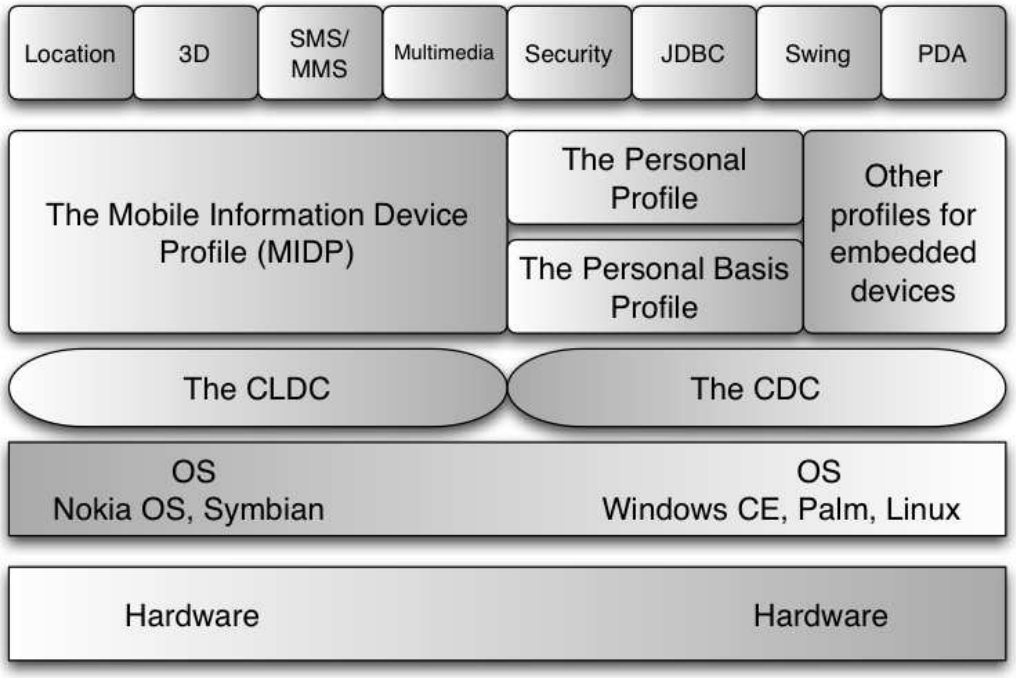

Figura 1.1: Architettura J2ME

L'architettura J2ME definisce configurazioni, profili e pacchetti opzionali, come riportato nella Figura 1.1

Ogni combinazione relativa ad una specifica categoria di dispositivi é ottimizzata secondo: capacitá di memoria, potenza del processore e capacitá d'interfacciamento. Il risultato é una comune piattaforma in grado di funzionare su ogni tipo di dispositivo.

1.2.2 Configurazioni

Le configurazioni sono composte da una *Virtual Machine* e da un set minimo di librerie. Esse forniscono le funzionalitá base per un particolare range di dispositivi che condividono caratteristiche simili, come quantitá di memoria e connettivitá alla rete. Le due configurazioni principali sono:

CLDC *(Connected Limited Device Configuration)*: é la piú "piccola" delle due configurazioni, progettata per dispositivi con connessioni di rete intermittente a banda ridotta, processori lenti e limitata capacitá di memoria. Caratteristiche tipiche:

- Processore a 16 o a 32 bit;
- 128kB di memoria;
- 32kB di memoria allocabile a *run-time*;
- Interfaccia utente ridotta.

CDC *(Connected Device Configuration)*: é progettata per dispositivi con maggior quantitá di memoria, processori piú veloci e una maggior larghezza di banda. Include inoltre una *Virtual Machine* completa e un piú ampio set di API. Caratteristiche tipiche:

- Processore a 32 bit;
- almeno 512kB di memoria;
- almeno 256kB di memoria allocabile a *run-time*;
- Interfaccia utente ad elevate prestazioni.

1.2.3 Profili

Al fine di fornire un ambiente run-time completo ed orientato alle specifiche caratteristiche dei dispositivi, le configurazioni devono essere combinate con un set di API di alto livello, o profili, che definiscono il ciclo di vita dell'applicazione, l'interfaccia utente e le specifiche di accesso al dispositivo.

Mobile Information Device Profile : il *Mobile Information Device Profile* (MIDP) é progettato per telefoni cellulari e palmari con ridotte capacitá. Offre le funzionalitá base per le applicazioni *Mobile* includendo la gestione dell'interfaccia utente, della connessione alla rete e del mantenimento dei dati. Combinato al CLDC fornisce un ambiente run-time completo minimizzando sia l'uso di memoria che il consumo di energia.

Personal Profile : il *Personal Profile* é un profilo appoggiato al CDC, mirato a tutti quei dispositivi che richiedono una completa e funzionale interfaccia grafica o che supportino le applet Java, come i palmari di ultima generazione. Include le librerie del *Java Abstract Window Toolkit* (AWT) facilitando l'esecuzione delle applicazioni Web-Based progettate per funzionare in ambienti desktop. PP sostituisce la tecnologia $PersonalJava^{TM}$ e fornisce alle applicazioni *PersonalJava* un percorso vantaggioso per l'espansione verso le piattaforme J2ME.

Personal Basis Profile : il *Personal Basis Profile* é un sottoinsieme del PP, fornisce un ambiente per tutti quei dispositivi in grado di supportare un livello basilare d'interfaccia grafica o richiedere l'uso di toolkit grafici per specifiche applicazioni. Sia PP che PBP sono mappati in cima alla configurazione CDC.

Foundation Profile : si appoggia allo strato CDC e fornisce funzionalitá alle applicazioni in base ai differenti tipi di dispositivo. Il *Foundation*

Profile é il piú basso livello per la configurazione CDC, fornisce un'implementazione del CDC adattata alle reti per tutti quei dispositivi non forniti d'interfaccia utente. Puó essere abbinato con il *Personal Basis Profile* ed il *Personal Profile* per i dispositivi che necessitano di un'interfaccia utente grafica (GUI).

1.2.4 Pacchetti opzionali

La piattaforma J2ME puó essere estesa mediante la combinazione di numerosi pacchetti opzionali ed i loro corrispondenti profili del CDC e del CLDC. I pacchetti opzionali offrono API standard facendo uso sia della tecnologia esistente che di quella in via di sviluppo *(Bluetooth, Web services, wireless messaging, multimedia)* e connessioni ai *database*. Grazie alla modularitá di questi pacchetti, i produttori di dispositivi possono includerli, a seconda dei bisogni, all'interno delle caratteristiche di ogni dispositivo.

1.3 CLDC

L'obiettivo delle specifiche CLDC é quello di standardizzare la piattaforma di sviluppo delle applicazioni, rendendola altamente portabile ed economica per tutti i dispositivi con scarse risorse. I dispositivi considerati nelle specifiche devono garantire le seguenti caratteristiche:

- Processore a 16 o a 32 bit;
- Almeno 192kB di memoria disponibile per la piattaforma Java;
- Basso consumo di energia;
- Connettivitá con qualche tipo di rete senza fili anche se discontinua e a banda ridotta.

