Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Realizzazione di un sistema software per la localizzazione di dispositivi GPS e la gestione di contenuti context aware in un sistema di Grid Computing

Giovanni Aiello, Giuseppe Lo Re, Giampiero Rizzo,
Pietro Storniolo, Alfonso Urso

Rapporto Tecnico N.:12
RT-ICAR-PA-04-12 **ottobre 2004**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sezione di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Realizzazione di un sistema software per la localizzazione di dispositivi GPSe la gestione di contenuti context aware in un sistema di Grid Computing

Giovanni Aiello[2], Giuseppe Lo Re[1] ,Giampiero Rizzo[1],
Pietro Storniolo[1], Alfonso Urso[1]
e-mail: aiello.giovi@virgilio.it, lore@pa.icar.cnr.it,
storniolo@pa.icar.cnr.it, grizzo@pa.icar.cnr.it, urso@pa.icar.cnr.it,

Rapporto Tecnico N.:12 ***Data:***
RT-ICAR-PA-04-12 **ottobre 2004**

---

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sezione di Palermo Viale delle Scienze edificio 11 90128 Palermo
[2] Università degli Studi di Palermo Dipartimento di Ingegneria Informatica Viale delle Scienze 90128 Palermo

Sommario

Nel presente documento viene proposto un sistema software, sviluppato in Java, in grado di localizzare, utilizzando la tecnologia GPS, un utente sulla superficie terrestre e di gestire contenuti *context aware*. Il context awaring è una branca di ricerca in fase di sviluppo che consiste nel fornire determinati servizi ad un utente, in base al contesto di fruizione del servizio.

Il sistema, creato per l'utilizzo in un sito archeologico, si pone come obbiettivo quello di accompagnare il visitatore e di fornigli a richiesta tutte le informazioni disponibili dei *Punti di Interesse* in cui è prossimo. Sul computer palmare connesso in rete un'applicazione del sistema interroga un'antenna GPS bluetooth dalla quale ottiene la posizione geografica espressa in latitudine e longitudine. Una volta ricevuti, i valori della posizione per mezzo di pacchetti UDP sono inviati ad un'applicazione intermedia che risieede sul server. Quest'applicazione intermedia elabora le informazioni sulla posizione e restituisce alla parte server del sistema un file contenente la *zona* ove è localizzato l'utente. Nel server, una servlet spedisce pagine HTML contenenti informazioni relative alla *zona* su cui insiste il visitatore.

Capitolo 1

Introduzione al Global Positioning System (GPS)

Grazie ad una più spinta miniaturizzazione delle periferiche elettroniche ed allo sviluppo di tecnologie wireless il mercato è fortemente stimolato all'utilizzo di dispositivi ed i relativi servizi orientati alla mobilità, cioè che permettano di svolgere una vasta gamma di attività senza particolari vincoli strutturali. Un esempio sono i computer palmari detti PDA (acronimo di Personal Data Assistants), dispositivi dalle dimensioni sempre più ridotte e con sempre più elevate prestazioni e funzionalità. Naturalmente le performance dei PDA non sono ancora equivalenti a quelle dei personal computer, ma la loro peculiarità li rende unici nel loro utilizzo, un esempio è il Palmare utilizzato nella fase di test che si integra con una periferica wireless quale un ricevitore *GPS (Global Positioning System)*con tecnologia Bluetooth.

Il *Global Positioning System*, come descritto in [1], è un sistema di radiolocalizzazione e posizionamento sviluppato dal Dipartimento della difesa degli Stati Uniti che permette di determinare la posizione con un grado di precisione che va dai 10 ai 20 metri. Il concetto di autonomia ha avuto la sua importanza nello sviluppo di un sistema che permettesse al soldato di conoscere la sua posizione senza l'utilizzo di altre comunicazioni radio intercettabili dai potenziali nemici. La necessità che il sistema funzionasse a livello *globale* e per 24 ore al giorno ed il basso costo hanno consentito l'utilizzo di questa tecnologia tanto da sostituirne altre meno recenti.

Il sistema GPS è utilizzato da molti anni nella navigazione aerea, marittima e terrestre; esso è basato su una costellazione di 24 satelliti orbitanti su sei piani precisi ad un'altezza di circa 20.200 km con un periodo orbitante di 12 ore. La loro posizione perciò è la stessa ogni giorno allo stesso tempo siderale con un anticipo di quattro minuti ogni giorno. I satelliti emettono segnali a microonde sulla frequenza di $1.575,42\ MHz$ in direzione del nostro pianeta, che possono essere ricevuti ed elaborati *gratuitamente* dai ricevitori GPS.

Il sistema di localizzazione proposto in questo documento utilizza la capacità che ha un utente ad ottenere dai ricevitori GPS la posizione geografica, quindi potersi dirigere verso determinati *Punti di Interesse* ed usufruire di contenuti messi a disposizione dai servizi disponibili.

Capitolo 2

Panoramica sul Global Positioning System

Il sistema GPS (Global Positioning System), noto anche come NAVSTAR (Navigation System with Time And Ranging), è un sistema satellitare basato su una costellazione di 24 satelliti (figura 2.1) orbitanti ad una quota di circa 20.200 Km. Esso rappresenta un sistema di navigazione globale, continuo e tridimensionale e quindi è in grado di fornire, con estrema precisione, le coordinate geografiche, la quota e la velocità di qualsiasi mezzo mobile in ogni punto della Terra e per l'intero arco delle ventiquattro ore. Pur essendo stato concepito per scopi militari, i progettisti del sistema fecero in modo che anche i civili potessero utilizzarlo anche se con una precisione minore. I primi 11 satelliti di tipo sperimentale furono lanciati dal 1978 al 1985 e furono sostituiti successivamente da quelli operativi, a partire dal 1989. A seguito del completamento della prevista costellazione di 24 satelliti, l'8 dicembre 1993 fu fatta una prima dichiarazione di inizio operatività del sistema. Al completamento di tutti i test per la verifica, in particolare, delle specifiche militari del sistema, esso fu definitivamente dichiarato operativo il 27 aprile 1995. Le ricerche basate sul Global Positioning System hanno goduto di numerosi finanziamenti, infatti, oltre agli utilizzi militari di questa nuova tecnologia, ben presto il GPS divenne il sistema di riferimento per misurazioni di posizione e tempo oltre che in ambito scientifico anche in svariate applicazioni civili.

Il GPS:

- è utilizzato quale sistema base per il controllo della navigazione aerea ed anche di quella marittima nelle aree di maggior traffico;
- consente il controllo della posizione di automezzi, che viene comunicata via radio ad una centrale di controllo. Tale sistema è utilizzato da alcune società di trasporti come strumento antifurto per i propri automezzi;
- è utilizzato dai servizi di assistenza e di protezione civile per una migliore gestione ed utilizzo dei mezzi di soccorso. E' prevedibile che esso potrà essere utilizzato anche nel controllo del traffico ferroviario.
- Considerate le caratteristiche strutturali degli odierni ricevitori GPS e dei computer palmari è possibile utilizzare il sistema GPS *comodamente* come

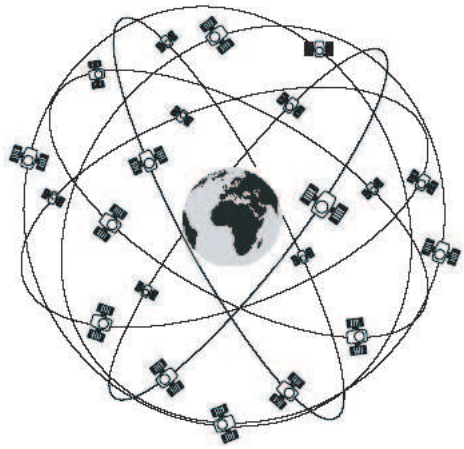

Figura 2.1: *Costellazione dei satelliti*

sistema di navigazione e guida di un utente verso una meta prestabilita o verso un determinato POI (Point of Interest).

Le applicazioni elencate sono soltanto alcune di quelle in uso, molte altre sono in fase di studio per cui il sistema GPS avrà sempre maggiore rilevanza in ambito civile.

Come descritto in [1], il sistema è diviso in tre segmenti denominati: 1) Segmento spaziale, che è formato da una costellazione nominale di 24 satelliti, che trasmettono dei codici di distanza a radio frequenza e dei dati di navigazione. 2) Segmento di controllo, che consiste in una rete di monitoraggio e di mezzi di controllo, per il mantenimento della costellazione e l'aggiornamento dei messaggi di navigazione dei satelliti. 3) Segmento dell'utente, che consiste in una varietà di ricevitori, di decodificatori e di elaboratori di segnali GPS. I tre segmenti sono illustrati in figura 2.2. Il segmento spaziale è progettato per avere sempre visibili in ogni parte della terra almeno quattro satelliti con un angolo d'elevazione di 15° rispetto all'orizzonte.A bordo di ciascuno satellite è funzionante un orologio atomico la cui frequenza (10.23 Mhz) viene utilizzata per generare i segnali emessi. I satelliti trasmettono due onde portanti nella banda L e sono la L1 (1575,42 Mhz) e la L2 (1227,60 Mhz) esse sono modulate da dei codici (C/A 1,023 Mhz e P 10,23 Mhz per L1 e P 10,23 Mhz per L1) che derivano dalla frequenza fondamentale dell'orologio atomico. I ricevitori GPS utilizzano tali codici per determinare il satellite che si sta tracciando. Il segmento di controllo è basato su 5 stazioni di monitoraggio sparse su tutta la Terra:

- Colorado Spring (Stati Uniti);
- Isole Hawaii (Oceano Pacifico);
- Isola di Ascension (Oceano Atlantico);
- Isola Diego Garcia (Oceano Indiano);
- Isola Kwajalein (Oceano Pacifico).

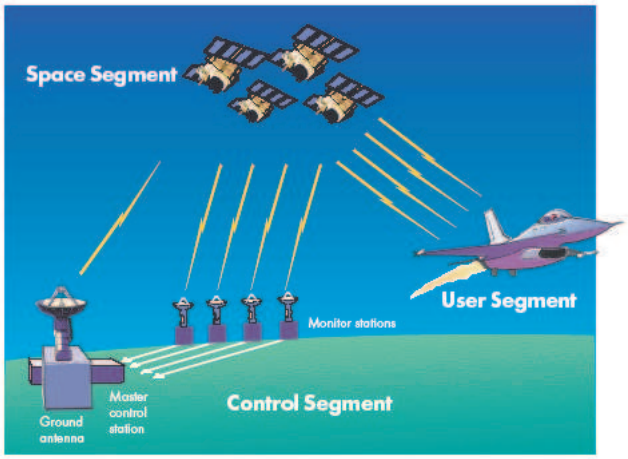

Figura 2.2: *Segmenti del GPS*

Queste stazioni ricevono i segnali dai satelliti, li elaborano e trasmettono ai satelliti stessi gli aggiornamenti delle orbite da seguire.

La prima, a Colorado Spring, è quella master alla quale vengono convogliati tutti i dati delle altre.

Per quanto riguarda la precisione del segnale trasmesso, questa dipende da alcune variabili tra le quali:

1. Tempo impiegato nella misura;
2. Tipo di ricevitori utilizzati;
3. Algoritmo di correzione applicato alle misure.

Sostanzialmente il grado di precisione del sistema GPS può essere così riassunto:

1. Da 30 a 100 metri per qualunque ricevitore utilizzato in modo autonomo;
2. Da 1 a 5 metri per ricevitori in modalità differenziale DGPS;
3. Precisione < 1 cm per i sistemi più sofisticati quali la posizione differenziale di fase.

2.1 Principio di funzionamento del GPS

Un elemento importante per la determinazione della distanza tra un satellite ed il ricevitore è il *tempo*. Sia un ricevitore GPS posizionato in un qualunque punto della Terra, tutti i satelliti trasmettono dei segnali opportunamente codificati che contengono svariate informazioni quali i dati orbitali per il calcolo della posizione del satellite ed un'informazione *temporale* per la determinazione degli istanti di partenza dei segnali stessi. E' essenziale quindi l'estrema precisione degli orologi a bordo dei satelliti poichè un errore ad esempio di 1 μsec equivale a

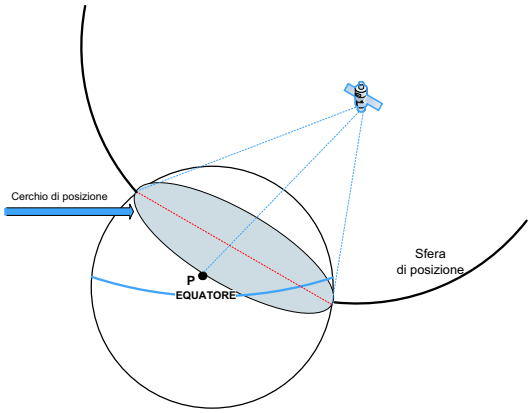

Figura 2.3: *Rappresentazione della sfera di posizione e del cerchio di posizione*

un errore di 30 m in coordinate spaziali. Per ridurre il più possibile l'errore, nei satelliti sono installati orologi atomici, costantemente controllati dalla centrale di controllo situato negli USA a Colorado Spring.

I ricevitori terrestri, mediante il proprio orologio interno, calcolano la differenza tra l'istante di invio del segnale e l'istante di arrivo del segnale stesso; poi moltiplicano tale differenza per la velocità di propagazione delle onde elettromagnetiche ottenendo così la distanza tra il satellite ed il ricevitore.

In tal modo è possibile identificare nel sistema una sfera di centro pari alla posizione del satellite e di raggio pari alla distanza tra il satellite ed il ricevitore. Tale sfera intersecherà la superficie terrestre formando una circonferenza che è il luogo dei punti in cui si trova l'utilizzatore terrestre. In figura 2.3 è illustrata tale situazione supponendo che il ricevitore terrestre si trovi in un punto P della superficie terrestre. Dal sistema Global Positioning System è possibile ottenere informazioni bidimensionali o tridimensionali a seconda che dal ricevitore siano visibili tre o a quattro satelliti.

Nel caso in cui dal ricevitore GPS siano visibili contemporaneamente tre satelliti, verranno individuati tre cerchi di posizione che a loro volta identificheranno un triangolo nella superficie terrestre rappresentato dal volume ottenuto dall'intersezione dei tre cerchi di posizione, mediante una formula di triangolazione, è possibile calcolare le due incognite: *latitudine* e *longitudine* della posizione del ricevitore GPS; tale scenario è illustrato in figura 2.4. Nel caso in cui dal ricevitore GPS siano visibili almeno quattro satelliti, è possibile determinare una terza incognita: *l'altitudine*. In queste condizioni si dice che è possibile determinare le coordinate tridimensionali della posizione del ricevitore GPS.

2.2 Standard NMEA 0183

Lo Standard NMEA (National Marine Electronics Association) 0183, come descritto in [2], è un protocollo di comunicazione tra dispositivi digitali, e la sua prima release risale al Marzo 1983. Lo Standard NMEA 0183 contempla i segnali elettrici, il protocollo e la temporizzazione della trasmissione mediante frasi,

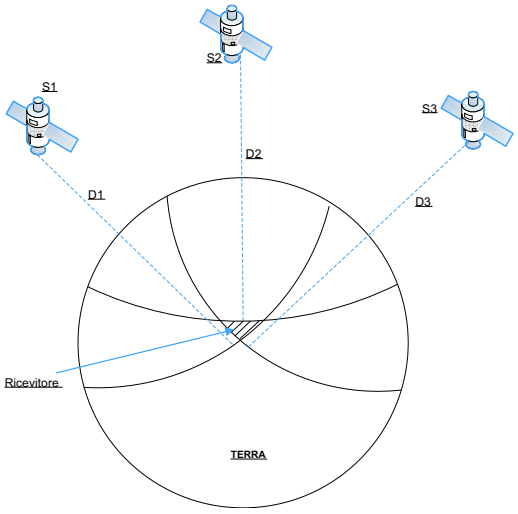

Figura 2.4: *Determinazione posizione bidimensionale (latitudine - longitudine)*

dette *sentences*, trasmesse attraverso BUS di dati seriali (ad esempio $RS232$) a velocità di trasmissione di 4800 baud.

I settaggi NMEA per RS232 sono illustrati nella tabella 2.1.

BaudRate	4800
Data bits	8 (Bit 7 settato a 0)
Stop bits	1 o 2
Parity	None
Handshake	none

Tabella 2.1: Settaggi NMEA per RS232

Le informazioni NMEA sono trasmesse da un dispositivo chiamato *talker* ad un dispositivo chiamato *listener* mediante *frasi* formate da un massimo di 80 caratteri. Qualche pubblicazione riporta tuttavia che la frase di prefisso $GPGSV$ può avere una lunghezza massima di 210 caratteri; infatti le informazioni di tipo $GPGSV$, quando più lunghe di 80 caratteri, sono suddivise in diverse frasi $GPGSV$, tutte di lunghezza massima di 80 caratteri.

Ogni frase o *sentence* NMEA inizia e finisce con $[CR][LF]$.

Il prototipo di una frase NMEA è del tipo:

$$\$Prefisso, campo[1], campo[2], ..., campo[n] * checksum[CR][LF].$$

Una frase NMEA riporta come primi 6 caratteri il *prefisso* , i restanti caratteri dei campi separati da virgole, rappresentano specifiche informazioni: tempo, latitudine, longitudine, numero del satellite che invia il messaggio, velocità, ecc. Alla fine di ogni sentence è presente un campo di checksum per ovviare agli errori durante la trasmissione del messaggio. Il campo checksum è preceduto da un * ed è una combinazione XOR (in notazione esadecimale) di tutti i caratteri tra $ e *.

Tutte le *sentences* NMEA iniziano con un prefisso costituito da un carattere $, seguito da due caratteri che identificano l'entità che le genera(talker). Le *frasi* che iniziano con $GP, indicano che sono generate da un dispositivo GPS. I successivi tre caratteri del prefisso della frase indicano il tipo di frase. Per esempio una frase che inizia con $GPGLL è inviato da un dispositivo di tipo GPS (GP) ed è del tipo *Geographic Position Longitude and Latitude (GLL)*.

I principali prefissi GPS NMEA sono illustrati nella tabella 2.2.

GPRMC	Recommended Minimum specific GPS/TRANSIT data.
GPRMB	Recommended Minimum navigation info (waypoint based navigation active).
GPGGA	Global Positioning System fix data.
GPGSA	GPS DOP (Dilution of Precision) and Satellites Active.
GPGLL	Geographic Position Latitude/Longitude.
GPGSV	Satellites in View.

Tabella 2.2: Prefissi NMEA GPS

Dopo il prefisso, sono presenti i vari campi separati, da virgole. Poichè tra le varie versioni le lunghezze dei campi possono essere variabili ed essere o meno presenti, per il sistema proposto si è deciso di implementare, per ottenere i valori della posizione geografica, una decodifica che tenesse conto, per la separazione dei campi, delle virgole, in modo da svincolarsi dalla variabilità della lunghezza delle singole *sentences*. Inoltre, alcuni dispositivi GPS includono gli zeri davanti ai numeri, altri no; poichè per le prove sperimentali si è utilizzato un dispositivo GPS TOMTOM Bluetooth, è stato necessario introdurre gli zeri negli spazi mancanti davanti ai numeri; modificando opportunamente quest'ultima caratteristica, è possibile rendere compatibile il sistema con altre marche di dispositivi GPS, ad esempio *Holux*.

I formati dei dati usati dalle frasi NMEA GPS sono illustrati nella tabella 2.3.

hhmmss.ss	Ore, minuti, secondi e centesimi di secondo (es: 132957.94 = 13:29:57.94)
ddmmyy	Campo data giorno, mese, anno (es: 151005 = 15 ottobre 2005)
A	Campo numerico di lunghezza fissa di un carattere
A-A	Campo alfanumerico di lunghezza variabile
llll.ll	Campo latitudine (es: 4531.47 = 45°31.47')
yyyyy.yy	Campo longitudine (es:00917.21 = 009°17.21')
x	Campo numerico con n cifre intere
x.x	Campo numerico con n cifre intere e n cifre decimali (es: 123.45)
n	Campo numerico, singola cifra
nn	Campo numerico, due cifre
nnnn	Campo numerico, quattro cifre

Tabella 2.3: Campi NMEA

Riferimenti a UTC o GMT si intendono relativi all'ora universale.

Capitolo 3

Sistema proposto

Negli ultimi anni si sta assistendo a una rapida evoluzione degli strumenti di comunicazione, si è avuta una crescita esponenziale nella diffusione dei dispositivi mobili, dovuta anche al continuo sviluppo delle reti di comunicazione. La diversità tra i vari terminali disponibili e le molteplici modalità di collegamento alla rete, determinano che i contenuti ed i servizi devono essere erogati in modo **context-aware**, cioè devono tenere conto di fattori specifici del contesto di erogazione quali per esempio le peculiarità dei terminali utilizzati, le preferenze degli utenti finali, la loro collocazione fisica, etc.

L'oggetto di ricerca descritto in questo documento tratta della realizzazione di un sistema basato sulla *context aware* technology, un settore di ricerca sviluppato in diversi centri accademici, che utilizza, nel nostro caso, il sistema di posizionamento *GPS* affinchè sia identificato in ogni istante uno stato del sistema che nella fattispecie rappresenterà la circostanza in cui si trova l'utente che lo utilizza.

Nella fase di *testing* del sistema proposto è stato utilizzato, per il lato client, un computer palmare *HP ipaq 5550* avente come sistema operativo Windows Pocket Pc 2003, un'antenna *GPS TOMTOM Bluetooth* e, per il lato middle e server, un notebook *Toshiba, Intel Celeron 1,3 GHz.*

3.1 Utilizzo del GPS e gestione di contenuti *contex aware*

Il progetto ha come obiettivo lo sviluppo di un sistema per la localizzazione di utenti mediante ricevitori *GPS* e la gestione di contenuti web context aware, adattandosi al contesto di fruizione del servizio e quindi alla posizione geografica dell'utente che utilizza il servizio stesso. A tal fine è possibile ipotizzare servizi che, automaticamente, sulla base dell'analisi della fruizione del servizio (posizione), siano in grado di proporre contenuti alternativi e opportuni al contesto in cui l'utente si trova.

Il sistema proposto è stato sviluppato con l'idea di accompagnare un utente nella visita di una specifica area siciliana, proponendogli dinamicamente contenuti relativi ai siti e specifici della posizione tenuta dal visitatore stesso. Chiaramente il contesto è dettato dalla posizione dell'utente all'interno della zona da visitare coperta dal servizio.

Il sistema presuppone che l'utente in questione sia in possesso di un computer palmare con sistema operativo Windows Pocket Pc 2003 su cui viene eseguita l'applicazione che comunica con il dispositivo *GPS* mediante protocollo Bluetooth. Quest'applicazione ottiene le informazioni dal dispositivo *GPS* che rappresentano la posizione dell'utente sulla superficie terrestre espressa in termini di latitudine e longitudine.

Per lo sviluppo del sistema quindi è stato necessario implementare un'applicazione per computer palmare, in grado di comunicare con il dispositivo *GPS* ed inviare informazioni opportune ad un'applicazione *middle* le cui funzionalità verranno illustrate nelle sezioni successive. L'applicazione che comunica con il dispositivo *GPS* è l'applicazione client del sistema proposto chiamata *Pocket Visit v.1.0*.

3.2 Overview del sistema

Il sistema proposto è stato sviluppato interamente in Java. La scelta di questo linguaggio di programmazione è motivata dal fatto che il compilatore Java non produce direttamente codice eseguibile dal calcolatore, ma produce un codice denominato Java bytecode, indipendente dal tipo di piattaforma di calcolo utilizzata.

Il risultato della compilazione di un programma scritto in Java è quindi indipendente dalla piattaforma. Ciò comporta l'uso di un apposito programma per eseguire il bytecode: l'interprete del bytecode, noto come *Macchina Virtuale Java* (Java Virtual Machine).

La portabilità del codice è una delle caratteristiche che hanno indotto alla scelta di questo linguaggio di programmazione.

Esistono in letteratura [3], [4] parecchie librerie, di seguito se ne elencano due che hanno agevolato lo sviluppo del sistema proposto:

- **javax.comm:** Le *Java Communications API (JCA)* permettono di creare delle applicazioni o applet portabili, che necessitano di comunicazioni attraverso porta seriale e/o parallela. Dopo la realizzazione delle JNI (Java Native Interface), che permettono di creare implementazioni native (in C o C++) di metodi Java, le JCA rappresentano un ulteriore passo verso la possibilità di utilizzare Java come linguaggio di programmazione completo (precedentemente era necessario fare comunicare codice java con codice C/C++).

  Interagire con le porte seriali o parallele di un sistema è un'operazione solitamente molto legata al sistema operativo stesso. Accedere alla seriale, ad esempio, in Windows è diverso dal farlo in un ambiente Unix o Windows CE. A tal fine è necessario fornirsi di una JVM che possa funzionare su sistema operativo Windows CE e che supporti le librerie javax.comm per la comunicazione con le porte seriali; una JVM che risponde a questi requisiti è stata sviluppata dalla $NSICOM$ ed è stata chiamata *CRE-ME v3.25*. Questa JVM si basa su JDK 1.1.8 ed è in grado di eseguire i files di estensione *jar*, precedentemente creati da macchine diverse, basati sulle API Java 1.1.8 [5].

