TESI DI LAUREA

Sistema Informativo Mobile Geolocalizzato per Supporto Mnemonico Basato su Architettura Android e RESTful Web Service

Corso di laurea specialistica in Ingegneria Informatica

Relatore:

Chia.mo Prof. Mauro Migliardi

Laureando:

Alessio Toso – matr. 586739-IF

Anno Accademico

2010-2011


DEPARTMENT OF
INFORMATION
ENGINEERING
UNIVERSITY OF PADOVA

A Barbara
e ai miei genitori

Abstract

Nel campo psico-fisiologico esistono molti studi che correlano alti livelli di stress a difficoltà nel trasferire le informazioni dalla memoria a breve termine a quella a lungo termine. Questo, in moltissime situazioni quotidiane può portare a non sfruttare correttamente la memoria diminuendo drasticamente l'efficienza personale. La situazione può degenerare sino all'Activity Thrashing: l'incapacità di organizzare gli impegni e di portarli a termine. Il presente progetto si prefigge il fine di dare un ausilio agli utenti supportandoli nel ricordare le attività da loro identificate nel luogo e nel tempo adatto a svolgerle. Il sistema fornisce un ausilio attivo, in quanto partendo dalle esigenze dell'utente espresse in linguaggio pseudo-naturale, ne estrae automaticamente delle parole chiave per l'interrogazione di servizi online e, tramite la geolocalizzazione, suggerisce, in base alla posizione, cosa è possibile portare a termine nelle immediate vicinanze.

In the field of psyco-physiology many studies relates high stress levels with the difficulty to transfer information from short-term memory to long term memory. This, in many daily situations, may lead to inefficient use of the human memory and may severely reduce the personal efficiency. This effect may even degenerate into an activity thrashing, where the user is not capable any longer to organize in a coherent and efficient way his activities. This project aims at providing support to users by reminding them the tasks they have previously identified in the site and at the time where and when those tasks may efficiently performed. The system provide an active support, in fact it starts from user needs expressed in pseudo-natural language, it extracts keywords from those needs, it queries online services and uses geo-localization to suggest what can be efficiently performed around the user current location.

Indice

Indice delle figure

Capitolo 1 . Introduzione

1.1 . Definizione del problema

La frenesia della vita quotidiana e gli innumerevoli impegni, personali e professionali, portano le persone a dover utilizzare ogni giorno capacità mnemoniche e organizzative complesse: tali capacità si sviluppano con l'esperienza ma, in determinate fasi della vita, possono essere limitate a causa di situazioni particolari di stress che non permettono di mantenerne l'efficienza.

Così ci si trova a dover affrontare problemi o disagi legati all'impossibilità di ricordare puntualmente le attività che, nel quotidiano di ogni persona, dovrebbero essere portate a termine; o spesso la memoria sembra riattivarsi quando è ormai troppo tardi per poter svolgere una particolare attività.

In casi estremi, ma non rari, tali situazioni possono rivelarsi degenerative: livelli molto alti di stress, se non correttamente gestiti, possono determinare un costante danneggiamento delle funzioni della memoria esplicita cosciente.

Approfonditi studi medici hanno dimostrato che le situazioni di stress inducono le ghiandole surrenali alla produzione dei cosiddetti ormoni dello stress, quali ad esempio adrenalina, noradrenalina e ormoni steroidi: la produzione persistente ed elevata di tali ormoni può determinare danni all'ippocampo, che rappresenta la principale struttura coinvolta nella memorizzazione dei ricordi coscienti (memoria esplicita).

Senza ricorrere a estremi esempi di degenerazione, ogni persona ha sicuramente sperimentato situazioni di disagio dovute al non ricordare nel momento giusto le azioni da portare a termine (o a totalmente dimenticarle), sprechi di tempo dovuti al

dover ripercorrere una strada, al dover tornare in un luogo dove si è stati poco prima o dove non si è stati ma al quale si è passati vicini.

Si tratta nella maggior parte dei casi di piccoli disagi, di piccole perdite di tempo, che però sommati tra loro, in una società in cui il tempo sembra ed è sempre poco, possono risultare irritanti quando non onerosi.

Il presente lavoro di tesi si prefigge quindi l'obiettivo di realizzare un sistema che, con l'ausilio di strumenti portatili sempre più evoluti e sempre più diffusi, agevolino l'utilizzatore ad affrontare quotidianamente le situazioni descritte. Con strumento portatile evoluto ci si riferisce in particolar modo agli smartphone, terminali in grado di svolgere molteplici compiti e dotati di una serie di sensori e dispositivi che permettono un'interazione con l'utente molto proficua, in particolare: antenne GPS integrate che permettono di geolocalizzare in qualsiasi momento il terminale, connessioni a banda larga, sistemi di acquisizione vocale che consentono di comandare il cellulare attraverso il semplice utilizzo della voce.

Al fine di aumentare la qualità e l'efficienza dell'interazione uomo-macchina, il presente progetto ha previsto l'utilizzo di alcune di queste funzionalità.

1.2 . Scopo del progetto

Il presente contributo è il proseguimento di un progetto iniziato qualche anno fa grazie al prezioso lavoro svolto da Giorgio Ravera (con la sua tesi di laurea dal titolo "Sistema di supporto Mnemonico") e da Petrus Prasetyo Anggono (con la sua tesi di laurea dal titolo "A Mobile, Context Aware System for Memory Support and Planning").

Basandosi sul software creato da Petrus Prasetyo Anggono e alla luce del contesto di riferimento descritto nel primo paragrafo, il presente lavoro di tesi si è dato l'obiettivo di sviluppare un sistema che permetta a chi lo utilizza di portare a termine i propri impegni in modo completo e soddisfacente, affidando ad esso il compito di ricordarli.

Brevemente, il sistema permette all'utilizzatore di memorizzare gli impegni, i compiti, le attività da eseguire (task). Grazie all'utilizzo del sistema GPS, quando l'utente si troverà ad una certa distanza da punti di interesse (negozi, ristoranti, uffici o quant'altro di necessità) che possono soddisfare uno o più dei task memorizzati, il sistema lo segnalerà attraverso delle notifiche.

Le notifiche sono di diverse tipologie (audio, vocali, visuali, vibrazioni) che possono essere scelte, anche in combinazione, dall'utente stesso a seconda delle proprie esigenze e preferenze.

Grazie all'ausilio di questo sistema, l'utilizzatore può concentrarsi in modo più sereno e rilassato in altre attività, limitando così non solo la concentrazione di informazioni da tenere a mente e pianificare ma anche, conseguentemente, il livello di stress ad

essa collegato.

Dal un lato il sistema si inserisce quindi nella vita quotidiana delle persone, cercando di creare nuove opportunità di collegamento tra l'utente, le sue attività ed esigenze ed il contesto in cui egli, momento dopo momento, si trova; dall'altro lato è necessario che tale sistema non sia invasivo, in modo da portare all'utilizzatore benefici e non ulteriori fastidi e scompensi.

1.3 . Struttura della tesi

Nelle pagine seguenti sono illustrate le caratteristiche del sistema di partenza relative a piattaforma client e server utilizzate e strumenti di sviluppo; sono inoltre presentate le funzionalità supportate sia dal terminale mobile che dal server (Capitolo 2 .).

Nel Capitolo 3 . sono descritte le proposte sorte dal gruppo di lavoro attraverso l'analisi del software di partenza, il riconoscimento dell'esigenza di migliorarne la stabilità, e l'individuazione delle potenzialità di sviluppo delle funzionalità.

Sono poi riportate le descrizioni tecniche delle soluzioni implementate (Capitolo 4 .) e la descrizione dei punti di sviluppo futuro (Capitolo 5 .).

Al fine di valutare diversi aspetti del sistema, sono stati attuati una serie di test: dall'offline test del codice, per verificare che funzionamento atteso ed effettivo (delle varie unità del sistema) corrispondessero, ad un test di prova dell'applicazione, con dei terminali, nelle sue funzionalità e potenzialità, seguendo un prefissato protocollo sperimentale che potesse permettere la riproducibilità del test stesso al fine di poterlo confrontare con test futuri. In questa seconda serie di test sono state coinvolte persone esterne al progetto per valutarne facilità di utilizzo, efficienza, percezione di utilità e gradimento da parte di potenziali utenti finali. Le modalità di svolgimento dei test e i risultati ottenuti sono descritti nel Capitolo 6 .

Capitolo 2 . Il sistema attuale

2.1 . Piattaforma client utilizzata

2.1.1 . Scelta della piattaforma

La presente applicazione è stata sviluppata per una serie di terminali per così dire avanzati. La scelta di questa tipologia di device è legata al fatto che il terminale deve poter aver accesso ad una serie di risorse, in particolare la possibilità di posizionamento geografico tramite utilizzo del segnale GPS e la connessione ad internet, preferibilmente a banda larga per evitare tempi di latenza troppo lunghi. Al giorno d'oggi è inoltre risaputo che il cellulare, ormai alla portata di tutti, è un oggetto di utilizzo quotidiano, sia per lavoro che per uso personale. Lo smartphone è pertanto lo strumento ideale per concentrare in un unico e compatto strumento tutte le funzionalità sopra elencate.

I brand che costruiscono smartphone sono molti. Il presente progetto è completamente open-source, perciò si è scelto di utilizzare il sistema operativo Google Android, uno stack comprensivo di strumenti utili che permettono ad uno sviluppatore di progettare ed eseguire applicazioni ricche di funzionalità. Essendo Android un sistema operativo sviluppato da Google, sono presenti, inoltre, una serie di automatismi ed API che permettono di interfacciare qualsiasi applicazione con una delle più grosse banche dati esistenti al mondo per quanto riguarda punti di interesse e business: Google Maps.

Android ha anche un altro punto di forza: nonostante sia un sistema piuttosto giovane, nel giro di pochi mesi ha letteralmente scalato le classifiche di vendita posizionandosi al secondo posto dopo un colosso come Symbian ma soprattutto superando il sistema operativo dell'azienda di Cupertino, l'IOS per gli Apple IPhone (Figura 2.1) e rispetto ai quali è completamente open. Si può prevedere pertanto una forte e capillare

diffusione del sistema operativo targato Google in tempi piuttosto brevi.

Worldwide Smartphone Sales to End Users by Operating Systems in 2010 (thousand of Units)

Company	2010 Units	2010 Market Share (%)	2009 Units	2009 Market Share (%)
Symbian	111,576.7	37.6	80,878.3	46.9
Android	67,224.5	22.7	6,798.4	3.9
Research In Motion	47,451.6	16.0	34,346.6	19.9
iOS	46,598.3	15.7	24,889.7	14.4
Microsoft	12,378.2	4.2	15,031.0	8.7
Other Oss	11417,4	3.8	10432.01.00	6.1
Total	**296,646,6**	**100.0**	**172,376.1**	**100.0**

Source: Gartner (February 2011)

Figura 2.1: Vendite smartphone 2010. Fonte gartner (Febbraio 2011)

2.1.2 . Struttura sistema operativo Android

Android è una piattaforma open-source per telefoni cellulari, basata sul sistema operativo Linux ed è sviluppata dall'Open Handset Alliance. La piattaforma ha a disposizione delle librerie dedicate, come il database SQLite o SGL e OpenGL dedicate alla grafica, un application framework, la Dalvik virtual machine (una Java virtual machine modificata) come ambiente di runtime e una serie di applicazioni preinstallate come un browser, una rubrica e un calendario. Il 12 novembre 2007 l'OHA ha rilasciato il software development kit (SDK) che include: strumenti di sviluppo, librerie, un emulatore del dispositivo, la documentazione (in inglese), alcuni progetti di esempio, tutorial, FAQ, e altro. Pesa 58 Megabyte ed è installabile su qualsiasi computer x86 che usa come sistema operativo Windows XP o Vista, Mac OS X 10.4.8 e successivi, o Linux. È anche possibile utilizzare il plug-in per Eclipse. L'SDK è stato aggiornato alla versione 0.9 il 18 agosto 2008: questa nuova versione pesa 90 Megabyte, ed e ormai la definitiva, saranno infatti minime le modiche effettuate con il rilascio della versione 1.0.

Il primo dispositivo mobile dotato della piattaforma Android è il T-Mobile G1, prodotto dalla società taiwanese HTC e commercializzato dal carrier telefonico T-Mobile. Il prodotto è stato presentato il 23 Settembre a New York. Allo stato dell'arte si è giunti alla versione 2.2 denominata Froyo e di recente al rilascio della versione 2.3 Gingerbread che, a detta degli sviluppatori, ha apportato importanti innovazioni come ad esempio il supporto per la tecnologia Near Field Communication, un sistema di trasmissione wireless a cortissimo raggio (fino a 4cm di distanza) che potrà sostituire in futuro le normali carte di credito.

L'architettura di Android si presenta suddivisa in diversi livelli, come mostrato in Figura 2.2.

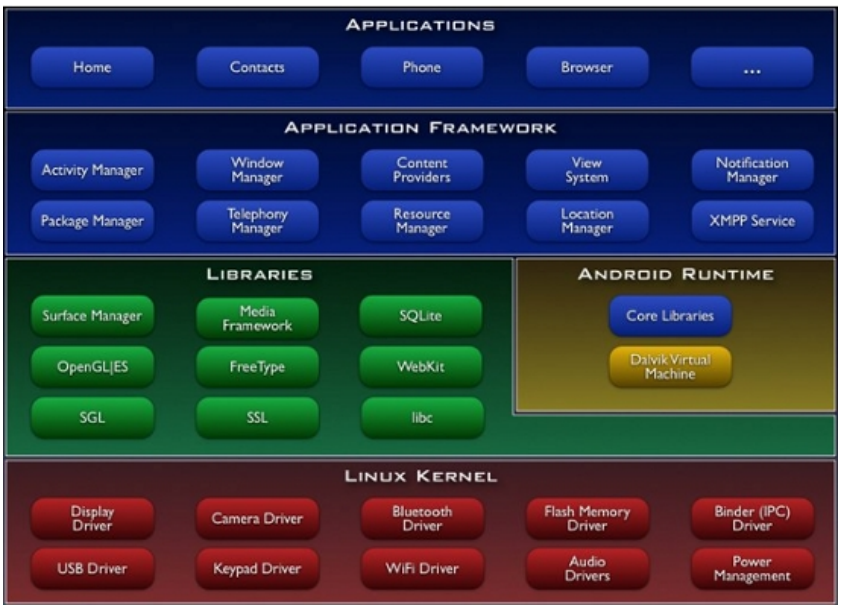


Figura 2.2: Architettura Android

Alla base dello stack Android troviamo un kernel Linux nella versione 2.6. La scelta di una simile configurazione è nata dalla necessità di disporre di un vero e proprio sistema operativo che fornisse gli strumenti di basso livello per la virtualizzazione dell'hardware sottostante attraverso l'utilizzo di diversi driver. A differenza di un kernel Linux standard per Android sono stati aggiunti ulteriori moduli come:

- *Binder IPC Driver*, un driver dedicato che permette a processi di fornire servizio ad altri processi attraverso un insieme di API di più alto livello rispetto a quelle presenti su un kernel linux standard, ciò permette la comunicazione tra processi con un costo computezionale minore e un relativo minore consumo di batteria;
- *Low Memory Killer*, un sistema che si preoccupa di terminare i processi liberando così spazio nella memoria centrale per soddisfare le richieste di un altro processo. La terminazione dei processi avviene secondo un sistema di ranking che assegna dei punteggi ai processi; i processi che verranno terminati saranno quelli con punteggio più alto. Ad esempio, un processo che controlla la UI (User Interface) di un'applicazione visibile sarà sicuramente più basso di quello relativo ad un'applicazione non visibile sullo schermo. Il processo init non può essere terminato;
- *Ashmem*, sistema di memoria condiviso anonimo (Anonimous Shared Memory) che definisce interfacce che consentono ai processi di condividere zone di memoria attraverso un nome. Il vantaggio di Ashmem rispetto ai sistemi Linux che fanno uso della memoria condivisa, è che viene fornito al kernel uno strumento per recuperare dei blocchi di memoria non utilizzati;
- *RAM Console e Log devices*, per agevolare il debugging, Android fornisce la capacità di memorizzare i messaggi di log generati dal kernel in un buffer RAM. È disponibile inoltre un modulo separato che permette ai processi

utente di leggere e scrivere messaggi di log;
- *Android Debug Bridge*, uno strumento che permette di gestire in maniera versatile un'istanza dell'emulatore o eventualmente di un dispositivo reale;
- *Power Management*, sezione progettata per permettere alla CPU di adattare il proprio funzionamento al fine di non consumare energia se nessuna applicazione o servizio ne fa richiesta.