Piú nello specifico, la piattaforma di sviluppo delle applicazioni deve:

- Mantenere un uso economico delle risorse. Essendo i dispositivi integrati, come i telefoni cellulari, prodotti in grande quantità, le case costruttrici, al fine di mantenere un margine di profitto cercano di mantenere il costo di produzione il più basso possibile. Ciò significa che la potenza di calcolo e la quantità di memoria dinamica non possono essere aumentate senza un sostanzioso incremento del prezzo. Per questo si definisce un minimo comune denominatore che include solo le caratteristiche minime della piattaforma Java e un set di API per un'ampia gamma di dispositivi;

- Focalizzare piú sulla programmazione delle applicazioni piuttosto che sulla programmazione del sistema. Le specifiche CLDC mirano ad inserire solo librerie di alto livello in grado di fornire un'elevata potenza di programmazione, ed enfatizzano l'importanza della generalizzazione e della portabilità;

- Abilitare il download dinamico delle applicazioni e incoraggiare lo sviluppo di applicazioni di terze parti. Contrariamente al passato, quando i piccoli dispositivi nascevano con una serie di caratteristiche mantenute nascoste dai produttori, ora le aziende stanno cercando soluzioni che permettano di scaricare contenuti interattivi da sviluppatori esterni. Con la nascita di dispositivi in grado di accedere ad Internet uno degli obiettivi del CLDC è diventato quello di definire un ambiente adattabile al download sicuro delle applicazioni passando attraverso diversi tipi di reti.

1.3.1 Requisiti

Le specifiche CLDC non impongono altre caratteristiche hardware se non quelle relative a capacità di memoria, delle quali definisce solo il limite minimo:

- Almeno 160kB di memoria non volatile, disponibile per la *Virtual Machine* e per le librerie CLDC;
- Almeno 32kB di memoria volatile per l'allocazione run-time.

Anche per quanto riguarda le caratteristiche software si definiscono solo dei requisiti minimi, tra i quali un Sistema Operativo minimale in grado di gestire le risorse hardware e il saper allocare una applicazione in via di esecuzione nella *Virtual Machine*.

1.3.2 Architettura ad alto livello

L'architettura ad alto livello tipica di un dispositivo J2ME è illustrata in Figura 1.2. Nel cuore dell'implementazione CLDC c'è la java virtual machine che, a parte qualche piccola differenza, nasce dal connubio tra le *Java Virtual Machine Specification* e le *Java Language Specification*. La macchina virtuale è generalmente eseguita in cima al *Host Operating System* che si trova fuori dagli obiettivi del CLDC. Sopra alla macchina virtuale stanno le librerie Java, definite dal CLDC ed estendibili dai vari profili del J2ME.

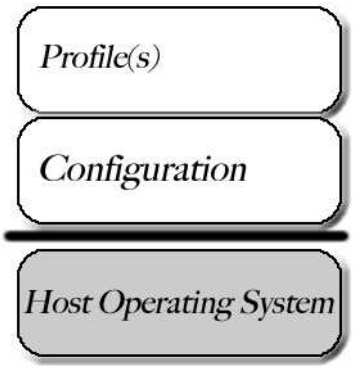

Figura 1.2: Architettura CLDC

1.3.3 Gestione delle applicazioni

Non tutti i dispositivi posseggono un *File System* o un altro tipo di meccanismo per il mantenimento delle informazioni scaricate dinamicamente. Per questo l'implementazione CLDC non richiede che le applicazioni siano mantenute in modo persistente nel dispositivo, é sufficente che queste vengano caricate ed immediatamente cancellate dopo la loro esecuzione. Assumendo peró che i dispositivi CLDC abbiano la possibilitá di stoccare i dati, al livello di gestione delle applicazioni sono demandate le seguenti operazioni:

- scaricare ed installare le applicazioni Java;
- ispezionare se esistono applicazioni Java salvate nel dispositivo;
- selezionare e lanciare una applicazione java;
- eliminare un'applicazione Java.

Dipendentemente dalle risorse disponibili nel dispositivo, il sistema puó permettere l'esecuzione concorrente di applicazioni, oppure permettere la singola esecuzione di un'applicazione alla volta. Una soluzione per eseguire applicazioni multitasking, qualora non sia pienamente supportato dal sistema, é quella d'istanziare piú *Virtual Machines* che eseguono ognuna un'applicazione.

1.3.4 Sicurezza

Il modello di sicurezza del CLDC é definito in tre livelli differenti:

Low-Level Security : conosciuto anche come *Virtual Machine Security*, assicura che l'esecuzione delle applicazioni nella macchina virtuale seguano la semantica della programmazione Java e che eventuali classi non permettano un crash del sistema o il suo danneggiamento.

Application-level Security : assicura che le applicazioni in esecuzione possano accedere solo a librerie, risorse di sistema e altri componenti a cui é permesso accedere.

End-to-end Security : si riferisce ad un modello che garantisce che ogni transazione iniziata su un dispositivo sia protetta per tutto il percorso dallo stesso fino all'entitá remota.

1.4 MIDP

Il*Mobile Information Device Profile* é un profilo *Java 2 Micro Edition* che si basa sulla configurazione CLDC ed é diretto allo sviluppo di applicazioni Java per dispositivi wireless. Consiste in un set APIs che fornisce un ambiente di sviluppo completo. La CLDC lascia irrisolti alcuni aspetti fondamentali per la realizzazione di applicazioni reali; le librerie MIDP si integrano ed estendono quelle della configurazione permettendo la realizzazione, la gestione e l'esecuzione degli applicativi Java. In particolare il MIDP si occupa di:

- Gestire il ciclo di vita delle applicazioni (caricamento - esecuzione - distruzione);
- Gestire l'interfaccia utente (dispositivi di input / output);
- Gestire il salvataggio persistente dei dati;
- Gestire il networking (implementazione dei protocolli).

1.4.1 Requisiti

La lista dei requisiti che segue specializza quelli giá esistenti e specificati dal CLDC.

Hardware Ogni applicazione scritta in MIDP Java ha bisogno dei seguenti requisiti minimi per poter funzionare correttamente:

- Dimensioni dello schermo: 96 x 54;
- Definizione dello schermo: 1 bit;
- Forma dei pixel: quadra 1:1;
- Dispositivo di input: tastiera (a una o due mani) o touch-screen;
- Memoria:
  - 128 kB di memoria non volatile per componenti MIDP;
  - 8 kB di memoria non volatile per applicazioni che utilizzano lo stoccaggio di dati in modo persistente;
  - 32 kB di memoria volatile per allocazioni run-time.
- Rete: a due vie, senza cavo, con connessione intermittente e a banda limitata.

Software Per tutti i dispositivi che supportano le caratteristiche *Hardware* menzionate c é ancora un elevato numero di caratteristiche *Software* da implementare:

- Un *kernel* minimale in grado di gestire l'*Hardware* e fornire la schedulazione di un'entitá che giri nella *virual machine*;
- Un meccanismo per leggere e scrivere dalla memoria non volatile, in grado di gestire i dati memorizzati in modo persistente;
- Un meccanismo in grado di accedere ad una rete *wireless*;
- Una capacitá minima di scrivere in un display mappato a bit;
- Un meccanismo in grado di gestire il ciclo di vita delle applicazioni.

1.4.2 Architettura

In Figura 1.3 é si puó vedere la struttura dell'architettura MIDP. Alla base troviamo, ovviamente, l'hardware del dispositivo host e il software nativo, cioé il sistema operativo e le librerie native. Al di sopra di questi due livelli si posa tutta l'architettura J2ME. La KVM (*Kilobyte Virtual Machine*) fornisce l'ambiente run-time per l'esecuzione delle applicazioni Java, mentre CLDC e MIDP forniscono il set completo di API Java di cui il programmatore potrá disporre. Come si puó osservare, oltre alle librerie Java CLDC e MIDP, possono essere disponibili librerie proprietarie specifiche per un dato dispositivo che, se da un lato aumentano potenzialitá e prestazioni, dall'altro ne pregiudicano la portabilitá.