- **javax.servlet.http:** Molto trattate in letteratura Java [4],[6] ed utilizzate per la creazione di *Java servlets*, soluzione adatta per le applicazioni che

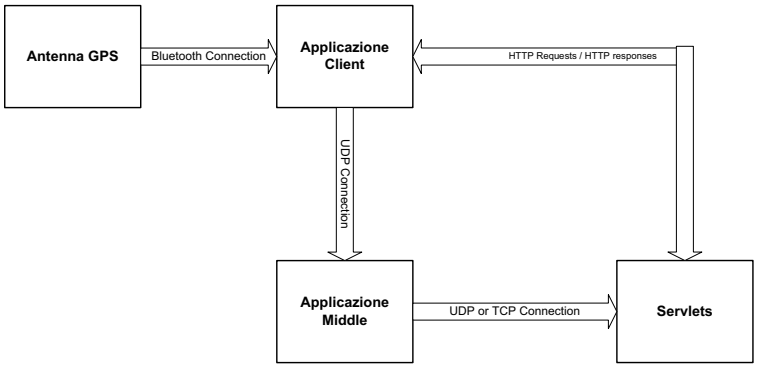

Figura 3.1: *Architettura sistema proposto*

usano massivamente i database e che comunicano con applicazioni che richiedono un supporto minimo per quanto riguarda il lato client. Il client si connette al server usando protocolli standard disponibili su quasi tutte le piattaforme. Nel caso in esame le servlets sono necessarie per soddisfare richieste *HTTP 1.0 e HTTP 1.1* e per la spedizione di pagine web legate alle richieste stesse.

3.3 Architettura del sistema proposto

La scelta dell'architettura è vincolata dall'analisi del sistema; nel caso in esame si è pensato ad un sistema distribuito in cui si distinguono dei moduli client che richiedono dati ad un modulo server; naturalmente i moduli client possono risiedere su macchine differenti (computer palmari). Per la comunicazione tra client e server è stata utilizzata una variante dell'architettura Client-Server di tipo Three-Tier; quindi si è considerato un livello intermedio (detto middle) rappresentato da un'applicazione che riceve i dati dai client utilizzando il protocollo UDP, li elabora opportunamente con la procedura che verrà descritta nelle sezioni successive, restituendo alla parte server la *zona* ove si trova ogni client e la rispettiva posizione in coordinate cartografiche. L'utilizzo di questa tipologia di architettura sgravia notevolmente l'elaborazione della parte server che, come verrà illustrato, sarà implementata con servlet invocate direttamente dai singoli clients mediante richieste HTTP (una connessione TCP per ogni client).

L'architettura è rappresentata dallo schema a blocchi della figura 3.1, mentre uno scenario esemplificativo è illustrato in figura 3.2. La parte client del sistema è formata dalla coppia palmare-dispositivo GPS che comunica con la parte middle mediante socket UDP; la scelta di utilizzare questo protocollo è vincolata dal fatto che UDP è *connectionless* e passa dei messaggi che vengono incapsulati in un'unica unità informativa (datagrammi), è veloce in quanto non esiste fase di setup della connessione (handshaking), ha un piccolo overhead di header, ed è possibile inviare datagrammi ad un ritmo arbitrario. In caso di perdita di pacchetti, non si avranno problemi al sistema poichè i pacchetti UDP vengono spediti dall'applicazione client ogni ΔT fissato in fase di configurazione del dispositivo. La comunicazione tra computer palmare e dispositivo GPS avviene,

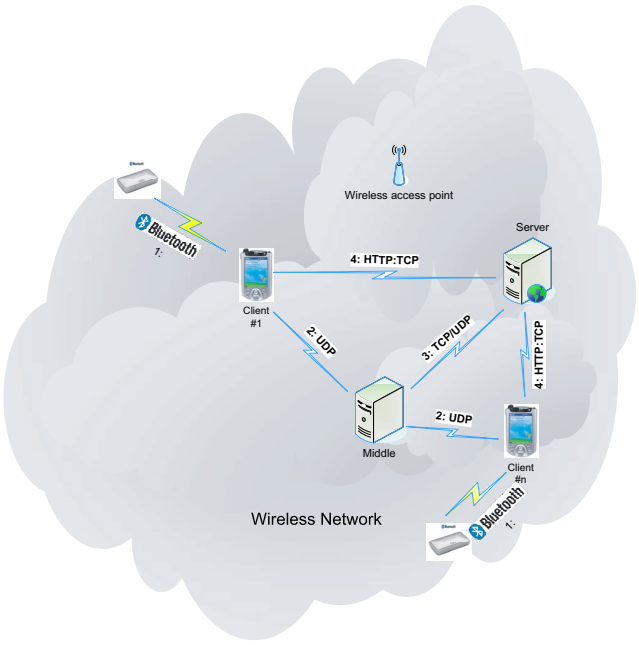

Figura 3.2: *Uno scenario del sistema proposto*

come verrà descritto in seguito, utilizzando il protocollo Bluetooth. Questa connessione è realizzata mediante comunicazione seriale utilizzando la libreria java *javax.comm* della *Sun*.

La comunicazione tra l'applicazione middle ed il server può avvenire analogamente a prima, mediante protocollo UDP o mediante protocollo TCP in funzione dell'utilizzo del sistema stesso.

Quando la parte server conosce le posizioni dei client, attende che questi invochino le *servlet* implementate per fornire il servizio di *context aware* web.

Si presuppone che la zona in cui si utilizzerà il sistema sia coperta da una rete Wireless. Ciascun computer palmare fruitore del servizio avrà un indirizzo di rete IP fisso. Tale informazione verrà utilizzata dall'applicazione middle per far leggere alla servlet i dati comunicati relativi al client identificato mediante l'indirizzo. Questo sistema di corrispondenze è realizzato mediante la creazione di una *Hash Table*.

La comunicazione diretta tra client e server, ovviamente, avviene mediante protocollo HTTP.

Nelle sezioni seguenti sono descritte in dettaglio le parti del sistema proposto: 1) la parte client rappresentata dall'applicazione per computer palmare Pocket Visit; 2)la parte middle rappresentata dall'applicazione di gestione della localizzazione; 3) la parte server rappresentata da due Servlet per la spedizione delle pagine web relative al contesto (posizione) in cui si trova il client.

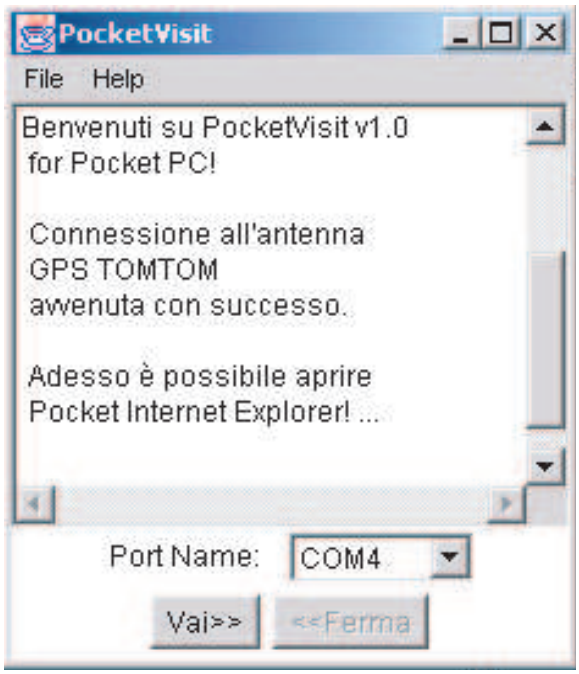

Figura 3.3: *GUI Pocket Visit v.1.0*

3.4 Pocket Visit v.1.0 for Pocket Pc

L'applicazione client, chiamata Pocket Visit, ha la funzione di comunicare con l'antenna GPS e successivamente trasferire all'applicazione middle la posizione in cui si trova l'utente. La connessione tra antenna GPS e computer palmare avviene mediante protocollo Bluetooth, è necessario che il computer palmare sia dotato di dispositivo Bluetooth affinchè si possa interfacciare con il sistema. Per aprire la connessione Bluetooth, l'applicazione client utilizza un collegamento seriale (RS232) tra il palmare e l'antenna GPS; poichè il numero di porta com può cambiare fra le tipologie di computer palmari, è necessario un settaggio manuale del numero di porta da utilizzare.

In figura 3.3 è illustrata la GUI dell'applicazione client Pocket Visit v.1.0. Una volta scelta la porta di comunicazione relativa alla connessione Bluetooth, che può variare fra le tipologie di computer palmare, selezionando il pulsante $Vai >>$ inizia la comunicazione con l'antenna GPS che trasferirà al palmare, ad intervalli $\Delta T = 1s$, la posizione dell'utente; nell'area testo della GUI viene notificata l'avvenuta connessione con l'antenna.

Le informazioni relative alla posizione vengono ricevute secondo lo standard NMEA 0183 [2] nella forma:

$$\$Prefisso, campo[1], campo[2], ..., campo[n] * checksum[CR][LF]$$

per cui è stato necessario implementare la funzionalità di *tokenizzazione* delle frasi NMEA. Questa funzionalità estrae opportunamente dalle *sentences* soltanto i dati di interesse che sono: *latitudine, longitudine, emisfero della posizione*

attuale e verso della posizione attuale, considerando che il numero di campi delle *sentences* è variabile.

I dati di interesse non vengono estratti da tutte le *sentences*, bensì da quelle con il seguente prefisso:

- $GPGGA;
- $GPGLL;
- $GPRMC.

Una volta estratti i dati di interesse dalle *sentences*, l'applicazione client apre un socket UDP e, mediante un datagramma UDP, invia queste informazioni all'applicazione middle ad un preciso indirizzo IP che l'amministratore del sistema ha precedentemente indicato.

L'applicazione client gestisce infine ogni tipo di eccezione tra cui la gestione della *ownership* (padronanza) della porta seriale tra le applicazioni in esecuzione nel computer palmare.

La connessione Bluetooth viene chiusa selezionando il pulsante $<< Ferma$.

Una volta stabilita la connessione è possibile aprire il browser web del computer palmare.

L'applicazione client è formata da sei classi Java:

1. *AlertDialog*: utilizzata per la gestione degli avvertimenti all'utente mediante finestra.
2. *PocketVisit*: è la classe principale (*main class*) in cui è implementata l'interfaccia grafica dell'applicazione client. Questa classe gestisce anche gli eventi associati ai pulsanti e richiama la classe *SerialConnection*.
3. *SerialConnection*: è la classe centrale per il controllo dell'accesso alla porta Bluetooth. In questa classe sono implementati tutti i metodi per la gestione della connessione Bluetooth, gestione delle eccezioni relative alla *ownership* della porta, spedizione delle informazioni, mediante socket UDP, all'applicazione middle. Questa classe contiene l'indirizzo IP della macchina ove è in esecuzione l'applicazione middle.
4. *PortRequestedDialog*: Informa l'utente che un'altra applicazione ha richiesto la porta che sta utilizzando e domanda se si è disposti a lasciare libera la porta. Se l'utente risponde Si la porta è chiusa ed il dialogo è chiuso, se l'utente risponde No il dialogo è chiuso e nessun'altra azione è intrapresa. Questa classe è invocata dalla classe *SerialConnection*.
5. *SerialConnectionException*: Classe ereditata dalla classe java.lang.Exception che implementa l'eccezione di connessione seriale non riuscita.
6. *SerialParameters*: Classe che contiene i parametri della porta seriale: nome della porta, baud rate, controllo del flusso in ingresso, controllo del flusso in uscita, data bits, stop bits e controllo di parità.

Il codice sorgente dell'applicazione client è descritto in appendice A.1.

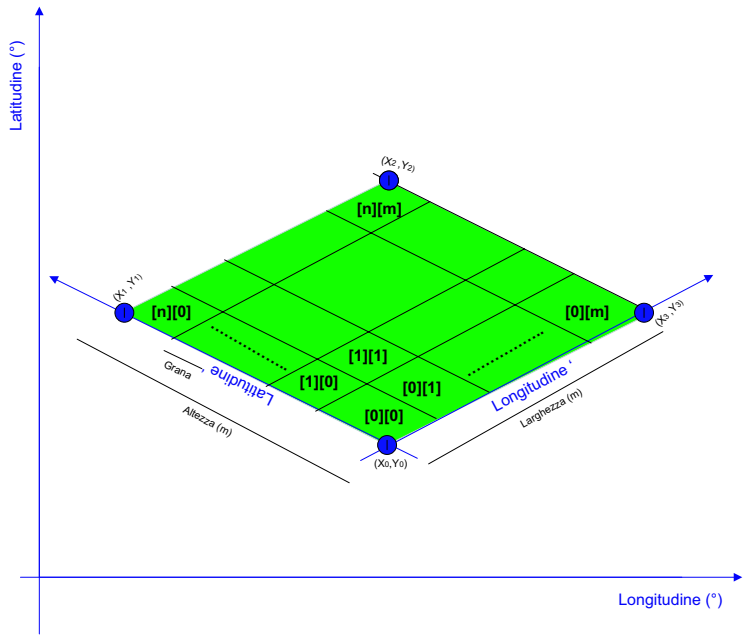

Figura 3.4: *Scenario della suddivisione in griglia dell'area in cui deve essere utilizzato il sistema proposto*

3.5 Applicazione middle

L'applicazione middle è il *core* del sistema proposto poichè trasmette alla parte server la *zona* ove è localizzato il client e la sua posizione espressa in coordinate cartografiche.

Un importante compito dell'applicazione middle è quello di caricare la *mappa* della zona ove deve essere installato il sistema. Questa mappa è contenuta in un file chiamato *map.txt* e consiste soltanto di quattro coppie (latitudine,longitudine) relative ai quattro *vertici* di un rettangolo che include il luogo stesso. Questo rettangolo, poi, viene suddiviso in griglia in funzione di una *grana* stabilita dall'amministratore del sistema. In questa maniera, ogni *cella* che forma il rettangolo potrà contenere al suo interno uno o più *Punti di Interesse* (POI).

Lo scenario è illustrato in figura 3.4. Siano (X_i, Y_i) rispettivamente latitudine e longitudine di un vertice v_i del rettangolo. Durante la fase preliminare di acquisizione della mappa, per rendere il più semplice possibile il calcolo della cella che contiene un client, l'applicazione middle effettua una *traslo-rotazione* del sistema di riferimento cartografico in maniera tale da svincolarsi dal problema della posizione del rettangolo rispetto al mondo che potrebbe complicare la procedura di individuazione della cella che contiene il client. Per fare questo, l'applicazione middle, effettua inizialmente una traslazione tale da fare coincidere il primo vertice v_0 del rettangolo, di coordinate (X_0, Y_0), con l'origine del sistema di riferimento traslato, e poi una rotazione del sistema di riferimento traslato di un angolo α tale che i vertici v_0 e v_3 abbiano stesso valore di Y

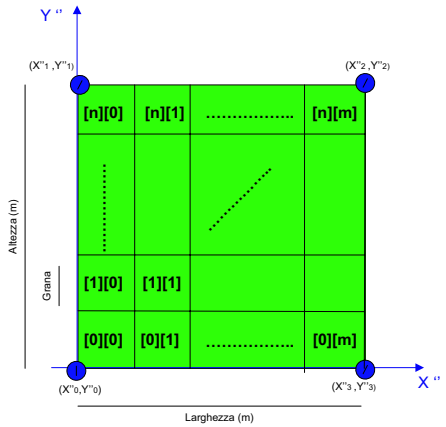

Figura 3.5: *Nuovo scenario considerato dall'applicazione middle effettuata la traslo-rotazione del sistema di riferimento cartografico*

rispetto al sistema di riferimento traslo-ruotato.

Effettuata questa traslo-rotazione, il nuovo scenario considerato dall'applicazione middle è illustrato in figura 3.5. Terminata la fase di acquisizione della mappa, l'applicazione middle apre un socket UDP mettendosi in ascolto alla porta # 9999. Su tale porta potrà ricevere i pacchetti UDP, inviati dalle applicazioni client, contenenti le informazioni relative alla posizione dei vari utenti fruitori del sistema.

Ogni qualvolta l'applicazione middle riceve un pacchetto UDP da un client, prima di tutto legge dall'header del pacchetto l'indirizzo IP del mittente ed aggiorna una *Hash table* che spedirà, ogni ΔT, alla parte server. Questa Hash Table è formata da n righe e due colonne dove $n = max$(numero di client collegati fino a quel momento). Gli *attributi* della Hash Table sono: 1)*Indirizzo IP* del client e 2) identificatore *ID=1,2,.....n* per potere effettuare un'associazione *client*↔*fileClient*, dove *fileClient* rappresenta il file che deve leggere la servlet, lato server, e che contiene la cella ove è contenuto quel client. Chiaramente se arriva un pacchetto UDP da un mittente avente un indirizzo IP contenuto nella Hash Table, a questa non verrà apportato alcun aggiornamento.

Aggiornata la Hash Table, l'applicazione middle legge le coordinate del client, in formato NMEA, contenute nel pacchetto UDP. La frase NMEA è del tipo:

xxxx.xxxx N xxxxx.xxxx E

quindi viene fatta una *tokenizzazione* di questa stringa con argomento $'.'$ e da quì vengono espresse le coordinate in gradi.

Fatto questo, l'applicazione middle esprime le coordinate del client rispetto al sistema di riferimento traslo-ruotato, esprime la grana in gradi e calcola la cella del rettangolo che contiene quel client come:

$$riga = (int)(Math.ceil(Y_c''/g));$$
$$colonna = (int)(Math.ceil(X_c''/g)).$$

dove (X_c'', Y_c'') rappresentano le coordinate del client rispetto al sistema di riferimento traslo-ruotato.

L'applicazione middle, infine, inserisce queste informazioni e la posizione del client in un file di nome *cella+ID.txt* dove ID è il valore associato alla chiave identificata dall'indirizzo IP del client nella Hash Table. Questo file viene spedito alla parte server in maniera tale che la servlet, quando il client effettuerà una richiesta HTTP a questa, possa rispondere in base al *contesto* (posizione) di quel client.

Ovviamente l'applicazione middle lavora in maniera *concorrente*.

Le due classi Java che costituiscono l'applicazione middle sono:

1. *CalcoloDistanza*: Classe invocata dalla classe *Middle* utile per calcolare la distanza, in linea d'aria, tra due punti nella superficie terrestre. Per effettuare questa misura si suppone la terra come una sfera perfetta di raggio 6371 Km. La formula utilizzata per il calcolo della distanza tra due punti $P_1 = (lat_1, lon_1)$ e $P_2 = (lat_2, lon_2)$ è:

   $p_1 = cos(lat_1) * cos(lon_1) * cos(lat_2) * cos(lon_2);$
   $p_2 = cos(lat_1) * sin(lon_1) * cos(lat_2) * sin(lon_2);$
   $p_3 = sin(lat_1) * sin(lat_2);$
   $distanza = arcos(p_1 + p_2 + p_3) * 6371;$

2. *Middle*: Classe principale (main class) che svolge tutte le attività descritte precedentemente.

Il codice sorgente dell'applicazione middle è descritto in appendice A.2.

3.6 Creazione file di mappa (map.txt)

Per la creazione del file contenente la mappa è stata sviluppata un'applicazione che richiede all'amministratore del sistema le quattro coppie (latitudine,longitudine), in gradi, relative ai quattro vertici del rettangolo e la grana, in metri, che identificherà il numero di righe e di colonne della matrice che rappresenta il rettangolo. L'applicazione, mediante un ordinamento dei vertici in base a determinati criteri, crea un file (map.txt) contenente le informazioni suddette secondo la sintassi:

$lat_1\ lon_1\ lat_2\ lon_2$
$lat_3\ lon_3\ lat_4\ lon_4$
$grana$

La GUI di questa applicazione è illustrata in figura 3.6. Nel caso si inserissero dati in un formato non valido, l'applicazione lancia un'eccezione invitando l'utente (amministratore del sistema) ad immettere i dati nel formato corretto.

Le due classi java che costituiscono quest'applicazione sono:

1. *Panels*: Costruisce e gestisce l'interfaccia grafica dell'applicazione:

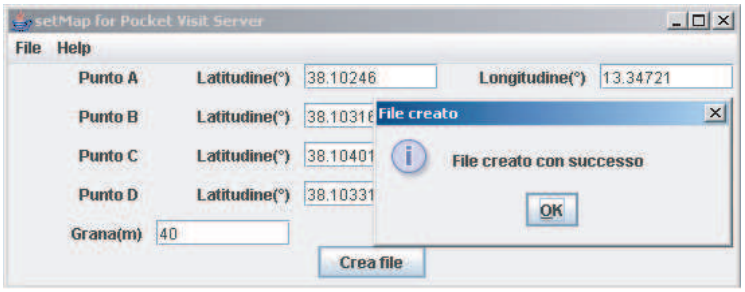

Figura 3.6: *GUI Creatore file mappa*

2. *setMap*: Classe principale (main class) dell'applicazione. La classe *setMap* si occupa della lettura dei dati dai *TextFields* contenuti nel Frame, del controllo della validità dei formati dei vari campi, dell'ordinamento dei vertici secondo un determinato criterio, e della creazione del file *map.txt*.

Una volta creato il file *map.txt* è compito dell'amministratore del sistema copiare questo file nella directory ove è presente il file class dell'applicazione middle.

Il codice sorgente di quest'applicazione è descritto in appendice A.4.

3.7 Memorizzazione coordinate dei Punti di Interesse

Per la memorizzazione delle coordinate dei *Punti di Interesse (POI)* è stata sviluppata un'applicazione per computer palmare capace di aprire una connessione Bluetooth con l'antenna *GPS* e scaricare le informazioni relative alla posizione dell'antenna *GPS* stessa. Mediante quest'applicazione è possibile memorizzare in un file, secondo una certa sintassi, le coordinate relative ai punti di interesse.

La GUI di quest'applicazione è illustrata in figura 3.7. L'applicazione opera nella maniera seguente: l'amministratore del sistema raggiunge il punto di interesse con il palmare, inserisce nel campo di testo la denominazione del punto di interesse e, mediante il pulsante *Salva POI*, salva le coordinate del punto di interesse nel file *POIs.txt* costruito secondo la sintassi:

$$
\begin{aligned}
& n \\
lat_1 \quad & lon_1 \\
lat_2 \quad & lon_2 \\
lat_3 \quad & lon_3 \\
& . \\
& . \\
& . \\
& .
\end{aligned}
$$

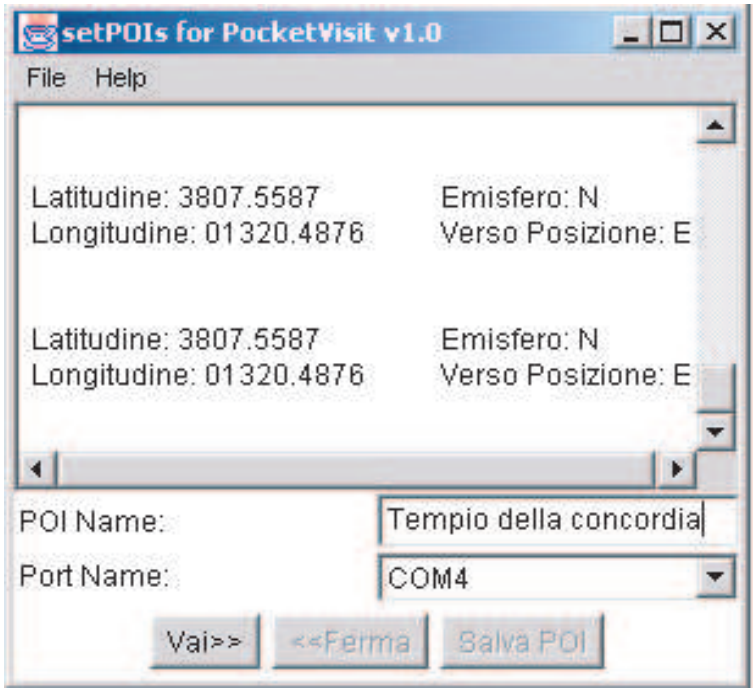

Figura 3.7: *GUI memorizzazione coordinate dei Punti di Interesse*

$$lat_n \quad lon_n$$

Ogni qualvolta viene premuto il pulsante *Salva POI*, viene aggiornato il file $POIs.txt$.

Terminata l'operazione di memorizzazione delle coordinate dei Punti di Interesse, è compito dell'amministratore del sistema copiare il file POIs.txt presente nel computer palmare nella directory *bin* di $TomCat$, web server che implementa gli standard relativi a servlet e JSP supposto installato nella parte server del sistema proposto.

Le sei classi Java che costituiscono quest'applicazione sono:

1. *AlertDialog*: Serve per la gestione di avvertimenti all'utente mediante finestra.

2. *setPOIs*: Classe principale (*main class*) ove è implementata l'interfaccia grafica dell'applicazione. Questa classe gestisce anche gli eventi associati ai pulsanti per cui richiama la classe *SerialConnection*.