Esistono poi delle librerie native, realizzate in C-C++, che si trovano sopra al livello kernel e che rappresentano il vero e proprio core del sistema oeprativo:
- *Surface Manager*: il modulo che gestisce le View, cioè i componenti di una interfaccia grafica. Funziona praticamente come uno scheduler per la gestione della visualizzazione delle interfacce grafiche e previene eventuali problemi di accavallamento scoordinato sul display. E' un componente di vitale importanza per un terminale che fa delle interfacce grafiche un punto di forza del suo funzionamento;
- *Open GL SE*: è una libreria grafica, versione light della libreria OpenGL per terminali mobili, che permette di utilizzare grafica 3D;
- *SGL (Scalable Graphics Library)*: libreria scritta in C++ che permette di gestire la grafica 2D;
- *Media Framework*: componente in grado di gestire i contenuti multimediali (Codec per l'acquisizione e riproduzione di contenuti audio e video);
- *FreeType*: componente che gestisce i font;
- *SQLite*: Database relazionale piccolo ed efficiente messo a disposizione dello sviluppatore per la persistenza dei dati nelle varie applicazioni sviluppate;
- *WebKit*: browser engine open source (utilizzato anche per i più conosciuti browser web Safari e Chrome). Da sottolineare che non si tratta di un browser web, quindi dovrà essere integrato nelle diverse applicazioni;
- *SSL*: si tratta della famosa libreria per la gestione del Secure Socket Layer.

Le applicazioni Android vengono sviluppate in Java. Serve pertanto una Macchina Virtuale in grado di eseguire le applicazioni. È presente infatti una Macchina Virtuale ottimizzata dalla Sun Microsystem (la Dalvik Virtual Machine) per l'esecuzione di software su sistemi con risorse limitate, quali possono essere ad esempio i dispositivi portatili come gli smartphone. Tale macchina virtuale è in grado di eseguire codice contenuto in file con estensione `.dex`, generati a partire dal bytecode Java. I file `.dex`, hanno un tipo di compressione che permette di dimezzare lo spazio utilizzato rispetto ai file `.jar` non compressi. Anche la Dalvik Virtual Machine, similmente per quanto accade per la JVM, implementa un Garbage Collector, liberando pertanto lo sviluppatore dall'onere della gestione della memoria.

Per quanto riguarda, infine, l'Application Framework, lo sviluppatore ha a disposizione una serie di API che sfruttano le librerie sottostanti di Android. Sono presenti i seguenti moduli:
- *Activity Manager*: modulo che gestisce tutto il ciclo di vita delle activity. Le activity sono entità associate ad una schermata, rappresentano quindi l'interfaccia verso l'utente. Il compito dell'Activity Manager è quello di gestire

le varie activity sul display del terminale e di organizzarle in uno stack in base all'ordine di visualizzazione sullo schermo;
- *Package Manager*: modulo che gestisce i processi di installazione e rimozione delle applicazioni dal sistema;
- *Telephony Manager*: modulo che gestisce l'interazione con le funzioni tipiche di un cellulare;
- *Content Provider*: modulo che gestisce la condivisione di informazioni tra i vari processi attivi. Il suo utilizzo è simile a quello di un repository comune nel quale i vari processi possono leggere e scrivere informazioni;
- *Resource Manager*: modulo deputato alla gestione delle informazioni relative ad una applicazione (file di configurazione, file di definizione dei layout, immagini utilizzate, …);
- *View System*: gestisce l'insieme delle viste utilizzate nella costruzione dell'interfaccia verso l'utente (bottoni, griglie, text boxes, …);
- *Location Manager*: modulo che mette a disposizione dello sviluppatore una serie di API che si occupano della localizzazione. Esistono due provider per la localizzazione: GPS e NETWORK. GPS utilizza i satelliti geostazionari per il posizionamento geografico, ha bisogno però della vista del cielo e risente delle cattive condizioni atmosferiche. NETWORK utilizza punti dei quali si conosce la posizione geografica, come ad esempio celle GSM oppure reti wireless geolocalizzate (Hot Spot);
- *Notification Manager*: mette a disposizione una serie di meccanismi utilizzabili dalle applicazioni per notificare eventi al dispositivo che intraprenderà delle particolari azioni in conseguenza della notifica ricevuta.

2.1.3 . Componenti di una applicazione Android

Una caratteristica innovativa del modello di Android, che si differenzia da altre piattaforme, è un supporto per il riutilizzo dei componenti. Con componente si intende un qualsiasi programma, applicazione o servizio implementati. Le applicazioni non sono più pensate come entità a sé stanti, ma come un insieme di componenti disponibili nel sistema e richiamabili all'occorrenza, nel momento in cui servono. Ne scaturisce un vantaggio non indifferente per lo sviluppatore, che si traduce in una maggiore velocità nella realizzazione di una applicazione in quanto non serve reimplementare dall'inizio. Nel suo piccolo, Android implementa il paradigma, mutuato dall'ingegneria del software, di Software Integration, tipico di progetti di grosse dimensioni per i quali non si punta più a sviluppare una unica applicazione, ma all'integrazione ed interazione di più componenti preesistenti.

In una applicazione Android esistono sostanzialmente quattro tipologie di componenti:
- Activities: come indicato in precedenza, una Activity è una entità alla quale è collegata una particolare interfaccia utente. Ogni applicazione per l'interazione con l'utente mostra una serie di activity attraverso le quali è possibile interagire con il sistema sottostante. All'avvio Android esegue l'Activity

marcata come principale e da questa, in seguito alle azioni intraprese dall'utente, esegue le successive Activity formando così l'intera interfaccia utente;
- Services: sono dei task che vengono eseguiti in background per un indefinito periodo di tempo. Sono ad esempio i processi che lavorano sotto all'interfaccia grafica per elaborare le informazioni inserite dall'utente. Buona norma è far eseguire un Service, soprattutto per compiti di lunga durata, in un thread separato rispetto al main thread;
- Broadcast Receivers: è un componente che ha il compito di eseguire delle azioni solo in seguito alla ricezione di un determinato evento, una sorta di gestione degli interrupt. Un evento può essere ad esempio la notifica del basso livello della batteria o il cambio di fuso orario. Un'applicazione può avere più di un broadcast receiver per rispondere agli annunci che si reputano importanti. Ogni ricevitore estende la classe `BroadcastReceiver`;
- Content Providers: utilizzati per mettere a disposizione uno specifico sottoinsieme di dati della propria applicazione ad altre applicazioni.

Nel momento in cui viene richiesto uno specifico componente di una applicazione, Android fa sì che il processo legato a questa sia in esecuzione, iniziandolo se necessario.

2.1.3.1 . Ciclo di vita dei componenti

Tutti i componenti di un'applicazione hanno un ciclo di vita che ha inizio nel momento in cui Android le istanzia con lo scopo di rispondere ad un intent e termina quando le loro istanze vengono distrutte.

Activity

Il ciclo di vita di una Activity, ad esempio, è indicato in Figura 2.3. Come è possibile notare, una Activity si trova in uno degli stati indicati in figura. Per sua natura Android tenta di mantenere attive più applicazioni possibile, per minimizzare il dispendio di energia per riattivare da zero una applicazione in caso di necessità. È compito del Garbage Collector scegliere quale applicazione rimuovere dalla memoria in caso di necessità. Il punteggio, accennato precedentemente, viene calcolato anche in base allo stato in cui si trovano le activity dell'applicazione in oggetto.

Service

Un servizio può invece essere utilizzato in due modi:
- può essere lanciato da un client invocando il metodo `Context.startService(Intent)` e fermato utilizzando il metodo `Context.stopService(Intent);` oppure il servizio si può anche fermare in maniera autonoma attraverso la chiamata al metodo `Service.stopSelf()` oppure `Service.stopSelfResult()`. Per

fermare un servizio occorre chiamare `stopService()` una volta sola mentre il metodo `startService()` può essere invocato più volte;

- se il client ha bisogno di interagire con le funzionalità che il servizio mette a disposizione attraverso la pubblicazione della sua interfaccia, deve stabilire una connessione con il servizio stesso utilizzando il metodo `Context.bindService()`. Questa connessione può essere chiusa invocando il metodo `Context.unbindService()`. Più client possono registrarsi allo stesso servizio il quale, se non attivo, può essere lanciato con il metodo `bindService()`.

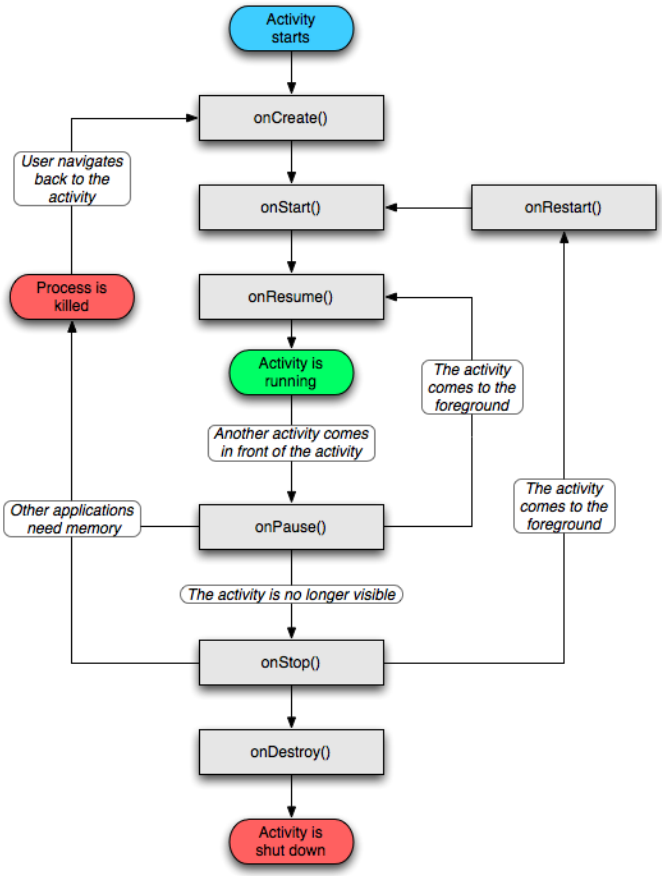


Figura 2.3: Ciclo di vita di una Activity

Broadcast Receiver

Quando un messaggio broadcast giunge al ricevitore, Android chiama il relativo metodo `onReceive()` passando l'oggetto `Intent` contenente in messaggio. Il broadcast receiver viene considerato attivo solo durante l'esecuzione del metodo `onReceive()`, quando questo metodo termina la sua esecuzione il receiver torna ad essere inattivo. Un processo che detiene un broadcast receiver attivo non può essere ucciso.

2.2 . Piattaforma server utilizzata

2.2.1 . Web Service

Secondo la definizione data dal World Wide Web Consortium (W3C) un Web Service (o servizio web) è un sistema software progettato per supportare l'interoperabilità tra diversi elaboratori su di una medesima rete; caratteristica fondamentale di un Web Service è quella di offrire un'interfaccia software (descritta in un formato automaticamente elaborabile quale, ad esempio, il WSDL, Web Services Description Language) utilizzando la quale altri sistemi possono interagire con il Web Service stesso, attivando le operazioni descritte nell'interfaccia tramite appositi messaggi utilizzando dei particolari protocolli (il più famoso è il SOAP).

La caratteristica dei Web Service di fornire dei servizi indipendentemente dalla piattaforma utilizzata, lo rende il candidato ideale per il progetto in esame. Si è deciso di utilizzare come architettura client il sistema operativo Android, ma nulla vieta di poter utilizzare qualsiasi altra piattaforma (altrettanto versatile) come ad esempio i diffusi I-Phone oppure Symbian. Un Web Service permette, difatti, di essere trasparente al linguaggio di programmazione. Nella fattispecie viene utilizzato Java Android lato client e Java lato server, ma potrebbe venire utilizzato lato client un qualsiasi altro linguaggio di programmazione. Lo scambio dei dati avviene tramite il metodo GET o POST messi a disposizione dal protocollo HTTP oppure utilizzando direttamente il formato XML, tutti metodi indipendenti dal linguaggio di programmazione utilizzato. Per rendere, inoltre, più semplice l'implementazione dell'applicazione lato client e per un migliore utilizzo della banda disponibile (spesso si utilizzano formati più compatti del SOAP), si è scelto di realizzare un REST (Representational State Transfer) Web Service.

2.2.2 . RESTful Web Service

REST è una particolare tipologia di architettura software per la comunicazione nei Web Service. Tale architettura utilizza principalmente, a livello di applicazione, il protocollo HTTP e non un protocollo proprietario (es. SOAP). Inizialmente REST venne descritto da Fielding nel contesto del protocollo HTTP; un sistema RESTful, però, si può tranquillamente appoggiare ad un qualunque altro protocollo che fornisca un vocabolario altrettanto ricco. A differenza di altre specifiche per Web Service (es. SOAP), REST sfrutta infatti appieno la semantica e la ricchezza dei comandi HTTP e le sue funzionalità, come ad esempio la negoziazione dei contenuti. Perché un Web Service sia conforme alle Specifiche REST deve avere alcune specifiche caratteristiche:

- architettura basata su client/server;
- stateless, cioè ogni ciclo di request/response deve rappresentare un'interazione completa del client con il server;
- uniformemente accessibile, cioè ogni risorsa deve avere un indirizzo univoco

ed ogni risorsa di ogni sistema presenta la stessa interfaccia, precisamente quella individuata dal protocollo HTTP.

Il RESTful rappresenta una serie di principi architetturali per la progettazione dei Web Service il cui concetto centrale è quello di risorsa, che rappresenta una qualunque entità che possa essere indirizzata tramite Web, in parole povere accessibile e trasferibile tra client e server. Spesso rappresenta un oggetto appartenente al dominio del problema che si sta trattando. Durante un'interazione tra client e server, quello che viene trasferito è una rappresentazione dello stato interno della risorsa. Vengono inoltre utilizzate, al fine di agevolare il processo di sviluppo del software, dei supporti quali ad esempio JAX-RS e JAXB.

JAX-RS (Java API for RESTful Web Service) sono una serie di API in liguaggio Java che permettono il supporto per la creazione di Web Service in accordo con le specifiche dell'architettura REST. JAX-RS utilizza le annotazioni[1], introdotte dalla versione 1.5 di Java, per semplificare lo sviluppo e il deploy dei web service. Dalla versione 1.1, JAX-RS fa ufficialmente parte di Java EE.

JAX-RS rende disponibile l'utilizzo di particolari annotazioni per mappare una classe che contiene delle risorse accessibili via web. Le annotazioni includono:

- *@Path* – specifica il path relativo per la risorsa;
- *@GET, @PUT, @POST, @DELETE* – specificano le tipologie di richieste HTTP;
- *@Produces* – specifica la tipologia di media ritornato;
- *@Consumes* – specifica la tipologia di media accettato;
- *@PathParam, @QueryParam, @HeaderParam, @CookieParam, @MatrixParam, @FormParam* – specificano la provenienza dei parametri passati al metodo remoto; ad esempio con *@PathParam* provengono dall'URL, *@QueryParam* provengono dai parametri di tipo query dell'URL, *@HeaderParam* provengono dall'header del messaggio HTTP.

Nel sistema descritto, si fa ampio uso delle API JAX-RS. La tecnica utilizzata per la sincronizzazione (data binding) dei dati tra il server ed il client avviene utilizzando dei file XML, creati a partire dagli oggetti Java creati durante l'elaborazione. Lato client viene successivamente utilizzato SAX[2] per recuperare le informazioni contenute in un file XML e ricreare l'oggetto Java di partenza. La creazione del file XML avviene utilizzando JAXB (Java Architecture for XML Binding) che permette di realizzare una mappatura tra le classi Java e una loro corrispondente rappresentazione sotto forma di file XML. JAXB permette perciò di serializzare oggetti Java in XML (effettua il marshalling) e di effettuare l'operazione inversa (unmarshalling), quindi dalla rappresentazione XML riottenere l'oggetto Java senza dover implementare alcuna routine per l'elaborazione di file XML.

1 Annotazione: particolare tipologia di metadato che in Java è possibile aggiungere al codice sorgente di un programma. Le annotazioni possono essere aggiunte a classi, metodi, parametri, variabili o pacchetti. A differenza dei tag aggiunti dalla documentazione Java, sono completamente accessibili al programmatore mentre il software è in esecuzione.

2 SAX: Simple API for XML, rappresenta un parser sequenziale per l'accesso alle informazioni contenute in un file XML. SAX implementa un meccanismo per l'accesso in lettura al file XML.

2.3 . Piattaforma di sviluppo

2.3.1 . Repository Google, SVN

In generale, quando si decide di sviluppare un progetto (anche di ridotte capacità), è sempre buona norma affiancare ad esso un gestore del controllo della versione. Si tratta di sistemi che gestiscono la continua evoluzione del codice sorgente del progetto, tenendo traccia di tutte le modifiche apportate e delle versioni.

Esistono vari tool in questo campo: CVS, SVN, Mercurial sono solo alcuni esempi. Le motivazioni che spingono alla scelta di un sistema di controllo della versione sono molteplici, le principali potrebbero però essere:

- Evitare perdite di dati e di tempo per involontarie cancellazioni;
- Programmazione asincrona: quando più sviluppatori fanno parte del progetto, la gestione concorrente dello sviluppo è un fattore cruciale al fine di evitare i cosiddetti salvataggi fantasma;
- Evitare di dover fare dei backup quando viene realizzata una release stabile e di doverli distribuire tra i vari componenti del gruppo di sviluppo;
- Confrontare velocemente le varie versioni, visto che solitamente i sistemi di controllo della versione utilizzano i diff, cioè un meccanismo che permette di memorizzare solo le differenze tra un file di una versione e quello modificato.

L'utilizzo di sistemi di controllo è molto utile e consigliato, ma solitamente dev'essere accompagnato dalla possibilità di avere un repository online (accessibile tramite Internet) e non in locale, per dare la possibilità anche a persone che si trovano a chilometri di distanza di poter partecipare allo sviluppo del codice del progetto.