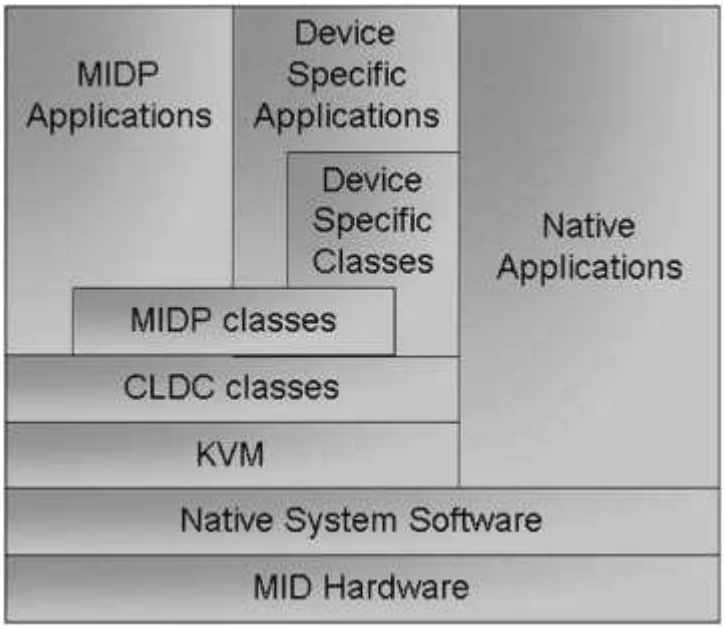

Figura 1.3: Architettura MIDP

1.5 MIDLet

Si tratta di un'applicazione scritta per la *Java 2 Micro Edition Mobile Information Device Profile*. Le applicazioni *MIDP* condividono la stessa memoria e le stesse funzioni con altre applicazioni del dispositivo.

1.5.1 Ciclo di vita

Una MIDLet non possiede un metodo main, ma deve estendere la classe javax.microedition.midlet.MIDlet ed implementare i suoi metodi astratti relativi allo stato in cui la MIDLet può trovarsi:

- `protected abstract void startApp()`
  segnala alla MIDLet che é stato posto nello stato di attivitá;
- `protected abstract void pauseApp()`
  segnala alla MIDLet il suo inserimento nello stato di pausa;
- `protected abstract void destroyApp(boolean unconditional)`
  impone alla *MIDLet* di terminare le proprio attivitá ed entrare nello strato destroyed.

Il cambiamento di stato di una *MIDLet* é controllato dall'*AMS* (*Application Management Software*), ossia un'applicazione software che fa parte dell'implementazione del *MIDP* stesso e svolge le funzioni di "gestore delle MIDLets" essendo responsabile della loro installazione, esecuzione e rimozione.

Piú precisamente l'*AMS*:

- Garantisce la possibilitá d'installare e disinstallare *MIDLet* dal dispositivo;
- Prepara l'ambiente d'esecuzione per la *MIDLet* (*KVM*, librerie *CLDC-MIDP*, librerie di terzi, file *JAD*, ecc);

- Cattura gli errori che si possono verificare sia in fase d'installazione della *MIDLet* che durante la sua esecuzione.

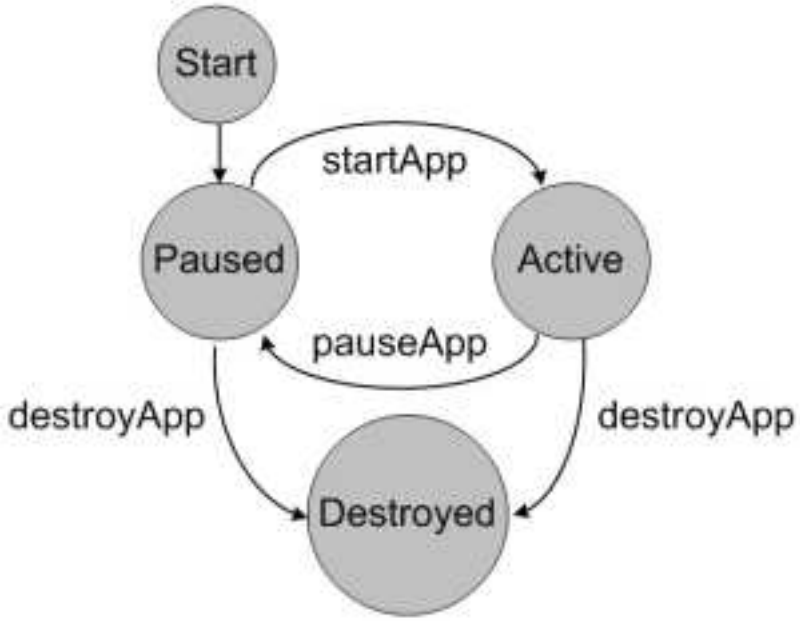

Figura 1.4: Ciclo di vita di una MIDLet

In Figura 1.4 é riportato il diagramma dei possibili stati di una MIDLet. Nel momento in cui é pronta per essere eseguita, l'*AMS* ne crea un'istanza e pone la MIDLet nello stato di pausa. In seguito con la chiamata del metodo `startApp()` la MIDLet passa allo stato di attivitá ricevendo tutte le risorse di cui necessita per il funzionamento. A questo punto possono verificarsi altre due condizioni: può essere distrutta oppure può entrare nello stato di pausa. Nel secondo caso rilascia temporaneamente le risorse di cui dispone e diventa inattiva.

Una nuova chiamata al metodo `startApp()` può riportare la MIDLet allo stato di attivitá.

La MIDLet stessa può decidere il proprio cambiamento di stato, informandone l'*AMS*, attraverso i metodi:

- `notifyPaused()`: informa che é entrato nello stato di pausa;
- `resumeRequest()`: richiede la possibilitá di rientrare in attivitá;
- `notifyDestroyed()`: informa che verrá distrutto.

1.5.2 MIDLet Suite

Occupiamoci ora del passaggio dalla realizzazione dalla MIDLet alla sua installazione ed esecuzione su di un dispositivo *MIDP*. La situazione é abbastanza differente dalle applicazione *Java Standard Edition*.

Tutti i file class della MIDLet devono essere inseriti in un file *JAR*, questo può contenere una o più applicazioni MIDLet il cui insieme prende il nome di MIDLet Suite. A questo punto il file *JAR* può essere caricato sul dispositivo in modi diversi a seconda dell'implementazione *MIDP* e del tipo di dispositivo, ad esempio via cavo, via rete, via porta infrarossi, ecc. Il file *JAR*, come detto, puó contenere piú MIDLet, al suo interno troveremo tutti i file class ad essi relativi, file di risorsa (file di testo, immagini), eventuali librerie di terze parti ed infine un *MANIFEST*. All'interno di una MIDLet Suite le singole MIDLet possono condividere file class e risorse, questo permette di ridurre le dimensioni totali del file *JAR*; cosa estremamente importante in caso di download dell'applicativo da reti con larghezza di banda ridotta. Il file MANIFEST presente all'interno del *JAR* descrive il contenuto del file *JAR* stesso e fornisce inoltre informazioni circa la *MIDLet Suite*.

Riportiamo qui sotto un esempio di file *MANIFEST* relativo ad una *MIDLet Suite* costituita da tre MIDLet: MIDLet1, MIDLet2, MIDlet3.

File MANIFEST:

```
Manifest-Version: 1.0
MIDlet-Version: 1.0
MIDlet-Vendor: My Vendor
MIDlet-2: MIDlet2, , midletsuiteexample.MIDlet2
```

```
MIDlet-3: MIDlet3, , midletsuiteexample.MIDlet3
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
MIDlet-1: MIDlet1, , midletsuiteexample.MIDlet1
MIDlet-Name: My MIDlet Suite
```

Oltre al file *JAR* una applicazione *MIDP* è costituita anche da un file *JAD* (*Java Application Descriptior*). A titolo di esempio è riportato il file *JAD* relativo alla *MIDLet Suite* di cui abbiamo appena visto il file *MANIFEST*.