3. *SerialConnection*: è la classe centrale per il controllo dell'accesso alla porta Bluetooth. In questa classe sono implementati tutti i metodi per la gestione della connessione Bluetooth, gestione delle eccezioni relative alla *ownership* della porta, costruzione e aggiornamento del file POIs.txt contenente le coordinate dei punti di Interesse.

4. *PortRequestedDialog*: Informa l'utente che un'altra applicazione ha richiesto la porta che sta utilizzando e domanda se si è disposti a lasciare libera la

porta. Se l'utente risponde Si la porta è chiusa ed il dialogo è chiuso, se l'utente risponde No il dialogo è chiuso e nessun'altra azione è intrapresa. Questa classe è invocata dalla classe *SerialConnection*.

5. *SerialConnectionException*: Classe ereditata dalla classe java.lang.Exception che rappresenta l'eccezione di connessione seriale non riuscita.

6. *SerialParameters*: Classe che contiene i parametri della porta seriale: nome della porta, baud rate, controllo del flusso in ingresso, controllo del flusso in uscita, data bits, stop bits e controllo di parità.

Il codice sorgente di quest'applicazione è descritto in appendice A.5.

3.8 Parte Server del sistema proposto

La parte server del sistema proposto è rappresentata da due servlet:

1. Servlet che gestisce la Home Page;

2. Servlet invocata dal client che spedisce le pagine web in base al contesto in cui si trova il client stesso. La modalità di invio di queste pagine web può essere:

   - *client pull*:Consiste nel fare eseguire direttamente al client un refresh ogni ΔT secondi;
   - *server push*: Consiste nella spedizione di pagine HTML al client ogni ΔT secondi. Questa modalità è gestita interamente dalla parte server.

Queste servlet sono invocabili dal computer palmare di ciascun utente dopo l'avvenuta connessione con l'antenna GPS. Si suppone che sul lato server sia installato il web server e contenitore di servlet *TomCat -Apache software licence-* cosicchè il client può effettuare richieste HTTP, mediante il proprio browser web, al web server in ascolto alla porta # 8080.

Nelle sezioni 3.8.1 e 3.8.2 sono descritte le due servlet in dettaglio.

3.8.1 Servlet di Home Page

La servlet di Home Page viene invocata una sola volta da ogni client che usufruisce del servizio; le sue funzionalità sono:

- conteggiare i client che hanno usufruito del servizio fino al momento della connessione;
- far prelevare il software necessario all'utente per l'utilizzo del sistema e un manuale che guidi l'utente al suo corretto uso;
- assegnare ad ogni client un cookie affinchè le successive richieste HTTP del client contengano, nel campo *set-cookie*, l'identificativo precedentemente assegnato. Questo aiuta ad identificare il client;
- permettere all'utente di far invocare la servlet che spedirà al client la pagina web relativa al contesto in cui si trova il client stesso.

Figura 3.8: *Esempio di Home Page*

Una Home Page di esempio è illustrata in figura 3.8. Cliccando su *Entra*, il client invocherà una nuova servlet, descritta nella sezione 3.8.2.

3.8.2 Servlet di context management

La Servlet di context management si occupa della spedizione delle pagine web ai client, in base al loro contesto (posizione). Questa Servlet è quella che deve essere implementata in funzione della zona ove si vuole installare il sistema proposto poichè è compito di questa la spedizione delle *risorse* relative a quella determinata zona. Si suppone che l'amministratore del sistema sia in possesso di una *planimetria* del luogo ove applicare il sistema proposto per permettere di adattare opportunamente la Servlet di context management proposta in questo documento alla zona ove intende installare il sistema.

Le attività di questa servlet sono molteplici:

1. carica dal file POIs.txt, precedentemente creato dall'applicazione descritta nella sezione 3.7, le coordinate cartografiche dei Punti di interesse (POI).

2. legge la Hash Table dal file Hash.txt, spedito dall'applicazione middle, in maniera tale da potere identificare i client e leggere correttamente i contesti.

3. lettura, per ciascun client, delle informazioni relative alla *cella* del rettangolo che contiene il client ed alla sua posizione, dal file *cella+ID.txt* dove *ID* è il numero associato alla chiave rappresentata dall'indirizzo IP del client contenuto nella Hash Table.

Figura 3.9: *Esempio di spedizione pagina HTML contenente le risorse relative ad un Punto di Interesse*

4. calcolo della distanza, in metri, tra il client e il punto di interesse contenuto nella cella che contiene il client e più vicino al client stesso.

5. spedizione delle risorse relative al contesto del client mediante risposte HTTP e possibilità di accedere a risorse multimediali, audio e video, relative al Punto di Interesse che il client sta visitando.

In base alla posizione di un client, possono verificarsi casi differenti:

- *client contenuto in una cella contenente un solo Punto di Interesse*: in questo caso viene spedita al client una pagina HTML contenente le risorse relativi al Punto di Interesse contenuto in quella cella e la distanza dal Punto di Interesse stesso. Inoltre il client ha la possibiltà di visionare contenuti multimediali audio e video relative al Punto di Interesse stesso. Un esempio è illustrato in figura 3.9.

- *client contenuto in una cella contenente due o più Punti di Interesse*: in questo caso viene spedita al client una pagina web intermedia contenente i collegamenti ipertestuali alle risorse di ciascun Punto di Interesse. Selezionando il collegamento ipertestuale, verrà spedita al client una pagina HTML contenente le risorse relative a quel Punto di Interesse e viene data al client la possibilità di visionare i contenuti multimediali audio e video attinenti al Punto di Interesse selezionato.

- *client contenuto in una cella che non contiene Punti di Interesse*: in questo caso viene spedita al client una pagina HTML che lo informa della sua

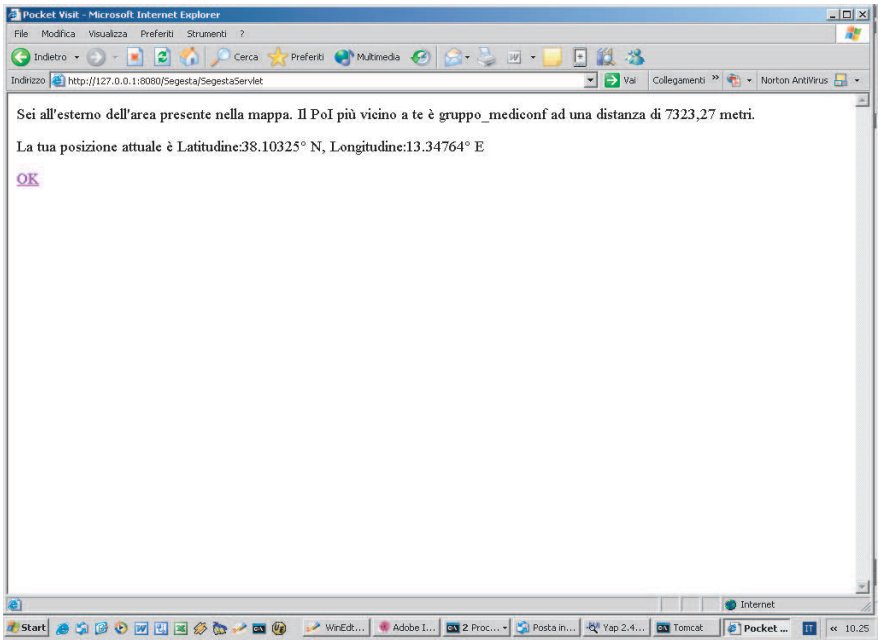

Figura 3.10: *Caso in cui il client si trova all'esterno della mappa*

posizione e del Punto di Interesse più vicino a lui. Questo può essere inteso come una sorta di *guida* dell'utente quando la zona ove è applicato il sistema è estesa e i Punti di Interesse situati in modo sparso.

- *client non contenuto in celle che formano il rettangolo*: supposto che la posizione del client sia coperta dalla rete Wireless, viene spedita al client una pagina HTML che lo informa della sua posizione all'esterno dell'area dotata del servizio e della distanza dal Punto di Interesse più vicino a lui. Questo caso d'uso è stato implementato ma, in situazioni normali, non dovrebbe essere invocato. Un esempio è illustrato in figura 3.10.

3.9 Simulatore antenna GPS

Per potere effettuare delle prove è stato implementato un simulatore di antenna *GPS*. Quest'applicazione legge le coordinate dei vertici del rettangolo dal file map.txt e genera, in maniera *Random* delle posizioni all'interno del rettangolo o immediatamente all'esterno di esso. Ogni posizione generata viene spedita, mediante datagramma UDP, all'applicazione middle per potere essere elaborata.

Il codice sorgente di quest'applicazione è descritto in appendice A.6.

3.10 Conclusioni

In questo documento è stato proposto un sistema di localizzazione e di gestione di contenuti context aware che, in maniera dinamica, spedisce informazioni ad

un computer palmare client in base alla posizione in cui si trova. Questo sistema consiste di un'applicazione client che comunica con l'antenna GPS, un'applicazione middle che elabora le informazioni spedite dall'applicazione client e localizza il client stesso nella zona in cui è stato installato il sistema, e di una parte server che gestisce il *context awaring* spedendo pagine web contenenti risorse relative a determinati Punti di Interesse.

La *distribuzione* del sistema è stata fatta tenendo conto di molteplici fattori tra cui l'onerosità nell'utilizzo di un protocollo di comunicazione rispetto ad un altro, multithreading, leggerezza della Servlet di context management, comunicazione semplice tra le varie parti del sistema.

Un pregio del sistema è la sua adattabilità, infatti, modificando la servlet di context management è possibile il suo utilizzo in ambiti diversi.

Infine, aggiornando opportunamente la classe *SerialConnection* delle applicazioni che la implementano, è possibile rendere compatibili con il sistema proposto altre marche di antenne GPS.

Appendice A

Codice sorgente Java

A.1 Applicazione Client Pocket Visit v.1.0

A.1.1 AlertDialog.java

```
//PocketVisit 1.0 for PocketPc

import java.awt.*;
import java.awt.event.*;

/**
Questa classe è configurabile per messaggi e titoli.
La larghezza del Dialog sarà quanto la lunghezza della linea del messaggio.
*/
public class AlertDialog extends Dialog implements ActionListener {

    /*
    Crea un nuovo <code>AlertDialog</code> con tre linee di messaggio e titolo.

    parent = Frame passato.
    title = Titolo che appare nel bordo del dialogo.
    lineOne = La prima linea del messaggio nel dialogo.
    lineTwo = La seconda linea del messaggio nel dialogo.
    lineThree = La terza linea del messaggio nel dialogo.
    */
    public AlertDialog(Frame parent,
       String title,
       String lineOne,
       String lineTwo,
       String lineThree) {
super(parent, title, true);

Panel labelPanel = new Panel();
labelPanel.setLayout(new GridLayout(3, 1));
labelPanel.add(new Label(lineOne, Label.CENTER));
labelPanel.add(new Label(lineTwo, Label.CENTER));
labelPanel.add(new Label(lineThree, Label.CENTER));
add(labelPanel, "Center");

Panel buttonPanel = new Panel();
Button okButton = new Button("OK");
okButton.addActionListener(this);
buttonPanel.add(okButton);
add(buttonPanel, "South");

FontMetrics fm = getFontMetrics(getFont());
int width = Math.max(fm.stringWidth(lineOne),
 Math.max(fm.stringWidth(lineTwo), fm.stringWidth(lineThree)));

setSize(width + 40, 150);
setLocation(parent.getLocationOnScreen().x + 30,
```

```
    parent.getLocationOnScreen().y + 30);
setVisible(true);
    }

    /**
   Quando è pressato OK il messaggio diviene invisibile,  e ritorna.
    */
    public void actionPerformed(ActionEvent e) {
setVisible(false);
dispose();
    }
}
```

A.1.2 PortRequestedDialog.java

```
//PocketVisit 1.0 for PocketPc

import java.awt.*;
import java.awt.event.*;
import javax.comm.*;

/*
Informa l'utente che un'altra applicazione ha richiesto la porta che sta utilizzando
e dopo domanda se si è disposti a lasciare libera la porta.
Se l'utente risponde "Si" la porta è chiusa ed il dialogo è chiuso,se l'utente
risponde "No" il dialogo è chiuso e nessun'altra azione è intrapresa
*/
public class PortRequestedDialog extends Dialog implements ActionListener {

    private PocketVisit parent;

    /*
    Crea il dialogo con due pulsanti ed un messaggio richiedente all'utente se è disposto a
    lasciare la porta.

    parent= L'oggetto principale PocketVisit.
    */
    public PortRequestedDialog(PocketVisit parent) {
super(parent, "Porta richiesta!", true);
this.parent = parent;

String lineOne = "La tua porta è stata richiesta";
String lineTwo = "da un'altra applicazione.";
String lineThree = "Vuio cedere la porta?";
Panel labelPanel = new Panel();
labelPanel.setLayout(new GridLayout(3, 1));
labelPanel.add(new Label(lineOne, Label.CENTER));
labelPanel.add(new Label(lineTwo, Label.CENTER));
labelPanel.add(new Label(lineThree, Label.CENTER));
add(labelPanel, "Center");

Panel buttonPanel = new Panel();
Button yesButton = new Button("Si");
yesButton.addActionListener(this);
buttonPanel.add(yesButton);
Button noButton = new Button("No");
noButton.addActionListener(this);
buttonPanel.add(noButton);
add(buttonPanel, "South");

FontMetrics fm = getFontMetrics(getFont());
int width = Math.max(fm.stringWidth(lineOne),
 Math.max(fm.stringWidth(lineTwo), fm.stringWidth(lineThree)));

setSize(width + 40, 150);
setLocation(parent.getLocationOnScreen().x + 30,
    parent.getLocationOnScreen().y + 30);
setVisible(true);
    }

    /**
    Ascolta gli eventi generati dai bottoni. Se è pressato il pulsante "Si"
    è chiamata la routine di chiusura della porta.
```

```
    */
    public void actionPerformed(ActionEvent e) {
String cmd = e.getActionCommand();

    if (cmd.equals("Si")) {
parent.portClosed();
    }

    setVisible(false);
    dispose();
    }
}
```

A.1.3 SerialConnection.java

```
//*****************************************
//*    PocketVisit v1.0 for Pocket Pc       *
//*                                         *
//*         ICAR-CNR di Palermo             *
//*                                         *
//*     Sviluppatore: Giovanni Aiello       *
//*                                         *
//*e-mail address: aiello.giovi@virgilio.it *
//*****************************************

import javax.comm.*;
import java.io.*;
import java.awt.TextArea;
import java.awt.event.*;
import java.util.TooManyListenersException;
import java.util.StringTokenizer;
import java.net.*;
import java.lang.Exception;


public class SerialConnection implements SerialPortEventListener,
                    CommPortOwnershipListener { /*La SerialPortEventListener
                                        propaga eventi di porta seriale mentre
                                        l'interfaccia CommPortOwnershipListener
                                        fa si che quando una porta è aperta, un
                                        evento CommPortOwnership di tipo PORT_OWNED
                                        (porta occupata) sarà propagato. Quando una
                                        porta è chiusa, un evento
                                        CommPortOwnership di tipo PORT_UNOWNED
                                        (porta libera) sarà propagata */

private PocketVisit parent; //Oggetto PocketVisit

private TextArea messageAreaOut; //TextArea dei messaggi in uscita che passiamo
                                        //dalla classe PocketVisit
private TextArea messageAreaIn; //TextArea dei messaggi in uscita che passiamo dalla
                                        //classe PocketVisit
private SerialParameters parameters; //oggetto della classe SerialParameters che rappresenterà
                                           // i parametri che passiamo
private OutputStream os; //Flusso in uscita
private InputStream is; //Flusso in ingresso
String old;//Serve per evitare errori dovuti a mancanze di informazioni dall'antenna GPS

private CommPortIdentifier portId; /*CommPortIdentifier è un gestore di comuniczioni di porte
                                       CommPortIdentifier è la classe centrale per il controllo
                                       dell'accesso alle porte ci comunicazione*/
private SerialPort sPort; // una porta di comunicazione seriale RS-232.

private boolean open; //Booleano

int flag;//Controlla se la connessione è avvenuta con successo

/*
Crea un oggetto SerialConnection e inizializza le variabili passate come parametri

parent= Oggetto PocketVisit.
parameters= Un oggetto SerialParameters.
messageAreaIn= Il TextArea che visualizza i messaggi in ingresso dalla porta seriale
```

```
*/
/*Nel messageAreaOut ci sono i messaggi che vengono mandati fuori la porta seriale*/
public SerialConnection(PocketVisit parent, /*COSTRUTTORE che accetta un oggetto PocketVisit,
                                             i parametri, la TextArea di input(Area dove i
                                             mex arrivano) e la TextArea di output */
SerialParameters parameters,
TextArea messageAreaOut,
TextArea messageAreaIn) {
old=new String("");
flag=0;
this.parent = parent; //Inizializza parent a parent di tipo SerialDemo
this.parameters = parameters;
//this.messageAreaOut = messageAreaOut;
this.messageAreaIn = messageAreaIn;
open = false; //è un booleano

}

/*Tenta di aprire una connessione seriale e flussi usando i parametri della porta
seriale (oggetto di tipo SerialParameters). Se la connesisone non va a buon fine,
esso torna la porta attraverso un'eccezione*/

public void openConnection() throws SerialConnectionException {  //Metodo che apre
//                                                                  una connessione

// Ottiene un oggetto CommPortIdentifier (javax.comm) per la porta che si vuole aprire.
try {
/*Gestione delle porte di comunicazione. CommPortIdentifier è una classe centrale per
il controllo all'accesso alle porte di comunicazione.*/
portId =
CommPortIdentifier.getPortIdentifier(parameters.getPortName());/*getPortIdentifier(String)
                                                                 è un metodo che ottiene
                                                                 un oggetto CommPortIdentifier
                                                                 mediante il nome della porta */

} catch (NoSuchPortException e) {
throw new SerialConnectionException(e.getMessage());
}


//Apre la porta rappresentata dall'oggetto CommPortIdentifier, viene dato all'apertura della
// chiamata un timeout relativamente lungo di 30 sec per permettere un'applicazione differente di
//rilasciare la porta se l'utente la vuole
try {
sPort = (SerialPort)portId.open("PocketVisit", 30000); /*il metodo open() della classe
                                                        CommPortIdentifier
                                                        apre una porta di comunicazione
                                                        e accetta il nome dell'applicazione
                                                        e un intero che indica il timeout (msec).
                                                        IL METODO OPEN OTTIENE LA COMPLETA
                                                        PADRONANZA DELLA PORTA da qualche
                                                        altra applicazione
                                                        , è propagato un evento
                                                        PORT_OWNERSHIP_REQUESTED usando il
                                                        meccanismo dell'evento
                                                        CommPortOwnershipListener.
                                                        C'è un InputStream ed un outputStream
                                                        associato a ogni porta. Dopo che una porta
                                                        è aperta mediante il metodo open,
                                                        tutte le chiamate al metodo getInputStream
                                                        ritornerà lo stesso stream fino a che è
                                                        chiamato il metodo close*/

messageAreaIn.append("    **************************\n ");
messageAreaIn.append("    *      PocketVisit v1.0 \"Client\"   *\n ");
messageAreaIn.append("    *         ICAR-CNR - Palermo      *\n ");
messageAreaIn.append("    ************************** \n\n\n");
messageAreaIn.append(" Attendere conferma connessione \n con antenna GPS....");
messageAreaIn.append("\n Se tale conferma non dovesse essere data,\n assicurarsi che la connessione
Bluetooth \n possa avvenire.. \n\n\n ");

} catch (PortInUseException e) { // l'evento di porta in uso lanciando un'eccezione se in uso
throw new SerialConnectionException(e.getMessage());
}
```

```
// Setta i parametri della connessione, se essi non si settano, la porta viene chiusa e viene
//lanciata un'eccezione
try {
setConnectionParameters(); //
} catch (SerialConnectionException e) {
sPort.close();
throw e;
}

//Apre i flussi di ingresso e uscita per la connessione. Se essi non si aprono, viene lanciata
//un'eccezione
try {
os = sPort.getOutputStream();
is = sPort.getInputStream();

} catch (IOException e) {
sPort.close();
throw new SerialConnectionException("Errore apertura flussi I/O");
}

// Aggiunge questo oggetto come un ascolto di evento per la porta seriale.
try {
sPort.addEventListener(this);
} catch (TooManyListenersException e) {
sPort.close();
  throw new SerialConnectionException("troppi listeners aggiunti");
}

// Setta notifyOnDataAvailable a true per permettere event driven di input.
sPort.notifyOnDataAvailable(true);

// Setta notifyOnDataAvailable a true per permettere event driven di break.
sPort.notifyOnBreakInterrupt(true);

try {
sPort.enableReceiveTimeout(30);
} catch (UnsupportedCommOperationException e) {
}


portId.addPortOwnershipListener(this);

open = true;
}


public void setConnectionParameters() throws SerialConnectionException{ /*Metodo di utilità
                                                                          che setta i parametri di connessione
                                                                          utilizzando l'oggetto parameters.
                                                                          Se il settaggio fallisce ritorna
                                                                          l'oggetto parameters
                                                                          ai suoi settaggi originali e
                                                                          lancia l'eccezione*/

// Salva lo stato dei parametri prima di tentare il settaggio.
int oldBaudRate = sPort.getBaudRate(); //salva il BaudRate
int oldDatabits = sPort.getDataBits();
int oldStopbits = sPort.getStopBits();
int oldParity   = sPort.getParity();
int oldFlowControl = sPort.getFlowControlMode();

// Setta i parametri di connessione, se il set fallisce ritorna l'oggetto parameters
// allo stato originale.
try {
sPort.setSerialPortParams(parameters.getBaudRate(), //Setta i parametri della porta seriale
//                                                    passando tutti i parametri.Ricordiamo che
parameters.getDatabits(),            //sPort è un oggetto CommPortIdentifier
parameters.getStopbits(),
parameters.getParity());
} catch (UnsupportedCommOperationException e) { //Gestisce l'eccezione di non suporto dei parametri
parameters.setBaudRate(oldBaudRate); //Riporta parameters al suo stato originale
parameters.setDatabits(oldDatabits);
parameters.setStopbits(oldStopbits);
```

```
parameters.setParity(oldParity);
throw new SerialConnectionException("Parametro non supportato");
}

// Setta il controllo di flusso dal settaggio contenuto in parameters
try {
sPort.setFlowControlMode(parameters.getFlowControlIn()
| parameters.getFlowControlOut());
} catch (UnsupportedCommOperationException e) { //lancia l'eccezione se non supportato
throw new SerialConnectionException("Flusso di controllo non supportato");
}
}//Fine metodo setConnectionparameters

/**
Chiude la porta e pulisce gli elementi associati.
*/
public void closeConnection() { //Metodo che chiude la connessione
// se la porta è già chiusa esce dalla funzione.
flag=0;
if (!open) {
return;
}


// Contolla che sPort ha referenza per evitare un NPE.
if (sPort != null) {//Se sPort si riferisce ad una porta
try {
// chiudi gli i/o streams.
            is.close();
} catch (IOException e) {
System.err.println(e);
}

// chiude la porta.
sPort.close(); //Chiude la porta

// Rimuove l'ascoltatore di padronanza della porta (ownership).
portId.removePortOwnershipListener(this);
}

open = false; //Dichiara la porta chiusa
}// fine metodo closeConnection


/*
Rapporta lo stato di apertura della porta.
ritorna true se la porta è aperta, false se la porta è chiusa.
*/
public boolean isOpen() {
return open;
}

/* Testa gli eventi SerialPortEvents. I due tipi di SerialPortEvents che questo programma è capace
di ascoltare sono DATA_AVAILABLE e BI. Durante il DATA_AVAILABLE il buffer della porta è letto
fino al suo svutamento, quando non sono disponibili più dati e sono passati 30 msec il metodo
ritorna. Quando un evento BI accade, la parola BREAK RECEIVED è scritta nel messageAreaIn. */

public void serialEvent(SerialPortEvent e) {
// Crea una StringBuffer e int per ricevere dati in input.
StringBuffer inputBuffer = new StringBuffer();
int newData = 0;
String toSend;

String toSent=new String("");
String confronto=new String("$GPGGA");
String confronto1=new String("$GPGLL");
String confronto2=new String("$GPRMC");

// determina il tipo di evento.
//if(!e.getEventType()) System.out.println("Non connesso");
switch (e.getEventType()) {

// Legge dati fino a quando è ritornato -1. Se è ricevuto \r sostituisce
// \n per un corretto maneggio dell newline.
```

```
case SerialPortEvent.DATA_AVAILABLE:
while (newData != -1) {
try {
newData = is.read();
if (newData == -1) {
break;
}
if ('\r' == (char)newData) { //Ha finito di trasmettere la riga
//System.out.println(inputBuffer+"\n\n\n");
StringTokenizer st = new StringTokenizer(new String(inputBuffer),",");
toSend=new String("");//Svuota la String precedente
String tag=new String(st.nextToken());
while(st.hasMoreTokens()){
if(tag.equals(confronto)){ /*Mediante queste righe si vengono presi soltanto 4 elementi:
a) Latitudine della posizione attuale
b) Emisfero della posizione attuale (Nord o Sud)
c) Longitudine della posizione attuale
d) Verso della posizione attuale (Esto o Ovest)
*/

                st.nextToken();
                 for(int i=0;i<4;i++){
toSend=toSend.concat(st.nextToken()+" ");
}
  old=new String(toSend);
break;

}else if(tag.equals(confronto1)){ /*Mediante queste righe si vengono presi soltanto 4 elementi:
a) Latitudine della posizione attuale
b) Emisfero della posizione attuale (Nord o Sud)
c) Longitudine della posizione attuale
d) Verso della posizione attuale (Esto o Ovest)
*/
for(int i=0;i<4;i++){
toSend=toSend.concat(st.nextToken()+" ");
}
//System.out.println(toSend);
  old=new String(toSend);
break;
}else if(tag.equals(confronto2)){ /*Mediante queste righe vengono presi soltanto 4 elementi:
a) Latitudine della posizione attuale
b) Emisfero della posizione attuale (Nord o Sud)
c) Longitudine della posizione attuale
d) Verso della posizione attuale (Esto o Ovest)
*/
st.nextToken();
st.nextToken();
for(int i=0;i<4;i++){
toSend=toSend.concat(st.nextToken()+" ");

}
old=new String(toSend);
//System.out.println(toSend);
break;
}
else{
toSend=new String(old);
//System.out.println("Continua ciclo");
break;
}
}
toSent=toSend;
//StringTokenizer s=new StringTokenizer(toSend);
//while(s.hasMoreElements())
//System.out.println(toSent);
//System.out.println(toSend);//Non si sa perchè ma toSend è formato da 4 righe uguali; a noi
//interessa soltanto la prima riga
inputBuffer.append('\n');
//Bisogna spedire al Server lo la Stringa toSend che contiene le
//coordinate del client, poi il server se lo gestisce

} else {
inputBuffer.append((char)newData);
}
```

```
} catch (IOException ex) {
System.err.println(ex);
return;
}

}
         try{
conn(toSent);

}
// Append received data to messageAreaIn.
finally{
//messageAreaIn.append(new String(inputBuffer)); //Appende i dati in ingresso nel
break;
}

}

}

/* Se è ricevuto un evento PORT_OWNERSHIP_REQUESTED è creato un box di dialogo domandando
all'utente se essi sono disposti a cedere la porta*/

public void ownershipChange(int type) {
if (type == CommPortOwnershipListener.PORT_OWNERSHIP_REQUESTED) {
PortRequestedDialog prd = new PortRequestedDialog(parent);
}
}


public   void conn(String arg) throws Exception, IOException{

if (flag==0){
messageAreaIn.append("********************************** \n\n\n");
messageAreaIn.append("Benvenuti su PocketVisit v1.0 \n for Pocket PC! \n \n
Connessione all'antenna\n GPS TOMTOM \n avvenuta con successo. \n \n Adesso è
possibile aprire\n Pocket Internet Explorer! ...\n\n");
flag++;
}
DatagramSocket datSocket;
byte[] data=new byte[1024];
data =arg.getBytes(); //Converte il messaggio in un array di byte
datSocket=new DatagramSocket(9998);
//crea il pacchetto
InetAddress address=InetAddress.getByName("169.254.69.4");
//InetAddress myAddress=InetAddress.getByName("Pocket_Pc_1.mshome.net");
DatagramPacket sendPacket= new DatagramPacket(data, data.length,/*InetAddress.getLocalHost()*/address,9999);
datSocket.send(sendPacket);
datSocket.close();
//messageAreaIn.append("Pacchetto Spedito a " + address.getHostAddress() +" contenente: " + arg + "\n");
//System.out.println("Io ho l'indirizzo: " + myAddress.getHostAddress());

}
}
```