Per il presente progetto si è scelto, anche per continuare sulla strada intrapresa da chi ha dato un contributo in precedenza, di utilizzare SVN come sistema di gestione della versione e di ospitare il codice scritto nei repository online di Google Code. Google Code mette a disposizione 1GB di spazio online per ogni progetto, con le uniche condizioni di avere un account Google per l'accesso al repository e di sviluppare un software che abbia una licenza approvata dall'OSI (Open Source Initiative).

2.3.2 . Eclipse

Per lo sviluppo del codice, si è scelto di utilizzare Eclipse[3], un ambiente di sviluppo integrato che permette di agevolare la scrittura di codice, nella fattispecie Java.

Eclipse è un progetto IDE (Integrated Development Environment) open-source e fornisce una serie di strumenti che aiutano lo sviluppatore nella programmazione, fornendo ad esempio la possibilità di includere e creare JavaDoc (sistema di documentazione del codice), suggerimenti sulle classi e metodi da utilizzare,

3 Eclipse: IDE open-source disponibile presso http://www.eclipse.org/

inclusione di librerie.

Tramite una serie di plugin permette di interfacciarsi con tutti i componenti che si sono resi necessari allo sviluppo dell'applicativo, in particolare:

- Subclipse[4]: plugin per l'accesso ed il controllo direttamente da Eclipse di SVN;
- Maven Integration for Eclipse: plugin per l'integrazione in Eclipse di Apache Maven, il sistema per la gestione dei progetti, dalla forma (archetipo Maven) alla gestione dei repository per le varie dipendenze necessarie, al rilascio del pacchetto finale;
- Eclipse Web Developer Tool: pacchetto che comprende tutti i componenti necessari alla programmazione Web, in particolare con Java EE ed il supporto per la creazione di server locali;
- Android Developer Tools (ADT): plugin per l'integrazione dell'Android SDK in Eclipse, per poter gestire l'emulatore Android e una serie di altri componenti (ad esempio DDMS, Dalvik Debug Monitor Server) utili nello sviluppo di applicazioni per Android.

In APPENDICE è presente una guida dettagliata sull'installazione dell'ambiente di sviluppo.

2.3.3 . Maven

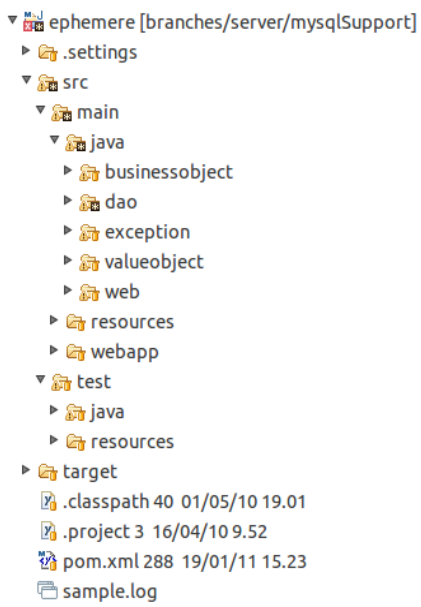
Si tratta di un sistema completo per la gestione dei progetti, che ingloba automatismi per la documentazione, distribuzione e collaborazione. Per funzionalità è similare ad Apache Ant ma basato su concetti differenti. Maven è ospitato da Apache Software Foundation, dove faceva parte dell'ex progetto Jakarta. E' un tentativo di riunire in un unico prodotto una serie di pattern ben consolidati e collaudati da applicare all'infrastruttura di build di progetti Java.

Il concetto principale è la dichiarazione di alcune convenzioni sulla configurazione del progetto, definendo ad esempio una organizzazione standard della directory dei progetti, cioè la collocazione del codice sorgente, dei file di configurazione, della documentazione e degli artefatti generati. Questo definisce l'Archetipo del progetto. Con Archetipo si intende il modello dal quale tutti gli altri oggetti dello stesso tipo sono creati. L'Archetipo definisce pertanto lo standard di un progetto. Coerentemente con l’impostazione di Maven, applicare tale standard non è obbligatorio, sebbene sia fortemente consigliato: Maven fornisce una serie di strumenti che permettono di utilizzare strutture diverse e, in casi molto particolari, è anche possibile non conformarsi agli standard. Ma occorre comunque ricordare che deviare dalle impostazioni standard tende a invalidare, o comunque a ridurre, la portata di tutta una serie di vantaggi, quali quelli tipici derivanti dal fatto che si tratta di uno “standard”.

La struttura del presente progetto, ricalca fedelmente il pattern descritto dagli

4 Subclipse: disponibile presso http://subclipse.tigris.org/update_1.6.x

archetipi di Maven, definendo pertanto una struttura fissa del progetto stesso, come in Figura 2.4.



Per sua natura, Maven è un framework progetto-centrico ed il POM (Project Object Model) è la descrizione del singolo progetto. Qualsiasi aspetto viene specificamente dichiarato e configurato nel file `pom.xml` ubicato nella radice del progetto. Mediante questo file di configurazione Maven gestisce ogni singolo aspetto e stadio del ciclo di vita del progetto.

Altra interessantissima funzionalità di Maven è la gestione e l'organizzazione delle dipendenze. Viene in questo contesto definito il concetto di artefatto, come una specifica parte di software (che può essere un file jar, war, sar, …). Una dipendenza non è altro che un riferimento ad un artefatto specifico disponibile in un preciso repository remoto. Maven cerca di interpretare le coordinate delle dipendenze indicate nel file POM e cerca di soddisfare tali riferimenti cercando gli artefatti richiesti nei repository remoti disponibili nel contesto del progetto, togliendo allo sviluppatore il compito di importare direttamente ogni singola dipendenza ed includerla successivamente nel pacchetto di rilascio. Se la dipendenza richiesta viene correttamente localizzata, Maven provvede al suo downlaod e all'installazione presso il repository locale.

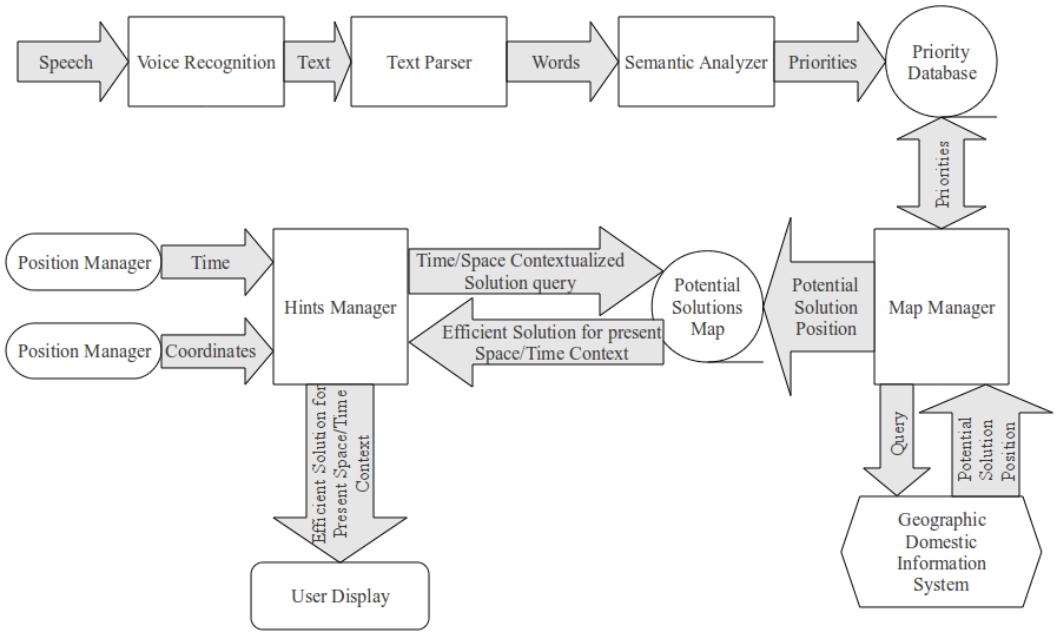
2.4 . Funzionalità supportate

In Figura 2.5 è presente uno schema a blocchi illustrativo del funzionamento del sistema nel suo complesso.

È possibile identificare una parte iniziale di speech recognition residente nel client (il

riconoscimento vocale avviene direttamente nel terminale). C'è una successiva sezione dedicata al riconoscimento dei componenti della frase inserita e di estrazione delle parole chiave dal linguaggio naturale, eseguita lato server. Il cuore del sistema (sempre lato server) utilizza il Map Manager per interagire con un servizio mappe ed estrarre i possibili punti di interesse dove poter soddisfare i task inseriti e creare delle notifiche per l'utente (Hint Manager) da restituire poi alla richiesta del terminale.

La fase di inserimento dei task, non rispetta rigorosamente lo schema indicato, in quanto se l'inserimento del task non avviene in linguaggio naturale (sia esso parlato o scritto) ma utilizzando l'apposita interfaccia nel client (stile form), si saltano i primi tre blocchi per inserire direttamente le informazioni nel database.



2.4.1 . Lato client

Il client è composto da una interfaccia grafica che interagisce con l'utente suddivisa principalmente in tre parti: Activities, Maps e Preferences.

La prima parte mostra all'utente una schermata riassuntiva nella quale possono essere controllati Eventi e Task inseriti. Tramite l'apposito pulsante, si apre un menù nel quale è possibile scegliere tra le seguenti sottovoci:

- Create Event
- Create Task
- Voice Input
- EXIT

Come facilmente intuibile, le voci Create Task e Create Event servono alla creazione di Task e Eventi rispettivamente. Le schermate per l'inserimento di Task o Eventi sono sostanzialmente simili: vengono inseriti i dati relativi ad ogni oggetto e con il pulsante Save è possibile inviare i dati al server il quale provvede alla memorizzazione e, a seconda dei casi, alla elaborazione delle informazioni per la ricerca dei punti di interesse.

La sezione dedicata a Voice Input provvede all'acquisizione di una frase in linguaggio naturale e la invia alla sezione del server, dove risiede il parser che effettua l'analisi sintattica della frase inserita alla ricerca dei vari componenti per successivamente memorizzare le informazioni. L'inserimento può avvenire sia con la tastiera alfanumerica del terminale sia utilizzando il riconoscitore vocale. In quest'ultimo caso, per il funzionamento il terminale deve necessariamente essere connesso a Internet infatti la traduzione dal parlato a stringa testuale non avviene in locale, ma viene fatta una registrazione della voce per poi inviarla al server Google che provvede a restituire la stringa testuale collegata alla registrazione appena fatta.

Il pulsante Exit, esplicativo di per se stesso, provvede a far uscire dall'applicazione.

Nella sezione Maps è possibile visualizzare la mappa nella quale saranno presenti dei markers per identificare la posizione del terminale e la posizione di Task ed Eventi inseriti per un dato utente. E' possibile così avere una panoramica della situazione nelle immediate vicinanze del terminale.

L'ultima sezione, Preferences, è dedicata al settaggio dei parametri di funzionamento del programma. Da qui è possibile impostare il Query Period, l'intervallo di tempo con il quale il client fa il download di eventuali dati dal server, e la Maximum Distance, che identifica il raggio entro il quale cercare i punti di interesse relativi alle keyword inserite dall'utente.

2.4.1.1 . Login

La schermata di Login prevede l'inserimento di nome utente e password per l'autenticazione presso il server. Se il server è raggiungibile e risponde affermativamente alla richiesta di login, si passa alla successiva schermata del programma, Activities, altrimenti vengono indicati eventuali messaggi di errore.

2.4.1.2 . Task ed Event

Task ed Eventi sono gli oggetti con i quali funziona l'applicazione. Gli eventi sono in tutto e per tutto degli impegni geolocalizzati, quindi hanno una determinata posizione nella quale poter soddisfare l'evento, un lasso temporale che indica data e orario di inizio e fine dell'evento, un titolo che identifica l'evento, una descrizione arbitraria e un livello di priorità impostabile da 1 a 5.

I task invece sono degli impegni “generali”. Generali nel senso che esprimono una

necessità dell'utente e che quindi sono soggetti a un reperimento di possibili posti nei quali soddisfarli, generato dal server grazie ad una ontologia impostata. Il sistema in questo caso non si preoccupa solamente di ricordare il Task, ma di cercare, attraverso il motore del server, qualsiasi posizione che possa permettere di soddisfarlo. Sono esempi di Task: “prendere il latte”, “prendere il pane”, … Come è possibile notare, si tratta di attività che possono essere effettuate in qualsiasi posizione geografica che consenta di portarlo a termine, non necessariamente in una posizione specifica. Potrebbero essere quindi visti come una generalizzazione del concetto di Evento. Hanno come parametri il titolo del Task, che lo identifica, una deadline, data e ora entro la quale portare a termine il Task, un orario di notifica (orario nel quale il Task viene notificato, al di fuori del quale non viene visualizzato), una priorità impostabile da 1 a 5 e una descrizione.

2.4.1.3 . Mappa

Questa visualizzazione è quella fornita da Google Maps attraverso uno specifico componente sviluppato per Android.

Nel sistema di base, qualora l'utente avesse ricevuto nuove notifiche utili al soddisfacimento delle proprie richieste, queste sarebbero state geolocalizzate con un puntatore sulla mappa, utilizzando dei markers a forma di casa.

2.4.1.4 . Preferences

Il sistema permette di impostare dei particolari parametri indicati come preferenze. In Android questo obiettivo si raggiunge attraverso un componente messo a disposizione dal sistema chiamato “SharedPreferences”, nel quale è possibile salvare coppie chiave valore attraverso le quali l'utente ha la possibilità di impostare dei parametri che cambiano runtime il funzionamento dell'applicazione.

Le variabili messe a disposizione dell'utente sono:

- *Max research distance*: ad ogni richiesta del client, il server riceve l'identificativo dell'utente e, a partire da questo, verifica se vi siano task per i quali si possa fornire una notifica di un qualche punto di interesse. Per poter stabilire l'ampiezza del raggio di ricerca, e quindi i task notificati, l'utente ha a disposizione tale parametro. Se il parametro non viene impostato dall'utente, la richiesta al server viene effettuata specificando un parametro di default, esplicitato nel file preferences.xml e dal codice Java quando richiamato;
- *Query period*: questo parametro specifica l'intervallo di tempo che trascorre tra due richieste di download Hints. Tale approccio è stato immediatamente considerato sbagliato ed in seguito sostituito con un parametro spaziale. Il download degli Hints non avviene pertanto cadenzato nel tempo, ma in seguito allo spostamento del terminale di una determinata distanza (impostabile dall'utente).

2.4.2 . Lato server

2.4.2.1 . Gestione richieste

Il server è composto principalmente da tre livelli che cooperano tra di loro per soddisfare le richieste effettuate dal client. I tre livelli sono:

- Livello Web: si occupa di fornire una interfaccia delle risorse verso il web, quindi verso la rete;
- Livello Management: si occupa di elaborare le informazioni che provengono dal client, creare una interfaccia con il livello sottostante (database) e creare gli oggetti che andranno poi restituiti al client in soddisfacimento della richiesta;
- Livello Database: il livello più basso del sistema che si occupa di rendere persistenti i dati e di offrire le opportune interrogazioni della base di dati che servono al funzionamento. Non è necessario che le interfacce che definiscono i metodi di accesso a database e il database vero e proprio, siano residenti fisicamente sulla stessa macchina.

Per sua stessa natura, tale sistema rispecchia l'architettura Multi-Tier (in particolare l'architettura Three-Tier[5]) poiché esiste una differenziazione dei tre livelli tipici di applicazioni che si basano su tale design-pattern. Nei sistemi basati su Web Service questo tipo di architettura viene intesa con una accezione più specifica, in particolare:

1) l'interfaccia è rappresentata dai servizi forniti dal Web Service;
2) la business logic corrisponde ad una serie di moduli integrati in una application server (ad esempio moduli Java su Tomcat);
3) i dati (ai quali viene garantito l'accesso alla business logic) sono memorizzati in un database relazionale.

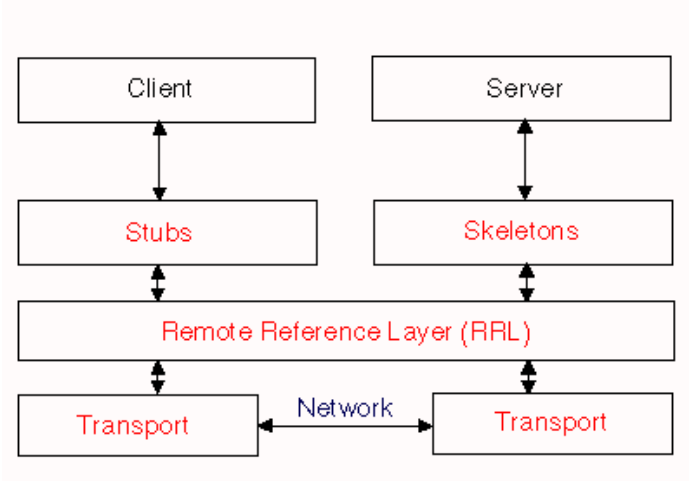


Figura 2.6: Remote Method Invocation

5 Architettura Three-Tier: in ingegneria del software viene indicata una particolare architettura software suddivisa in tre moduli dedicati rispettivamente all'interfaccia utente, alla businness logic e alla gestione della persistenza dei dati (database).