File JAD

```
MIDlet-Name: My MIDlet Suite
MIDlet-Version: 1.0
MIDlet-Vendor: My Vendor
MIDlet-1: MIDlet1, , midletsuiteexample.MIDlet1
MIDlet-2: MIDlet2, , midletsuiteexample.MIDlet2
MIDlet-3: MIDlet3, , midletsuiteexample.MIDlet3
MIDlet-Jar-Url: MIDletSuiteExample.jar
MIDlet-Jar-Size: 240
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
```

Quelli riportati sono gli attributi predefiniti ed obbligatori di un file JAD, altri attributi predefiniti ed opzionali sono:

```
MIDlet-Description
MIDlet-Icon
MIDlet-Info-URL
MIDlet-Data-Size
```

Come si può notare i due file sono molto simili ma svolgono ruoli differenti. Il file *JAD* gioca un ruolo molto importante, soprattutto nel caso del download dell'applicazione da reti con larghezza di banda ridotta; il suo scopo é, infatti, quello di informare l'*Application Manager Software* delle caratteristiche richieste per l'installazione e l'esecuzione della *MIDlet Suite* prima ancora dell'effettivo download del file *JAR*. Sulla base degli attributi contenuti nel file *JAD*, l'*AMS* é in grado di valutare la possibilitá o meno

di procedere all'installazione dell'applicativo. Ad esempio l'*AMS* verifica se esiste spazio di memoria sufficiente per il file *JAR*, valuta la corrispondenza delle versioni di configurazione e profilo, si assicura che nel dispositivo non sia presente la stessa versione dell'applicativo che s'intende installare.

I concetti appena esposti sono rappresentati in Figura 1.5 dove é possibile osservare la struttura di una tipica applicazione *MIDP*.

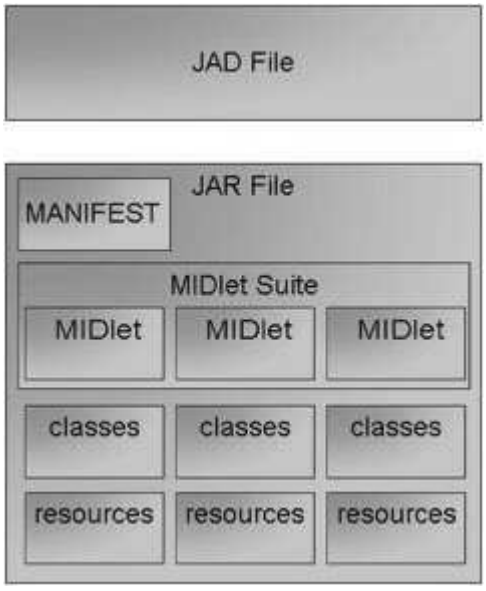

Figura 1.5: Applicazione MIDP

Oltre agli attributi predefiniti presenti nei file *MANIFEST* e *JAD* é possibile inserirne dei propri nel formato nome-valore. La possibilitá di definire attributi addizionali puó essere importante in alcuni contesti; in fase di runtime é infatti possibile leggere il valore di tali attributi utilizzando il metodo:
`String getAppProperty(String key)`
della classe MIDlet, che ritorna il valore dell'attributo passato come parametro. Tenendo conto che anche i file di risorsa (file di testo, immagini) sono accessibili mediante il metodo della classe:

`Class getResourceAsStream(String file)`

É evidente come le MIDLet possano auto-configurarsi a *run-time*.

Capitolo 2

Analisi delle specifiche di sistema

In questo capitolo verranno descritte le specifiche richieste dal sistema software, partendo dall'analisi delle attivitá che dovrá svolgere. Come si vede in Figura 2.1, l'utente, collegato al proprio gestore di telefonia mobile, si connette via GPRS o UMTS ai servizi internet e tramite l'applicazione, che rileverá la sua posizione utilizzando il dispositivo GPG integrato o collegato via Bluetooht, potrá visualizzare la posizione dei POI e come raggiungerli, oltre che ricevere automaticamente informazioni di carattere turistico.

2.1 Analisi del problema

2.1.1 Lato Client

Il sistema software che s'intende sviluppare deve fornire un servizio a tutti gli utilizzatori possessori di un telefono cellulare non troppo obsoleto. Tra le principali caratteristiche si piazza al primo posto l'elevata facilità d'utilizzo che svincoli l'utente dal settaggio di parametri (di cui può addirittura non conoscerne l'esistenza) e renda immediato l'utilizzo dell'applicazione. Un altro fattore di notevole importanza nasce dal fatto che il servizio possa essere fruibile in modo indipendente dal tipo di dispositivo posseduto; per questo

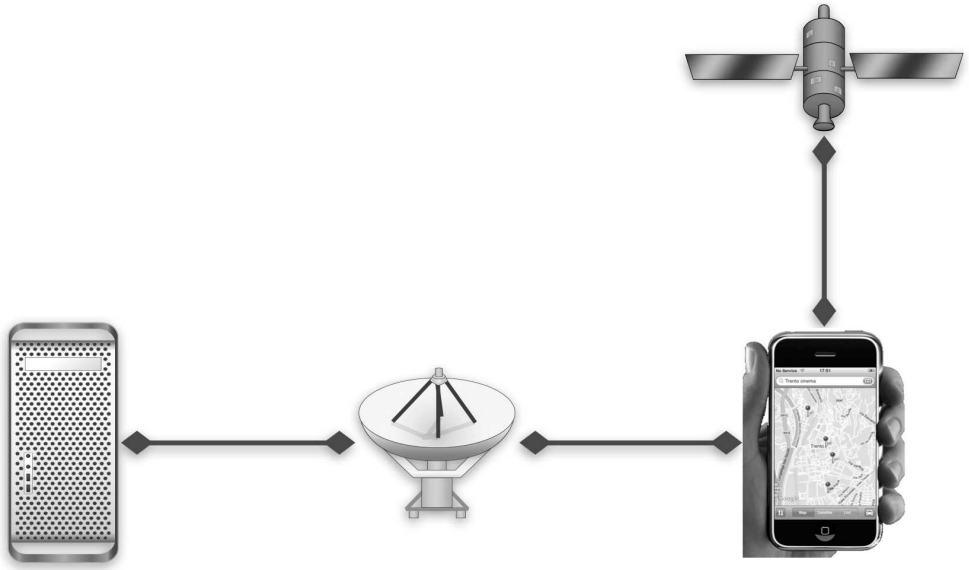

Figura 2.1: Schema dei vari componenti del sistema

durante la fase di progettazione risulterà necessario utilizzare componenti e moduli universali indipendenti quindi dalla piattaforma. Un altro aspetto da non trascurare è l'implementazione dell'interfaccia grafica: dovrà essere il più funzionale possibile permettendo all'utente di poter navigare all'interno dell'applicazione in modo veloce e dinamico.

2.1.2 Lato Server

Dal lato server deve esistere un'applicazione in grado di gestire le connessioni con i dispositivi mobili, il cui compito è quello di ricevere tutte le richieste ed elaborare la risposta, ricercando il percorso minimo per raggiungere il punto d'interesse specificato. Si tratta quindi di un server concorrente, in grado di gestire piú richieste contemporaneamente.

2.1.3 Scelta degli strumenti software

Per quanto riguarda la compatibilità, si è scelto l'utilizzo di applicazioni appoggiate sul profilo MIDP 2.0 in quanto, almeno teoricamente, garantiscono il funzionamento su tutti i dispositivi java-based.