A.1.4 SerialParameters.java

```
//PocketVisit 1.0 for PocketPc

import javax.comm.*;

/**
Classe che salva i parametri della porta seriale.
*/
public class SerialParameters {

    private String portName;
    private int baudRate;
    private int flowControlIn;
    private int flowControlOut;
    private int databits;
    private int stopbits;
    private int parity;
```

```
/**
Costruttore di default. Setta i parametri a no port, baud=9600, no controllo
flusso, 8 data bits, 1 stop bit, parità no.
*/
public SerialParameters () {
this("",
     9600,
     SerialPort.FLOWCONTROL_NONE,
     SerialPort.FLOWCONTROL_NONE,
     SerialPort.DATABITS_8,
     SerialPort.STOPBITS_1,
     SerialPort.PARITY_NONE );

}

/**
Costruttore parametrizzato.

portName= Nome della porta.
baudRate = Il baud rate.
flowControlIn = Tipo di controllo per il flusso in ingresso.
flowControlOut= Tipo di controllo per il flusso in uscita.
databits = Numero di data bits.
stopbits = Numero di stop bits.
parity=Tipo di parità
*/
public SerialParameters(String portName,
            int baudRate,
            int flowControlIn,
            int flowControlOut,
            int databits,
            int stopbits,
            int parity) {

    this.portName = portName;
    this.baudRate = baudRate;
    this.flowControlIn = flowControlIn;
    this.flowControlOut = flowControlOut;
    this.databits = databits;
    this.stopbits = stopbits;
    this.parity = parity;
}

/*
Setta il nome della porta.
portName = Nuovo nome della portaNew.
*/
public void setPortName(String portName) {
this.portName = portName;
}

/**
Restituisce nome della porta.
ritorna il corrente nome della porta.
*/
public String getPortName() {
return portName;
}

/**
Analogo per gli altri
*/
public void setBaudRate(int baudRate) {
this.baudRate = baudRate;
}


public void setBaudRate(String baudRate) {
this.baudRate = Integer.parseInt(baudRate);
}

public int getBaudRate() {
return baudRate;
```

```
}


public String getBaudRateString() {
return Integer.toString(baudRate);
}


public void setFlowControlIn(int flowControlIn) {
this.flowControlIn = flowControlIn;
}


public void setFlowControlIn(String flowControlIn) {
this.flowControlIn = stringToFlow(flowControlIn);
}


public int getFlowControlIn() {
return flowControlIn;
}


public String getFlowControlInString() {
return flowToString(flowControlIn);
}


public void setFlowControlOut(int flowControlOut) {
this.flowControlOut = flowControlOut;
}


public void setFlowControlOut(String flowControlOut) {
this.flowControlOut = stringToFlow(flowControlOut);
}


public int getFlowControlOut() {
return flowControlOut;
}


public String getFlowControlOutString() {
return flowToString(flowControlOut);
}


public void setDatabits(int databits) {
this.databits = databits;
}


public void setDatabits(String databits) {
if (databits.equals("5")) {
    this.databits = SerialPort.DATABITS_5;
}
if (databits.equals("6")) {
    this.databits = SerialPort.DATABITS_6;
}
if (databits.equals("7")) {
    this.databits = SerialPort.DATABITS_7;
}
if (databits.equals("8")) {
    this.databits = SerialPort.DATABITS_8;
}
}


public int getDatabits() {
return databits;
}
```

```
public String getDatabitsString() {
switch(databits) {
    case SerialPort.DATABITS_5:
    return "5";
    case SerialPort.DATABITS_6:
    return "6";
    case SerialPort.DATABITS_7:
    return "7";
    case SerialPort.DATABITS_8:
    return "8";
    default:
    return "8";
}
}


public void setStopbits(int stopbits) {
this.stopbits = stopbits;
}


public void setStopbits(String stopbits) {
if (stopbits.equals("1")) {
    this.stopbits = SerialPort.STOPBITS_1;
}
if (stopbits.equals("1.5")) {
    this.stopbits = SerialPort.STOPBITS_1_5;
}
if (stopbits.equals("2")) {
    this.stopbits = SerialPort.STOPBITS_2;
}
}


public int getStopbits() {
return stopbits;
}


public String getStopbitsString() {
switch(stopbits) {
    case SerialPort.STOPBITS_1:
    return "1";
    case SerialPort.STOPBITS_1_5:
    return "1.5";
    case SerialPort.STOPBITS_2:
    return "2";
    default:
    return "1";
}
}


public void setParity(int parity) {
this.parity = parity;
}


public void setParity(String parity) {
if (parity.equals("None")) {
    this.parity = SerialPort.PARITY_NONE;
}
if (parity.equals("Even")) {
    this.parity = SerialPort.PARITY_EVEN;
}
if (parity.equals("Odd")) {
    this.parity = SerialPort.PARITY_ODD;
}
}


public int getParity() {
return parity;
}
```

```
    public String getParityString() {
    switch(parity) {
        case SerialPort.PARITY_NONE:
        return "None";
        case SerialPort.PARITY_EVEN:
        return "Even";
        case SerialPort.PARITY_ODD:
        return "Odd";
        default:
        return "None";
    }
    }

    /**
    Converte una String descrivente un tipo di flusso di controllo in un tipo
    int definito in SerialPort.
    flowControl = Una String descrivente un tipo di flusso di controllo.
    ritorna un int descrivente un tipo di flusso di controllo.
    */
    private int stringToFlow(String flowControl) {
    if (flowControl.equals("None")) {
        return SerialPort.FLOWCONTROL_NONE;
    }
    if (flowControl.equals("Xon/Xoff Out")) {
        return SerialPort.FLOWCONTROL_XONXOFF_OUT;
    }
    if (flowControl.equals("Xon/Xoff In")) {
        return SerialPort.FLOWCONTROL_XONXOFF_IN;
    }
    if (flowControl.equals("RTS/CTS In")) {
        return SerialPort.FLOWCONTROL_RTSCTS_IN;
    }
    if (flowControl.equals("RTS/CTS Out")) {
        return SerialPort.FLOWCONTROL_RTSCTS_OUT;
    }
    return SerialPort.FLOWCONTROL_NONE;
    }

     /**
    Converte un int descrivente un tipo di flusso di controllo in un tipo
    String definito in SerialPort.
    flowControl = Una String descrivente un tipo di flusso di controllo.
    ritorna un int descrivente un tipo di flusso di controllo.
    */
    String flowToString(int flowControl) {
    switch(flowControl) {
        case SerialPort.FLOWCONTROL_NONE:
        return "None";
        case SerialPort.FLOWCONTROL_XONXOFF_OUT:
        return "Xon/Xoff Out";
        case SerialPort.FLOWCONTROL_XONXOFF_IN:
        return "Xon/Xoff In";
        case SerialPort.FLOWCONTROL_RTSCTS_IN:
        return "RTS/CTS In";
        case SerialPort.FLOWCONTROL_RTSCTS_OUT:
        return "RTS/CTS Out";
        default:
        return "None";
    }
    }
}
```

A.1.5 SerialConnectionException.java

```
//PocketVisit 1.0 for PocketPc


public class SerialConnectionException extends Exception {

    /**
     * Costrusce un <code>SerialConnectionException</code>
```

```
     * con il messaggio associato.
     *
     * s =  Messaggio.
     */
    public SerialConnectionException(String str) {
        super(str);
    }

    /**
     * Costruisce un <code>SerialConnectionException</code>
     * senza messaggio
     */
    public SerialConnectionException() {
        super();
    }
}
```

A.1.6 PocketVisit.java

```
//*****************************************
//*    PocketVisit v1.0 for Pocket Pc       *
//*                                         *
//*         ICAR-CNR di Palermo             *
//*                                         *
//*     Sviluppatore: Giovanni Aiello       *
//*                                         *
//*e-mail address: aiello.giovi@virgilio.it *
//*****************************************

import javax.comm.*;
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.util.Properties;
import java.util.Enumeration;



public class PocketVisit extends Frame implements ActionListener {

    final int HEIGHT = 500;
    final int WIDTH  = 460;

    private MenuBar mb;
    private Menu fileMenu;
    private Menu helpMenu;

    private MenuItem exitItem;
    private MenuItem aboutItem;

    private Button openButton;
    private Button stopButton;

    private Panel buttonPanel;

    private Panel    messagePanel;
    private TextArea messageAreaOut;
    private TextArea messageAreaIn;

    private ConfigurationPanel configurationPanel;
    private SerialParameters parameters;
    private SerialConnection connection;

    private Properties props = null;/*La classe Properties estende la classe Hashtable e
                                      rappresenta un set di proprietà persistente.
                                      Le proprietà possono essere salvate in uno Stream
                                      o caricate da uno Stream. Ogni chiave ed il suo
                                      corrispondente valore nella lista delle proprietà è
                                      una String */
```

```
    /*
    Metodo main. Controlla se come argomenti vengono passate le stringhe (-h, -help),
    e, se è così, visualizza il messaggio di utilizzo ed esce, altrimenti crea un nuovo
    PocketVisit e lo setta come visibile.
    */
    public static void main(String[] args) {
if ((args.length > 0)
    && (args[0].equals("-h")
    || args[0].equals("-help"))) {
    System.out.println("utilizzo: java PocketVisit");
    System.exit(1);
}

PocketVisit pocket_visit = new PocketVisit();
pocket_visit.setVisible(true);
pocket_visit.repaint();
    }


    public PocketVisit(){
super("PocketVisit");

parameters = new SerialParameters();/*inizializza parameters mediante il costruttore
                                      della classe SerialParameters che imposta i settaggi
                                      di default: Setta i parametri come: no port, 9600 baud,
                                      no flow control, 8 data bits, 1 stop bit, no parity. */
parameters.setBaudRate("4800"); //Baud Rate GPS

// Setta la GUI per il programma
addWindowListener(new CloseHandler(this));

mb = new MenuBar();

fileMenu = new Menu("File");
helpMenu = new Menu("Help");

exitItem = new MenuItem("Exit");
exitItem.addActionListener(this);
fileMenu.add(exitItem);

aboutItem=new MenuItem("About PocketVisit");
aboutItem.addActionListener(this);
helpMenu.add(aboutItem);


mb.add(fileMenu);
mb.add(helpMenu);

setMenuBar(mb);

messagePanel = new Panel();
messagePanel.setLayout(new GridLayout(1, 1));

messageAreaOut = new TextArea();
//messagePanel.add(messageAreaOut);

messageAreaIn = new TextArea();
messageAreaIn.setEditable(false);
messagePanel.add(messageAreaIn);

add(messagePanel, "Center");

configurationPanel = new ConfigurationPanel(this);

buttonPanel = new Panel();

openButton = new Button("Vai>>");
openButton.addActionListener(this);
buttonPanel.add(openButton);

stopButton = new Button("<<Ferma");
stopButton.addActionListener(this);
stopButton.setEnabled(false);
```

```
buttonPanel.add(stopButton);

Panel southPanel = new Panel();

GridBagLayout gridBag = new GridBagLayout();/* La classe GridBagLayout è un gestore
                                               flessibile di layout che allinea le
                                               componenti verticalmente e orizzontalmente
                                               senza il bisogno che le componenti siano
                                               della stessa grandezza. Ogni oggetto di
                                               questa classe mantiene una griglia
                                               rettangolare dinamica di celle con ogni
                                               campo occupante una o più celle,
                                               chiamato suo "DisplayArea" */

GridBagConstraints cons = new GridBagConstraints();/*la classe GridBagConstraints
                                                     specifica l'obbligo per le componenti
                                                     di usare la classe GridBagLayout*/

southPanel.setLayout(gridBag);

cons.gridwidth = GridBagConstraints.REMAINDER;/* cons.gridwidth specifica il numero di
                                                 celle in una riga per il displayArea del
                                                 componente GridBagConstraints.REMAINDER
                                                 indica che questo componente è l'ultimo
                                                 nella sua riga o colonna */


gridBag.setConstraints(configurationPanel, cons);//Configura in questa maniera il
                                                 //     configurationPanel
cons.weightx = 1.0;
southPanel.add(configurationPanel);
gridBag.setConstraints(buttonPanel, cons);
southPanel.add(buttonPanel);

add(southPanel, "South");


connection = new SerialConnection(this, parameters,
  messageAreaOut, messageAreaIn);/*Inizializza connection della classe
                                   SerialConnection passando i parametri,
                                   ed i textArea messageAreaIn e
                                   messageAreaOut*/
setConfigurationPanel();//Chiama la funzione di utilità

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();/*La classe Dimension
                                                                     racchiude la larghezza
                                                                     e l'altezza di un componente
                                                                     (in precisione intera)
                                                                     in un singolo oggetto*/

setLocation(/*screenSize.width/9*/ 5,
    /*screenSize.height/9*/ 25);       /* Muove questo componente in una nuova locazione.
                                           l'angolo in alto a sinistra della nuova locazione
                                           è specificato dai parametri x e y
                                           nello spazio delle coordinate del componente genitore
                                         */
setSize(WIDTH/2 , HEIGHT/2 );//Setta le dimensioni della finestra
    }//Fine costruttore di PocketVisit

    /**
    Setta gli elementi GUI nel configurationPanel.
    */
    public void setConfigurationPanel() {
configurationPanel.setConfigurationPanel();/*Richiama il metodo setConfigurationPanel della classe
                                             ConfigurationPanel*/
    }

    /**
    Risponde agli MenuItem e ai pulsanti
    */
    public void actionPerformed(ActionEvent e) {
String cmd = e.getActionCommand();// Restituisce il tipo di evento avvenuto

// Chiama shutdown, in modo da termianre il programma.
```

```
if (cmd.equals("Exit")) {
    shutdown();
}

if (cmd.equals("About PocketVisit")){
AlertDialog al=new AlertDialog(this, "About PocketVisit",
"Pocket Visit v1.0 for Pocket Pc", "Proprietà di: ICAR di Palermo",
"Istituto di Calcolo e Reti ad alte prestazioni");
}
// Apre una porta.
if (cmd.equals("Vai>>")) {
    openButton.setEnabled(false);
    Cursor previousCursor = getCursor();/*Cursor è una classe che racchiude la rappresentazione
                                          bitmap di un cursore del mouse e la associa a previousCursor*/
    setNewCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));/*chiama funzione di utilità che
                                                                   accetta dal sistema il cursore di default*/
    configurationPanel.setParameters();/*Funzione di utilità che riempie parameters dai settaggi
                                          presenti nel configurationPanel*/
    try {
     connection.openConnection();      /*Apre la connessione. connection è un oggetto di tipo
                                          SerialConnection quindi viene invocato il metodo
                                          openConnection() della classe SerialConnection*/
    } catch (SerialConnectionException e2) {//Gestisce l'evento di errore di apertura della porta
AlertDialog ad = new AlertDialog(this,
 "Errore nell'apertura della porta!",
 "Errore nell'apertura della porta,",
 e2.getMessage() + ".",
 "Selezioan un nuovo settaggio e riprova.");
        openButton.setEnabled(true);
setNewCursor(previousCursor);
return;
    }
    portOpened();
    setNewCursor(previousCursor);
}

// Chiude la porta.
if (cmd.equals("<<Ferma")) {
    portClosed();
}

}

    /*
    Setta i pulsanti nello stato di porta aperta.
    */
    public void portOpened() {
openButton.setEnabled(false);
stopButton.setEnabled(true);

    }

    /*
    Chiama closeConnection in SerialConnection e setta i bottoni nello stato.
    di porta chiusa
    */
    public void portClosed() {
connection.closeConnection();
openButton.setEnabled(true);
stopButton.setEnabled(false);
    }

    /**
    Setta il Cursor per l'applicazione.
    c= nuovo Cursor
    */
    private void setNewCursor(Cursor c) {//Setta un nuovo cursore e accetta un altro cursore
setCursor(c);//setta l'immagine del cursore per il cursore selezionato
messageAreaIn.setCursor(c);/*Setta l'immagine del cursore selezionato. Questa immagine è
                             visualizzata quando il metodo "contains"
                             per questo componente torna true per la locazione del cursore
                             corrente e questo componente è visibile, visualizzabile
                             e abilitato*/
messageAreaOut.setCursor(c);
```

```
    }



    /**
    Cleanly shuts down the applicaion. first closes any open ports and
    cleans up, then exits.
    */
    private void shutdown() {
connection.closeConnection();
System.exit(1);
    }


    class ConfigurationPanel extends Panel implements ItemListener {/*Classe che eredita i metodi
                                                                       e gli attributi da Panel
                                                                       (quindi è un pannello) e
                                                                       implementa l'interfaccia
                                                                       che gestisce gli eventi di
                                                                       selezione degli item */

private Frame parent;

private Label portNameLabel;
private Choice portChoice; /*Un oggetto Choice è un oggetto che contiene tante voci
    -----------
    |     | > | e serve per la selezione delle opzioni
    -----------

    */


/*
Crea e inizializza il pannello di configurazione.
*/
public ConfigurationPanel(Frame parent) {/*Il costruttore di questa classe accetta un Frame
                                           che nel nostro caso è PocketVisit*/
    this.parent = parent;//Fa una copia del Frame PocketVisit e lo chiama parent

    setLayout(new GridLayout(1, 2));

    portNameLabel = new Label("Port Name:", Label.LEFT);/*Inizializza il Label portNameLabel
                                                          e immette la stringa "Port Name:" e lo mette
                                                          a sinistra della cella*/
    add(portNameLabel);//aggiunge nel Pannello ConfigurationPanel l'etichetta portNameLabel

    portChoice = new Choice();//Inizializza il Choice portChoice relativo alla lista di porte
    portChoice.addItemListener(this);/*Aggiunge un ascoltatore di item per le porte in questo
                                       pannello ConfigurationPanel*/
    add(portChoice);
    listPortChoices();/*Serve per rilevare le porte disponibili, è una funzione di utilità di
                        questa classe ConfigurationPanel*/
    portChoice.select(parameters.getPortName());/*Viene selezionato l'item che corrispode alla
                                                  porta contenuta in parameters*/

}

/*
Setta il pannello di configurazione con i settaggi dell'oggetto parameters
*/
public void setConfigurationPanel() {
    portChoice.select(parameters.getPortName());
}


        public void setParameters() {/*Riempie parameters con i settaggi selezionati nel
                                       configurationPanel*/
            parameters.setPortName(portChoice.getSelectedItem());/*Salva in parameters la
                                                                   configurazione del nome della porta
                                                                    da utilizzare*/
        }


        void listPortChoices() { //Serve per rilevare le porte disponibili
```

```
            CommPortIdentifier portId;/*Dichiara un oggetto CommPortIdentifier (identificatore di
                                       porte di comunicazione)la classe relativa è nelle
                                       API javax.comm  ed è una classe per la gestione delle
                                       porte di comunicazione. Questa è la classe centrale per
                                       il controlo dell'accesso alle porte di comunicazione
                                       Esso include metodi per:
                                        a)Determinare le porte disponibili dal driver
                                        b) Apertura delle porte di comunicazione per operazioni di I/O
                                        c) Determinazione padronanda porta (ownership)
                                        d) Risolve il problema di contendimento della padronanza della
                                           porta
                                        e) Eventi di gestione che indica cambiamenti negli stati della
                                           padronanza della porta
                                        */

            Enumeration en = CommPortIdentifier.getPortIdentifiers();/*Classe di enumerazione di
                                                                       elementi (è un'interfaccia).
                                                                       Un oggetto che implementa questa
                                                                       classe genera
                                                                       una serie di elementi; una alla
                                                                       volta chiamate al metodo nextElement()
                                                                       ritorna il successivo elemento
                                                                       Il metodo getPortIdentifiers() ottiene
                                                                       un oggetto di tipo Enumerazion che
                                                                       contiene un oggetto CommPortIdentifier
                                                                       per ogni porta nel sistema.  */

  // itera attraverso le porte.
            while (en.hasMoreElements()) {/*attraversa tutto l'oggetto della classe Element fino a quando
                                            ci sono elementi*/
                portId = (CommPortIdentifier) en.nextElement();/*Assegna ogni oggetto di en a portId
                                                                 (il cast necessita xchè en è un oggetto generico)*/
                if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {/* ritorna il tipo di porta
                                                                               (seriale o parallela) e se è una porta seriale*/
                    portChoice.addItem(portId.getName()); /*Aggiunge il nome della porta nel choice relativo
                                                            all'elenco di porte*/
                }
            }
            portChoice.select(parameters.getPortName());/*setta l'item selezionato in questo menu choice per
                                                          essere l'item il quale nome èuguale alla Stringa
                                                          specificata, quindi parameters.getPortName()
                                                          restituisce una Stringa con il nome della porta*/
        }

/* Gestione dell'evento di  cambiamento di selezione nei Choice
        Se una porta è aperta, la porta non può essere cambiata
        se il Choice non è supportato nella piattaforma l'utente è notoficato ed i settaggi
        saranno riportati al loro stato precedente.
   */
public void itemStateChanged(ItemEvent e) {
  // Controlla se la porta è aperta.
    if (connection.isOpen()) {
 // Se la porta è aperta non permette il cambiamento della porta.
     if (e.getItemSelectable() == portChoice) {/*Se viene selezionato il Choice relativo alla porta
                                                 mentre che questa è aperta, viene notificato l'utente*/

    AlertDialog ad = new AlertDialog(parent, "Porta Aperta!",
       "La porta non può",
     "essere cambiata",
     "mentre la porta è aperta.");

    // Il configurationPanel torna alla vecchia impostazione.
    setConfigurationPanel();//Funzione di utilità che setta  il configurationPanel da parameters
    return;
}
// Se la connessione non è aperta, setta parameters dal configurationPanel.
setParameters();
try {
    // prova a cambiare i settaggi di una porta aperta.
    connection.setConnectionParameters();/*Setta i parametri di connessione dall'oggetto
                                           parameters che lui sa al momento della costruzione
                                           dell'oggetto connection*/
} catch (SerialConnectionException ex) {
     // Se i settaggi non possono essere cambiati, si avverte l'utente e ritorna
```

```
         // i settaggi che precedentemente erano nel Choice.
    AlertDialog ad = new AlertDialog(parent,
"Configurazione non supportata!",
"Parametrozione non supportato,",
"seleziona nuovo valore.",
"Ritorno alla precedente configurazione.");
    setConfigurationPanel();//Setta il configurationPanel da parameters
}
    } else {
 // Poichè la porta non è aperta setta l'oggetto parameters.
     setParameters();
    }
}
    }

    /*
    Permette all'applicazione di essere chiusa mediante il box di chiusura
    */
    class CloseHandler extends WindowAdapter {

PocketVisit sd;

public CloseHandler(PocketVisit sd) {
    this.sd = sd;
}

public void windowClosing(WindowEvent e) {
    sd.shutdown();
}
    }
}
```