L'architettura realizzata è di tipo client/server e non si discosta molto dal paradigma RMI (Remote Method Invocation) la cui struttura è esemplificata in Figura 2.6. Di fatto il client richiede al server di elaborare dei dati che gli vengono spediti via rete (nella fattispecie viene utilizzata la rete mobile del cellulare) e restituiti dopo l'elaborazione effettuata dal server.

Richieste Task

Le richieste da parte del client per ottenere le informazioni necessarie ai Task, avvengono all'url http://indirizzoserver:8080/{username}/task. Tali richieste sono contenute nella classe `TaskResource` del package `web` del server. Le varie risorse sono accessibili in alcune sottocartelle, in particolare:

- `/add`: aggiunge un task al database. Necessita in input di un file XML con i parametri del task da inserire e dei parametri presi dall'url. Non restituisce nulla al client;
- `/all`: effettua il reperimento di tutti i task presenti per un dato utente. Necessita in input dei parametri presi dall'url. Restituisce un file XML al client con l'elenco dei task presenti;
- `/first`: effettua il reperimento dei primi 5 task (in ordine di priorità) presenti per un dato utente. Necessita in input dei parametri dall'url. Restituisce un file XML come sopra;
- `/update`: effettua la modifica dei parametri di un Task. Necessita in input di un file XML con i parametri da modificare del Task e dei parametri dall'url. Non restituisce nulla al client;
- `/erase`: cancella un task dal sistema, in questo modo non è più reperibile dall'utente. Necessita in input di un file XML con i dati identificativi del Task e dei parametri dall'url. Non restituisce nulla al client.

In Appendice A sono presenti alcuni diagrammi di sequenza che descrivono il percorso delle informazioni tra le varie classi del server.

Richieste Eventi

Le richieste da parte del client per ottenere le informazioni necessarie agli Eventi, avvengono all'url http://indirizzoserver:8080/{username}/event. Tali richieste sono contenute nella classe `EventResource` del package `web` del server. Le varie risorse sono accessibili in alcune sottocartelle, in particolare:

- `/add`;
- `/all`;
- `/today`;
- `/between`;
- `/update`;
- `/erase`.

Non viene esplicitato il funzionamento di ogni risorsa in quanto molto simili al caso

precedente relativo ai Task.

Richieste Input

Tale sezione è responsabile dell'inserimento dei Task o Eventi nel sistema, derivanti dalle richieste del client in seguito all'inserimento del Task o Evento in linguaggio naturale oppure tramite l'opportuna interfaccia vocale. L'unica risorsa è contenuta nella classe `InputResource` del package `web` del server ed è reperibile all'url http://indirizzoserver:8080/{username}/input. Tale risorsa si preoccupa di interpretare la stringa inserita in linguaggio naturale tramite un parser e di inserire il task o evento richiesto.

Richieste Login

All'avvio dell'applicazione nel terminale vengono richieste le credenziali di accesso. Lato server, la risorsa che si occupa di effettuare la procedura di autentificazione dell'utente è reperibile all'url http://indirizzoserver:8080/{username}/login. Tale risorsa è contenuta nella classe `LoginResource` del package `web` del server e si preoccupa di controllare l'esistenza dell'utente nel database e di restituire al client un oggetto di tipo `LoginReply` che comunica l'avvenuta autenticazione o l'eventuale codice di errore in modo tale che l'errore possa essere notificato all'utente.

Richieste Context

La sezione reperibile all'url http://indirizzoserver:8080/{username}/location mette a disposizione due risorse disponibili rispettivamente in `/all` e `/single` che provvedono alla ricerca nel servizio utilizzato (per ora solo Google Maps) di tutti i punti dove è possibile portare a termine uno (single) oppure tutti (all) i task memorizzati da un utente. Tale risorsa utilizza l'ontologia e la rete semantica, per inferire dalle parole chiave inserite dall'utente una serie di parole chiave correlate ed effettuare la ricerca, e le Google Maps API, per la ricerca nel database di Google delle possibili attività che possono soddisfare un task.

2.4.2.2 . Parser

La possibilità di inserimento di Task ed Eventi utilizzando il linguaggio naturale (sia attraverso l'inserimento di una stringa manualmente che attraverso il riconoscimento vocale), necessita della presenza di un parser testuale che permetta di riconoscere i vari token della frase inserita e decidere, dopo una opportuna analisi sintattica della frase, l'operazione da intraprendere (può essere ad esempio l'inserimento di un Task oppure l'inserimento di un Evento). Il parser si frappone tra l'utente ed il sistema lato server per fare da “middleware” ed interpretare il linguaggio naturale in linguaggio macchina, nella fattispecie la creazione della query che inserisce i dati provenienti dall'utente nella Base di Dati.

Il funzionamento del parser implementato ricalca sotto qualche aspetto il funzionamento del parser implementato dal Mozilla Labs per Ubiquity[6], l'add-ons per il famoso browser web Mozilla Firefox.

Esiste però una differenza tra il parser implementato per Ubiquity ed il parser implementato nel presente sistema. Il parser di Ubiquity tenta di anticipare l'azione dell'utente analizzando i caratteri inseriti al fine di capire cosa si vuole fare. Ad esempio se l'utente inserisce:

to Jono

Ubiquity interpreta i caratteri inseriti e propone come possibili azioni:

email to jdicarlo@mozilla.com

twitter to jono

translate to jono

Il parser implementato nel sistema, invece, prende in input una stringa completa in linguaggio naturale e da questa tenta di estrarre delle parole chiave che permettano di capire qual è l'azione da intraprendere nel sistema e quali sono gli oggetti a cui si riferisce.

Il parser di Ubiquity basa il suo funzionamento processando il testo in input seguendo in ordine una serie di passi:

1) Separare le parole/argomenti;
2) Scegliere i possibili verbi;
3) Scegliere i possibili clitici;
4) Raggruppare in argomenti;
5) Sostituire le anafore;
6) Suggerire degli argomenti normalizzati;
7) Suggerire dei verbi;
8) Definire la tipologia di sostantivi;
9) Sostituire gli argomenti con il loro sostantivo;
10) Effettuare un ranking.

Senza dilungarsi troppo nell'analisi approfondita[7] del parser di Ubiquity, verranno ora analizzati i passi che si rendono necessari al funzionamento del parser per il presente sistema. In particolare:

- Riconoscimento dei verbi;
- Riconoscimento degli argomenti.

Il primo passo, riconoscimento dei verbi, permette di scegliere l'azione da

6 Ubiquity: add-ons per Mozilla Firefox. E' una collezione di semplici comandi derivati dal linguaggio naturale che permetto all'utente di ottenere informazioni da utilizzare nella pagina web corrente o di indirizzarlo ad altre pagine web o servizi.

7 Si rimanda alla pagina web dove viene descritto il funzionamento del parser di Ubiquity: https://wiki.mozilla.org/Labs/Ubiquity/Parser_2

intraprendere nel sistema (di seguito indicata con comando); il secondo passo permette di trovare quali siano gli oggetti (argomenti) ai quali l'azione è riferita. Ad esempio se l'utente inserisce la seguente frase:

ricordami di prendere il latte prima delle 20

nel primo passo il sistema deve riconoscere il verbo *ricordami* e legarlo al comando *"aggiungi task"*, mentre nel secondo passo il sistema deve riconoscere *"prendere il latte"* e *"20"* che saranno rispettivamente l'oggetto e il limite temporale del Task.

Analizzando più approfonditamente il funzionamento del parser, si nota che il l'individuazione del comando da eseguire è identificato dal verbo che viene riconosciuto. In particolare per il comando *"aggiungi Task"* vengono cercati i seguenti verbi:

- I have;
- remind me;
- add task.

mentre per il comando *"aggiungi Evento"* vengono cercati i seguenti verbi:

- I have an appointment;
- add appointment;
- add event.

Data una stringa di input, il parser esegue le seguenti operazioni:

1) Per ogni tipologia di comando, tenta di riconoscere il verbo che più si avvicina a soddisfarlo, utilizzando una serie di espressioni regolari che vengono confrontate con la stringa in ingresso;
2) Viene assegnato un certo punteggio al riconoscimento di una espressione regolare (che corrisponde al riconoscimento di un possibile verbo che identifica un comando) che serve alla successiva scelta del verbo che più si avvicina al corretto matching con il comando;
3) Viene scelto come candidato il verbo con punteggio più alto (calcolato precedentemente);
4) Per ognuno dei verbi trovati, si cerca di estrarre gli argomenti che si riferiscono al comando relativo. I possibili argomenti che si riesce ad estrarre sono di tipo OBJECT (oggetto al quale si riferisce il comando) e TIME (riferimento temporale per il comando);
5) Si aggiusta il punteggio del comando aggiungendo un punto al punteggio calcolato precedentemente se dalla stringa sono stati estratti entrambi gli argomenti sopra indicati;
6) Viene infine scelto il comando che ha punteggio più alto. Il punteggio più alto viene calcolato in base al matching dell'espressione regolare con il verbo al punto 2) e al numero di argomenti estratti dalla stringa al punto 4). Ad esempio, se in una stringa si ha un matching perfetto con una espressione regolare che mi identifica il verbo di un comando e vengono estratti dalla stringa entrambi gli argomenti, allora il comando associato avrà un punteggio decisamente alto e sarà un ottimo candidato ad essere scelto.

Il parser presente riconosce solo stringhe inserite in lingua inglese ed utilizza un file di configurazione (`en.lang`) che contiene una serie di delimitatori (le entità con le quali il parser cerca di distinguere i vari argomenti) per ogni argomento. In fase di estrazione degli argomenti, il parser controlla ed identifica la presenza di un delimitatore. Se presente, significa che la porzione di stringa successiva si riferisce all'argomento al quale il delimitatore è collegato.

2.4.2.3 . Reasoner - Ontologie

Esiste una distinzione tra i concetti di conoscenza concettuale, nomologica e fattuale:

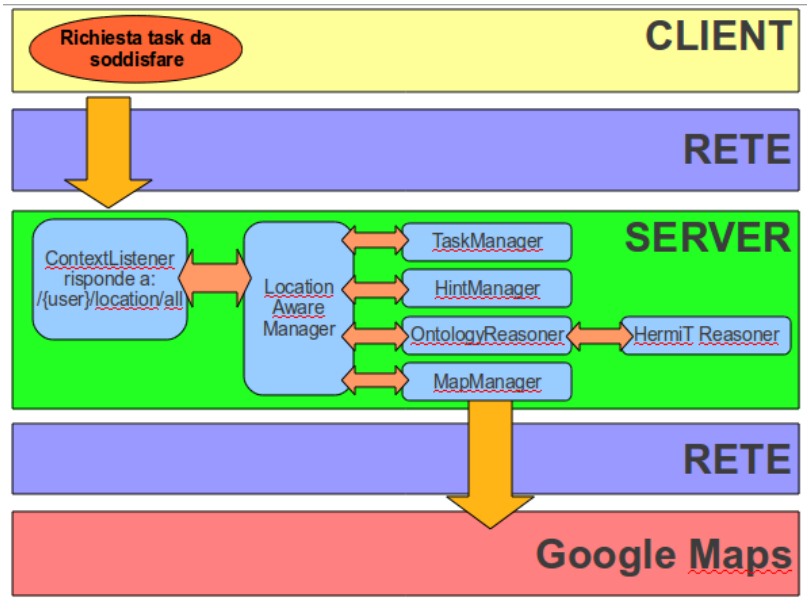
- conoscenza concettuale: è la conoscenza dei concetti che si utilizzano per interpretare la realtà (es. una madre è donna con almeno un figlio);
- conoscenze nomologiche: è la conoscenza di regolarità o leggi generali che regolano il mondo (es. una madre ama i propri figli);
- conoscenze fattuali: si riferiscono a particolari fatti esistenti (es. Alice è la madre di Bob).

Per attuare delle, anche semplici, deduzioni, in campo informatico si possono utilizzare degli strumenti quali le ontologie. Nel significato più comune, ontologia corrisponde alle conoscenze concettuali e nomologiche, mentre non comprende le conoscenze fattuali. Ad esempio, possono essere espressi i concetti di relazione fra i concetti di “stato” e “città” e la relazione “capitale di”, ma non il fatto che “Roma” sia “la capitale dell'Italia”. In parole povere, una ontologia descrive uno schema, di tipo concettuale, che permette di descrivere un mondo, ma non comprende le situazioni fattuali. Inoltre, fuori da un determinato contesto, le conoscenze possono essere piuttosto generali e risulta pertanto difficile trovare una rigorosa descrizione per tali tipologie di conoscenze. Le ontologie dovranno pertanto escludere questo tipo di conoscenze.

Nel presente progetto, il concetto di ontologia è stato utilizzato per poter ampliare il parco di conoscenza derivante direttamente dai task inseriti e pertanto poter inferire concetti più complessi. Ad esempio, se l'utente dovesse richiedere al sistema di ricordargli di prendere il pane, sarebbe riduttivo utilizzare solo la parola chiave “pane” per la ricerca di attività commerciali dove possa soddisfare questa esigenza. Tramite l'utilizzo delle ontologie è possibile ampliare il parco di richieste (verso un database commerciale come Google Maps) riuscendo a ottenere delle parole chiave strettamente collegate a quanto richiesto dall'utente. È così possibile inferire dalla parola “pane” anche le parole chiave “panetteria, panificio, supermercato, supermarket, alimentari” e poter estendere la ricerca del pane anche alle entità così identificate. Come indicato sopra, generalmente una ontologia non è in grado di comprendere le conoscenze fattuali. Nel ristrettissimo mondo in analisi, è possibile raggirare questo ostacolo definendo ad esempio delle relazioni dirette che derivano da conoscenze che si possono estendere a tutti, come ad esempio il fatto che il “pane” si può acquistare anche presso “Coop, Despar, Auchan”.

Il ragionamento che sta alla base dell'inferenza fa ampio utilizzo delle DL (Logiche

Descrittive), ossia una famiglia di formalismi utilizzati per rappresentare la conoscenza in un dominio di applicazione detto mondo. In particolare sono dei linguaggi di rappresentazione adatti alla definizione di ontologie. Un particolare tipo di DL è identificato dal linguaggio OWL. Il linguaggio OWL[8] (Web Ontology Language) è lo standard raccomandato nel 2004 dal W3C[9] per la definizione di ontologie per il web semantico[10]. Il linguaggio OWL 2[11] è una estensione di OWL (a volte chiamato OWL 1. E' raccomandato dal W3C a partire dal 2009.



Nel progetto in esame viene utilizzata la rappresentazione XML del linguaggio OWL, attraverso la quale vengono definite le relazioni che stanno alla base dell'estrapolazione delle parole chiave, con le quali verranno effettuate poi le query al servizio esterno. In particolare, il diagramma a blocchi che definisce l'interazione con il client ed il server (e quindi l'ontologia ed il servizio Google Maps) è definito come in Figura 2.7.

Un aspetto che distingue le basi di conoscenze dalle basi di dati è la possibilità di condurre ragionamenti in modo automatico. Nel contesto della logica quando si parla di "ragionamento" ci si riferisce a ragionamenti di tipo deduttivo (o più semplicemente deduzioni). Un ragionamento è dunque un procedimento che porta a verificare se un enunciato o asserzione X (ad esempio una relazione di sottoclasse) è conseguenza logica di una base di conoscenze.

Nel presente progetto è stato utilizzato il reasoner HermiT[12] per rispondere alle

8 OWL: http://www.w3.org/TR/owl-ref/
9 W3C: World Wide Web Consortium. Http://www.w3c.org/
10 http://www.w3c.org/2001/sw
11 http://www.w3.org/TR/owl2-direct-semantics/
12 HermiT reasoner: http://hermit-reasoner.com/

interrogazioni. La definizione delle relazioni logiche avviene tramite il file `HintsOntology.owl` contenunto nel package `resource` del progetto. Le librerie necessarie per il reasoner HermiT sono scaricate all'occorrenza da Maven e sono incluse direttamente nel pacchetto war del Web Service per il deploy sul servlet container (Apache Tomcat).

2.4.2.4 . Ricerca Google Maps

Una volta estratte le parole chiave per le varie richieste, il server, in seguito alla richiesta di Hints da parte del Client (servita dalla classe `ContextListener` del package `web`), effettua una query attravero l'utilizzo delle Google Maps API. Il risultato della query al servizio Maps di Google, viene restituito in formato JSON. Il server si preoccupa di riconoscere le varie parti del file JSON e di restituire il risultato al client in formato XML.

La richiesta al servizio Google maps avviene al seguente URL:

http://ajax.googleapis.com/ajax/services/search/local?sll=45.666096%2C12.22924&mrt=localonly&key=ABQIAAAAWuzz0Iu2TplTMtzxmJOuUBQusdbP4Fg6ujkVg6Gs4Mes9VMUQBRE-2eyibREfaskHZRxvoQb-5Hluw&q=panetteria&v=1.0

nel quale vengono specificati il servizio richiesto, le coordinate geografiche alle quali effettuare la ricerca, la chiave Google API Key (richiesta dalle specifiche Google API per identificare il richiedente della risorsa), la parola chiave con la quale effettuare la richiesta e la versione delle API da utilizzare.