2.2 Casi d'uso

Si presume che l'utente possegga già la versione client-side (MIDLet) del sistema, e che questa sia installata sul telefono cellulare. In Figura 2.2 sono mostrati i casi d'uso del sistema.

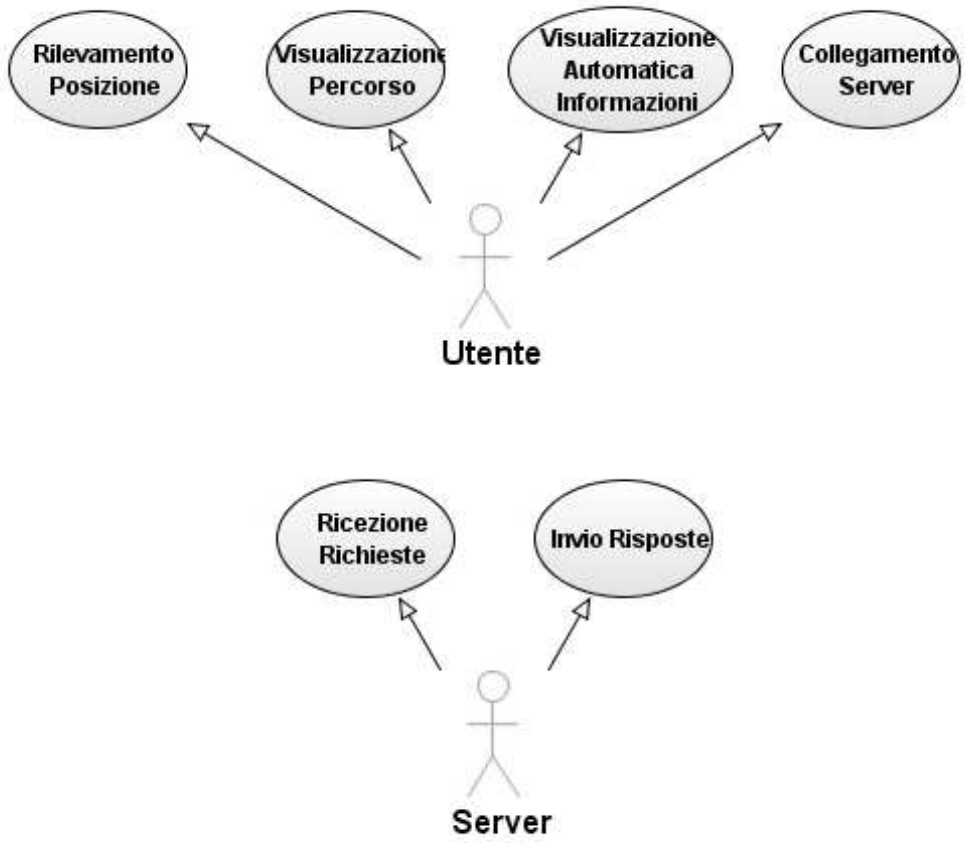

Figura 2.2: Casi d'uso del sistema

2.3 Scenari

Gli scenari si dividono in due categorie, una riferita all'attore utente e l'altra riferita all'attore server.

2.3.1 Scenari Utente

Nelle Tabelle 2.1, 2.2, 2.3, 2.4 sono riportati gli scenari relativi all'utente.

Rilevamento propria posizione	
Attore	Utente
Obiettivo	Rilevare la propria posizione tramite il dispositivo GPS
Frequenza	Alta
Precondizione	Avere un dispositivo GPS collegato
Postcondizione	Visualizzazione della propria posizione
Condizione di successo	La propria posizione viene visualizzata sulla mappa
Condizione di fallimento	La propria posizione non viene visualizzata

Tabella 2.1: Descrizione scenario *Rilevamento propria posizione*

Collegamento Server	
Attore	Utente
Obiettivo	Ottenere informazioni dal server
Frequenza	Media
Precondizione	Avere un dispositivo GPS collegato
Postcondizione	Ricezione risposta dal server
Condizione di successo	Connessione di rete e GPS disponibili
Condizione di fallimento	GPS o connessione di rete non disponibile

Tabella 2.2: Descrizione scenario *Collegamento Server*

Visualizzazione percorso	
Attore	Utente
Obiettivo	Visualizzare sul display il percorso per raggiungere i POI
Frequenza	Media
Precondizione	Aver selezionato il POI da raggiungere
Postcondizione	Visualizzazione del percorso
Condizione di successo	Sul display viene visualizzato il percorso per raggiungere il POI
Condizione di fallimento	Errore di connesione con il server

Tabella 2.3: Descrizione scenario *Visualizzazione percorso*

Visualizzazione Informazioni	
Attore	Utente
Obiettivo	Visualizzare automaticamente informazioni turistiche
Frequenza	Alta
Precondizione	Avere un dispositivo GPS collegato
Postcondizione	Visualizzazione delle informazioni turistiche
Condizione di successo	Vengo visualizzate le informazioni sul POI
Condizione di fallimento	Non vengono visualizzate le informazioni sul POI

Tabella 2.4: Descrizione scenario *Visualizzazione informazioni*

2.3.2 Scenari Server

Nelle Tabelle 2.5, 2.6 sono riportati gli scenari relativi al server.

Ricezione Richiesta	
Attore	Server
Obiettivo	Ricevere le richieste dai client
Frequenza	Alta
Precondizione	Server avviato e configurato
Postcondizione	Ricezione della richiesta
Condizione di successo	Viene ricevuta correttamente la richiesta del client
Condizione di fallimento	Errore di connessione

Tabella 2.5: Descrizione scenario *Ricezione richieste*

Invio Risposta	
Attore	Server
Obiettivo	Inviare la risposta al client
Frequenza	Alta
Precondizione	Aver ricevuto la richiesta dal client
Postcondizione	Invio della risposta al client
Condizione di successo	La richiesta viene elaborata e la risposta inviata al client
Condizione di fallimento	Errori di elaborazione o di connessione di rete

Tabella 2.6: Descrizione scenario *Invio risposta*

Capitolo 3

Progetto ed implementazione

In questo capitolo verranno descritte le tecniche utilizzate per ottenere le funzionalità indicate in fase di progetto. Si descriverà l'ambiente di sviluppo motivando le scelte intraprese per poi passare alla descrizione di ogni modulo del progetto, dal client al server, entrando più approfonditamente all'interno di ognuno di essi.

3.1 Ambiente di sviluppo

L'ambiente di sviluppo utilizzzato, sia per la programmazione lato client che per la programmazione lato server, é NetBeans; si tratta di un IDE (*Integrated Development Environment*) di sviluppo multilinguaggio scritto interamente in Java. Sono disponibili anche un gran numero di moduli per estendere le funzionalitá dell'IDE. NetBeans é organizzato in modo da fornire un facile accesso a tutti quei tools necessari allo sviluppo delle applicazioni ed é suddiviso in quattro workspaces:

Editing : Nel quale si organizzano i files del progetto ed il codice sorgente.

GUI-Editing : Dove si progettano e sviluppano le interfacce grafiche.

Debugging : Le cui funzioni comprendono il settaggio di breakpoints, l'a-

nalisi del codice step-by-step, l'esame delle variabili, l'esame delle classi e l'esecuzione di threads in sessioni multiple.

Running : Mostra tutti i programmi in esecuzione ed il loro output.