A.2 Applicazione middle

A.2.1 CalcoloDistanza.java

```
//*****************************************
//*         CalcoloDistanza classv          *
//*                                         *
//*         ICAR-CNR di Palermo             *
//*                                         *
//*     Sviluppatore: Giovanni Aiello       *
//*                                         *
//*e-mail address: aiello.giovi@virgilio.it *
//*****************************************

/*
* Classe di utilità per il calcolo della distanza tra duwe punti nelle superficie terrestre
*/

import java.util.*;

public class CalcoloDistanza{


double lat1rad;
double distanza;
double lon1rad;
double lat2rad;
double lon2rad;

public CalcoloDistanza(double lat1, double lon1, double lat2, double lon2){
lat1rad=lat1*Math.PI/180;

lon1rad=lon1*Math.PI/180;

lat2rad=lat2*Math.PI/180;

lon2rad=lon2*Math.PI/180;

getDistance();
```

```
}

public double getDistance(){
double p1 = Math.cos(lat1rad)*Math.cos(lon1rad) *Math.cos(lat2rad)*Math.cos(lon2rad);
   double p2 = Math.cos(lat1rad)*Math.sin(lon1rad) *Math.cos(lat2rad)*Math.sin(lon2rad);
   double p3 = Math.sin(lat1rad)*Math.sin(lat2rad);
distanza=(Math.acos(p1+p2+p3)*6371);
System.out.println("Distanza="+distanza);
return distanza;
}
}
```

A.2.2 Middle.java

```
//*****************************************
//*           Middle side                 *
//*                                       *
//*         ICAR-CNR di Palermo           *
//*                                       *
//*     Sviluppatore: Giovanni Aiello     *
//*                                       *
//*e-mail address: aiello.giovi@virgilio.it *
//*****************************************


import java.net.*;
import java.io.*;
import java.util.*;
import java.lang.*;
import java.text.*;


public class Middle{

private DatagramSocket datSocket;

private ObjectOutputStream output;

private double vertici[][]; //Matrice 4x4 che contiene latitudine e longitudine dei 4 vertici espressi
//nel sistema di coordinate cartografico

private double vertici1[][];//Matrice 4x2 che contiene latitudine e longitudine dei 4 vertici espressi
//nel sistema di coordinate traslato

private double vertici2[][];//Matrice 4x2 che contiene latitudine e longitudine dei 4 vertici espressi
//nel sistema di coordinate rototraslato

private double verticiGradi[][]; //Matrice 4x2 contenete le coordinate in gradi

private double verticiGradi2[][]; //Matrice 4x2 contenete le coordinate in gradi

int finder; //Identifica la coordinata rispetto alla quale bisogna fare la traslazione degli assi

private double larghezza;//Contiene la larghezza espressa in metri del luogo
private double altezza; //Contiene l'altezza espressa in metri del luogo

int ID; //Identificativo file client
private double grana; //Grana espressa in metri. La grana corrisponderà al lato della singola cella
//che divide il luogo

private double g; //grana espressa in gradi
private double alpha; //Angolo di rotazione assi cartografici
//Costruttore della classe Middle. Il costruttore si deve occupare dell' inizializzazione
//del sistema, infatti carica la mappa dal file ed effettua la trasformazione cartografica delle
//coordinate dei punti. Il costruttore accetta il nome del file della mappa
double latitudine; //Latitudine corrente dell'antenna GPS espressa in gradi
double longitudine;//Longitudine corrente dell'antenna GPS espressa in gradi
double latitudine1; //Latitudine corrente dell'antenna GPS espressa in gradi e nel sistema di
//coordinate traslato

double longitudine1;//Longitudine corrente dell'antenna GPS espressa in gradi e nel sistema di
//coordinate traslato
double latitudine2; //Latitudine corrente dell'antenna GPS espressa in gradi e nel sistema di
```

```
//coordinate rototraslato
double longitudine2;//Longitudine corrente dell'antenna GPS espressa in gradi e nel sistema di
//coordinate rototraslato

public Middle(String mappa) throws IOException{

BufferedReader filebuf =
new BufferedReader(new FileReader(mappa)); //Legge la mappa dal file

/* equivale alla coppia di istruzioni:
 *
 * FileReader filein = new FileReader("giovanni.txt");
 * BufferedReader filebuf = new BufferedReader(filein);
 */

String vertici12 = new String(filebuf.readLine()); // Legge la Prima riga del file che contiene
// le coordinate dei primi 2 punti
String vertici34 = new String(filebuf.readLine()); // Legge la Seconda riga del file che contiene
// le coordinate degli altri 2 punti
String dimensioni=filebuf.readLine();//Legge le dimensioni del luogo
grana=Double.parseDouble(filebuf.readLine()); //Legge la grana
  filebuf.close();  // chiude il file
//
//Bisogna sistemare i vertici e le dimensioni
StringTokenizer dim=new StringTokenizer(dimensioni);
altezza=Double.parseDouble(dim.nextToken());
larghezza=Double.parseDouble(dim.nextToken());

//Viene suddivisa il token la stringa vertici ottenendo 4 tokens  del tipo xx,xx.xxxx
StringTokenizer coordinate=new StringTokenizer(vertici12);
//
//        //Il primo token rappresenta la latitudine del primo punto in gradi, minuti primi e millesimi di
// primi
//        //Bisogna ulteriormente dividere in tokens questo primo token affinchè si possano individuare i
//gradi e i minuti primi
//        StringTokenizer temp=new StringTokenizer(coordinate.nextToken(),","); //Token nella forma xx,xx.xxxx
//        vertici[0][0]=Double.parseDouble(temp.nextToken()); //Gradi della latitudine
//        vertici[0][1]=Double.parseDouble(temp.nextToken()); //Minuti primi della latitudine
//
//
//        temp=new StringTokenizer(coordinate.nextToken(),","); //Token nella forma xx,xx.xxxx. Contiene la
// longitudine del primo punto
//        vertici[0][2]=Double.parseDouble(temp.nextToken()); //Gradi della latitudine
//        vertici[0][3]=Double.parseDouble(temp.nextToken()); //Minuti primi della latitudine
//
//        temp=new StringTokenizer(coordinate.nextToken(),","); //Token nella forma xx,xx.xxxx
//        vertici[1][0]=Double.parseDouble(temp.nextToken()); //Gradi della latitudine
//        vertici[1][1]=Double.parseDouble(temp.nextToken()); //Minuti primi della latitudine
//
//        temp=new StringTokenizer(coordinate.nextToken(),","); //Token nella forma xx,xx.xxxx. Contiene la
//longitudine del primo punto
//        vertici[1][2]=Double.parseDouble(temp.nextToken()); //Gradi della latitudine
//        vertici[1][3]=Double.parseDouble(temp.nextToken()); //Minuti primi della latitudine
//
//        //Alla prossima riga del file vi sono le coordinate degli altri 2 punti
//        coordinate=new StringTokenizer(vertici34);
//
//        //Il primo token rappresenta la latitudine del primo punto in gradi, minuti primi e millesimi di primi
//        //Bisogna ulteriormente dividere in tokens questo primo token affinchè si possano individuare i
//gradi e i minuti primi
//        temp=new StringTokenizer(coordinate.nextToken(),","); //Token nella forma xx,xx.xxxx
//        vertici[2][0]=Double.parseDouble(temp.nextToken()); //Gradi della latitudine
//        vertici[2][1]=Double.parseDouble(temp.nextToken()); //Minuti primi della latitudine
//
//        temp=new StringTokenizer(coordinate.nextToken(),","); //Token nella forma xx,xx.xxxx. Contiene la
//longitudine del terzo punto
//        vertici[2][2]=Double.parseDouble(temp.nextToken()); //Gradi della latitudine
//        vertici[2][3]=Double.parseDouble(temp.nextToken()); //Minuti primi della latitudine
//
//        temp=new StringTokenizer(coordinate.nextToken(),","); //Token nella forma xx,xx.xxxx
//        vertici[3][0]=Double.parseDouble(temp.nextToken()); //Gradi della latitudine
//        vertici[3][1]=Double.parseDouble(temp.nextToken()); //Minuti primi della latitudine
//
//        temp=new StringTokenizer(coordinate.nextToken(),","); //Token nella forma xx,xx.xxxx. Contiene la
```

```
//longitudine del quarto punto
//        vertici[3][2]=Double.parseDouble(temp.nextToken()); //Gradi della latitudine
//        vertici[3][3]=Double.parseDouble(temp.nextToken()); //Minuti primi della latitudine
//
//        //Le coordinate sono espresse rispetto al sistena di riferimento cartografico
//
//        //Adesso bisogna effettuare una rototraslazione del sistema di coordinate cartografico affichè si
// possa facilitare
//        //la gestione delle coordinate e la griglizzazione secondo la prescelta grana g.
//
//        //Prima si effettua la traslazione in entrambe le direzioni (x,y).
//        //Per fare questo bisogna individuare il punti con ordinata (latitudine) più piccola rispetto al sistema di
//        //coordinate cartografico. Questo punto sarà l'origine degli assi per il nuovo sistema di coordinate che
//        //andremo a determinare
//

verticiGradi=new double[4][2];//verticiGradi contiene latitudini e longitudini in gradi
int finder=0;
//       for(int i=0;i<4;i++){
//       verticiGradi[i][0]=vertici[i][0]+((vertici[i][1])/60);
//       verticiGradi[i][1]=vertici[i][2]+((vertici[i][3])/60);
//      }

StringTokenizer coord12=new StringTokenizer(vertici12);
StringTokenizer coord34=new StringTokenizer(vertici34);
verticiGradi[0][0]=Double.parseDouble(coord12.nextToken());
verticiGradi[0][1]=Double.parseDouble(coord12.nextToken());
verticiGradi[1][0]=Double.parseDouble(coord12.nextToken());
verticiGradi[1][1]=Double.parseDouble(coord12.nextToken());
verticiGradi[2][0]=Double.parseDouble(coord34.nextToken());
verticiGradi[2][1]=Double.parseDouble(coord34.nextToken());
verticiGradi[3][0]=Double.parseDouble(coord34.nextToken());
verticiGradi[3][1]=Double.parseDouble(coord34.nextToken());
//System.out.println(verticiGradi[3][0]+"  ,  "+verticiGradi[3][1]);

for(int i=1;i<4;i++) //Cerca il punto con ordinata minore
if (verticiGradi[finder][0]>verticiGradi[i][0]) finder=i;

//System.out.println("finder="+finder);

//       verticiGradi[1][0]=((verticiGradi[0][1]-verticiGradi[2][1])+(verticiGradi[2][0]+verticiGradi[0][0]))/2;
//       verticiGradi[3][0]=(verticiGradi[2][0]+verticiGradi[0][0])-verticiGradi[1][0];
//       verticiGradi[3][1]=(verticiGradi[0][1]-verticiGradi[0][0])+verticiGradi[3][0];
//       verticiGradi[1][1]=(verticiGradi[2][1]+verticiGradi[0][1])-verticiGradi[3][1];
DecimalFormat fiveDigits=new DecimalFormat("0.00000");
verticiGradi2=new double[4][2];
for(int i=0;i<4;i++)
for(int j=0;j<2;j++)
verticiGradi2[i][j]=Double.parseDouble((fiveDigits.format(verticiGradi[i][j])).replace(',','.'));

//        for(int i=0;i<4;i++)
//        System.out.println(verticiGradi2[i][0]+"  "+verticiGradi2[i][1]);
//       double t=Double.parseDouble(two.format(verticiGradi[1][0]));
//        System.out.println("t="+t);

       //finder è l'indice di riga che rappresenta il punto con minore valore di latitudine

/*
 * Bisogna effettuare la traslazione del sistema di coordinate di una quantità pari a latitudine[finder]
 * e longitudine[finder] in maniera tale da fare coincidere il punto identificato da finder con l'origine
 * del nuovo sistema di riferimento. Per fare questo si pongono subito a zero le nuove coordinate di finder
 */
vertici1=new double[4][2]; //Matrice inizializzata a zero
for(int i=0;i<4;i++)
for(int j=0;j<2;j++)
vertici1[i][j]=0.0;

//Adesso bisogna esprimere le coordinate dei punti rispetto al nuovo sistema di riferimento traslato
for(int i=0;i<4;i++){
if (i==finder) continue;
for(int j=0;j<2;j++)
vertici1[i][j]=verticiGradi2[i][j]-verticiGradi2[finder][j];
}
```

```
/*
 * Adesso bisogna fare una rotazione del nuovo sistema di coordinate però prima bisogna
 * determinare il punto a est del punto finder con ascissa (longitudine) più alta rispetto al sistema di
 * coordinate cartografico.
 */

int finder2=0;

for(int i=1;i<4;i++) //Cerca il punto con ascissa maggiore
if (verticiGradi2[finder2][1]<verticiGradi2[i][1]) finder2=i;

//        System.out.println("finder2= "+finder2);

/*
* Determinato il punto con maggiore valore di ascissa, si procede con l'effettuare la rotazione del
* sistema di riferimento
* Questa rotazione deve essere di un angolo alpha tale che:
  */

if(verticiGradi2[finder2][0]<verticiGradi2[finder][0]) System.out.println("Attenzione ERRORE!!");
//Non dovrebbe mai accadere
  alpha=Math.atan((verticiGradi2[finder2][0]-verticiGradi2[finder][0])/(verticiGradi2[finder2][1]-
  verticiGradi2[finder][1]));
  //Angolo in radianti

//        System.out.println("alpha="+alpha);

/*
* A questo punto è possibile esprimere tutti i punti rispetto al nuovo sistema di riferimento
* Il punto identificato da finder sarà l'origine degli assi rispetto al nuovo sistema di riferimento
*traslo-ruotato
*/

vertici2=new double[4][2]; //Matrice inizializzata a zero
for(int i=0;i<4;i++)
for(int j=0;j<2;j++)
vertici2[i][j]=0.0;

for(int i=0;i<4;i++){//Per tutti i punti
vertici2[i][0]=Math.cos(alpha)*vertici1[i][0]-Math.sin(alpha)*vertici1[i][1];
vertici2[i][1]=Math.cos(alpha)*vertici1[i][1]+Math.sin(alpha)*vertici1[i][0];
}//Coordinate traslo-ruotate

//        for(int i=0;i<4;i++)
//         System.out.println("Latitudine:"+vertici2[i][0]+" Longitudine:"+vertici2[i][1]);


/*
* Adesso entra in gioco la grana g. Questa grana la si deve esprimere in gradi.
* Dato che abbiamo l'altezza in metri, e sappiamo a quanti gradi corrispondono (latitudine di un
*qualsiasi punto la
* cui latitudine non è nulla), allora: latitudine:L(m)=x:g(m)
  */

CalcoloDistanza dist=new CalcoloDistanza(verticiGradi2[0][0],verticiGradi2[0][1],verticiGradi2[1][0],
verticiGradi2[1][1]);
altezza=dist.getDistance()*1000;
//System.out.println("Altezza: "+altezza);
dist=new CalcoloDistanza(verticiGradi2[0][0],verticiGradi2[0][1],verticiGradi2[3][0],verticiGradi2[3][1]);
larghezza=dist.getDistance()*1000;
//System.out.println("Larghezza: "+larghezza);

//for(int i=0;i<4;i++)
//  if(vertici2[i][0]>0.0 && vertici2[i][0]>0.0){ //Dovrebbe prendere o il vertice opposto all'origine
// degli assi traslo-ruotati, oppure il punto 2
// g=vertici2[i][0]*grana/altezza;
g=vertici2[1][0]*grana/altezza;
// break;
//  }

// System.out.println("Grana="+g);

//La mappa è stata creata, è possibile adesso iniziare la lettura dal socket UDP
}
```

```
public void waitForPacket(){
try{
datSocket= new DatagramSocket(9999);
System.out.println("Server UDP in ascolto alla porta 9999");
}
catch(SocketException e){
e.printStackTrace();
System.exit(1);
}
int riga,colonna;


/*
* Crea una Hash table che contiene gli indirizi IP registrati e connessi e un identificativo del file che
* poi dovrà leggere la servlet. La struttura della hash table sarà del tipo:

IP_Address_1 ----> 1
IP_Address_2 ----> 2
IP_Address_3 ----> 3
.          .
.          .
.          .
IP_Address_n ----> n

* Il valore i relativo all'i-esimo IP Address identificherà univocamente un file che conterrà le informazioni
*  del client con quell'indirizzo IP
*/

Hashtable table=new Hashtable();

while(true){
//Riceve pacchetto, visualizza contenuto e salva in un file

try{
//imposta pacchetto
byte data[]=new byte[100];
DatagramPacket receive=new DatagramPacket(data, data.length);
datSocket.receive(receive); //attende il pacchetto
//synchronized(this){
//Controlla a quale client appartiene il pacchetto

// LA STRINGA DI SOTTO E' DA METTERE, ERA STATA SOSTITUITA CON LA SUCCESSIVA PER FARE LE PROVE IN MACCHINA LOCALE
String IPHost=(receive.getAddress()).toString();//Indirizzo IP del client
// String IPHost=new String("/127.0.0.1");


/*
* Controlla se questo indirizzo è contenuto nella Hash Table
* Se la tabella contiene la parola, estrapola l'ID associato
*/

if(!table.containsKey(IPHost))
table.put(IPHost, new Integer(table.size()+1)); //Aggiunge coppia chiave-valore
ID=((Integer) table.get(IPHost)).intValue(); //Ottiene valore della chiave

//Immette la hash in un file (hash.txt) per essere letto dalla servlet
Enumeration keys = table.keys();
String output;
FileWriter fileoutput = new FileWriter(new File(new URI("file:/C:/jakarta-tomcat-4.1.27/bin/hash.txt")));
BufferedWriter filebuf = new BufferedWriter(fileoutput);
PrintWriter printout = new PrintWriter(filebuf);
// itera attraverso le chiavi
  while ( keys.hasMoreElements() ) {
output=new String("");
Object currentKey = keys.nextElement();
  // crea stringa con tutte le coppie chiave-valore
output=output.concat(currentKey + " " + table.get( currentKey ));
printout.println(output);
}
printout.close();


// //Testa Hash Table
```

```
// Enumeration keys = table.keys();
//
// // itera attraverso le chiavi
//  while ( keys.hasMoreElements() ) {
// Object currentKey = keys.nextElement();
//
// // visualizza le coppie chiave-valore
// System.out.println(currentKey + "\t" + table.get( currentKey ));
// }

//Visualizza informazioni sul pacchetto ricevuto
System.out.println("Pacchetto ricevuto da:\n" +
"Host: " + receive.getAddress()+"\n" +
"Contenente la Stringa: " + new String(receive.getData(),0,receive.getLength()));
String nextStr;
synchronized(this){
String str=new String(receive.getData(),0,receive.getLength()); //Stringa contenente le coordinate
del client
if(str==null || str.equals("")) continue; //Non dovrebbe mai accadere, ma se dovesse accadere lo ignora
//Si suddivide in token str ottenendo 4 tokens
StringTokenizer divided=new StringTokenizer(str);

//lat è la stringa xxxx.xxxx
String lat=divided.nextToken();

//Si suddivide ulteriormente in token con argomento "."
StringTokenizer latitude=new StringTokenizer(lat,"."); //Si ottengono i token xxxx e xxxx

//Il token xxxx lo si deve dividere in due parti xx xx corrispondenti ai gradi ed ai minuti primi
String temp=new String(latitude.nextToken());
String latgrad=temp.substring(0,2);
String latprimi=temp.substring(2);
double gradi=Double.parseDouble(latgrad);//Gradi
double minutiPrimi=(Double.parseDouble(latprimi));//Minuti primi
double millesimiPrimi=(Double.parseDouble(latitude.nextToken()))/10000;//millesimi di minuti primi
double decGrad=(minutiPrimi+millesimiPrimi)/60;
latitudine=gradi+decGrad; //Latitudine dell'antenna GPS espressa in gradi

//La stessa cosa per la longitudine
divided.nextToken();//Emisfero della posizione attuale(indica il verso di positività degli assi cartografici

//lon è la stringa xxxxx.xx
String lon=divided.nextToken();

//Si suddivide ulteriormente in token con argomento "."
StringTokenizer longitude=new StringTokenizer(lon,"."); //Si ottengono i token xxxxx e xx

//Il token xxxxx lo si deve dividere in due parti xxx xx corrispondenti ai gradi ed ai minuti primi
String temp2=new String(longitude.nextToken());
String longrad=temp2.substring(0,3);
String lonprimi=temp2.substring(3);
gradi=Double.parseDouble(longrad);//Gradi
minutiPrimi=(Double.parseDouble(lonprimi));//Minuti primi
millesimiPrimi=(Double.parseDouble(longitude.nextToken()))/10000;//millesimi di primi
decGrad=(minutiPrimi+millesimiPrimi)/60;
longitudine=gradi+decGrad;//longitudine dell'antenna GPS espressa in gradi

/*
* Per convenienza le coordinate sono espresse in gradi con 5 cifre di precisione
*/
DecimalFormat fiveDigits=new DecimalFormat("0.00000");
latitudine=Double.parseDouble(fiveDigits.format(latitudine).replace(',','.'));
longitudine=Double.parseDouble(fiveDigits.format(longitudine).replace(',','.'));

//        System.out.println(latitudine+"   "+longitudine);

/*
* Avendo il punto espresso in gradi, bisogna portarlo rispetto al sistema di riferimento traslo-ruotato
*/

latitudine1=latitudine-verticiGradi2[finder][0];
longitudine1=longitudine-verticiGradi2[finder][1];

//Rotazione
```

```
latitudine2=Math.cos(alpha)*latitudine1-Math.sin(alpha)*longitudine1;
longitudine2=Math.cos(alpha)*longitudine1+Math.sin(alpha)*latitudine1;


/*
* Adesso che le coordinate sono espresse rispetto al sistema di riferimento traslo-ruotoato,
* basta fare un ceil del rapporto tra la coordinata espressa nel sistema di coordinate traslo-ruotato
* e la grana espressa in gradi
*/

if(latitudine2>0)
riga=(int)(Math.ceil(latitudine2/g));
else
riga=(int)(Math.floor(latitudine2/g));
if(longitudine2>0)
colonna=(int)(Math.ceil(longitudine2/g));
else
colonna=(int)(Math.floor(longitudine2/g));

/*
* Bisogna salvare la riga e la colonna in un file secondo la sintassi: riga colonna
*/

nextStr=String.valueOf(riga);
String nextStr2=String.valueOf(colonna);
nextStr=nextStr.concat(" "+nextStr2+" ");//Stringa contenente riga colonna
nextStr=nextStr.concat(latitudine+" "+longitudine); //Cella di appartenenza e posizione antenna GPS
}
try{
File file=new File(new URI("file:/C:/jakarta-tomcat-4.1.27/bin/cella"+ID+".txt"));
FileWriter fileout = new FileWriter(file);
for (int i = 0; i < nextStr.length(); i++)
fileout.write(nextStr.charAt(i));
fileout.close();  // chiude il file

}catch(IOException e){
System.out.print(e.toString());
e.printStackTrace();
}
}
catch(IOException e2){
System.out.print(e2.toString());
e2.printStackTrace();
}
catch(URISyntaxException e3){}
}

}
public static void main (String args[]) throws IOException {
if(args.length!=1){
System.out.println("\n\n\n ****************************************** ");
System.out.println(" *   PocketVisit v1.0 \"Middle\"         * ");
System.out.println(" *        ICAR-CNR  -  Palermo           * ");
System.out.println(" *    Sviluppatore: Giovanni Aiello      * ");
System.out.println(" *e-mail address: aiello.giovi@virgilio.it* ");
System.out.println(" ****************************************** \n\n\n");
System.out.println("Utilizzo: Middle \"file_mappa\" ");
System.exit(0);
}
Middle middle=new Middle(args[0]);
middle.waitForPacket();
}
}
```