La risposta (formato JSON) è del tipo:


```
{"responseData":
        {"results"
                [{"GsearchResultClass":"GlocalSearch",
                        "viewportmode":"computed",
                        "listingType":"local",
                        "lat":"45.670359","lng":"12.235108",
                        "accuracy":"8",
                        "title":"Fradis Sas Di Bernardi A. \u0026amp; C.",
                        "titleNoFormatting":"Fradis Sas Di Bernardi A. \u0026 C.",
                        "ddUrl":"http://www.google.com/maps?
source\u003duds\u0026daddr\u003dViale+Monfenera,+11,+Treviso,+Veneto+
(Fradis+Sas+Di+Bernardi+A.+%26+C.)
+@45.670359,12.235108\u0026saddr\u003d45.666096,12.22924",
                        "ddUrlToHere":"http://www.google.com/maps?
source\u003duds\u0026daddr\u003dViale+Monfenera,+11,+Treviso,+Veneto+
(Fradis+Sas+Di+Bernardi+A.+%26+C.)+@45.670359,12.235108\u0026iwstate1\u003ddir:to",
                        "ddUrlFromHere":"http://www.google.com/maps?
source\u003duds\u0026saddr\u003dViale+Monfenera,+11,+Treviso,+Veneto+
(Fradis+Sas+Di+Bernardi+A.+%26+C.)+@45.670359,12.235108\u0026iwstate1\u003ddir:from",
                        "streetAddress":"Viale Monfenera, 11",
                        "city":"Treviso",
                        "region":"Veneto",
```

```
                        "country":"Italy",
                        "staticMapUrl":"http://maps.google.com/maps/api/staticmap?
maptype\u003droadmap\u0026format\u003dgif\u0026sensor\u003dfalse\u0026size\u003d150x100\u0
026zoom\u003d13\u0026markers\u003d45.670359,12.235108",
                        "url":"http://www.google.com/maps/place?
source\u003duds\u0026q\u003dpanetteria\u0026cid\u003d12804520337120632811",
                        "content":"",
                        "maxAge":604800,
                        "phoneNumbers":[{"type":"","number":"0422 22751"}],
                        "addressLines":["Viale Monfenera, 11","31100 Treviso, Italia"]
                },

                {"GsearchResultClass":"GlocalSearch",
                        "viewportmode":"computed",
                        "listingType":"local",
                        "lat":"45.666188",
                        "lng":"12.241609",
                        "accuracy":"8",
                        "title":"Panificio Di Fontan Dino",
                        "titleNoFormatting":"Panificio Di Fontan Dino",
                        "ddUrl":"http://www.google.com/maps?
source\u003duds\u0026daddr\u003dVia+Jacopo+Riccati,+31,+Treviso,+Veneto+
(Panificio+Di+Fontan+Dino)+@45.666188,12.241609\u0026saddr\u003d45.666096,12.22924",
                        "ddUrlToHere":"http://www.google.com/maps?
source\u003duds\u0026daddr\u003dVia+Jacopo+Riccati,+31,+Treviso,+Veneto+
(Panificio+Di+Fontan+Dino)+@45.666188,12.241609\u0026iwstate1\u003ddir:to",
                        "ddUrlFromHere":"http://www.google.com/maps?
source\u003duds\u0026saddr\u003dVia+Jacopo+Riccati,+31,+Treviso,+Veneto+
(Panificio+Di+Fontan+Dino)+@45.666188,12.241609\u0026iwstate1\u003ddir:from",
                        "streetAddress":"Via Jacopo Riccati, 31",
                        "city":"Treviso",
                        "region":"Veneto",
                        "country":"Italy",
                        "staticMapUrl":"http://maps.google.com/maps/api/staticmap?
maptype\u003droadmap\u0026format\u003dgif\u0026sensor\u003dfalse\u0026size\u003d150x100\u0
026zoom\u003d13\u0026markers\u003d45.666188,12.241609",
                        "url":"http://www.google.com/maps/place?
source\u003duds\u0026q\u003dpanetteria\u0026cid\u003d11339606163680129896",
                        "content":"",
                        "maxAge":604800,
                        "phoneNumbers":[{"type":"","number":"0422 542331"}],
                        "addressLines":["Via Jacopo Riccati, 31","31100 Treviso, Italia"]
                },
...
...
                ],
...
...
                "resultAttribution":"Schede attività commerciali fornite da \u003ca
href\u003d\"http://www.paginegialle.it/\"\u003ePagineGialle.it\u003c/a\u003e"},
        "responseDetails": null,
        "responseStatus": 200
}
```


Tale richiesta avviene per ognuna delle parole chiave inferite dall'ontologia.

Capitolo 3 . Nuove proposte

3.1 . Realizzazione di un ambiente server

3.1.1 . Virtualizzazione

La possibilità di avere a disposizione un server funzionante con il quale testare quanto realizzato è estremamente importante in un progetto come questo. Si è deciso pertanto di adibire delle macchine a tale scopo, installando un sistema operativo e tutti i componenti necessari all'utilizzo dell'applicazione.

Ciò che ha spinto alla scelta di virtualizzare il server, è stato principalmente la possibilità di distribuire l'applicazione lato server come pacchetto completo e preconfigurato da utilizzare con un software di virtualizzazione. L'installazione del server si riconduce ad una operazione estremamente semplice e alla portata di tutti, anche senza particolari conoscenze informatiche.

Si è scelto di utilizzare il software di virtualizzazione VMWare server versione 2.0.2. Una volta installato il server sulla macchina virtuale, è possibile trasportarlo da una piattaforma all'altra o da una macchina fisica (con installato VMWare) ad un'altra semplicemente spostando una cartella che contiene tutte le impostazioni ed i file necessari al suo funzionamento.

Sono state predisposte per i test due macchine virtuali installate rispettivamente a Padova e a Genova. Sono state equipaggiate con:

- Sistema operativo Linux Ubuntu
- Servlet container Apache Tomcat 7
- Database MySql

Nel server padovano è stato inoltre installato e configurato il server web Apache, il

motore PHP, PHPMyAdmin (uno script PHP per l'interazione con MySql), Postfix (un MTA, Mail Transfer Agent) per l'invio di posta elettronica, Mantis Bug Tracker (uno scritp in PHP che permette di tener traccia e stato dei bug che si trovano nello sviluppo).

I due server sono disponibili agli indirizzi:

- http://serverpd.dyndns.org:8080/
- http://serverge.dyndns.org:8080/ oppure http://zelda.openlab-dist.org:8080/

3.1.2 . Tomcat e sistema di logging

Apache Tomcat (o semplicemente Tomcat) è un web container open source sviluppato dalla Apache Software Foundation. Implementa le specifiche JavaServer Pages (JSP) e Servlet di Sun Microsystems, fornendo quindi una piattaforma per l'esecuzione di applicazioni Web sviluppate nel linguaggio Java. La sua distribuzione standard include anche le funzionalità di web server tradizionale, che corrispondono al prodotto Apache.

Durante il ciclo di sviluppo ed il successivo funzionamento, ogni sistema produce una serie di messaggi di output che possono essere utilizzati per poter comprendere meglio cosa stia accadendo, o per verificare se vi siano situazioni critiche che richiedono l'intervento del personale di amministrazione: il log.

Il log più semplice, dalle origini ad oggi, è un file sequenziale sempre aperto in scrittura, che viene chiuso e conservato a cadenze regolari e reso disponibile per:

- analisi delle segnalazioni di errore;
- produzione di statistiche di esercizio, come ad esempio quelle del traffico nei servizi web;
- ripristino di situazioni precedenti;
- analisi delle modifiche apportate nella base dati;
- analisi delle operazioni eseguite e dei responsabili di tali operazioni;
- riassunto di quanto successo in un determinato arco di tempo di esecuzione dell'applicazione.

Anziché utilizzare un file di test, il log può anche essere un segmento di database con accesso diretto mediante chiave cronologica (timestamp).

Tomcat permette di utilizzare il sistema di logging nativo oppure un qualsiasi altro sistema, importando le librerie e/o i componenti necessari al funzionamento. Si è scelto di utilizzare un sistema alternativo di logging, il Log4J, che intercetta i log prodotti da Tomcat e da tutti i sistemi inseriti e che crea i file. Tale sistema di logging è stato creato principalmente per essere utilizzato con Java e permette di distinguere i vari messaggi di log identificandone un livello di severità. In fase di programmazione si può scegliere pertanto quale livello di log attribuire ad un determinato messaggio. Tale struttura permette di identificare facilmente le tipologie di messaggi,

eventualmente filtrandole e permettendo allo sviluppatore di capire immediatamente cosa succede nel sistema. In Tabella 3.1 sono riportati i livelli (in ordine decrescente di severità) dei log.

Livello	Descrizione
FATAL	Errore importante che causa un prematuro termine dell'applicazione in esecuzione. È auspicabile che tale messaggio sia visibile immediatamente all'operatore.
ERROR	Errore di esecuzione o condizione imprevista. Anche questa tipologia di informazione deve essere immediatamente segnalata.
WARN	Utilizzato per ogni condizione imprevista o anomalia di esecuzione che però non comporta necessariamente un errore.
INFO	Segnalazione di eventi di esecuzione (ad esempio startup/shutdown). Dev'essere segnalato ma non necessariamente mantenuto a lungo.
DEBUG	Utilizzato nelle fasi di debug del programma, permette di tracciare il funzionamento del sorgente.
TRACE	Alcune informazioni dettagliate. Dev'essere scritto esclusivamente nei file di log. Introdotto dalla versione 1.2.12 di Log4J.

Tabella 3.1: Livelli di log definiti per Log4J. Ordine decrescente di severità

Utilizzando il file `log4j.properties` contenuto nella cartella `lib` di tomcat, è possibile definire la tipologia di file prodotti e lo standard adottato per i nomi dei file di log. Di default tomcat scrive tutti i messaggi di log in un unico file chiamato `tomcat.log`, contenuto nella cartella `logs`. Tramite il file di configurazione indicato, sono stati suddivisi i file di log per data, utilizzando `tomcat.log.AAAA-MM-GG` come formato per il nome del file di log. Al cambiare della data nel server, il file `tomcat.log` viene rinominato come indicato e viene creato un nuovo `tomcat.log`. In questo modo si evitano file di log di dimensioni troppo grosse per essere gestiti proficuamente.

3.1.3 . Strumenti di aiuto allo sviluppo

3.1.3.1 . Mantis Bug Tracker

Mantis Bug Tracker è un software lato server scritto in PHP che si può appoggiare a diversi database: MySQL, MSSQL, DB2, Oracle e PostgreSQL. Mantis Bug Tracker è rilasciato sotto la licenza GNU GPL. Viene utilizzato per tenere traccia di tutti i bug

che si verificano durante lo sviluppo di un'applicazione.

Utilizzando tale prodotto, è stato messo a disposizione del gruppo di sviluppo un sistema per tenere costantemente sotto controllo l'andamento del progetto. Utilizzando Mantis Bug Tracker è possibile inoltre assegnare la correzione di un bug ad un particolare componente, il quale se ne prende carico e porta a termine il lavoro notificando in Mantis eventuali messaggi o cambiamenti di stato del bug in esame. Essendo il gruppo di sviluppo dislocato in posizioni geografiche diverse e non vicine, tale sistema si è dimostrato estremamente utile per il coordinamento del lavoro.

Le funzionalità supportate da Mantis Bug Tracker in particolare sono:

- Creazione automatica di Changelog[13];
- Visualizzazione di report personalizzati per ogni utente;
- Completa personalizzazione di ogni opzione del bug: possibilità di aggiungere nuovi campi ed eliminare i predefiniti;
- Sistema di ricerca con filtri;
- Creazione di sottoprogetti o categorie di progetti;
- Esportazione in formato csv, MS Word, MS Excel;
- Possibilità di integrazione con una Wiki.

I requisiti per l'installazione, prevedono la presenza di un WebServer, del motore PHP e di un database. È stato pertanto installato nel server padovano ed utilizza Postfix come MTA per l'invio di posta elettronica al fine di notificare ai componenti del gruppo le varie azioni.

3.1.3.2 . Script realizzati

In fase di sviluppo, sono stati creati degli script ad hoc per poter agevolare i rilasci e le operazioni di programmazione. In particolare sono stati creati due script in PHP per:

- pubblicazione online delle varie versioni client (formato .apk), rilasciate per avere sempre a disposizione una versione in un repository condiviso da installare nel terminale. Tale script è raggiungibile all'indirizzo: http://serverpd.dyndns.org/client/ e permette di visualizzare l'elenco dei vari apk memorizzati, effettuare l'upload di una nuova release e scaricare direttamente l'ultima versione disponibile;
- creazione di file KML[14] utilizzabili con DDMS per simulare percorsi con l'emulatore. DDMS ha la possibilità, per testare l'applicazione Android sull'emulatore, di inviare coordinate geografiche GPS come se fosse il LocationProvider reale del terminale che le invia. La possibilità di creare percorsi simulati con Google Earth ha portato alla creazione di questo script

13 Changelog: di solito memorizzato su un file di testo, raccoglie, in ordine cronologico, tutti i possibili riferimenti di rintracciabilità di ogni modifica effettuata ad un progetto fino alla release attuale.

14 KML (Keyhole Markup Language) è un linguaggio basato su XML creato per gestire dati geospaziali in tre dimensioni.

che converte il file KML creato da Google Earth in un file KML utilizzabile con DDMS e l'emulatore Android. Si è notato però che esiste qualche problema nel componente DDMS in quanto non permette di identificare correttamente (e quindi inviare) le coordinate geografiche da file KML, pertanto questa tipologia di testing è stata abbandonata. Si è preferito effettuare i test che necessitano di posizione GPS direttamente con dei terminali Android.

3.2 . Bug Fixing e nuove funzionalità

Uno dei punti cardine di tutto il lavoro svolto, è quello di aver portato una applicazione ancora in fase pre-prototipo (una fase in cui sono state identificate le caratteristiche e definite le tecniche implementative, ma non ancora stabile da garantirne il funzionamento su un terminale reale) ad uno stato prototipale avanzato che ne garantisce il funzionamento senza troppi problemi di stabilità.

Tramite l'utilizzo del servizio offerto da Mantis Bug Tracker[15], lo sviluppo di nuove funzionalità supportate dal sistema è stato intervallato dalle correzioni di tutti i bug che si presentavano mentre si effettuavano dei test. Tali correzioni si riferiscono ad esempio a problemi derivanti da un non corretto utilizzo dei provider di ricerca della posizione (NETWORK e GPS), ad una non corretta gestione dei thread che effettuano le elaborazioni in background e a problemi di comunicazione con il server.

3.2.1 . Organizzazione del lavoro di gruppo

Il presente progetto è stato sviluppato (e seguito) da più persone, alcune afferenti al Dipartimento di Ingegneria dell'Informazione dell'Università degli Studi di Padova ed altri afferenti al Dipartimento di Informatica, Sistemistica e Telematica dell'Università degli Studi di Genova. La particolare dislocazione geografica dei vari componenti del gruppo, ha reso necessario l'utilizzo di sistemi di comunicazione a distanza. Sono stati scelti, in particolare, la creazione di un gruppo su Google Groups, con il quale tutti i componenti comunicavano, e l'utilizzo del software di VoIP Skype, per alternare le comunicazioni scritte, più formali e orientate alla creazione di una documentazione di quanto discusso e creato, a meeting di brainstorming per decidere metodologie da utilizzare e aspetti da considerare nella soluzione dei problemi che via via affioravano e nella creazione di nuove funzionalità a supporto del sistema.

Gli aspetti trattati nelle riunioni periodiche hanno sollevato la necessità di sviluppare nuovi componenti del sistema, quali ad esempio:

- una gestione social del sistema, includendo delle segnalazioni di punti di interesse personali, non presenti nei servizi esterni (ad esempio in Google Maps) fruibili da tutti gli utenti registrati al sistema; in questo modo è possibile avere a disposizione un parco più ampio di soluzioni da notificare senza dover sottostare a quanto scelto e presente in un servizio non governato

15 Confrontare § 3.1.3.1 .

dal sistema;
- la necessità di ampliare il numero di servizi esterni dai quali reperire gli hints (aggiungendo ad esempio PagineGialle, PagineBianche, 2spaghi, …);
- la necessità di mettere in comunicazione gli utenti registrati aggiungendo dei gruppi di utenti che perseguono uno stesso fine, che hanno pertanto dei task in comune;
- la necessità di ampliare la portata del reasoner, includendo sempre più collegamenti semantici a parole chiave, eventualmente estendendo anche agli utenti la possibilità di popolare l'ontologia;
- la necessità di creare un sistema per la registrazione degli utenti, in modo poco impegnativo per l'utente stesso, che possa raccogliere alcuni dati anagrafici della persona che vuole utilizzare il sistema;
- la necessità di rendere il sistema il più reattivo possibile includendo dei sistemi di notifica che richiamino l'attenzione degli utenti quando vengono notificati gli hints;
- la necessità di creare un protocollo di comunicazione che preveda l'insorgere di errori tra il client e il server;
- la necessità di non apportare ulteriore stress nella vita quotidiana degli utenti, cercando di stabilire un buon compromesso tra le notifiche e gli impegni dell'utente, quindi senza notificare troppo e troppo insistentemente il ritrovamento dei punti di interesse;
- la necessità di sopperire al mal funzionamento del ricevitore GPS in situazioni di scarsa visibilità del cielo (come ad esempio sotto i portici in città) utilizzando il posizionamento dato dalle antenne GSM;
- la necessità di far interagire l'utente con il sistema in modo da personalizzarlo il più possibile in base alle proprie esigenze e gusti.