Il modulo per la creazione di applicazioni mobile é il Mobility Pack, che integrato a NetBeans mette a disposizione del programmatore:

- Un sistema integrato di compilazione, preverifica ed esecuzione di MIDLet e MIDLet Suite;
- Creazione automatica dei file .jad e .jar;
- Un sistema di debug per le MIDLet;
- Completamento automatico del codice;
- Vari templates per la creazione delle MIDLet e MIDLet Suite;
- Un emulatore di default.

3.2 Ambiente di simulazione

3.2.1 Java Server Development Kit

La tecnologia Servlet offre i seguenti vantaggi:

- Permette l'utilizzo di un qualsiasi componente Java;
- É portabile (una Servlet puó funzionare su tutte le piattaforme Server Java-compatibili);
- Puó soddisfare piú richieste contemporaneamente tramite l'utilizzo di *thread*;
- Permette di generare dinamicamente documenti HTML in maniera piú semplice dei programmi CGI;

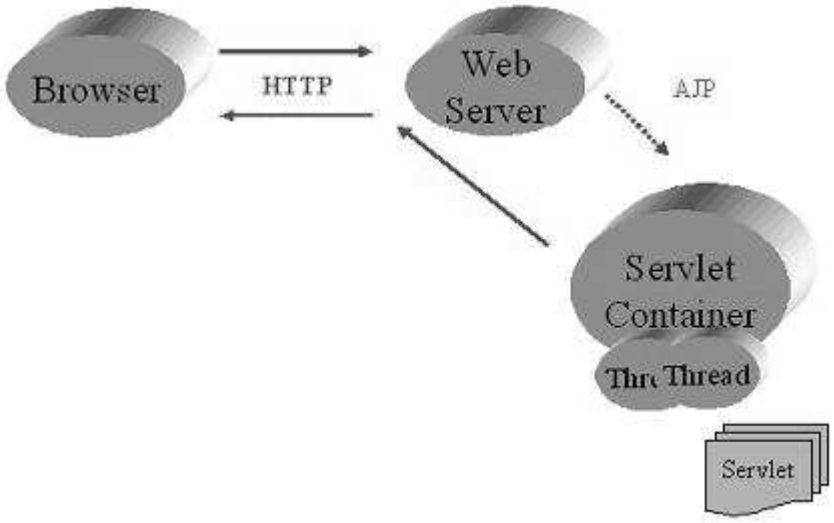

Figura 3.1: Schema di funzionamento di una Servlet

Il JSDK 2.1 é quindi un ambiente d'esecuzione per applicazioni servlet. É sempre in ascolto sulla porta 8080 e processa tutte le richieste HTTP, istanziando la relativa classe. Gestisce automaticamente la concorrenza eseguendo tante servlet, quante sono le richieste in coda.

3.3 Moduli del progetto

Come illustrato dalla Figura 3.2 il progetto puó essere suddiviso essenzialmete in due parti:

- La prima riguradante l'applicazione MIDLet appoggiata sul Wireless Toolkit e scomponibile in due ulteriori sotto-elementi di rilevante importanza:
  - Interfaccia grafica, che raccoglie le problematiche sulla visualizzazione dei vari componenti dell'applicazione;
  - Networking, che raccoglie le problematiche riguardanti la gestione della rete: invio richieste, ricezione risposte, ecc....

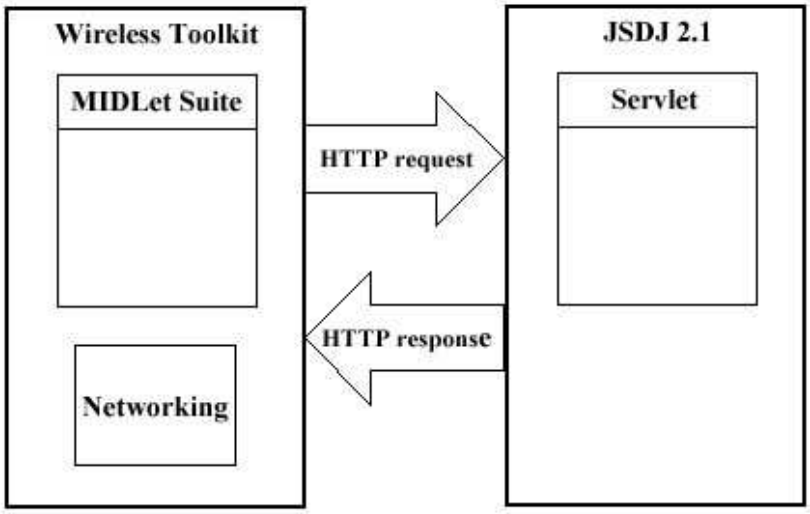

Figura 3.2: Moduli del progetto

- La seconda riguardante l'applicazione server-side appoggiata sul JSDK 2.1 in grado di gestire le richieste ricevute dalle MIDLets.

Tipico caso d'utilizzo del sistema: l'utente, nel momento in cui sceglie il POI da visitare, invia una *HTTP request* alla servlet, che dopo aver calcolato il percorso minimo per raggiungerlo utilizzando le linee degli autobus (partendo dalla posizione dell'utente rilevata tramite il dispositivo GPS), invia (mediante l'*HTTP response*) la risposta al client. Viene cosí visualizzato sul display la mappa, con indicate le varie fermate da percorrere con relativi cambi di linea. L'utente, inoltre, potrá visualizzare le istruzioni relative al percorso anche in formato testuale. Durante il percorso, inoltre, vengono visualizzate informazioni turistiche relative ad altri POI che si trovano nelle vicinanze.

3.3.1 MIDLet Suite

Si tratta di un set di classi di cui la principale è quella che implementa la superclasse javax.microedition.midlet.MIDlet (*MyBus*). Questa è l'unica classe istanziabile direttamente dalla *Virtual Machine* del dispositivo. Il suo compito è quello d'inizializzare le variabili necessarie all'esecuzione sul dispositivo e gestire, mediante l'implementazione dei rispettivi metodi, lo stato di esecuzione, di pausa e di distruzione della stessa. Fa uso delle classi accessorie necessarie alla visualizzazione sul display, alla gestione degli eventi (pulsanti premuti) e alla gestione della connessione. Un tipico caso di funzionamento dell'applicazione: viene instanziata la classe *MyBus*. La quale si occupa della visualizzazione del menu principale e d'istanziare a sua volta la classe *Business*, che si occuperá di rilevare periodicamente la posizione dell'utente e di visualizzare le informazioni inerenti ai punti d'interesse che si trovano nel raggio di cinquanta metri. Nel caso l'utente richieda il percorso per raggiungere uno specifico POI, viene istanziata la classe *ComunicatioPercorso* che si occupa d'inviare al server la richiesta dell'utente e di elaborare la risposta, al fine di visualizzare sulla mappa le informazioni ricevute.

Interfaccia utente : la MIDP User Interface è suddivisa in interfaccia ad alto-livello ed interfaccia a basso-livello.

Le API di basso-livello sono basate sull'uso della classe astratta *Canvas*. Permettono all'applicazione di avere maggior controllo sull'interfaccia grafica controllando: cosa c'è disegnato sul display e quali tasti vengono premuti sul tastierino numerico. É quindi una delle responsabilità del programmatore quella di assicurare la portabilità del codice su diversi dispositivi.

Le API di alto-livello come *Alert*, *Form*, *List* e *TextBox* sono estensioni della classe astratta Screen. Sono state progettate per fornire la massima astrazione rendendo ogni componente portabile su ogni piattaforma

MIDP prendendo cura di aspetti come: tipo di carattere, navigazione, scrolling, disegni, ecc....

Networking : le specifiche CLDC 1.0 definiscono un Generic Connection Framework. Questo framework definisce a sua volta una serie di API rivolte alle reti, ma nessuna implementazione di protocollo. Supporta due modalità di connessione:

- *Stream-based connection*;
- *Datagram*.