A.3 Parte Server del sistema proposto

A.3.1 HomePageServer.java

```
//*****************************************
```

```
//*          Home Page Servlet              *
//*                                         *
//*         ICAR-CNR di Palermo             *
//*                                         *
//*     Sviluppatore: Giovanni Aiello       *
//*                                         *
//*e-mail address: aiello.giovi@virgilio.it *
//*******************************************

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class HomePageServlet extends HttpServlet{

int count=0; //Contatore di utenti collegati durante il giorno

protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException{
count++;
Cookie myCookie=null;
String nome="mioCookie";
String valore="1";
myCookie=new Cookie(nome,valore);
myCookie.setMaxAge(1000);
response.addCookie(myCookie);
PrintWriter out=response.getWriter();
//Spedisce pagina HTML

response.setContentType("text/html");
String hostName=request.getRemoteHost();
out.println("<html>");
out.println("<head>");
out.println("<meta http-equiv=\"Content-Language\" content=\"it\">");
out.print("<meta http-equiv=\"Content-Type\" content=\"text/html; ");
out.println("charset=windows-1252\">");
out.println("<meta name=\"GENERATOR\" content=\"Microsoft FrontPage 5.0\">");
out.println("<meta name=\"ProgId\" content=\"FrontPage.Editor.Document\">");
out.println("<meta name=\"Microsoft Theme\" content=\"ice 011\">");
out.println("</head>");
out.println("<body>");
out.println("<p align=\"center\"></p>");
out.println("<p align=\"center\"><b><font size=\"5\">");
out.print("<img border=\"0\" src=\"images/animated.gif\" align=\"left\" ");
out.print("width=\"48\" height=\"55\"></font><font size=\"6\">Home Page di ");
out.print("prova." + "\n Oggi sei il visitatore n." + count + "&quot;</i>");
out.println("</font></b></p>");
out.println("<p align=\"center\">");
out.println(" </p>");
out.println("<p align=\"center\"><font size=\"1\">");
out.print("<!--webbot bot=\"Timestamp\" S-Type=\"EDITED\" ");
out.println("S-Format=\"%d/%m/%Y %H.%M\" --></font></p>");
out.println("<p align=\"center\"><b><font size=\"5\">");
out.print("<img border=\"0\" src=\"images/Pocket.jpg\" width=\"50\" height=\"70\" ");
out.println("hspace=\"30\"></font></b></p>");
out.println("<p align=\"center\">");
out.print("<img border=\"0\" src=\"images/PocketLogo.jpg\" width=\"87\" height=\"33\" ");
out.println("vspace=\"-20\"></p>");
out.println("<p align=\"center\"> </p>");

out.print("<p align=\"left\">Benvenuto "+ hostName +". Ricorda che per utilizzare ");
out.print("questo servizio bisogna installare sul proprio palmare il software PocketVisit ");
out.println("v1.0 for Pocket Pc scaricabile da <b>");
out.println("<a href=\"http://169.254.69.4:8080/home/PocketVisit/\">&quot;<i>qui</i>&quot;.");
out.println("</a>.");
out.print("Nello stesso sito è possibile anche scaricare una documentazione in formato pdf ");
out.println("riguardo l'utilizzo dello stesso. </b></p>");
out.println("<p align=\"left\">PocketPc v.1.0 è un software che permette di fare conoscere al ");
out.println("nostro web server il punto in cui sei nella mappa caricata al fine di poterti ");
out.println("inviare informazioni &quot;automaticamente&quot; riguardo ciò che stai guardando.</p>");
out.print("<p align=\"center\"><img border=\"0\" src=\"images/PocketLogo.jpg\" width=\"150\" ");
out.println("height=\"70\"></p>");
out.println("<p align=\"center\"> </p>");
out.print("<p align=\"center\"><i><b><font size=\"5\" face=\"Palatino Linotype\">Buona visita ");
```

```
out.println("!!</font></b></i></p>");
out.println("<p align=\"center\"> </p>");
out.println("<p align=\"center\"><b>");
out.println("<a href=\"http://169.254.69.4:8080/Segesta/SegestaServlet\">ENTRA</a></p>");
out.println("<p align=\"center\"> </p>");
out.println("<p align=\"center\"> </p>");
out.print("<p align=\"left\"><font size=\"2\">Servizio offerto in collaborazione con </font> ");
out.println("<a href=\"http://www.cere.pa.cnr.it/\">");
out.println("<font size=\"2\">ICAR ");
out.println("Istituto di Calcolo e Reti ad Alte Prestazioni di Palermo.</font></a></p>");
out.println("<p align=\"center\"><b> </b></p>");
out.println("<p align=\"center\"> </p>");
out.println("<p align=\"center\"> </p>");
out.println("</body>");
out.println("</html>");
out.close();
//response.sendRedirect("http://147.163.3.71:8080/home/index.htm");
}
}
```

A.3.2 SegestaServlet.java

```
//*****************************************
//*      Servlet contex aware management    *
//*                                         *
//*         ICAR-CNR di Palermo             *
//*                                         *
//*     Sviluppatore: Giovanni Aiello       *
//*                                         *
//*e-mail address: aiello.giovi@virgilio.it *
//*****************************************

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.lang.Thread;
import java.util.*;
import java.text.DecimalFormat;

public class SegestaServlet extends HttpServlet{

final double EARTH_RADIUS = 6371.0;//Raggio della terra in Km

Hashtable clients; //Contiene la hash table passata dall'applicazione di Middle
double POI[][];//Contiene le coordinate dei POIs
String denominazione[]; //Contiene le denominazioni dei POIs


int n;//Numero di POIs


// String nextStr; //Coordinate geografiche del client
// double latitudine; //Latitudine corrente espressa in gradi
// double longitudine;//Longitudine corrente espressa in gradi
// int riga;
// int colonna;
// int n;//Numero di POIs
// double POI[][];//Contiene le coordinate dei POI
// String denominazione[];
// double radLatitudine;
// double radLongitudine;
// Hashtable clients;
// String IPAddr;

/*
* Il metodo init viene eseguito una sola volta per tutto il ciclo di vita della servlet
*/

public void init(){
clients=new Hashtable();
try{
leggiPOI(); //Legge i POIs da file
}catch(IOException e6){}
```

```
}

public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException{
String nextStr; //Coordinate geografiche del client rispetto al sistema di
//riferimento cartografico
double latitudine; //Latitudine corrente espressa in gradi
double longitudine;//Longitudine corrente espressa in gradi
int riga;//Riga della matrice che contiene la cella
int colonna;//Colonna della matrice che contiene la cella

double radLatitudine; //Latitudine espressa in radianti
double radLongitudine;//Longitudine espressa in radianti

String IPAddr; //Indirizzo IP del client che effettua la richiesta get
int count=0;

// ServletOutputStream out = response.getOutputStream();
// PrintWriter out=new PrintWriter(response.getWriter());
//       MultipartResponse multi = new MultipartResponse(response);

IPAddr="/"+request.getRemoteAddr();
leggiHash();//Aggiorna Hash table
leggiPOI(); //Legge i POI da file

/*
* Adesso si portano le coordinate dei POIs in radianti
*/

double distanze[]=new double[n];
double minimo;//Minimo delle distanze

/*
* Costruisce la matrice che contiene le coordinate dei n POIs
* Le coordinate sono espresse in radianti
*/

for(int i=0;i<n;i++){ //porta le coordinate dei POIs in radianti
POI[i][0]=(POI[i][0]*Math.PI)/180;
POI[i][1]=(POI[i][1]*Math.PI)/180;
}
ServletOutputStream out;


//while(true){ //Utilizzato per il server pushing
out = response.getOutputStream();

// leggi(); //Legge le coordinate dell'antenna GPS da file temporaneo

int finder;
Integer ID=(Integer) clients.get(IPAddr); //restituisce il valore associato
//alla chiave IPAddr.

/*
* NOTA: Non vi è uno spreco di risorse poichè se un utente si scollega dal
*servizio, sicuramente
* successivamente un altro utente avrà assegnato lo stesso indirizzo IP del
* precedente cosicchè
*possa riutilizzare il file
* che precedentemente era reso inutilizzato.
*/

BufferedReader filebuf =
new BufferedReader(new FileReader("cella"+ID+".txt"));

/* equivale alla coppia di istruzioni:
         *
         * FileReader filein = new FileReader("giovanni.txt");
         * BufferedReader filebuf = new BufferedReader(filein);
         */

        nextStr = filebuf.readLine();     // legge una riga del file

        filebuf.close();  // chiude il file
```

```
        if(nextStr==null || nextStr.equals("")) return; //Non dovrebbe mai accadere ma se
        //dovesse accadere lo ignora

        /*
        * Si suddivide in token nextStr ottenendo 4 token
        */

        StringTokenizer divided=new StringTokenizer(nextStr);

        //riga
        String raw=divided.nextToken();//Legge la riga
        riga=Integer.parseInt(raw);

        //colonna
        String column=divided.nextToken();
        colonna=Integer.parseInt(column);

        latitudine=Double.parseDouble(divided.nextToken());
        longitudine=Double.parseDouble(divided.nextToken());
//System.out.println(latitudine+"   "+longitudine);

/*
* Adesso è possibile calcolare il punto di interesse più vicino
*/

radLatitudine=(latitudine*Math.PI)/180;
        radLongitudine=(longitudine*Math.PI)/180;

for(int i=0;i<n;i++){
distanze[i]=Math.acos((Math.cos(radLatitudine)*Math.cos(radLongitudine)*
Math.cos(POI[i][0])*Math.cos(POI[i][1]))
+ (Math.cos(radLatitudine)*Math.sin(radLongitudine) *Math.cos(POI[i][0])
*Math.sin(POI[i][1]))
+(Math.sin(radLatitudine)*Math.sin(POI[i][0])))*EARTH_RADIUS;
//System.out.println(distanze[i]);
}

minimo=distanze[0];
for(int i=1;i<n;i++){

/*
* Determina la minima distanza
*/
minimo=Math.min(distanze[i],minimo);
}

/*
* Determina l'indice del minimo delle n distanze
*/

finder=0;
for(int i=0;i<n;i++)
if(distanze[i]==minimo) finder=i; //finder identifica il POI a distanza minore
//}
response.setContentType("text/html");

//System.out.println(finder);
//response.setHeader("Refresh","5"); //Serve per il client pulling

/*
* In base alla cella in cui è il client, gli viene spedita la pagine web relativa o
* una pagina intermedia se nella cella
* sono presenti più di un punto di interesse
*/
DecimalFormat twoDigits=new DecimalFormat("0.00");
if (riga==1 && (colonna==1 || colonna==0 )){

out.println("<html>");
out.println("<head>");
out.println("<meta http-equiv=\"Content-Language\" content=\"it\">");
out.println("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=");
out.println("windows-1252\">");
out.println("<title>Pocket Visit</title>");
out.println("</head>");
```

```
out.println("");
out.println("<body>");
.
.
.
.
.
.
.
.
.
.
.
.
.
.
out.println("</body>");
out.println("");
out.println("</html>");
out.flush();
// try { Thread.sleep(5000); } catch (InterruptedException e) { }
out.close();


// response.sendRedirect( "http://192.168.0.250:8080/ValleDeiTempli/concordia.html" );
}else
if ( riga==2 && (colonna==1 || colonna==0 )){
out.println("<html>");
out.println("<head>");
out.println("<meta http-equiv=\"Content-Language\" content=\"it\">");
out.print("<meta http-equiv=\"Content-Type\" content=\"text/html; ");
out.println("charset=windows-1252\">");
out.println("<title>Pocket Visit</title>");
out.println("</head>");
out.println("");
out.println("<body>");
.
.
.
.
.
.
.
.
.
.
out.println("</body>");
out.println("");
out.println("</html>");
out.flush();
// try { Thread.sleep(5000); } catch (InterruptedException e) { }
out.close();


//response.sendRedirect( "http://192.168.0.250:8080/ValleDeiTempli/ercole.htm" );
}else if ( riga==3 && (colonna==1 || colonna==0 ) ){
out.println("<html>");
out.println("<head>");
out.println("<meta http-equiv=\"Content-Language\" content=\"it\">");
out.println("<meta http-equiv=\"Content-Type\" content=\"text/html;
charset=windows-1252\">");
out.println("<title>Pocket Visit</title>");
out.println("</head>");
out.println("");
out.println("<body>");
.
.
.
.
.
.
.
.
.
```

```
out.println("</body>");
out.println("");
out.println("</html>");
out.flush();
// try { Thread.sleep(5000); } catch (InterruptedException e) { }
out.close();

// response.sendRedirect( "http://192.168.0.250:8080/ValleDeiTempli/vulcano.htm" );
}else if ( riga==4 && (colonna==1|| colonna==0 ) ){

out.println("<html>");
out.println("<head>");
out.println("<meta http-equiv=\"Content-Language\" content=\"it\">");
out.println("<meta http-equiv=\"Content-Type\" content=\"text/html;
charset=windows-1252\">");
out.println("<title>Pocket Visit</title>");
out.println("</head>");
out.println("");
out.println("<body>");

.
.
.
.
.
.
.
.
out.println("</body>");
out.println("");
out.println("</html>");
out.flush();
// try { Thread.sleep(5000); } catch (InterruptedException e) { }
out.close();

}else if ( riga==5 && (colonna==1|| colonna==0 ) ){
out.println("<html>");
out.println("<head>");
out.println("<meta http-equiv=\"Content-Language\" content=\"it\">");
out.println("<meta http-equiv=\"Content-Type\" content=\"text/html;
charset=windows-1252\">");
out.println("<title>Pocket Visit</title>");
out.println("</head>");
out.println("");
out.println("<body>");
.
.
.
.
.
.
.
.
out.println("</body>");
out.println("");
out.println("</html>");
out.flush();
// try { Thread.sleep(5000); } catch (InterruptedException e) { }
out.close();

}else if ( riga==6 && (colonna==1 || colonna==0 )){
out.println("<html>");
out.println("<head>");
out.println("<meta http-equiv=\"Content-Language\" content=\"it\">");
out.println("<meta http-equiv=\"Content-Type\" content=\"text/html;
charset=windows-1252\">");
out.println("<title>Pocket Visit</title>");
out.println("</head>");
out.println("");
out.println("<body>");
.
.
.
.
```

```
.
.
.
.
.
out.println("</body>");
out.println("");
out.println("</html>");
out.flush();
// try { Thread.sleep(5000); } catch (InterruptedException e) { }
out.close();
      }
    //out = response.getWriter();
    else{
     response.setContentType( "text/html" );
     out.println("<html>");
out.println("<head>");
out.println("<meta http-equiv=\"Content-Language\" content=\"it\">");
out.println("<meta http-equiv=\"Content-Type\" content=\"text/html;
charset=windows-1252\">");
out.println("<title>Pocket Visit</title>");
//out.println("<base target=\"_blank\">");
out.println("</head>");
out.println("");
out.println("<body>");
.
.
.
.
.
.
out.println("</body>");
out.println("");
out.println("</html>");
out.flush();
out.close();

// try { Thread.sleep(5000); } catch (InterruptedException e) { }

}
//}
}

//    void leggi() throws IOException{//Legge la cella associata al client il
//cui mapping è presente
// nella hashtable
//  Integer ID=(Integer) clients.get(IPAddr); //restituisce il
//valore associato alla chiave IPAddr.
//
//  /*
//  NOTA: Non vi è uno spreco di risorse poichè se un utente si
//scollega dal servizio, sicuramente
//  successivamente un altro utente avrà assegnato lo stesso
//indirizzo IP del precedente cosicchè
possa riutilizzare il file
//  che precedentemente era reso inutilizzato.
//  */
//
//     BufferedReader filebuf =
//             new BufferedReader(new FileReader("cella"+ID+".txt"));
//
//        /* equivale alla coppia di istruzioni:
//         *
//         * FileReader filein = new FileReader("giovanni.txt");
//         * BufferedReader filebuf = new BufferedReader(filein);
//         */
//
//        nextStr = filebuf.readLine();     // legge una riga del file
//
//        filebuf.close();  // chiude il file
//        if(nextStr==null || nextStr.equals("")) return;
//        //Si suddivide in token nextStr ottenendo 4 token
//        StringTokenizer divided=new StringTokenizer(nextStr);
```

```
//
//        //riga
//        String raw=divided.nextToken();//Legge la riga
//        riga=Integer.parseInt(raw);
//
//        //colonna
//        String column=divided.nextToken();
//        colonna=Integer.parseInt(column);
//
//        latitudine=Double.parseDouble(divided.nextToken());
//        longitudine=Double.parseDouble(divided.nextToken());
//
////        //Si suddivide ulteriormente in token con argomento "."
////        StringTokenizer latitude=new StringTokenizer(lat,"."); //Si ottengono
//i token xxxx e xxxx
////
////        //Il token xxxx lo si deve dividere in due parti xx xx corrispondenti
//ai gradi ed ai minuti primi
////        String temp=new String(latitude.nextToken());
////        String latgrad=temp.substring(0,2);
////
////        String latprimi=temp.substring(2);
////
////        double primo=Double.parseDouble(latgrad);//Gradi
////        double secondo=(Double.parseDouble(latprimi));//Minuti primi
////        double terzo=(Double.parseDouble(latitude.nextToken()))/10000;//millesimi
//di grado
////        double quarto=(secondo+terzo)/60;
////        latitudine=primo+quarto;
////
////        //La stessa cosa per la longitudine
////        divided.nextToken();//Emisfero
////
////        //lat è la stringa xxxxx.xx
////        String lon=divided.nextToken();
////
////        //Si suddivide ulteriormente in token con argomento "."
////        StringTokenizer longitude=new StringTokenizer(lon,"."); //Si ottengono i
//token xxxxx e xx
////
////         //Il token xxxxx lo si deve dividere in due parti xxx xx corrispondenti
//ai gradi ed ai minuti primi
////        String temp2=new String(longitude.nextToken());
////        String longrad=temp2.substring(0,3);
////        String lonprimi=temp2.substring(3);
////        primo=Double.parseDouble(longrad);//Gradi
////        secondo=(Double.parseDouble(lonprimi));//Minuti primi
////        terzo=(Double.parseDouble(longitude.nextToken()))/10000;//millesimi di grado
////        quarto=(secondo+terzo)/60;
////        longitudine=primo+quarto;
////
////        //Si riportano latitudine e longitudine in radianti
////        latitudine=(latitudine*Math.PI)/180;
////        longitudine=(longitudine*Math.PI)/180;
//      }

void leggiPOI() throws IOException{ //Legge le coordinate dei POIs
 BufferedReader filebuf =
            new BufferedReader(new FileReader("POIs.txt"));
        String temp;
        StringTokenizer divide;
        String nPOIs=filebuf.readLine(); //legge numero di POIs
        n=Integer.parseInt(nPOIs); //numero di POIs espresso come int
        POI=new double[n][2];
        denominazione=new String[n];
        for(int i=0;i<n;i++){ //Legge i POIs
         temp=filebuf.readLine();//Contiene lat e lon in formato stringa
//         System.out.println(temp);
         divide=new StringTokenizer(temp);
         POI[i][0]=Double.parseDouble(divide.nextToken());
         POI[i][1]=Double.parseDouble(divide.nextToken());
         denominazione[i]=divide.nextToken();
         //System.out.println(POI[i][0]+" "+POI[i][1]+" "+denominazione[i]);
     }
```

```
        filebuf.close();  // chiude il file
}

private void leggiHash()throws IOException{ //Aggiorna in tempo reale
//la hash table in comunicazione
con l'applicazione middle
BufferedReader filebuf =
             new BufferedReader(new FileReader("hash.txt"));
        String temp;
        StringTokenizer divide;
        while(true){
        temp=filebuf.readLine();
        if(temp==null) break;
        divide= new StringTokenizer(temp);
        String IP=divide.nextToken();
        if(!clients.containsKey(IP))
clients.put(IP, new Integer(divide.nextToken()));//Aggiungi IP
//alla hash table
}
//Testa Hash Table
Enumeration keys = clients.keys();

// itera attraverso le chiavi
while ( keys.hasMoreElements() ) {
Object currentKey = keys.nextElement();

// visualizza le coppie chiave-valore
//System.out.println(currentKey + "   " + clients.get( currentKey ));
}
}
}
```

A.4 Creatore file di mappa *map.txt*

A.4.1 Panels.java

```
//*****************************************
//*          setMap for Pocket Visit        *
//*                                         *
//*         ICAR-CNR di Palermo             *
//*                                         *
//*     Sviluppatore: Giovanni Aiello       *
//*                                         *
//*e-mail address: aiello.giovi@virgilio.it *
//*****************************************

import java.awt.*;
import javax.swing.*;

public class Panels extends JPanel{

//Definisce le etichette per la GUI
protected final static String names[]={"Punto A ", "Latitudine(°)","Longitudine(°)",
"Punto B ", "Latitudine(°)","Longitudine(°)","Punto C ", "Latitudine(°)","Longitudine(°)",
"Punto D ", "Latitudine(°)","Longitudine(°)","Grana(m)"};

//Componenti GUI, dichiarati protected per l'accesso da sottoclassi
protected JLabel labels[]; //Array di JLabel
protected JTextField fields[]; //Array di campi di testo
protected JButton creaButton;
protected JPanel northPanel, centerPanel, southPanel;

//Imposta la GUI
public Panels(){
labels=new JLabel[13];
fields=new JTextField[9];

//Crea etichette
for(int i=0;i<13;i++)
labels[i]=new JLabel(names[i],JLabel.RIGHT);
```

```
//Crea campi di testo
for(int i=0;i<9;i++)
fields[i]=new JTextField();

//Crea pannello per disporre campi di testo ed etichette
//centerPanel=new JPanel();
//centerPanel.setLayout(new GridBagLayout());


//Attacca le etichette e campi di testo a northPanel

northPanel=new JPanel();
northPanel.setLayout(new GridLayout(5,5,10,10));

northPanel.add(labels[0]);
northPanel.add(labels[1]);
northPanel.add(fields[0]);
northPanel.add(labels[2]);
northPanel.add(fields[1]);
northPanel.add(labels[3]);
northPanel.add(labels[4]);
northPanel.add(fields[2]);
northPanel.add(labels[5]);
northPanel.add(fields[3]);

northPanel.add(labels[6]);
northPanel.add(labels[7]);
northPanel.add(fields[4]);
northPanel.add(labels[8]);
northPanel.add(fields[5]);
northPanel.add(labels[9]);
northPanel.add(labels[10]);
northPanel.add(fields[6]);
northPanel.add(labels[11]);
northPanel.add(fields[7]);

northPanel.add(labels[12]);
northPanel.add(fields[8]);

//Crea pulsante
creaButton=new JButton("Crea file");
southPanel=new JPanel();
southPanel.add(creaButton);

//Imposta layout del contenitore e attacca i pannelli
setLayout(new BorderLayout());
add(northPanel,BorderLayout.NORTH);
//add(centerPanel,BorderLayout.CENTER);
add(southPanel,BorderLayout.SOUTH);

validate(); //valida il layout
}//Fine costruttore

public void clearFields(){//Cancella il contenuto dei campi di testo
for(int i=0;i<9;i++)
fields[i].setText("");
}

//Imposta i valori dei campi di testo;lancia un'eccezione se il numeor di stringhe
//del'argomento non è corretto

public void setFieldValues(String strings[]) throws IllegalArgumentException{
if(strings.length!=9)
throw new IllegalArgumentException("Attenzione!! devono essere presenti
//8 stringhe  nell'array");

for(int i=0;i<9;i++)
fields[i].setText(strings[i]);
}

//Ottiene un array dui stringhe con il valore dei campi di testo

public String[] getFieldValues(){
String values[]=new String[9];
```

```
for(int i=0;i<9;i++)
values[i]=fields[i].getText();

return values;
}

public JButton getCreaButton(){
return creaButton;
}

public JTextField[] getFields(){
return fields;
}

}
```