Molti dei punti sopra indicati sono stati sviscerati e trattati nel progetto. I dettagli funzionali ed implementativi sono rimandati al capitolo seguente. Altri punti sono stati invece solamente accennati e non analizzati a fondo, a causa della mancanza di tempo e risorse coinvolte nel progetto. Sono stati riportati però degli spunti per dei futuri sviluppi in merito.

3.2.2 . Soluzioni identificate

Rispetto a quanto accennato sopra, sono stati presi in considerazione e ritenuti prioritari alcuni aspetti, affrontati al fine di rendere il sistema stabile per poter effettuare successivamente delle prove a valutazione della bontà del progetto.

Cambio DBMS

Il DBMS utilizzato, db4o, non offre grandi doti di stabilità e prestazioni, quindi si è valutato necessario la sostituzione del database, in particolare utilizzando MySQL, e al contempo la creazione di un metodo per lasciare libera la futura implementazione con altri DBMS.

Gestione di gruppi di utenti

La necessità di poter usufruire di task condivisi fra più utenti è una funzionalità utile per il sistema. Oltre a permettere all'utente di inserire task privati (visibili cioè solo dall'utente che li ha inseriti), abilitare l'inserimento di task da portare a termine in modo concorrente (il primo che lo soddisfa lo toglie dalla lista dei task notificati) in modo tale che sia un gruppo di persone a portare a termine un determinato impegno. Tale funzionalità prevede però un adattamento dell'applicazione esistente per poter inserire le informazioni sui gruppi e la creazione di una nuova sezione dedicata al funzionamento del gruppo, per quanto riguarda creazione dei gruppi e inviti e revoche degli utenti al gruppo. Si è scelto di optare per una sorta di safe social network (la creazione di un gruppo mette in relazione più persone, perciò indirettamente può essere visto come una sorta di social network), facendo in modo che, per quanto possibile, si eviti di abusare di questa funzionalità come succede oggi con le amicizie dei social network come Facebook o Twitter. Il gruppo di utenti è strettamente privato e composto da persone che si conoscono realmente (che pertanto possono avere la necessità di concorrere al soddisfacimento di un determinato impegno comune) e gli inviti sono strettamente personali e possibili solo conoscendo direttamente il nome utente della persona registrata che si vuole invitare a far parte del gruppo. In questo modo viene salvaguardata anche la privacy dei vari utenti.

Registrazione utente

Come ogni buon sistema informatico, anche in questo la sicurezza non viene sottovalutata, pertanto la garanzia di accesso univoco al sistema e la riservatezza dei dati inseriti sono un punto da non sottovalutare. La registrazione utente permette di poter usufruire del sistema scegliendo un proprio nome utente e una password di accesso per l'autenticazione. Questa procedura porta alla creazione di un account per ogni utente e quindi la possibilità di reperire solamente i task che interessano l'utente registrato.

Un futuro spin-off può portare alla distribuzione (eventualmente vendita) del sistema su larga scala. Proponendo tale sistema ad esempio nel Market Android, è possiible ottenere una conoscenza e diffusione dell'applicazione. Sempre più spesso però capita che, anche se una applicazione viene reputata interessante, viene comunque abbandonata a causa di una serie di problemi legati alla registrazione (a volte troppo macchinosa) e alla gestione di un ulteriore nome utente e password per l'ennesimo sistema utilizzato. Una prima modifica (attualmente non implementata nel sistema poiché è stato valutato non portasse grandi migliorie) è quella di permettere l'autenticazione con un qualche sistema SSO, utilizzando ad esempio OpenID per effettuare l'autenticazione tramite le credenziali di un altro servizio al quale l'utente è già registrato. La procedura di registrazione classica automatizzata (necessaria per non dover inserire gli utenti solamente a livello di amministrazione del server) prevede la possibilità di avere fin da subito il proprio account attivo per un numero limitato di login; in questo modo l'utente ha la possibilità di provare il software per un

periodo limitato di tempo (dopo il quale deve necessariamente attivare l'account creato per verificare l'identità) e verificare se risponde alle proprie aspettative. Questo sistema permette inoltre di non far allontanare subito potenziali nuovi utenti solo perché la procedura di registrazione risulta essere troppo complessa o macchinosa.

Notifiche Hints

La sola notifica visuale (un'icona che compare nella barra di stato del cellulare) si è rivelata essere insufficiente al riconoscimento, in una determinata posizione geografica, della possibilità di fare qualcosa. Tale limite è facilmente deducibile dal fatto che tale tipo di notifica è efficace solo se l'utente ha costantemente il cellulare davanti agli occhi, situazione che non si verifica mai, principalmente perché si tende a tenere il telefono in tasca e perché per il risparmio energetico lo schermo si oscura quando il terminale è in standby.

Si è valutato pertanto necessario notificare all'utente il ritrovamento di hints anche con altri metodi di notifica, quali ad esempio una suoneria (stile sms), la vibrazione e un sistema di sintesi vocale che legga all'utente il contenuto del task trovato. Quest'ultima tipologia di notifica, solitamente non presente nelle tradizionali applicazioni per cellulari, segna un primo passo verso una interazione uomo-macchina basata sul linguaggio naturale, in particolare il riconoscimento vocale. Studiando ed implementando un sistema di interazione vocale, sarà possibile, considerati i passi da gigante compiuti in questo ambito dall'ingegneria che hanno portato ad una maggiore precisione nel riconoscimento del parlato, comandare il terminale utilizzando sempre più dei comandi vocali, sentendo sempre meno l'esigenza della tastiera.

Protocollo di comunicazione

La necessità da parte dell'utilizzatore di capire cosa sta facendo il sistema, prevede la notifica del maggior numero di azioni possibile (senza abusi). Tale necessità porta alla definizione di un protocollo di comunicazione tra server e client il più preciso possibile. Ogni transazione portata a termine da parte del sistema prevede la definizione di uno stato nel quale il sistema si trova che permette di distinguere che azione sia stata effettuata e se tale azione sia stata completata correttamente oppure con degli errori. Sapendo precisamente cosa sta accadendo e in che modo, è possibile dare all'utente una serie di informazioni (visualizzate sul display del terminale) al fine di dimostrargli che il sistema sta effettivamente elaborando le informazioni e non è irrimediabilmente piantato.

Per alcune operazioni è inoltre possibile prevenirne gli errori che possono essere identificati a priori, per altre invece no. Ad esempio, dal client è possibile controllare se esiste connessione con il server (se la connessione a internet del cellulare è abilitata) e, in caso negativo, notificarlo direttamente all'utente. Contrariamente, se un utente richiede una registrazione, non è possibile sapere a priori se la registrazione

andrà a buon fine o meno, perché è necessario attendere la risposta del server che controllerà l'esistenza o meno dell'utente.

Per evitare alcuni problemi prevedibili, quali ad esempio la presenza o meno della connessione ad internet del terminale oppure l'incompatibilità delle versioni client e server utilizzate, si è reputato necessario implementare dei metodi che testassero tali situazioni preventivamente e notificassero all'utente il relativo errore prima che l'applicazione potesse terminare in modo imprevisto. Per quanto riguarda tutte le altre interazioni client-server, sono stati definiti una serie di codici di stato che identificassero al client l'esito di una operazione richiesta nel server (tale codice è stato pensato in analogia a quanto succede, ad esempio, per i codici di stato delle richieste HTTP).

Un esempio di tali codici di stato è riportato in Tabella 3.2.

Codici di stato			
0xxx – server; **1xxx** – login; **2xxx** – registration; **3xxx** – groups; **4xxx** – event; **5xxx** – task; **6xxx** – hint.			
0000	Server ok	2000	Registrazion successful
0001	Server ver. non compatible	2001	Registration not confirmed
0002	Server ver. compatible	2002	Registration failed, username already exist
1000	Login successful	2003	Registration failed, email address already exist
1001	Trial login successful		
1002	Login failed, username or password is wrong		
	...		...

Tabella 3.2: Codici di stato

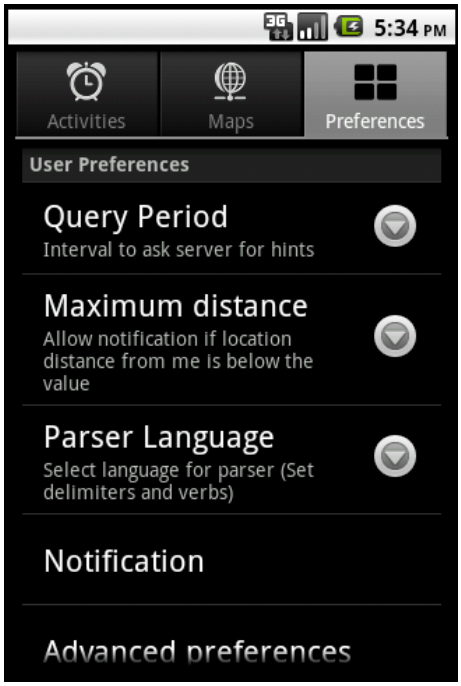
Geolocalizzazione tramite antenne GSM

Il posizionamento geografico avviene in modo estremamente preciso (con una accuratezza che sfiora i metri) per le esigenze del sistema, utilizzando il sistema satellitare GPS. Tale sistema soffre però di un grosso problema, legato alla visibilità del cielo e alle condizioni meteorologiche. Il segnale GPS è tanto peggiore quanto la visibilità del cielo scarseggia (ad esempio sotto i portici nei centri città, oppure in vie molto strette) e quanto peggiori sono le condizioni meteo (con il maltempo diminuisce l'accuratezza del sistema di puntamento fino al punto di non riuscire ad agganciare nemmeno i satelliti e quindi a perdere la geolocalizzazione).

Il terminale utilizzato, oltre ad avere la funzionalità di ricevitore di segnale GPS, è principalmente un telefono cellulare e come tale si aggancia alle antenne GSM per garantire la comunicazione. Le antenne GSM sono geolocalizzate in modo preciso e difficilmente cambiano la loro posizione, pertanto si possono sfruttare le informazioni date dall'antenna in uso per geolocalizzare il terminale. L'accuratezza non è ai livelli

del ricevitore GPS, ma consente comunque di non perdere del tutto la posizione e di far funzionare il sistema. In particolare Android permette un uso estremamente semplice delle informazioni di posizione derivanti da segnale GPS o da antenna GSM. Le informazioni relative alla posizione possono essere ottenute dal `LocationManager` (uno strumento messo a disposizione da Android che permette di ricevere notifiche sulla posizione del dispositivo o lanciare un Intent nel caso in cui lo stesso entri in prossimità di una particolare posizione geografica) in modi diversi a seconda dei `LocationProvider` disponibili. Si può pensare al LocationProvider come quel meccanismo fisico che è in grado di acquisire la posizione del terminale. I dispositivi che utilizza Android per acquisire la posizione sono due: GPS o basato sulla rete. Nel progetto è stata utilizzata la classe `LocationManager` che permette di avere delle indicazioni sui `LocationProvider` disponibili ed utilizzarli secondo le esigenze dell'applicazione. Perciò è stata data la possibilità di avere una posizione geografica, seppur non estremamente precisa, nel caso di utilizzo della rete anche in situazioni critiche per il ricevitore GPS.

Personalizzazione



Il livello di personalizzazione permette all'utente di configurare l'applicazione in modo tale che risponda alle proprie esigenze in fatto di usabilità e funzionalità.

Per permettere all'utente di personalizzare il funzionamento del software come meglio preferisce, è stato reputato necessario aggiungere alla schermata Preferences il maggior numero di opzioni possibili. Oltre alle già presenti distanze (raggio di ricerca e distanza di aggiornamento hints), sono stati aggiunti (come da Figura 3.1):

- Parser Language: specifica la lingua con la quale deve essere interpretata la stringa inserita in linguaggio naturale;
- Notification: specifica quali notifiche si desidera attivare (le notifiche visuali sono sempre attive):
  - Sound;
  - Vibrate;
  - Speech;
- Advanced preferences: specifica il server da utilizzare:
  - Select server from the list;
  - Edit server location (inserimento manuale).

Per quanto riguarda la lingua del parser, è facile implementare la scelta in base alle impostazioni presenti a livello di sistema del cellulare.

3.3 . Documentazione del codice

La prima fase del progetto è stata quella della documentazione del codice. Dopo aver sincronizzato il progetto con il repository online di Google Code, si è passati alla sua analisi e alla redazione di diagrammi UML (in particolare diagrammi di sequenza) per capire il percorso delle informazioni nel sistema e alla creazione di un JavaDoc per la documentazione del sorgente.

Tale fase si è resa necessaria ed è stata estremamente utile per capire il principio di funzionamento di tutto il sistema e poterlo successivamente adattare alle nuove esigenze che venivano discusse, visto che sin dall'inizio è stato scelto come modus operandi il refactoring[16] dell'applicazione e non il reengeneering[17], ossia non abbandonare le soluzioni implementate per riscriverle ex novo riprogettandone la struttura.

16 Refactoring: indica il processo di modifica della struttura interna di un software eseguito senza modificarne il comportamento funzionale esterno o le funzionalità esistenti.

17 Reengeneering: metodo utilizzato principalmente in campo manageriale per sopperire a risultati insoddisfacenti. Ciò avviene ripensando l'azienda per processi orizzontali e riprogettando gli stessi secondo precisi standard di efficienza.

Capitolo 4 . Soluzioni implementate

4.1 . Sostituzione DBMS

Nell'analisi del sistema di base, uno dei problemi riscontrati fin dall'inizio, era dato dal fatto che, utilizzando db4o come DBMS, le prestazioni e la stabilità del prodotto erano molto influenzate dalle scarse prestazioni di questo componente; per questo motivo, il primo passo relativo alle modifiche al sistema è stato quello di valutarne il cambiamento.

Prima di effettuare una scelta, sono stati analizzati due paradigmi che potevano risultare utili alla sostituzione di db4o; tali sistemi sono infatti gli esponenti di due modi diversi di realizzare un DBMS:

- Database NoSQL (Apache Cassandra);
- Database relazionale (MySQL).

In ogni caso il passaggio da un DBMS ad oggetti come db4o ad uno di un'altra categoria tra quelle citate, ha comportato notevoli sforzi di adattamento del codice, soprattutto dal punto di vista dell'aggiornamento dei dati: si pensi a questo proposito al fatto che se i dati sono immagazzinati in un oggetto, le modifiche verranno rese durature attraverso operazioni come una semplice invocazione al metodo `dbmsInstance.persist(object)`, mentre successivamente si è reso necessario strutturare in modo formale i dati da memorizzare per creare le tabelle e le relazioni tra di esse al fine di relazionare i dati tra di loro.

db4o è, infatti, un database ad oggetti opensource che supporta sia lo sviluppo in Java che .Net, con il quale non è necessario mantenere, come nel caso dei db relazionali, un datamodel fisso, in quanto qualsiasi oggetto viene automaticamente considerato come una riga della tabella del database, permettendo quindi un alto grado di flessibilità ed astrazione nello sviluppo delle applicazioni.

La suite dei prodotti liberamente disponibili è formata da:

- motore di base per la gestione di un'istanza del database;
- sistema di replicazione, utile a sincronizzare le varie istanze;
- plug-in per la visualizzazione del database in Eclipse o Visual Studio.

Esistono poi altri servizi disponibili per lo sviluppo professionale. Dal punto di vista delle licenze, il motore di base è disponibile con licenza General Public License oppure, se si vuole includere db4o in un sistema rilasciato sotto licenza commerciale, tramite la db4o Opensource Compatibility License.

Come già anticipato, nel decidere quale dovesse essere il nuovo DBMS, sono stati presi in considerazione prodotti che appartenessero a filosofie diverse per la gestione dei dati, in modo da sostituire il DBMS esistente con dei livelli di performance e stabilità superiori.

4.1.1 . Database “NO-Sql”

In generale, il termine "no-sql" indica tutto un insieme di DBMS che cercano di differenziarsi rispetto ai tradizionali database relazionali grazie all'uso di sistemi alternativi per la gestione dei dati, quali ad esempio le coppie chiave-valore, rispetto al più classico metodo tabellare, basato sull'algebra relazionale.

Tali sistemi in genere puntano ad una scalabilità orizzontale, incentivata sempre più dal decrescere del costo dell'hardware, così si punta sulla replicazione dei dati in modo da raggiungere due importanti obiettivi:

- maggiore tolleranza rispetto alla caduta di uno o più nodi del sistema;
- maggiore probabilità di trovare i dati richiesti in un nodo vicino a quello che riceve la richiesta.

Alcuni esempi di sistemi NoSQL che possano amministrare una grossa mole di dati sono: Google-BigTable, HyperTable, Voldemort ed Apache Cassandra.

Nello specifico, nel presente progetto si è scelto di valutare soltanto sistemi che fossero già stati testati da qualche altro player conosciuto, in modo da non dover tenere conto, in fase di analisi dei risultati del testing dell'applicazione e di eventuali problemi dovuti al non trovarsi di fronte ad un sistema maturo e collaudato.