Comunque, affinchè tutto funzioni correttamente, ogni connessione deve essere basata sul protocollo HTTP 1.1 stream-based connection che garantisce la completa portabilità del codice. Naturalmente è implicito che l'utilizzo di sole connessioni HTTP possa limitare le funzionalità delle applicazioni, ad esempio non è possibile che il server possa stabilire di sua iniziativa una comunicazione con una MIDLet, se non rispondendo ad una request.

Questi sono i metodi supportati per inviare una *request*:

GET : Utilizzato per richiedere risorse statiche al server. Inoltrando quindi la stessa richiesta, si avrá come risposta la stessa risorsa.

HEAD : Ha le stesse funzioni di GET con l'eccezione che la *response* non include la risorsa richiesta, ma solo le informazioni relative alla stessa.

POST : Utilizzata per richiedere risorse dinamiche. La POST *request* contiene il corpo del messaggio con i dati richiesti al server.

3.3.2 Servlet

Si tratta di una classe residente nell'ambiente JSDK 2.1 costantemente in ascolto sulla porta 8080. All'atto della ricezione di una richiesta da parte

dell'utente, controlla il tipo di richiesta, genera dinamicamente la risposta e la invia al client.

3.4 Client

Verrá ora illustrato il diagramma delle classi principali e la descrizione di ognuna di esse.

3.4.1 Diagramma delle classi

Come illustrato dalla Figura 3.3 la classe principale, *MyBus*, é l'unica avente diritto d'istanziare le altre classi. Passiamo alla descrizione delle singole classi:

MyBus : Classe principale dell'applicazione, si occupa della gestione degli eventi e dell'instanziazione delle altre classi.

Business : Classe che controlla periodicamente la posizione dell'utente e nel caso si trovi nelle vicinanze di un POI ne visualizza le informazioni.

ComunicationPercorso : Classe che controlla l'accesso alla rete, invia le richieste alla servlet e ne riceve le risposte.

DatiGPS : Classe che occupa del collegamento con il dispositivo GPS e rileva la propria posizione.

3.4.2 OTA Provisioning

Esistono tre possibilità per scaricare l'applicazione all'interno del dispositivo mobile: tramite connessione diretta via cavo, tramite il protocollo OTA (Over the Air) Provisioning, oppure tramite una connessione bluetooth, ma questo implica avere sul proprio pc il file JAR dell'applicazione. Il metodo piú largamente utilizzato é il secondo: si tratta essenzialmente del download di una MIDlet Suite da un sito Internet attraverso una rete wireless. Grazie

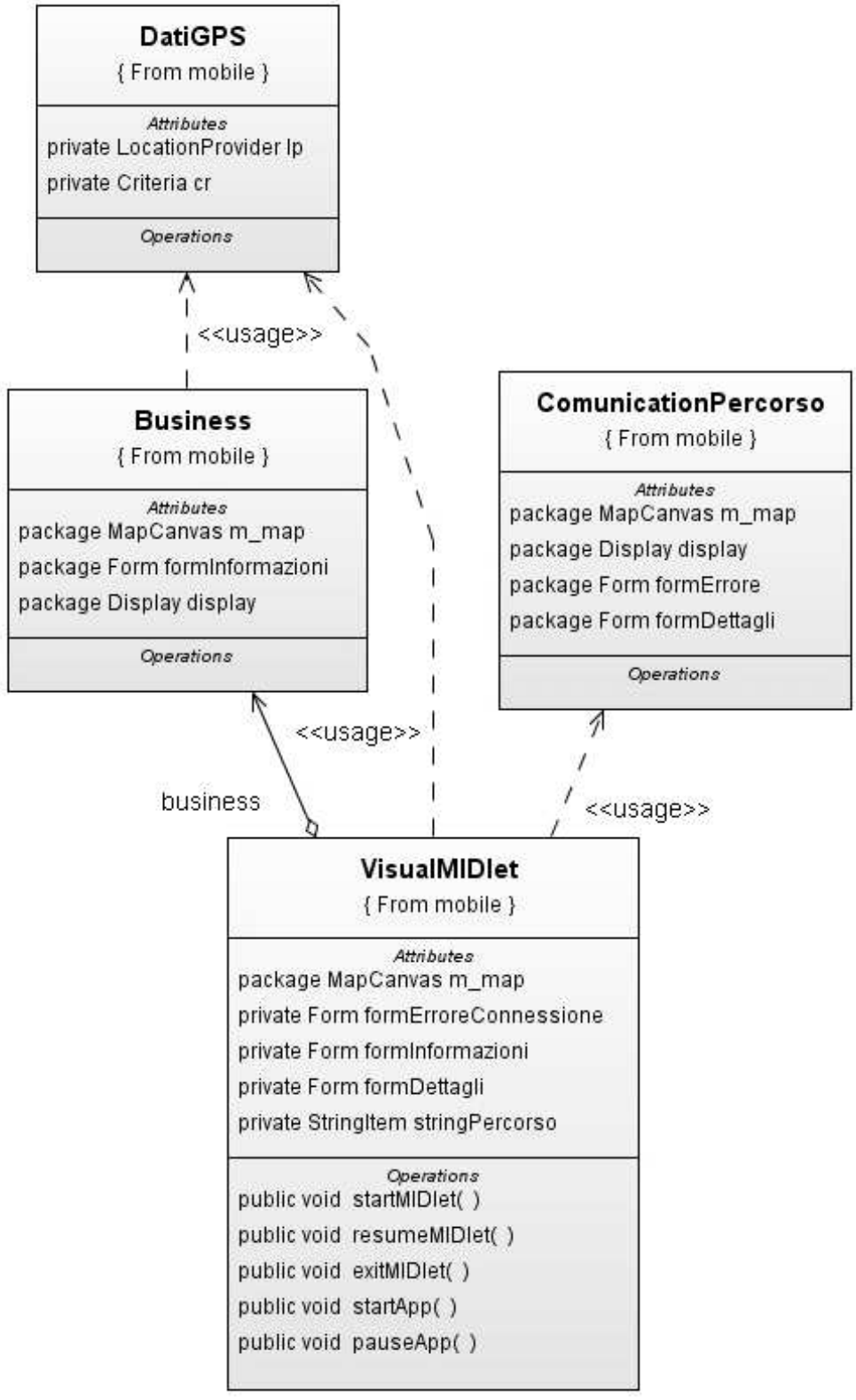

Figura 3.3: Diagramma delle classi della MIDLet: MyBus

ad OTA, non è più necessario che gli utenti si rechino nei centri assistenza o colleghino il cellulare al pc per scaricare o aggiornare il software del telefonino, in quanto possono accedere a tali tipi di servizi in ogni momento. Bisogna però fare una distinzione tra l'OTA Provisioning e il WAP OTA Provisioning in quanto, il primo supporta solo l'installazione di file JAR e JAD, mentre il secondo permette anche il download di moduli non java.