A.4.2 setMap.java

```
//*****************************************
//*          setMap for Pocket Visit        *
//*                                         *
//*         ICAR-CNR di Palermo             *
//*                                         *
//*     Sviluppatore: Giovanni Aiello       *
//*                                         *
//*e-mail address: aiello.giovi@virgilio.it *
//*****************************************

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class setMap extends JFrame{

private JButton createButton;
private Panels interfacciaUtente;
private JMenuBar mb;
private JMenu fileMenu;
private JMenu helpMenu;

private JMenuItem exitItem;
private JMenuItem aboutItem;


//Imposta GUI
public setMap(){
super("setMap for Pocket Visit Server");
interfacciaUtente=new Panels();

mb = new JMenuBar();

fileMenu = new JMenu("File");
helpMenu = new JMenu("Help");

exitItem = new JMenuItem("Exit");
//exitItem.addActionListener(this);
fileMenu.add(exitItem);

aboutItem=new JMenuItem("About PocketVisit");
//aboutItem.addActionListener(this);
helpMenu.add(aboutItem);


mb.add(fileMenu);
mb.add(helpMenu);

setJMenuBar(mb);


getContentPane().add(interfacciaUtente,BorderLayout.CENTER);
```

```
//Configura il pulsante creaButton per questa applicazione
createButton=interfacciaUtente.getCreaButton();

//Registra listener che chiami save() alla pressione del pulsante
createButton.addActionListener(
//classe interna anonima per gestire evento di createButton
new ActionListener(){
//Chiama save su pressione del pulsante
public void actionPerformed(ActionEvent event)
{
try{
save();
}catch(IOException e){}
}
}//Fine classe interna
);//fine chiamata addActionListener

exitItem.addActionListener(
//classe interna anonima per gestire evento di createButton
new ActionListener(){
//Chiama save su pressione del pulsante
public void actionPerformed(ActionEvent event)
{
System.exit(1);
}
}//Fine classe interna
);//fine chiamata addActionListener

aboutItem.addActionListener(
//classe interna anonima per gestire evento di createButton
new ActionListener(){
//Chiama save su pressione del pulsante
public void actionPerformed(ActionEvent event)
{
JOptionPane.showMessageDialog(null," MapSetup
for PocketVisit 1.0 \n Proprietà di ICAR-CNR di
 Palermo \n Developer: Giovanni Aiello \n e-mail address:
 aiello.giovi@virgilio.it","About PocketVisit",
 JOptionPane.INFORMATION_MESSAGE);
}
}//Fine classe interna
);//fine chiamata addActionListener


//registra ascoltatore per gestire chiusura finestra
addWindowListener(
//classe interna anonima per la gestione di chiusura della finestra
new WindowAdapter(){
public void windowClosing(WindowEvent event){
}
}
);
setSize(550,220);
setVisible(true);
}

public boolean save() throws IOException { //Salva il contenuto dei campi in un file

String fieldValues[]=interfacciaUtente.getFieldValues();

if(fieldValues.length!=9){
JOptionPane.showMessageDialog(null,"Errore! devi inserire tutti i campi",
"Errore!",JOptionPane.ERROR_MESSAGE);
return false;
}
//Il numero di campi riempiti è esatto

/*A questo punto bisogna ordinare le coordinate.
In fieldValues, le coordinate sono organizzate come segue:
fieldValues[0],fieldsValues[1]= latitudine,longitudine del punto A
fieldValues[2],fieldsValues[3]= latitudine,longitudine del punto B
fieldValues[4],fieldsValues[5]= latitudine,longitudine del punto C
fieldValues[6],fieldsValues[7]= latitudine,longitudine del punto D
*/
```

```
double coordinate[][]=new double[4][2];
try{
coordinate[0][0]=Double.parseDouble(fieldValues[0]);
coordinate[0][1]=Double.parseDouble(fieldValues[1]);
coordinate[1][0]=Double.parseDouble(fieldValues[2]);
coordinate[1][1]=Double.parseDouble(fieldValues[3]);
coordinate[2][0]=Double.parseDouble(fieldValues[4]);
coordinate[2][1]=Double.parseDouble(fieldValues[5]);
coordinate[3][0]=Double.parseDouble(fieldValues[6]);
coordinate[3][1]=Double.parseDouble(fieldValues[7]);
}catch(NumberFormatException err){
JOptionPane.showMessageDialog(null,"Formato dei campi non valido","Errore!",
JOptionPane.ERROR_MESSAGE);
String refresh[]=new String[9];
for(int i=0;i<9;i++)

refresh[i]=new String("");
interfacciaUtente.setFieldValues(refresh);
return false;
}

/*
 * Cerca il punto con latitudine minore
 */
int finder=0;

for(int i=1;i<4;i++)
if(coordinate[i][0]<coordinate[finder][0]) finder=i;

/*
 * cerca il punto con longitudine minore
 */
int finder2=0;
for(int i=1;i<4;i++)
if(coordinate[i][1]<coordinate[finder2][1]) finder2=i;

double coordinateOrd[][]=new double[4][2];
coordinateOrd[0][0]=coordinate[finder][0];
coordinateOrd[0][1]=coordinate[finder][1];
coordinateOrd[1][0]=coordinate[finder2][0];
coordinateOrd[1][1]=coordinate[finder2][1];

if(((finder<finder2) && !(finder==0 && finder2==3)) || (finder==3 && finder2==0)){
//JOptionPane.showMessageDialog(null,"sono quà!","sono quà",JOptionPane.ERROR_MESSAGE);
if(finder2==3){
coordinateOrd[2][0]=coordinate[0][0];
coordinateOrd[2][1]=coordinate[0][1];
coordinateOrd[3][0]=coordinate[1][0];
coordinateOrd[3][1]=coordinate[1][1];
}
else{//Caso generico
coordinateOrd[2][0]=coordinate[finder2+1][0];
coordinateOrd[2][1]=coordinate[finder2+1][1];
if(finder2==2){//Se nel caso generico finder2==2
coordinateOrd[3][0]=coordinate[0][0];
coordinateOrd[3][1]=coordinate[0][1];
}
else{
coordinateOrd[3][0]=coordinate[finder2+2][0];
coordinateOrd[3][1]=coordinate[finder2+2][1];
}
}
}
else{  //In questo caso si è nella situazione finder>finder2 oppure finder2=3 e finder=0
if(finder2==0){
coordinateOrd[2][0]=coordinate[3][0];
coordinateOrd[2][1]=coordinate[3][1];
coordinateOrd[3][0]=coordinate[2][0];
coordinateOrd[3][1]=coordinate[2][1];
}
else if(finder2==3){
coordinateOrd[2][0]=coordinate[2][0];
coordinateOrd[2][1]=coordinate[2][1];
```

```
coordinateOrd[3][0]=coordinate[1][0];
coordinateOrd[3][1]=coordinate[1][1];
}
else if(finder2==2){//Se nel caso generico finder2==2
coordinateOrd[2][0]=coordinate[1][0];
coordinateOrd[2][1]=coordinate[1][1];
coordinateOrd[3][0]=coordinate[0][0];
coordinateOrd[3][1]=coordinate[0][1];
}
else if(finder2==1){//Se nel caso generico finder2==2
coordinateOrd[2][0]=coordinate[0][0];
coordinateOrd[2][1]=coordinate[0][1];
coordinateOrd[3][0]=coordinate[3][0];
coordinateOrd[3][1]=coordinate[3][1];
}
}

FileWriter fileout = new FileWriter("map.txt");
        // ... che incapsulo in un BufferedWriter...
        BufferedWriter filebuf = new BufferedWriter(fileout);
        // ... che incapsulo in un PrintWriter
        PrintWriter printout = new PrintWriter(filebuf);

        printout.println(coordinateOrd[0][0]+" "+coordinateOrd[0][1]+" "+coordinateOrd[1][0]+" "
        +coordinateOrd[1][1]);
        printout.println(coordinateOrd[2][0]+" "+coordinateOrd[2][1]+" "+coordinateOrd[3][0]+" "
        +coordinateOrd[3][1]);
printout.println("0 0");
        printout.println(fieldValues[8]);

        printout.close();
String refresh[]=new String[9];
for(int i=0;i<9;i++)
refresh[i]=new String("");
JOptionPane.showMessageDialog(null,"File creato con successo","File creato",
JOptionPane.INFORMATION_MESSAGE);
interfacciaUtente.setFieldValues(refresh);
return true;
}

public static void main(String args[]){
setMap map=new setMap();
}
}
```

A.5 Creatore file POIs.txt for Pocket Pc

A.5.1 setPOIs.java

```
//*****************************************
//*          setPOIs for Pocket Visit       *
//*                                         *
//*         ICAR-CNR di Palermo             *
//*                                         *
//*     Sviluppatore: Giovanni Aiello       *
//*                                         *
//*e-mail address: aiello.giovi@virgilio.it *
//*****************************************

import javax.comm.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;
import java.text.DecimalFormat;
```

```
public class setPOIs extends Frame implements ActionListener {

    final int HEIGHT = 500;
    final int WIDTH  = 460;

    private MenuBar mb;
    private Menu fileMenu;
    private Menu helpMenu;

    private MenuItem exitItem;
    private MenuItem aboutItem;

    private Button openButton;
    private Button stopButton;
  private Button addButton;

    private Panel buttonPanel;

    private Panel messagePanel;

    private TextArea messageAreaOut;
    private TextArea messageAreaIn;

    private ConfigurationPanel configurationPanel;
    private SerialParameters parameters;
    private SerialConnection connection;

    private Properties props = null;


    public static void main(String[] args) {
if ((args.length > 0)
    && (args[0].equals("-h")
    || args[0].equals("-help"))) {
    System.out.println("utilizzo: java setPOIs");
    System.exit(1);
}

setPOIs set_pois = new setPOIs();
set_pois.setVisible(true);
set_pois.repaint();
    }


    public setPOIs(){
super("setPOIs for PocketVisit v1.0");

parameters = new SerialParameters();
parameters.setBaudRate("4800");


addWindowListener(new CloseHandler(this));

mb = new MenuBar();

fileMenu = new Menu("File");
helpMenu = new Menu("Help");

exitItem = new MenuItem("Exit");
exitItem.addActionListener(this);
fileMenu.add(exitItem);

aboutItem=new MenuItem("About PocketVisit");
aboutItem.addActionListener(this);
helpMenu.add(aboutItem);


mb.add(fileMenu);
mb.add(helpMenu);

setMenuBar(mb);

messagePanel = new Panel();
messagePanel.setLayout(new GridLayout(1, 1));
```

```
messageAreaOut=new TextArea();

messageAreaIn = new TextArea();
messageAreaIn.setEditable(false);
messagePanel.add(messageAreaIn);

add(messagePanel, "Center");

configurationPanel = new ConfigurationPanel(this);

buttonPanel = new Panel();

openButton = new Button("Vai>>");
openButton.addActionListener(this);
buttonPanel.add(openButton);

stopButton = new Button("<<Ferma");
stopButton.addActionListener(this);
stopButton.setEnabled(false);

buttonPanel.add(stopButton);

addButton=new Button("Salva POI");
addButton.addActionListener(this);
addButton.setEnabled(false);
buttonPanel.add(addButton);

Panel southPanel = new Panel();

GridBagLayout gridBag = new GridBagLayout();
GridBagConstraints cons = new GridBagConstraints();

southPanel.setLayout(gridBag);

cons.gridwidth = GridBagConstraints.REMAINDER;
gridBag.setConstraints(configurationPanel, cons);
cons.weightx = 1.0;
southPanel.add(configurationPanel);
gridBag.setConstraints(buttonPanel, cons);
southPanel.add(buttonPanel);

add(southPanel, "South");



connection = new SerialConnection(this, parameters,
  messageAreaOut, messageAreaIn);
setConfigurationPanel();

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

setLocation(/*screenSize.width/9*/ 5,
    /*screenSize.height/9*/ 25);

setSize(WIDTH/2 , HEIGHT/2 );
    }


    public void setConfigurationPanel() {
configurationPanel.setConfigurationPanel();
    }

    public void actionPerformed(ActionEvent e) {
String cmd = e.getActionCommand();

if (cmd.equals("Exit")) {
    shutdown();
}

if (cmd.equals("About PocketVisit")){
AlertDialog al=new AlertDialog(this, "About PocketVisit",
"setPOIs for Pocket Visit v1.0", "Proprietà di: ICAR di Palermo",
 "Istituto di Calcolo e Reti ad alte prestazioni");
```

```
}

if (cmd.equals("Vai>>")) {
    openButton.setEnabled(false);
    Cursor previousCursor = getCursor();
    setNewCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
    configurationPanel.setParameters();
    try {
     connection.openConnection();
    } catch (SerialConnectionException e2) {
AlertDialog ad = new AlertDialog(this,
 "Error Opening Port!",
 "Error opening port,",
 e2.getMessage() + ".",
 "Select new settings, try again.");
        openButton.setEnabled(true);
setNewCursor(previousCursor);
return;
    }
    portOpened();
    setNewCursor(previousCursor);
}


if (cmd.equals("<<Ferma")) {
    portClosed();
}

if(cmd.equals("Salva POI")){
try{
salva(); //Questo metodo deve aprire il file esistente di POIs.txt
//ed aggiornarlo con la nuova posizione e la denominazione del punto
//di interesse
}catch(IOException e4){}

}

}


    public void portOpened() {
openButton.setEnabled(false);
stopButton.setEnabled(true);
addButton.setEnabled(true);
configurationPanel.denField.setEditable(true);

    }


    public void portClosed() {
connection.closeConnection();
openButton.setEnabled(true);
stopButton.setEnabled(false);
addButton.setEnabled(false);
configurationPanel.denField.setEditable(false);
    }


    private void setNewCursor(Cursor c) {
setCursor(c);
messageAreaIn.setCursor(c);
messageAreaOut.setCursor(c);
    }

    private void shutdown() {
connection.closeConnection();
System.exit(1);
    }



    class ConfigurationPanel extends Panel implements ItemListener {

private Frame parent;
```

```
private Label den; //denominazione del POI da aggiungere
private Label portNameLabel;
private Choice portChoice;

public TextField denField;



public ConfigurationPanel(Frame parent) {
    this.parent = parent;

    setLayout(new GridLayout(2, 2));

den=new Label("POI Name:",Label.LEFT);
   add(den);

   denField=new TextField("",20);
   denField.setEditable(false);
   add(denField);

    portNameLabel = new Label("Port Name:", Label.LEFT);
    add(portNameLabel);

    portChoice = new Choice();
    portChoice.addItemListener(this);
    add(portChoice);
    listPortChoices();
    portChoice.select(parameters.getPortName());

}


public void setConfigurationPanel() {
    portChoice.select(parameters.getPortName());

}


        public void setParameters() {
            parameters.setPortName(portChoice.getSelectedItem());

        }


        void listPortChoices() {
            CommPortIdentifier portId;

            Enumeration en = CommPortIdentifier.getPortIdentifiers();

            // iterate through the ports.
            while (en.hasMoreElements()) {
                portId = (CommPortIdentifier) en.nextElement();
                if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
                    portChoice.addItem(portId.getName());
                }
            }
            portChoice.select(parameters.getPortName());
        }


public void itemStateChanged(ItemEvent e) {

    if (connection.isOpen()) {

     if (e.getItemSelectable() == portChoice) {

    AlertDialog ad = new AlertDialog(parent, "Porta aperta!",
       "La porta non può essere",
     "cambiata",
     "mentre una porta è aperta.");


    setConfigurationPanel();
```

```
    return;
}

setParameters();
try {

    connection.setConnectionParameters();
} catch (SerialConnectionException ex) {

    AlertDialog ad = new AlertDialog(parent,
"Configurazione non supportata!",
"Parametro di configurazione non supportato,",
"seleziona un nuovo valore.",
"Ritorno alla precedente configurazione.");
    setConfigurationPanel();
}
    } else {

     setParameters();
    }
}
    }


    class CloseHandler extends WindowAdapter {

setPOIs sd;

public CloseHandler(setPOIs sd) {
    this.sd = sd;
}

public void windowClosing(WindowEvent e) {
    sd.shutdown();
}
    }

    private void salva() throws IOException, FileNotFoundException{
     //Questo metodo legge il vecchio POIs.txt, legge le nuove coordinate e costruisce
     //un nuovo POIs.txt
     File f=new File("POIs.txt");
     BufferedReader filebuf =
             new BufferedReader(new FileReader(f));
        String temp;
        int n;
        String POI[][];
        String denominazione[];
        StringTokenizer divide;
        String nPOIs=filebuf.readLine(); //legge numero di POIs
        n=Integer.parseInt(nPOIs); //numero di POIs espresso come int
        if(n>0){
         POI=new String[n][2];
         denominazione=new String[n];
         for(int i=0;i<n;i++){ //Legge i POIs
         temp=filebuf.readLine();//Contiene lat e lon in formato stringa
         divide=new StringTokenizer(temp);
         POI[i][0]=(divide.nextToken());
         POI[i][1]=(divide.nextToken());
         denominazione[i]=divide.nextToken();
      }
        filebuf.close();  // chiude il file

      }
      else{

      filebuf.close();
       POI=new String[0][2];
       denominazione=new String[0];
      }

        //Legge le coordinate correnti e la denominazione dal TextField
          filebuf =
             new BufferedReader(new FileReader("temp.txt"));
        temp=filebuf.readLine(); //legge coordinate nuovo POI
```

```
        filebuf.close();  // chiude il file

        //Trasforma le coordinate in gradi
        StringTokenizer divided=new StringTokenizer(temp);

        //lat è la stringa xxxx.xxxx
        String lat=divided.nextToken();

        //Si suddivide ulteriormente in token con argomento "."
        StringTokenizer latitude=new StringTokenizer(lat,"."); //Si ottengono i token
        //xxxx e xxxx

        //Il token xxxx lo si deve dividere in due parti xx xx corrispondenti ai gradi
        //ed ai minuti primi
        String temp1=new String(latitude.nextToken());
        String latgrad=temp1.substring(0,2);
        String latprimi=temp1.substring(2);
        double gradi=Double.valueOf(latgrad).doubleValue();//Gradi
        double minutiPrimi=(Double.valueOf(latprimi)).doubleValue();//Minuti primi
        double millesimiPrimi=(Double.valueOf(latitude.nextToken())).doubleValue()/10000;
        //millesimi di minuti primi
        double decGrad=(minutiPrimi+millesimiPrimi)/60;
        double latitudine=gradi+decGrad; //Latitudine dell'antenna GPS espressa in gradi

        //La stessa cosa per la longitudine
        divided.nextToken();//Emisfero della posizione attuale(indica il verso di positività
        //degli assi cartografici

        //lon è la stringa xxxxx.xx
        String lon=divided.nextToken();

        //Si suddivide ulteriormente in token con argomento "."
        StringTokenizer longitude=new StringTokenizer(lon,"."); //Si ottengono i token xxxxx e xx

         //Il token xxxxx lo si deve dividere in due parti xxx xx corrispondenti ai gradi ed ai minuti primi
        String temp2=new String(longitude.nextToken());
        String longrad=temp2.substring(0,3);
        String lonprimi=temp2.substring(3);
        gradi=Double.valueOf(longrad).doubleValue();//Gradi
        minutiPrimi=(Double.valueOf(lonprimi)).doubleValue();//Minuti primi
        millesimiPrimi=(Double.valueOf(longitude.nextToken())).doubleValue()/10000;//millesimi di primi
        decGrad=(minutiPrimi+millesimiPrimi)/60;
       double longitudine=gradi+decGrad;//longitudine dell'antenna GPS espressa in gradi


        DecimalFormat fourDigit=new DecimalFormat("0.00000");
        latitudine=(Double.valueOf(fourDigit.format(latitudine).replace(',','.'))).doubleValue();
        longitudine=(Double.valueOf(fourDigit.format(longitudine).replace(',','.'))).doubleValue();
        temp=new String(latitudine+" ");
        String longStr=Double.toString(longitudine);
        temp=temp.concat(longStr);

        String newDen=configurationPanel.denField.getText();

        //Ricrea il file
        FileWriter fileout = new FileWriter("POIs.txt");
        // ... che incapsulo in un BufferedWriter...
        BufferedWriter filebuffered = new BufferedWriter(fileout);
        // ... che incapsulo in un PrintWriter
        PrintWriter printout = new PrintWriter(filebuffered);
        printout.println(n+1);
        if(n>0)
         for(int i=0;i<n;i++)
         printout.println(POI[i][0] + " " +POI[i][1]+ " " +denominazione[i]);
        printout.println(temp+" "+newDen);
        printout.close();
      }
    }
```