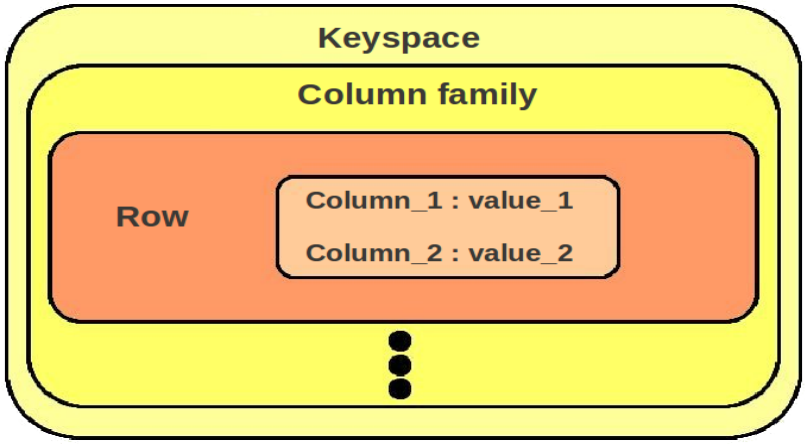
Dati questi presupposti, l'analisi ricade su Big-Table e Cassandra; la valutazione verte principalmente sul secondo in quanto dei due è l'unico ad essere anche liberamente disponibile, mentre Big-Table risulta utilizzato solo all'interno dei sistemi di Google.

Si tratta di un sistema nel quale una tabella è, in genere, una mappa multidimensionale dove la chiave è data da una stringa di qualsiasi dimensione, mentre il valore identificato dalla chiave è un insieme di array di Byte, che può quindi avere una struttura molto complessa.

Le operazioni su una riga del database restano comunque atomiche su ciascuna replica, considerando che il database ha tra i suoi punti focali il fatto di essere replicato su più nodi, a prescindere dalla quantità di dati che viene letta o scritta nell'operazione.

Per quanto riguarda il modo di organizzare i dati, Cassandra permette di strutturarli utilizzando tre tipi di oggetti:

- *Columns*: l'elemento base in cui vengono salvati i dati;
- *Column families*: gruppi di colonne; possono essere di due tipi, semplici o super (ulteriori raggruppamenti di colonne);
- *Keyspace*: è l'elemento di aggregazione di livello più alto. Solitamente ne è presente uno per tutta l'applicazione.



Column Family

USERS racchiude liste di key che puntano ad oggetti formati da coppie di chiave valore. Ad esempio: USERS{pippo->, pluto}; poi pippo e pluto possono avere all'interno un diverso numero di coppie chiave-valore, e.g. Pippo ha gli attributi nome, cognome, Pluto invece ha gli stessi ed, inoltre, ha anche password.

Super Column Family

In pratica è un livello ulteriore di direzione; ad esempio, posso avere la column family LooneyToons, che punta a una lista di keyword: bugs, duffy, etc; poi bugs si comporta come il pippo dell'esempio precedente e duffy come pluto.

CAP Theorem

Come ogni sistema distribuito, anche Cassandra deve sottostare al CAP (Consistency, Availability, Partition Tollerance) theorem, che recita:

> *"In un sistema distribuito è possibile avere contemporaneamente solo due tra Consistency, Availability, Partition tollerance"*

Il punto cruciale è capire se, a partire dalla scelta fatta tra queste tre caratteristiche, Cassandra possa essere utilizzato per il sistema in esame. Dalla documentazione presente nel sito di Apache Cassandra, è possibile assumere come la scelta sia ricaduta su Availability e Partition tollerance; si è quindi deciso di sacrificare in parte la consistenza, intendendo con questo che è necessario stabilire un trade-off tra le tre grandezze, che però tenga conto del fatto che la consistenza sarà quella che avrà meno probabilità di essere mantenuta, a meno di pagare molto in termini di A e P.

Per la gestione della consistenza dei dati, Cassandra utilizza la cosiddetta EVENTUAL CONSISTENCY, la cui definizione è indicativamente la seguente:

> *"Given a sufficiently long period of time, over which no updates are sent, we can expect that during this period, all updates will, eventually, propagate through the system and all the replicas will be consistent."*

In alternativa viene data invece quest'altra definizione:

> *"for a given accepted update and a given replica eventually either the update reaches the replica or the replica retires from service."*

Nei database, questo concetto si traduce poi nel cosiddetto BASE (Basically Available, Soft state, Eventual consistency) che appunto è in contrasto con quanto espresso dal termine ACID.

4.1.2 . Database relazionali

Si tratta di DBMS che si basano sul modello relazionale, un modello logico di rappresentazione dei dati fondato sull'algebra relazionale e la teoria degli insiemi. In letteratura esiste una ampia e aggiornata documentazione sui database relazionali, alla quale si rimanda per l'approfondimento di tale argomento.

Per quanto riguarda il panorama dei database relazionali, spicca il nome di MySQL, progetto iniziato dalla svedese MySQLLabs ed acquisito successivamente da Sun Microsystems e recentemente da Oracle.

Tale prodotto, considerando che la consistenza dei dati è punto cruciale per il progetto, è stato il candidato scelto per sostituire il vecchio DBMS db4o. Forti del funzionamento e dell'affidabilità di MySQL, un ulteriore punto a favore si è rivelato nella conoscenza del prodotto da parte dei componenti del gruppo di lavoro.

Il cambio del DBMS nel sistema, come indicato precedentemente, ha necessitato uno studio preliminare del funzionamento e un adattamento di quanto presente al modello E-R (modello entità-relazione). La struttura tabellare del database è riportata nel file

SQL in Appendice G. Il problema principale è stato quello di “tradurre” gli oggetti Java utilizzati dalle varie classi del server (oggetti che venivano memorizzati tali e quali con db4o) in una serie di tabelle e relazioni tra di esse che riproducessero il più fedelmente possibile la struttura del sistema. A tal proposito è stato dapprima definito uno schema del database che potesse sostituire le informazioni contenute negli oggetti memorizzati da db4o, successivamente realizzata l'interfaccia (`IFDatabaseManagement`) che definisce le operazioni necessarie per l'interazione con MySQL e infine implementata tale interfaccia (`MySQLDBManager`) scrivendo tutti i metodi definiti. L'interfaccia creata per la definizione dei componenti necessari per l'utilizzo di un DBMS, permette di svincolare l'intero progetto dal DBMS scelto e, in caso di futura sostituzione, implementare l'interfaccia con le specifiche del nuovo database scelto. Nell'implementazione delle funzionalità per MySQL, è stata utilizzata la libreria jdbc, specificamente creata per interfacciare MySQL con il linguaggio di programmazione Java. Anche questa libreria viene automaticamente importata (in fase di importazione del progetto) ed inserita nel pacchetto finale di rilascio da Maven.

4.2 . Task di gruppo

Ritenendo utile l'implementazione di Task condivisi, la soluzione che è stata implementata è quella dei gruppi. Con Task condiviso si intende una particolare attività geolocalizzata (la cui posizione dev'essere stabilita dal sistema analizzato) condivisa tra vari utenti. La soluzione implementata prevede che la condivisione di un Task avvenga attraverso la creazione di gruppi di utenti (gruppi chiusi, solo ad invito) e la creazione di un Task inserendo anche il gruppo di appartenenza. Sfruttando questo meccanismo, ereditato ad esempio dai gruppi di Skype, è possibile notificare un Task ad un utente anche non avendolo inserito personalmente: facendo parte di un gruppo, presumibilmente il task inserito interessa anche a tale utente. In questo modo è possibile creare delle attività che possono essere portate a termine da persone diverse: la prima che la porta a termine la elimina dal sistema e agli altri utenti non viene più notificata.

Nella creazione di un sistema che permetta la gestione di gruppi di utenti, si deve prevedere una serie di funzionalità di base:

- un mezzo attraverso il quale creare il gruppo;
- il metodo usato dagli utenti per venire a conoscenza del gruppo;
- le procedure da seguire per accettare o revocare l'invito ad un gruppo.

Il sistema già presente per l'inserimento dei Task è stato sostanzialmente mantenuto, aggiungendo alcune voci (nella schermata di inserimento da terminale e conseguentemente nel database lato server) per supportare questa nuova funzionalità. Il primo passo è stato quello di legare ogni Task ad uno specifico gruppo. Questa scelta si ripercuote in un nuovo parametro da inserire lato client e nella creazione di un nuovo attributo (UserGroup) nel database lato server nella tabella Task. Nel caso di inserimento di un task tradizionale (non condiviso), l'id del gruppo è pari a 0,

quindi viene mantenuta la funzionalità originale che permette ad un utente di inserire nel sistema task cosiddetti “privati”. L'attributo UserGroup nel database è di tipo autoincrement il cui valore parte da 1. E' pertanto facile distinguere quali sono i Task di gruppo dai Task tradizionali (personali).

4.2.1 . Implementazione

Al fine di mantenere la struttura dell'applicazione, le risorse necessarie al funzionamento dei gruppi nel sistema sono disponibili nella classe GroupResource. Tale classe mette a disposizione una serie di risorse che permettono l'interazione dell'utente con il sottosistema che gestisce i gruppi. Tutte le richieste pervenute vengono successivamente inoltrate alla classe GroupManager, un componente che si occupa di fare da cuscinetto tra la business logic dei gruppi e la sezione che gestisce la persistenza dei dati (la classe GroupDatabase) che crea le query da sottoporre poi al DBMS.

A livello database sono state create le seguenti tabelle:

- *UserGroup*: contiene i dati del singolo gruppo. Si noti il fatto che il nome qui non identifica il gruppo, come invece accade per gli utenti: questa scelta è stata fatta per permettere la presenza di nomi di gruppo duplicati (ad esempio il gruppo “famiglia” o “casa”). Viene inoltre memorizzato il nome dell'utente che crea il gruppo. Attualmente questo dato risulta ininfluente rispetto alla gestione dei gruppi, ma in futuro si potrà decidere di concedere a questo utente particolari privilegi circa la gestione del gruppo stesso, come ad esempio una sorta di privilegi di amministratore;
- *GroupMember*: oltre all'indicazione dell'id del gruppo e lo username dell'utente iscritto, si ha anche un'indicazione di quando questi si sia unito al gruppo; questo dato può risultare utile ad esempio nel momento in cui si vogliano fare statistiche sul modo in cui si formano i gruppi o su come, mediamente, questi vadano a crescere;
- *GroupRequest*: questa tabella viene utilizzata per accodare tutti gli inviti ricevuti da un utente che risultano in attesa di approvazione o rifiuto.

Quando un utente vuole invitarne un altro, questi può inviargli, unitamente all'invito, anche un messaggio, utile a spiegare il perché della creazione del gruppo, o ad identificare la provenienza della richiesta. Tale soluzione è stata scelta per ovviare al problema legato all'utilizzo di pseudonimi, non sempre facilmente riconducibili alla persona fisica. In futuro si potrebbe pensare di far sì che venga anche dato accesso ai dati anagrafici degli utenti che richiedono l'affiliazione ad un gruppo. Tali dati sono infatti presenti nel database dopo la fase di registrazione.

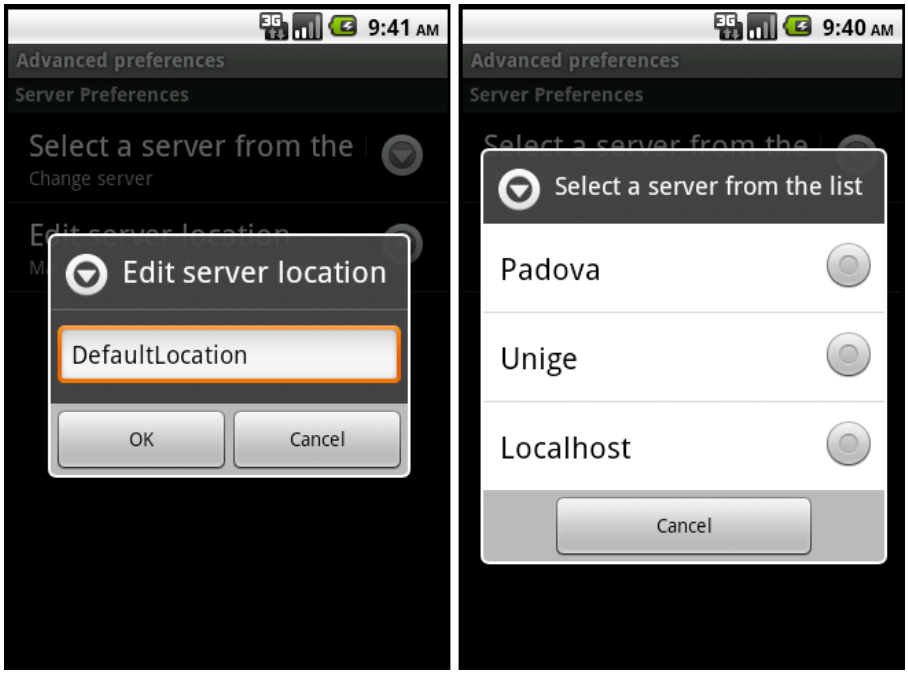
In Appendice B sono presenti alcuni screenshot del client che mostrano le varie fasi di utilizzo della sezione gruppi, in particolare le schermate di gestione operazioni gruppi (aggiunta gruppi, invito ad un gruppo, update inviti, visione membri appartenenti ad un gruppo).

4.3 . IP del server modificabile

Nella versione di base del client non era possibile definire questo parametro; l'ip del server era presente direttamente nel codice, nella classe `NetworkUtilities`. Cambiare l'indirizzo al quale è raggiungibile il server comporta pertanto una modifica del codice sorgente e un nuovo build di tutto l'applicativo lato client.
La soluzione è stata la creazione di due diverse modalità:

- selezione del server da una lista;
- inserimento manuale dell'indirizzo del server.

La prima modalità è di aiuto agli utenti medi: la selezione del server avviene semplicemente da una lista senza dover ricordare parametri particolari quali l'ip, la porta di comunicazione o la cartella dove risiedono le risorse. La seconda modalità è per utenti avanzati che conoscono tali parametri. La serie dei server disponibili è configurabile nel sorgente attraverso la modifica del file `array.xml` presente in `res/values/`.



Nel presente progetto sono disponibili tre server, due fisici ed uno virtuale residente nella macchina di sviluppo ed avviabile tramite Eclipse. In particolare il file di configurazione è:

```
<string-array name="serverList">
    <item>Padova</item>
    <item>Unige</item>
```

```
        <item>Localhost</item>
    </string-array>
    <string-array name="serverListValue">
        <item>serverpd.dyndns.org</item>
        <item>zelda.openlab-dist.org</item>
        <item>10.0.2.2</item>
```

Una volta selezionato un server dall'apposita interfaccia grafica (Figura 4.2), il cambiamento dell'indirizzo da contattare per le richieste non è immediato: viene utilizzato inizialmente un metodo (`TryConnection`), reperibile nel server al path `percorso_base/tryConn`, che permette di verificare la connessione con il server e notificare all'utente eventuali problemi. Se il test di connessione viene portato a termine positivamente, avviene il cambio dell'indirizzo, in caso contrario non viene utilizzato il nuovo server.

4.4 . Ricerca Hints forzata

L'introduzione di un sistema di aggiornamento degli Hints basato sullo spostamento e non più su un tempo, ha portato benefici in quanto a riduzione dell'impegno di banda non necessario (se l'utente resta fermo in un punto, il client non continua, ad intervalli di tempo fissati, a scaricare dal server la lista degli hints, occupando banda inutilmente, ma lo fa solamente dopo essersi spostato). Si è valutato che però l'utente possa avere la necessità o volontà di ricercare Hints in un momento particolare (ad esempio non appena riparte dopo una sosta in un punto). È stato creato pertanto un pulsante nel menù del client per permettere la ricerca degli hints in una posizione particolare.

A tal proposito è stato implementato un servizio che intercetta il thread che si occupa del download degli hints dal server e lo sblocca alla richiesta dell'utente. Tale thread è solitamente in stato inattivo durante l'utilizzo normale del sistema; viene risvegliato dal LocationManager quando si verifica uno spostamento del terminale maggiore della distanza impostabile da preferenze del sistema. Con questo servizio si va a svegliare il thread inattivo per il download degli hints a richiesta dell'utente, ma non si va a modificare il comportamento tradizionale; infatti se viene richiesto un download degli hints forzato, questo avviene nel momento in cui viene richiesto ed il successivo aggiornamento avverrà quando il LocationManager riattiverà il thread, cosa che succede in base alla distanza impostata nelle preferenze, indipendentemente dal fatto che sia stata effettuata o meno una ricerca forzata.

4.5 . Modalità di notifica degli hints

Punto cruciale del presente sistema è il reperimento dei punti su un servizio mappe, come ad esempio Google Maps, dove è possibile soddisfare i task, le esigenze dell'utente. A supporto di tale funzionalità, deve esistere un valido metodo di notifica dei punti di interesse, altrimenti il sistema risulta poco fruibile dall'utente. Con

notifiche si intendono tutte le azioni che intraprende il terminale per focalizzare l'attenzione dell'utente sul reperimento di un punto di interesse. Basti pensare ad esempio alle notifiche delle chiamate voce, utilizzando la suoneria oppure la vibrazione del cellulare, o, allo stesso modo, le notifiche per il ricevimento di un SMS. Se ciò non avvenisse, l'utente non avrebbe la possibilità di notare cosa succede. Per quanto riguarda ad esempio gli SMS, si tradurrebbero solo in una lettura differita del messaggio, per il sistema in oggetto però si traduce in un allontanamento dell'utente dal punto geografico nel quale può soddisfare una determinata esigenza e quindi la perdita della possibilità di portare a termine il task. Questo esempio mette in evidenza quanto importanti siano le notifiche da parte dell'applicazione.