3.5 Server

Verrà ora illustrata la parte server-side dell'applicazione. Si tratta essenzialmente di una Servlet. Le Servlet sono moduli software scritti in Java che vengono eseguiti in applicazioni lato server per esaudire le richieste dei client. Esse non sono legate ad un particolare protocollo per la comunicazione tra client e server, anche se più comunemente si utilizza il protocollo HTTP ed infatti si parla di HTTP Servlet. Per scrivere una Servlet si fa uso delle classi del package javax.servlet che è il framework di base per la scrittura delle servlet e del package javax.servlet.http che è l'estensione del framework di base, per la comunicazione via HTTP. Utilizzando un linguaggio portabile come Java, le Servlet consentono di realizzare soluzioni per estendere le funzionalità dei server web, indipendenti dal sistema operativo su cui esse vengono eseguite.
Le Servlet vengono più comunemente utilizzate per:

- Elaborare e salvare dati provenienti da form HTML;
- Provvedere alla creazione di pagine HTML dinamiche, ad esempio utilizzando i dati contenuti in un database, in seguito ad una richiesta di un client;
- Gestire informazioni con stato, ad esempio la gestione di un sistema di commercio elettronico, dove un certo numero di clienti fa degli acquisti

contemporaneamente e dove occorre quindi mantenere in modo corretto le informazioni sui clienti e sui loro acquisti (il classico carrello della spesa);

3.5.1 Architettura e ciclo di vita di una Servlet

Una Servlet, nella sua forma più generale, è un'istanza della classe che implementa l'interfaccia:
`javax.servlet.Servlet`: `javax.servlet.GenericServlet` ma come si è già detto quella più utilizzata è quella che usa il protocollo HTTP e che si costruisce estendendo la classe
`javax.servlet.http.HttpServlet`. All'avvio, il server carica in memoria la classe Servlet ed eventualmente le classi utilizzate da questa, ne crea un'istanza chiamando il costruttore senza argomenti. Subito dopo viene chiamato il metodo `init(ServletConfig config)`. Qui avvengono le inizializzazioni delle variabili globali, procedura che viene fatta una sola volta durante l'intero ciclo di vita della Servlet, e viene caricato l'oggetto ServletConfig che potrà poi essere recuperato più tardi mediante il metodo `getServletConfig()`. Quest'ultima operazione viene eseguita nel metodo init della classe GenericServlet ed è per questo che in ogni classe che la estende, come lo è HttpServlet, all'inizio del metodo init c'è una chiamata al metodo della superclasse e cioè `super.init(config)`. L'oggetto ServletConfig contiene i parametri della Servlet e il riferimento a ServletContext che rappresenta il contesto in cui gira la Servlet. Una volta inizializzata la Servlet, viene chiamato il metodo `service(ServletRequest req, ServletResponse res)` per ogni richiesta che arriva dai client; questo metodo viene chiamato in modo concorrente ovvero più Thread possono invocarlo nello stesso momento. In casi particolari come quando si lavora con risorse non condivisibili, è possibile implementare Servlet non concorrenti. Quando una Servlet deve essere arrestata, ad esempio se deve essere aggiornata o bisogna riavviare il server, viene chiamato il

metodo `destroy()` nel quale vengono rilasciate le risorse allocate nel metodo init. Anche questo metodo viene chiamato una sola volta durante il ciclo di vita della Servlet.

3.5.2 MyBusServer

La descrizione della classe é suddivisa in tre parti, eseguite in modo sequenziale.

Lettura della request : Viene aperto uno stream d'ingresso e viene letto il contenuto della request. Viene creato un buffer che conterrá i caratteri letti, alla fine della lettura si procede con la chiusura dello stream.

Elaborazione response : Terminata la lettura della request si passa alla generazione della response: utilizzando l'algoritmo di Dijkstra viene generato il percorso minimo tra la fermata piú vicina all'utente e la fermata piú vicina al POI da visitare, tenendo conto anche di relativi cambi di linea.

Invio response : Vengono settati i parametri relativi all'header della response, indicando il tipo di contenuto e la sua dimensione.
Dopo l'apertura dello stream di output viene inviato il contenuto e viene chiuso lo stream.

Capitolo 4

Risultati sperimentali

4.1 Il Sistema

Il sistema sul quale é stata sviluppata la Tesi é composto da un PC il cui hardware é costituito da un AMD Athlon XP 2000 a 1.67 GHz con 512 MB di RAM, SO Windows XP Professional.

4.2 Il sistema di emulazione

Al fine di testare l'applicazione sviluppata é stato utilizzato l'emulatore di default, fornito da NetBeans. Questo ha permesso di testare la parte relativa alla connessione con la servlet, dopo averne effettuato il *deploy* sul server TomCat installato sul PC, la corretta visualizzazione della mappa e la navigazione della stessa.
L'emulatore, peró, non permette di testare la parte relativa alla comunicazione con il dispositivo GPS e, di conseguenza, verificare il funzionamento del *pushing* di informazioni in *real-time*, si é quindi passati all'installazione della MIDLet su un telefono cellulare modello Nokia e61 ed al testing dell'applicazione per le vie della cittá di Trento. Nelle Figure dalla 4.1 alla 4.4 sono riportati alcuni screenshot dell'esecuzione dell'applicazione sull'emulatore fornito da NetBeans.

Figura 4.1: Menu principale

Figura 4.2: Visualizzazione della mappa con l'indicazione dei POI

Figura 4.3: Visualizzazione del percorso per raggiungere il POI con gli autobus

Figura 4.4: Visualizzazione dei dettagli dei POI

4.3 Risultati

Alla fine della fase di testing si avuti buoni risultati. Il sistema, grazie alla rete UMTS, si collega velocemente al server e visualizza rapidamente il percorso sulla mappa. Avvicinandosi ai POI, quando ci si trova ad una distanza inferiore ai cinquanta metri, vengono visualizzate automaticamente le informazioni ad essi associate.

Conclusione

Nel corso della presente Tesi é stato sviluppato un software che consente di ricevere automaticamente informazioni turistiche georeferenziate direttamente sul proprio telefono cellulare, visualizzare la posizione di eventuali POI ed il percorso per raggiungerli utilizzando i mezzi pubblici.
Il software sviluppato, suddiviso in due parti: server e client, é indipendente dalla piattaforma e puó essere installato sul proprio dispositivo con estrema faciliá.
Per quanto riguarda la comunicazione con il server, grazie alla velocitá della rete UMTS, si sono riscontrati risultati soddisfacenti, come anche per la navigazione della mappa.
Il sistema presenta delle limitazioni:

- Le informazioni relative ai punti d'interesse sono inserite nell'applicazione stessa, questo impone la reinstallazione della MIDLet in caso di aggiornamenti (inserimento di nuovi POI o cancellazione degli stessi);

- Il grafo delle linee dei trasporti pubblici, utilizzato per la ricerca del percorso minimo tra la posizione dell'utente e la destinazione scelta, non é generato dinamicamente tenendo conto dei cambi di linea in base all'orario ma si tratta di un grafo statico che come vertici ha le fermate ordinarie delle varie linee;
  Da qui la nascitá di numerose idee per sviluppi futuri tra le quali:

- Invio delle informazioni sui POI alla MIDLet in base alla base alla posizione dell'utente rilevata dal dispositivo GPS;
- Generare dinamicamente il grafo delle linee degli autobus, tenendo conto degli orari e delle festivitá;

Bibliografia

[1] SUN Microsystem, *J2ME Documentation*,
`http://java.sun.com/javame/reference/apis.jsp`.

[2] SUN Microsystem, *Java 2 Platform, Micro Edition (J2ME)*,
`http://java.sun.com/javame/index.jsp`.

[3] SUN Microsystem, *Wireless Java Programming with J2ME*,
`http://java.sun.com/developer/Books/J2MEwireless`.

[4] SUN Microsystem, *The Java platform for mobile information devices.*,
`http://java.sun.com/products/midp/midp-ds.pdf`.

[5] SUN Microsystem, *J2ME Building Block for Mobile Devices*,
`http://java.sun.com/products/cldc/wp/KVMwp.pdf`.

[6] SUN Microsystem, *Java Servlet Technology*,
`http://java.sun.com/products/servlet/`.

[7] J2MEMap,
`http://j2memap.landspurg.net/`.

[8] Bruce Eckel, "*Thinking In JAVA,*" 2002.