A.5.2 SerialConnection.java

```
//*****************************************
//*          setPOIs for Pocket Visit       *
```

```
//*                                        *
//*         ICAR-CNR di Palermo            *
//*                                        *
//*     Sviluppatore: Giovanni Aiello      *
//*                                        *
//*e-mail address: aiello.giovi@virgilio.it *
//******************************************

import javax.comm.*;
import java.io.*;
import java.awt.TextArea;
import java.awt.event.*;
import java.util.TooManyListenersException;
import java.util.StringTokenizer;
import java.net.*;
import java.lang.Exception;



public class SerialConnection implements SerialPortEventListener,
                    CommPortOwnershipListener { /*La SerialPortEventListener
                                         propaga eventi di porta seriale mentre
                                         l'interfaccia CommPortOwnershipListener
                                         fa si che quando una porta è aperta, un
                                         evento CommPortOwnership di tipo PORT_OWNED
                                         (porta occupata) sarà propagato. Quando una
                                         porta è chiusa, un evento
                                         CommPortOwnership di tipo PORT_UNOWNED
                                         (porta libera) sarà propagata */

    private setPOIs parent; //Oggetto setPOIs

String old;

    private TextArea messageAreaOut; //TextArea dei messaggi in ingeresso che passiamo
                                     //dalla classe setPOIs
    private TextArea messageAreaIn; //TextArea dei messaggi in uscita che passiamo dalla
                                     //classe setPOIs
    private SerialParameters parameters; //oggetto della classe SerialParameters che rappresenterà
                                       // i parametri che passiamo
    private OutputStream os; //Flusso in uscita
    private InputStream is; //Flusso in ingresso


    private CommPortIdentifier portId; /*
                                     CommPortIdentifier è la classe centrale per il controllo
                                     dell'accesso alle porte ci comunicazione*/
    private SerialPort sPort; // una porta di comunicazione seriale RS-232.

    private boolean open;

    int flag;//Controlla se la connessione è avvenuta con successo


    public SerialConnection(setPOIs parent, /*COSTRUTTORE che accetta un oggetto setPOIs,
                                          i parametri, la TextArea di input(Area dove i
                                          mex arrivano) e la TextArea di output */
                SerialParameters parameters,
                TextArea messageAreaOut,
                TextArea messageAreaIn) {
    old=new String("");
    flag=0;
    this.parent = parent; //Inizializza parent a parent di tipo setPOIs
    this.parameters = parameters;
//    this.messageAreaOut = messageAreaOut;
    this.messageAreaIn = messageAreaIn;
    open = false; //è un booleano

   }


    /*Tenta di aprire una connessione seriale e flussi usando i parametri della porta
    seriale (oggetto di tipo SerialParameters). Se la connesisone non va a buon fine,
    esso torna la porta  attraverso un'eccezione*/
```

```
   public void openConnection() throws SerialConnectionException {  //Metodo che apre
//                                                                     una connessione

    // Ottiene un oggetto CommPortIdentifier (javax.comm) per la porta che si vuole aprire.
    try {
    /*Gestione delle porte di comunicazione. CommPortIdentifier è una classe centrale per
   il controllo all'accesso alle porte di comunicazione.*/
        portId =
         CommPortIdentifier.getPortIdentifier(parameters.getPortName());/*getPortIdentifier(String)
                                                                  è un metodo che ottiene
                                                                  un oggetto CommPortIdentifier
                                                                  mediante il nome della porta */

    } catch (NoSuchPortException e) {
        throw new SerialConnectionException(e.getMessage());
    }


    //Apre la porta rappresentata dall'oggetto CommPortIdentifier, viene datoi all'apertura della
    // chiamata un timeout relativamente lungo di 30 sec per permettere un'applicazione differente di
//     rilasciare la porta se l'utente la vuole
    try {
        sPort = (SerialPort)portId.open("setPOIs", 30000); /*il metodo open() della classe
                                                           CommPortIdentifier
                                                           apre una porta di comunicazione
                                                           e accetta il nome dell'applicazione
                                                           e un intero che indica il timeout (msec).
                                                           IL METODO OPEN OTTIENE LA COMPLETA
                                                           PADRONANZA DELLA PORTA da qualche
                                                           altra applicazione
                                                           , è proparato un evento
                                                           PORT_OWNERSHIP_REQUESTED isando il
                                                           meccanismo dell'evento
                                                           CommPortOwnershipListener.
                                                           C'è un InputStream ed un outputStream
                                                           associato a ogni porta. Dopo che una porta
                                                           è aperta mediante il metodo open,
                                                           tutte le chiamate al metodo getInputStream
                                                           ritornerà lo stesso stream fino a che è
                                                           chiamato il metodo close*/

    messageAreaIn.append("    **************************\n ");
messageAreaIn.append("    *      setPOIs v1.0 \"Client\"   *\n ");
messageAreaIn.append("    *         ICAR-CNR - Palermo      *\n ");
messageAreaIn.append("    ************************* \n\n\n");
    messageAreaIn.append(" Attendere conferma connessione \n con antenna GPS....");
    messageAreaIn.append("\n Se tale conferma non dovesse essere data,\n assicurarsi che la connessione
    Bluetooth \n possa avvenire.. \n\n\n ");

    } catch (PortInUseException e) { // l'evento di porta in uso lanciando un'eccezione se in uso
        throw new SerialConnectionException(e.getMessage());
    }

    // Setta i parametri della connessione, se essi non si settano, la porta viene chiusa e viene
//    lanciata un'eccezione
    try {
        setConnectionParameters();
    } catch (SerialConnectionException e) {
        sPort.close();
        throw e;
    }


    //Apre i flussi di ingresso e uscita per la connessione. se essi non si aprono, viene lanciata
//    un'eccezione
    try {
//        os = sPort.getOutputStream();
        is = sPort.getInputStream();
    } catch (IOException e) {
        sPort.close();
        throw new SerialConnectionException("Error opening i/o streams");
    }
```

```
    try {
        sPort.addEventListener(this);
    } catch (TooManyListenersException e) {
        sPort.close();
        throw new SerialConnectionException("too many listeners added");
    }


    sPort.notifyOnDataAvailable(true);


    sPort.notifyOnBreakInterrupt(true);


    try {
        sPort.enableReceiveTimeout(30);
    } catch (UnsupportedCommOperationException e) {
    }


    portId.addPortOwnershipListener(this);

    open = true;
    }

    /**
    Sets the connection parameters to the setting in the parameters object.
    If set fails return the parameters object to origional settings and
    throw exception.
    */
    public void setConnectionParameters() throws SerialConnectionException{ //Metodo di utilità
//                                                                          che setta i parametri di connessione
//
 //utilizzando l'oggetto parameters.
//                                                                           Se il settaggio fallisce ritorna
//                                                                           l'oggetto parameters
//                                                                          //ai suoi settaggi originali e
//                                                                         lancia l'eccezione

    // Salva lo stato dei parametri prima di tentare il settaggio.
    int oldBaudRate = sPort.getBaudRate(); //salva il BaudRate
    int oldDatabits = sPort.getDataBits();
    int oldStopbits = sPort.getStopBits();
    int oldParity   = sPort.getParity();
    int oldFlowControl = sPort.getFlowControlMode();

    // Setta i parametri di connessione, se il set fallisce ritorna l'oggetto parameters
    // allo stato originale.
    try {
        sPort.setSerialPortParams(parameters.getBaudRate(), //Setta i parametri della porta seriale
//                                                            passando tutti i parametri.Ricordiamo che
                    parameters.getDatabits(),             //sPort è un oggetto CommPortIdentifier
                    parameters.getStopbits(),
                    parameters.getParity());
    } catch (UnsupportedCommOperationException e) { //Gestisce l'eccezione di non suporto dei parametri
        parameters.setBaudRate(oldBaudRate); //Riprta parameters al suo stato originale
        parameters.setDatabits(oldDatabits);
        parameters.setStopbits(oldStopbits);
        parameters.setParity(oldParity);
        throw new SerialConnectionException("Parametro non supportato");
    }

    // Setta il controllo di flusso dal settaggio contenuto in parameters
    try {
        sPort.setFlowControlMode(parameters.getFlowControlIn()
                    | parameters.getFlowControlOut());
    } catch (UnsupportedCommOperationException e) { //lancia l'eccezione se non supportato
        throw new SerialConnectionException("Flusso di controllo non supportato");
    }
    }//Fine metodo setConnectionparameters

    /*
    Chiude la porta e pulisce gli elementi associati.
```

```
    */
    public void closeConnection() { //Metodo che chiude la connessione
    // se la porta è già chiuse esce dalla funzione.
    flag=0;
    if (!open) {
        return;
    }


    // Contolla che sPort ha referenza per evitare un NPE.
    if (sPort != null) {//Se sPort si riferisce ad una porta
        try {
        // chiudi gli i/o streams.
//            os.close(); //Chiude gli Stream in ingresso e uscita
            is.close();
        } catch (IOException e) {
        System.err.println(e);
        }

        // chiude la porta.
        sPort.close(); //Chiude la porta

        // Rimuove l'ascoltatore di padronanza della porta.
        portId.removePortOwnershipListener(this);
    }

    open = false; //Dichiara la porta chiusa
    }// fine metodo closeConnection



    public boolean isOpen() {
    return open;
    }


    /* Testa gli eventi SerialPortEvents. i due tipi di SerialPortEvents che questo programma è capace
     di ascoltare sono DATA_AVAILABLE e BI. Durante il DATA_AVAILABLE il buffer della porta è letto
      fino al suo svutamento quando non sono disponibili più dati e sono passati 30 msec il metodo
       ritorna. quando un evento BI occorre, la parola BREAK RECEIVED è scritto nel messageAreaIn. */

    public void serialEvent(SerialPortEvent e) {
    // Crea una StringBuffer e int per ricevere dati in input.
    StringBuffer inputBuffer = new StringBuffer();
    int newData = 0;
    String toSend;
    String toSent=new String("");
    String confronto=new String("$GPGGA");
    String confronto1=new String("$GPGLL");
    String confronto2=new String("$GPRMC");
    // determina il tipo di evento.
    switch (e.getEventType()) {

        // Legge dati fino a quando è ritornato -1. Se è ricevuto \r sostituisce
        // \n per un corretto maneggio dell newline.
        case SerialPortEvent.DATA_AVAILABLE:
            while (newData != -1) {
                try {
                    newData = is.read();
                if (newData == -1) {
                break;
                }
                if ('\r' == (char)newData) { //Ha finito di trasmettere la riga
//                  System.out.println(inputBuffer+"\n\n\n");
                 StringTokenizer st = new StringTokenizer(new String(inputBuffer),",");
               toSend=new String("");//Svuota la String precedente
               String tag=new String(st.nextToken());
                while(st.hasMoreTokens()){
                 if(tag.equals(confronto)){ /*Mediante queste righe si vengono presi soltanto 4 elementi:
                 a) Latitudine della posizione attuale
                 b) Emisfero della posizione attuale (Nord o Sud)
                 c) Longitudine della posizione attuale
                 d) Verso della posizione attuale (Esto o Ovest)
                 */
```

```
                st.nextToken();
                 for(int i=0;i<4;i++){
                   toSend=toSend.concat(st.nextToken()+" ");
                       }
                old=new String(toSend);
//                System.out.println("toSend:" +toSend);
//                System.out.println("old:" +old);
                break;

              }else if(tag.equals(confronto1)){ /*Mediante queste righe si vengono presi soltanto 4 elementi:
                a) Latitudine della posizione attuale
                b) Emisfero della posizione attuale (Nord o Sud)
                c) Longitudine della posizione attuale
                d) Verso della posizione attuale (Esto o Ovest)
                */
                 for(int i=0;i<4;i++){
                   toSend=toSend.concat(st.nextToken()+" ");
                       }
                //System.out.println(toSend);
                old=new String(toSend);
//                System.out.println("toSend:" +toSend);
//                System.out.println("old:" +old);
                break;
              }else if(tag.equals(confronto2)){ /*Mediante queste righe vengono presi soltanto 4 elementi:
                a) Latitudine della posizione attuale
                b) Emisfero della posizione attuale (Nord o Sud)
                c) Longitudine della posizione attuale
                d) Verso della posizione attuale (Esto o Ovest)
                */
                 st.nextToken();
                 st.nextToken();
                 for(int i=0;i<4;i++){
                   toSend=toSend.concat(st.nextToken()+" ");

                       }
                //System.out.println(toSend);
                old=new String(toSend);
//                System.out.println("toSend:" +toSend);
//                System.out.println("old:" +old);
                break;
              }
              else{

//                System.out.println("toSend:" +toSend);
//                 System.out.println("old:" +old);
               toSend=new String(old);
//                System.out.println("Continua ciclo");
               break;
             }
             }
            // System.out.println(toSend);
             toSent=toSend;
             //StringTokenizer s=new StringTokenizer(toSend);
             //while(s.hasMoreElements())
//              System.out.println(toSent);

              inputBuffer.append('\n');
          //Bisogna spedire al Server la Stringa toSend che contiene le
          //coordinate del client, poi il server se lo gestisce

              } else {
                  inputBuffer.append((char)newData);
              }
              } catch (IOException ex) {
                  System.err.println(ex);
                  return;
              }
          }
        try{
 save(toSent);
}
```

```
       finally{
       StringTokenizer div=new StringTokenizer(toSent);

        messageAreaIn.append("\n\n Latitudine: "+div.nextToken()+"\t Emisfero: "+div.nextToken()+"\n
        Longitudine: "+div.nextToken()+"\t Verso Posizione: "+div.nextToken()+"\n"); //Appende i dati
        in ingresso nel

        break;
}
    }

    }


    /* Se è ricevuto un evento PORT_OWNERSHIP_REQUESTED è creato un box di dialogo domandando
     all'utente se essi sono disposti a cedere la porta*/

    public void ownershipChange(int type) {
    if (type == CommPortOwnershipListener.PORT_OWNERSHIP_REQUESTED) {
        PortRequestedDialog prd = new PortRequestedDialog(parent);
    }
    }

public void save(String arg) throws Exception, IOException{
if (flag==0){
messageAreaIn.append("********************************** \n\n\n");
messageAreaIn.append("Benvenuti su PocketVisit v1.0 \n for Pocket PC! \n \n
Connessione all'antenna\n GPS TOMTOM \n avvenuta con successo. \n \n Adesso
è possibile aprire\n Pocket Internet Explorer! ...\n\n");
flag++;
}
FileWriter fileout = new FileWriter("temp.txt");
        // ... che incapsulo in un BufferedWriter...
        BufferedWriter filebuf = new BufferedWriter(fileout);
        // ... che incapsulo in un PrintWriter
        PrintWriter printout = new PrintWriter(filebuf);
        printout.println(arg);
        printout.close();

          }
}
```

A.6 Simulatore antenna GPS

A.6.1 Simulator.java

```
//*****************************************
//*         Simulatore antenna GPS          *
//*                                         *
//*         ICAR-CNR di Palermo             *
//*                                         *
//*     Sviluppatore: Giovanni Aiello       *
//*                                         *
//*e-mail address: aiello.giovi@virgilio.it *
//*****************************************

import javax.comm.*;
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.util.Properties;
import java.util.Enumeration;



public class Simulator extends Frame implements ActionListener {
```

```
    final int HEIGHT = 500;
    final int WIDTH  = 460;

    private MenuBar mb;
    private Menu fileMenu;
    private Menu helpMenu;

    private MenuItem exitItem;
    private MenuItem aboutItem;

    private Button openButton;
    private Button stopButton;

    private Panel buttonPanel;

    private Panel    messagePanel;
    private TextArea messageAreaOut;
    private TextArea messageAreaIn;

    private ConfigurationPanel configurationPanel;
//    private SerialParameters parameters;
//    private SerialConnection connection;
private GPSSimulator sim;


    private Properties props = null;/*La classe Properties estende la classe Hashtable e
                                      rappresenta un set di proprietà persistente.
                                      Le proprietà possono essere salvate in uno Stream
                                      o caricate da uno Stream. Ogni chiave ed il suo
                                      corrispondente valore nella lista delle proprietà è
                                      una String */

    /*
    Metodo main. Controlla se come argomenti vengono passate le stringhe (-h, -help),
    e, se è così, visualizza il messaggio di utilizzo ed esce, altrimenti crea un nuovo
    Simulator e lo setta come visibile.
    */
    public static void main(String[] args) {
if ((args.length > 0)
    && (args[0].equals("-h")
    || args[0].equals("-help"))) {
    System.out.println("utilizzo: java Simulator");
    System.exit(1);
}

Simulator pocket_visit = new Simulator();
pocket_visit.setVisible(true);
pocket_visit.repaint();
    }

    public Simulator(){
super("GPS Simulator");

// Setta la GUI per il programma
addWindowListener(new CloseHandler(this));

mb = new MenuBar();

fileMenu = new Menu("File");
helpMenu = new Menu("Help");

exitItem = new MenuItem("Exit");
exitItem.addActionListener(this);
fileMenu.add(exitItem);

aboutItem=new MenuItem("About Simulator");
aboutItem.addActionListener(this);
helpMenu.add(aboutItem);


mb.add(fileMenu);
mb.add(helpMenu);

setMenuBar(mb);
```

```
messagePanel = new Panel();
messagePanel.setLayout(new GridLayout(1, 1));

messageAreaOut = new TextArea();
//messagePanel.add(messageAreaOut);

messageAreaIn = new TextArea();
messageAreaIn.setEditable(false);
messagePanel.add(messageAreaIn);

add(messagePanel, "Center");

configurationPanel = new ConfigurationPanel(this);

buttonPanel = new Panel();

openButton = new Button("Vai>>");
openButton.addActionListener(this);
buttonPanel.add(openButton);

stopButton = new Button("<<Ferma");
stopButton.addActionListener(this);
stopButton.setEnabled(false);
buttonPanel.add(stopButton);

Panel southPanel = new Panel();

GridBagLayout gridBag = new GridBagLayout();/* La classe GridBagLayout è un gestore
                                               flessibile di layout che allinea le
                                               componenti verticalmente e orizzontalmente
                                               senza il bisogno che le componenti siano
                                               della stessa grandezza. Ogni oggetto di
                                               questa classe mantiene una griglia
                                               rettangolare dinamica di celle con ogni
                                               campo occupante una o più celle,
                                               chiamato suo "DisplayArea" */

GridBagConstraints cons = new GridBagConstraints();/*la classe GridBagConstraints
                                                     specifica l'obbligo per le componenti
                                                     di usare la classe GridBagLayout*/

southPanel.setLayout(gridBag);

cons.gridwidth = GridBagConstraints.REMAINDER;/* cons.gridwidth specifica il numero di
                                                 celle in una riga per il displayArea del
                                                 componente GridBagConstraints.REMAINDER
                                                 indica che questo componente è l'ultimo
                                                 nella sua riga o colonna */


gridBag.setConstraints(configurationPanel, cons);//Configura in questa maniera il
                                                 //     configurationPanel
cons.weightx = 1.0;
southPanel.add(configurationPanel);
gridBag.setConstraints(buttonPanel, cons);
southPanel.add(buttonPanel);

add(southPanel, "South");


setConfigurationPanel();//Chiama la funzione di utilità

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();/*La classe Dimension
                                                                     racchiude la larghezza
                                                                     e l'altezza di un componente
                                                                     (in precisione intera)
                                                                     in un singolo oggetto*/

setLocation(/*screenSize.width/9*/ 5,
    /*screenSize.height/9*/ 25);       /* Muove questo componente in una nuova locazione.
                                           l'angolo in alto a sinistra della nuova locazione
                                           è specificato dai parametri x e y
                                           nello spazio delle coordinate del componente genitore
```

```
                                          */
setSize(WIDTH/2 , HEIGHT/2 );//Setta le dimensioni della finestra
    }//Fine costruttore di Simulator

    /**
    Setta gli elementi GUI nel configurationPanel.
    */
    public void setConfigurationPanel() {
configurationPanel.setConfigurationPanel();/*Richiama il metodo setConfigurationPanel della classe
                                             ConfigurationPanel*/
    }

    /**
    Risponde agli MenuItem e ai pulsanti
    */
    public void actionPerformed(ActionEvent event) {
String cmd = event.getActionCommand();// Restituisce il tipo di evento avvenuto

// Chiama shutdown, in modo da terminare il programma.
if (cmd.equals("Exit")) {
    shutdown();
}

if (cmd.equals("About Simulator")){
AlertDialog al=new AlertDialog(this, "About Simulator",
"Pocket Visit v1.0 for Pocket Pc", "Proprietà di: ICAR di Palermo",
"Istituto di Calcolo e Reti ad alte prestazioni");
}
// Apre una porta.
if (cmd.equals("Vai>>")) {
    openButton.setEnabled(false);
    Cursor previousCursor = getCursor();/*Cursor è una classe che racchiude la rappresentazione
                                         bitmap di un cursore del mouse e la associa a previousCursor*/
    setNewCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));/*chiama funzione di utilità che
                                                                   accetta dal sistema il cursore di default*/

portOpened();
setNewCursor(previousCursor);


sim=new GPSSimulator(this,messageAreaIn);
sim.start();
}

// Chiude la connessione.
if (cmd.equals("<<Ferma")) {

    portClosed();

}

}

    /*
    Setta i pulsanti nello stato di connessione aperta.
    */
    public void portOpened() {
openButton.setEnabled(false);
stopButton.setEnabled(true);

    }

    /*
    Setta i pulsanti nello stato di connessione chiusa.
    */
    public void portClosed() {
//connection.closeConnection();

sim.close();
openButton.setEnabled(true);
stopButton.setEnabled(false);

    }
```

```
    /**
    Setta il Cursor per l'applicazione.
    c= nuovo Cursor
    */
    private void setNewCursor(Cursor c) {//Setta un nuovo cursore e accetta un altro cursore
setCursor(c);//setta l'immagine del cursore per il cursore selezionato
messageAreaIn.setCursor(c);/*Setta l'immagine del cursore selezionato. Questa immagine è
                            visualizzata quando il metodo "contains"
                            per questo componente torna true per la locazione del cursore
                            corrente e questo componente è visibile, visualizzabile
                            e abilitato*/
messageAreaOut.setCursor(c);
    }


    private void shutdown() {
//sim.close();
System.exit(1);
    }


    class ConfigurationPanel extends Panel implements ItemListener {/*Classe che eredita i metodi
                                                                       e gli attributi da Panel
                                                                       (quindi è un pannello) e
                                                                       implementa l'interfaccia
                                                                       che gestisce gli eventi di
                                                                       selezione degli item */

private Frame parent;


/*
Crea e inizializza il pannello di configurazione.
*/
public ConfigurationPanel(Frame parent) {/*Il costruttore di questa classe accetta un Frame
                                           che nel nostro caso è Simulator*/
    this.parent = parent;//Fa una copia del Frame Simulator e lo chiama parent

    setLayout(new GridLayout(1, 2));

}


public void setConfigurationPanel() {

}



public void itemStateChanged(ItemEvent e) {

}
    }

    /*
    Permette all'applicazione di essere chiusa mediante il box di chiusura
    */
    class CloseHandler extends WindowAdapter {

Simulator sd;

public CloseHandler(Simulator sd) {
    this.sd = sd;
}

public void windowClosing(WindowEvent e) {
    sd.shutdown();
}
}
}
```

A.6.2 AlertDialog.java

```
//Simulator 1.0 for PocketPc

import java.awt.*;
import java.awt.event.*;

/**
Questa classe è configurabile pèer messaggi e titoli.
La larghezza del Dialog sarà quanto la lunghezza della linea del messaggio.
*/
public class AlertDialog extends Dialog implements ActionListener {

    /*
    Crea un nuovo <code>AlertDialog</code> con tre linee di messaggio e titolo.

    parent = Frame passato.
    title = Titolo che appare nel bordo del dialogo.
    lineOne = La prima linea del messaggio nel dialogo.
    lineTwo = La seconda linea del messaggio nel dialogo.
    lineThree = La terza linea del messaggio nel dialogo.
    */
    public AlertDialog(Frame parent,
       String title,
       String lineOne,
       String lineTwo,
       String lineThree) {
super(parent, title, true);

Panel labelPanel = new Panel();
labelPanel.setLayout(new GridLayout(3, 1));
labelPanel.add(new Label(lineOne, Label.CENTER));
labelPanel.add(new Label(lineTwo, Label.CENTER));
labelPanel.add(new Label(lineThree, Label.CENTER));
add(labelPanel, "Center");

Panel buttonPanel = new Panel();
Button okButton = new Button("OK");
okButton.addActionListener(this);
buttonPanel.add(okButton);
add(buttonPanel, "South");

FontMetrics fm = getFontMetrics(getFont());
int width = Math.max(fm.stringWidth(lineOne),
 Math.max(fm.stringWidth(lineTwo), fm.stringWidth(lineThree)));

setSize(width + 40, 150);
setLocation(parent.getLocationOnScreen().x + 30,
    parent.getLocationOnScreen().y + 30);
setVisible(true);
    }

    /**
   Quando è pressato OK il messaggio diviene invisibile,  e ritorna.
    */
    public void actionPerformed(ActionEvent e) {
setVisible(false);
dispose();
    }
}
```

A.6.3 GPSSimulator.java

```
//*****************************************
//*         Simulatore antenna GPS          *
//*                                         *
//*         ICAR-CNR di Palermo             *
//*                                         *
//*     Sviluppatore: Giovanni Aiello       *
//*                                         *
//*e-mail address: aiello.giovi@virgilio.it *
//*****************************************
```

```
import java.net.*;
import java.util.*;
import java.io.*;
import java.awt.*;
import java.text.DecimalFormat;

public class GPSSimulator extends Thread{

private Simulator parent; //Oggetto Simulator

private TextArea messageAreaIn; //TextArea dei messaggi in uscita che passiamo dalla
//classe Simulator
private boolean open; //Booleano

double vertici[][];
double latitudine,longitudine; //Latitudine e longitudine estratti dinamicamente


int flag;//Controlla se la connessione è avvenuta con successo

public GPSSimulator(Simulator parent, TextArea messageAreaIn){
this.parent = parent; //Inizializza parent a parent di tipo Simulator

this.messageAreaIn = messageAreaIn;
open = false; //è un booleano
try{
load();


}catch(IOException e){}

  }

public void run(){
try{
open();
}
catch(IOException e1){}
catch(InterruptedException e2){}
}

private void load()throws IOException {
BufferedReader filebuf =
new BufferedReader(new FileReader("map.txt")); //Legge la mappa dal file

/* equivale alla coppia di istruzioni:
*
* FileReader filein = new FileReader("giovanni.txt");
* BufferedReader filebuf = new BufferedReader(filein);
*/


String vertici12 = new String(filebuf.readLine()); // Legge la Prima riga del file che contiene
//le coordinate dei primi 2 punti
String vertici34 = new String(filebuf.readLine()); // Legge la Seconda riga del file che contiene
//le coordinate degli altri 2 punti
filebuf.close();  // chiude il file

vertici=new double[4][2];
StringTokenizer coord12=new StringTokenizer(vertici12);
StringTokenizer coord34=new StringTokenizer(vertici34);
vertici[0][0]=Double.valueOf(coord12.nextToken()).doubleValue();
  vertici[0][1]=Double.valueOf(coord12.nextToken()).doubleValue();
vertici[1][0]=Double.valueOf(coord12.nextToken()).doubleValue();
vertici[1][1]=Double.valueOf(coord12.nextToken()).doubleValue();
vertici[2][0]=Double.valueOf(coord34.nextToken()).doubleValue();
vertici[2][1]=Double.valueOf(coord34.nextToken()).doubleValue();
vertici[3][0]=Double.valueOf(coord34.nextToken()).doubleValue();
vertici[3][1]=Double.valueOf(coord34.nextToken()).doubleValue();
}


public void open()throws IOException,InterruptedException{
```

```
Random ran=new Random();//Il seme è il tempo corrente
DecimalFormat four=new DecimalFormat("00.0000");
open=true;
while (true){
if(open==false){
break;
}
/*
* Le coordinate sono state caricate dal file map.txt
* A questo punto bisogna generare i numeri casuali all'interno del range del rettangolo
*/

latitudine= vertici[0][0]+ (ran.nextDouble()*(vertici[2][0]-vertici[0][0]));
longitudine=vertici[1][1]+ (ran.nextDouble()*(vertici[3][1]-vertici[1][1]));

/*
* Adesso bisogna portare queste informazioni nel formato NMEA 0183
*/

double latDec=latitudine-38;
double lonDec=longitudine-13;
double latPrimi=latDec*60;
double lonPrimi=lonDec*60;


latPrimi=Double.valueOf(four.format(latPrimi).replace(',','.')).doubleValue();
lonPrimi=Double.valueOf(four.format(lonPrimi).replace(',','.')).doubleValue();
String Str=new String(Double.toString(latPrimi));
int index=Str.indexOf('.');
String subLatStr=Str.substring(0,index);
String primo;
if(subLatStr.length()==1)
primo=new String("380"+latPrimi);
else
   primo=new String("38"+latPrimi);


//subLatStr=Str.substring(index+1);
//System.out.println(primo);
String terzo;
if(primo.length()==7)
terzo=new String(primo+"00");
else if(primo.length()==8)
terzo=new String(primo+"0");
else terzo=new String(primo);

//System.out.println(terzo);

String secondo;
Str=new String(Double.toString(lonPrimi));
index=Str.indexOf('.');
String subLonStr=Str.substring(0,index);
if(subLonStr.length()==1)
secondo=new String("0130"+lonPrimi);
else
secondo=new String("013"+lonPrimi);
//System.out.println(secondo);

String quarto;
if(secondo.length()==8)
quarto=new String(secondo+"00");
else if(secondo.length()==9)
quarto=new String(secondo+"0");
else quarto=new String(secondo);
//System.out.println(quarto);

Str=new String(terzo+" N "+quarto+" E");

DatagramSocket datSocket;
byte[] data=new byte[1024];
data =Str.getBytes(); //Converte il messaggio in un array di byte
datSocket=new DatagramSocket(9998);
//crea il pacchetto
InetAddress address=InetAddress.getByName("127.0.0.1");
```

```
//  InetAddress myAddress=InetAddress.getByName("Pocket_Pc_1.mshome.net");
DatagramPacket sendPacket= new DatagramPacket(data, data.length,/*InetAddress.getLocalHost()*/address,9999);
datSocket.send(sendPacket);
messageAreaIn.append("\n Pacchetto UDP spedito a "+address+"\n contenente "+Str+"\n");
datSocket.close();
Thread.sleep(4000); //Ogni 4 secondi spedisce un pacchetto UDP
}
 }

public void close(){
open=false;
}

 }
```

Bibliografia

[1] Gregory T. French, *Understanding the GPS: An Introduction to the Global Positioning System*, GeoResearch 1996.

[2] Fastrax Oy, "NMEA Protocol Specification", December 2002.

[3] Sun Microsystems, "Java Communication API" .

[4] Sun Microsystems, "Java Servlets API" .

[5] Sun Microsystems, "JDK 1.1.8 API" .

[6] Jason Hunter, William Crawford, *Java Servlet Programming 2Ed.*, O'Reilly 2001.