Sono state prese in considerazione notifiche di diverse tipologie, a scelta dall'utente a seconda delle specifiche abitudini:

- notifiche visuali: visualizzazione nella parte superiore del display del cellulare, nella barra di stato, di una icona che identifica una notifica proveniente dall'applicazione;
- notifiche audio: suoneria legata alla ricezione dal server di alcuni punti di interesse nei quali poter soddisfare dei task, stile sms;
- notifiche tramite vibrazione: vibrazione del terminale nel momento in cui si riceve qualcosa dal server;
- notifiche vocali: utilizzo dell'altoparlante del terminale per leggere la notifica quando viene ricevuta.

Le notifiche visuali, audio e vocali, non sono molto personalizzabili, in quanto viene visualizzata una icona o riprodotto un suono, in stile sms a come ogni utente è abituato utilizzando un cellulare; le notifiche vocali prevedono la possibilità di tradurre in voce delle stringhe passate al sistema, utilizzando i servizi di speech (in base alla lingua impostata nel telefono) messi a disposizione da Google. Per quanto riguarda la vibrazione, invece, c'è la possibilità di personalizzarla in base a moltissime variabili. Si è focalizzato l'attenzione, pertanto, sulla modulazione della vibrazione per far percepire all'utente diverse modalità di notifica.

È importante aggiungere che, in campo psicofisiologico, l'essere umano è portato ad assuefarsi ad alcuni stimoli che prevedono l'uso dei sensi. Ad esempio, l'essere costantemente immersi in un rumore continuo, porta inevitabilmente a non sentire più tale rumore. Lo stesso vale per la vibrazione: essendo costante e sempre uguale, non permette di distinguere notifiche provenienti da contesti diversi e porta ad una minore sensibilità a questa tipologia di stimolo fino alla degenerazione che non la fa più percepire.

4.5.1 . Gestione della vibrazione

Android mette a disposizione due tipologie diverse di vibrazione del terminale:

- vibrazione di default: vibrazione standard con la quale non sarebbe possibile distinguere due notifiche provenienti da applicazioni diverse;

- vibrazione secondo pattern: modulazione della vibrazione effettuata con periodi di vibrazione alternati a pause, ripetuti un certo numero di volte.

Per rendere chiaramente identificabile la vibrazione proveniente dall'applicazione, si è scelto di utilizzare la seconda tipologia. In Android tale tipologia viene facilmente realizzata con un array, contenente due periodi base, uno di vibrazione e uno di pausa, ripetuti un certo numero di volte pari alla durata della vibrazione. Si è scelto inoltre di modulare la vibrazione in base alla priorità del task notificato oppure in base alla distanza rispetto alla posizione dell'utente.

Vibrazione - Notifica per priorità

Tale notifica richiama l'attenzione dell'utente solo sul task che ha priorità impostata più alta. Proprio per questo motivo, la notifica all'utente avviene solo dopo aver ricevuto l'intera lista dei task da portare a termine dal server ed avviene utilizzando un pattern in grado di dare all'utente una indicazione del livello di priorità del task, utilizzando un periodo di vibrazione pari a

$$minVibrate + minVibrate * (1 + (1 * priority))$$

con $minVibrate$ una costante che indica il valore base della vibrazione.

Il periodo di pausa è invece calcolato con

$$minPause + minPause * (1 + (1 / priority))$$

con $minPause$ una costante che indica il valore base della pausa.

Vengono prodotti così dei pattern che alternano periodi di vibrazione crescenti con la priorità alternati a periodi di pausa decrescenti con la priorità.

Vibrazione - Notifica per distanza

La situazione è identica alla precedente, tranne per il fatto che le formule per il calcolo dei periodi base di vibrazione e pausa sono ottenuti utilizzando la distanza del task dall'utente anziché la priorità.

Tale metodologia di notifica è presumibilmente più apprezzata dagli utenti, in quanto in fase di inserimento del task nel sistema non si utilizza sempre il campo priorità ed inoltre perché potrebbe essere più utile, al fine di risparmiare del tempo, notificare i punti di interesse in base alla distanza effettiva alla quale l'utente può soddisfarlo (questo si traduce in un risparmio di tempo più vicino è il task da soddisfare, in quanto la deviazione rispetto alla destinazione da raggiungere è minore).

4.6 . Verifica compatibilità versione

Oltre al test di connessione trattato nel precedente paragrafo 4.3 ., si è resa necessaria la verifica della compatibilità di versione tra i software client e server. Tale funzionalità è stata pensata per evitare spiacevoli interruzioni di funzionamento dell'applicazione, sia lato client che lato server, derivanti da richieste fatte dal client a risorse non disponibili (versione client troppo recente per il server) oppure risposte del server non correttamente interpretabili (versione del server troppo recente per il client).

Lo standard adottato per la numerazione delle versioni dell'applicazione sia lato client che lato server, prevede l'utilizzo di 3 numeri progressivi separati da punti, secondo lo schema A.B.C con:

- A: Major release number. Tale numero viene incrementato nel caso ci siano dei forti cambiamenti allo schema oppure API radicalmente cambiate;
- B: Minor release number. Tale numero viene incrementato nel caso siano state implementate aggiunte o estensioni allo schema o alle API, nuove funzionalità, aggiornamenti sulla sicurezza o aggiornamenti su bug presenti nella Major release;
- C: Bugfix release number. Tale numero viene incrementato nel caso di implementazioni atte al consolidamento di altre tipologie di bug non identificate da B.

Il controllo della compatibilità della versione avviene in fase di selezione del server nella schermata di login e in caso di cambiamento del server runtime dalla schermata Preferences del client.

Sia nel client che nel server, esiste un file di configurazione (`Constants.java` e `system.conf` rispettivamente) nei quali vengono memorizzati i numeri di versione dell'applicazione e il numero di versione minimo richiesto per il corretto funzionamento. Lato server, ad esempio, nel file `system.conf` saranno memorizzati i valori `VERSION` (versione corrente del server) e `MIN_CLIENT_VER` (versione minima del client per garantire il corretto funzionamento).

Lato client, avviene una richiesta al servizio reperibile nel server all'url http://indirizzoserver:8080/ephemere/checkVer comunicando (metodo GET) il proprio numero di versione.

Lato server, la richiesta viene gestita dalla classe `CheckVersionResource` del package `web` che confronta il numero di versione comunicato dal client con il valore `MIN_CLIENT_VER`. Nel caso il numero di versione comunicato dal client sia maggiore o uguale a `MIN_CLIENT_VER`, allora il server risponde al client comunicando il proprio numero di versione e la possibilità di compatibilità del client con il server.

Non appena il client riceve la risposta dal server, anch'esso controlla la versione del

server comunicata con il proprio valore `MIN_SERVER_VER`, per controllare, in questo caso, la compatibilità del server con il client. In caso positivo è garantito l'utilizzo del server selezionato, in caso contrario viene notificato all'utente un messaggio di errore che identifica quale software non è compatibile e suggerisce l'aggiornamento o del client o del server.

4.7 . Registrazione nuovo utente

Tutti i sistemi basati su architettura client/server necessitano di una autenticazione per l'accesso alle risorse. Tramite l'autenticazione è possibile identificare univocamente un determinato utente e garantirgli l'accesso solo alle proprie informazioni. Parimenti, per garantire la segretezza delle proprie informazioni, un utente non ha accesso ai dati inseriti da altri utenti.

Per rendere più User Friendly possibile l'applicazione, si è pensato di inserire una procedura guidata di registrazione nuovo utente direttamente disponibile nel terminale mobile, svincolando così la necessità di centralizzare le registrazioni tramite ad esempio una interfaccia web disponibile nel sito di riferimento dell'applicazione (cosa sulla quale si appoggiano tantissimi servizi, quali ad esempio Skype). La possibilità di distribuire l'applicazione completa, lato client tramite archivio `.apk` oppure attraverso il Market di Android e lato server come macchina virtuale preconfezionata (da utilizzare con sistema di virtualizzazione come VMWare) oppure tramite archivio `.war` da installare su un server Tomcat già esistente,elimina la centralità del server, pertanto risulta comodo potersi registrare al sistema tramite terminale. La registrazione nuovo utente, che poco si discosta da una registrazione classica, necessita dell'inserimento di alcuni dati personali, quali ad esempio: nome, cognome, indirizzo di posta elettronica e un nome utente ed una password da utilizzare per l'autenticazione nel sistema.

La procedura di registrazione realizzata, dopo l'inserimento dei dati da parte dell'utente, trasmette tali dati al server che li controlla e memorizza ed invia una mail con un link per confermare la registrazione all'indirizzo di posta elettronica indicato in fase di registrazione. Dopo aver portato a termine la registrazione è data la possibilità all'utente di avere l'account appena creato immediatamente funzionante, potendo usufruire di 5 login di prova (trial login) con i quali l'utente può provare l'applicazione e, se di suo gradimento, passare all'attivazione del proprio account. Questa funzionalità è stata inserita per evitare di dover richiedere all'utente l'immediata conferma della registrazione.

4.7.1 . Implementazione

4.7.1.1 . Lato client

L'implementazione della registrazione nuovo utente lato client si riconduce alla creazione di una nuova Activity che definisce l'interfaccia grafica per l'inserimento

dei dati necessari alla registrazione dell'utente (Figura 4.3). L'activity indicata viene richiamata dalla schermata di login tramite la pressione del pulsante *Register*. Nella schermata di registrazione viene richiesto l'inserimento di tutti i parametri presenti. La correttezza dei valori inseriti viene garantita tramite il confronto delle stringhe inserite con delle espressioni regolari create ad hoc:

- per i campi nome, cognome, username, password: `.*\\S.*`
- per il campo email: `[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,4}`

Il doppio campo password serve al controllo del corretto inserimento della password utente.

Il pulsante Cancel permette di annullare la procedura di registrazione e ritornare alla schermata precedente.

La pressione del pulsante Submit, avvia il processo di registrazione: viene creato un thread, definito nella classe `RegistrationResource`, che crea l'url al quale è disponibile la risorsa (lato server) che gestisce il processo di registrazione preoccupandosi di aggiungere i dati che verranno elaborati. Tutti i campi inseriti, ad eccezione del campo password, sono inseriti in coda all'url, utilizzando così il metodo di richiesta GET; la password invece viene inserita nell'header del messaggio HTTP inviato al server.

4.7.1.2 . Lato server

Al server perviene una richiesta GET del tipo:

```
http://indirizzo:8080/ephemere/registration?fn=nome&
ln=cognome&e=email@email.com&u=user
```

La richiesta viene gestita dalla classe `RegistrationResource`, in particolare dal metodo `Registration` che provvede a recuperare i parametri della richiesta (il processo è automatizzato mediante l'utilizzo di JAX-RS che utilizza le annotazioni per reperire le corrette informazioni) e a invocare il metodo `register` della classe `RegistrationManager` dove avviene l'elaborazione vera e propria delle informazioni pervenute dal client. Viene successivamente restituito al client un oggetto di tipo `RegistrationReply` che contiene il codice di stato dell'operazione di registrazione. Tale codice di stato definisce se l'operazione di registrazione è andata a buon fine. I codici di stato per la registrazione sono i seguenti:

- 2000 – Registration successful
- 2001 – Registration not confirmed
- 2002 – Registration failed, username already exist
- 2003 – Registration failed, email already exist

Il metodo `register` effettua inizialmente il controllo dell'esistenza del nome utente o dell'indirizzo di posta elettronica nel database (nel primo caso l'utente deve cambiare username perché già utilizzato, nel secondo caso l'utente si era già registrato al sistema); in caso positivo segnala, tramite l'oggetto `RegistrationReply`, il codice dell'errore così che possa venire notificato all'utente dal terminale. Nel caso non fossero presenti username o indirizzo di posta elettronica, viene creato un codice di verifica per la registrazione calcolato utilizzando l'algoritmo di criptazione md5 partendo dalla stringa formata da "`thesisug-YYYY-MM-DDThh:mm:ss`". Tale stringa, insieme ai dati forniti dall'utente in fase di registrazione, vengono memorizzati nella tabella `User` del database invocando il metodo `addUser` della classe `RegisteredUsers`, che insieme alla classe `MySQLDBManager` è responsabile della connessione al database e della sottomissione della corretta query per l'effettiva memorizzazione. Viene inoltre inviata all'indirizzo di posta elettronica indicato dall'utente una mail nella quale è presente un link per attivare l'account creato e garantire l'accesso incondizionato all'utente (nel caso non venisse confermata la registrazione si hanno a disposizione 5 trial login, dopo dei quali bisogna necessariamente registrare l'account).

Per la conferma della registrazione, cliccando sul link inviato all'indirizzo di posta elettronica in fase di registrazione, viene richiamata una ulteriore risorsa presente all'url http://indirizzoserver:8080/ephemere/verification?code=verificationcode&email=email@email.com che tramite il metodo verify della classe `RegistrationManager` controlla nel database la presenza dell'indirizzo di posta elettronica indicato e del codice di verifica e avverte l'utente dell'avvenuta attivazione o meno dell'account.

In Appendice A sono disponibili i diagrammi di sequenza per la procedura di registrazione nuovo utente.

4.8 . Implementazione parser in italiano

Il parser esistente (derivato dal parser di Ubiquity) è stato parzialmente modificato, adattandolo alle esigenze dell'italiano.

Al fine di facilitare l'internazionalizzazione del parser, sono stati creati due file di configurazione per il parser per ogni lingua (per ora italiano e inglese, ma per far funzionare il parser con una nuova lingua basta creare ed aggiungere i due file relativi alla lingua da aggiungere).

Il primo file `xx.lang` (dove `xx` è il codice ISO 3166-1 alpha-2[18] che identifica il paese) contiene i delimitatori che contribuiscono al riconoscimento sintattico degli argomenti di una frase da analizzare, mentre il secondo file `xx_verbs.lang` contiene tutti i verbi che identificano una determinata azione, sia per i task che per gli eventi. Ad esempio:

```
filename=en.lang
name=English
roles=GOAL,OBJECT,SOURCE,LOCATION,LOCATION,LOCATION,TIME,TIME
delimiters=to,to,from,in,around,near,at,before
```

e

```
filename=en_verbs.lang
name=EnglishVerbs
task_verbs=i have,remind me,add task
event_verbs=add event,i have appointment,add appointment
```

Nell'inserimento di un task o di un evento, ciò che serve è essenzialmente l'oggetto (OBJECT), cioè cosa si va a fare, e il limite temporale (TIME, per ora solamente di orario, ma si può facilmente estendere impostando una deadline sulla data).

Il parser si articola principalmente in due passi:

- STEP1: estrazione dei possibili verbi dalla frase;
- STEP2: estrazione degli argomenti riferiti ai verbi precedentemente estratti.

C'è una sostanziale differenza rispetto al parser di Ubiquity, dove il parser viene utilizzato per “indovinare” l'azione che l'utente vuole intraprendere, ad esempio scrivendo:

```
to Jono
```

ubiquity suggerisce:

```
email to jdicarlo@mozilla.com
twitter to jono
translate to jono
```

In questo progetto si vuole riconoscere invece l'azione che l'utente vuole fare. In questo caso la struttura sintattica della frase da analizzare è un po' più rigorosa e risulta più semplice effettuare l'analisi sintattica.

18 ISO 3166-1 alhpa-2: è la parte più famosa dello standard ISO 3166-1 che conta 248 codici a due cifre che identificano la nazione.

La facilità con la quale si aggiunge una nuova lingua al parser sta nella creazione di una coppia di file, uno che identifichi i delimitatori e l'altro che identifica i verbi utilizzati nelle frasi.

Per l'italiano, vengono riconosciute le frasi che contengono i seguenti verbi:

```
filename=it_verbs.lang
name=ItalianVerbs
task_verbs=devo,ricordami,aggiungi task
event_verbs=aggiungi evento,ho un appuntamento, aggiungi
appuntamento
```

e i seguenti delimitatori

```
filename=it.lang
name=Italian
roles=GOAL,OBJECT,OBJECT,OBJECT,OBJECT,OBJECT,OBJECT,SOURCE,LOCATIO
N,LOCATION,LOCATION,TIME,TIME,TIME,TIME
delimiters=a,di,devo,aggiungi task,aggiungi evento,ho un
appuntamento,aggiungi appuntamento,da,in,vicino,nei dintorni,prima
delle,prima di,prima dell\',alle
```

Step 1

Step1: estrazione verbi

Input: frase da analizzare

Output: Lista di coppie <comando> - <verbo>

Il parser riconosce due tipologie di comandi, *Add Task* e *Add Event*, che si traducono successivamente in comandi veri e propri per inserire nel database le tuple necessarie alla memorizzazione del task o evento da portare a termine.

Per ogni tipologia di comando, allo step1 si tenta di riconoscere il verbo che più si avvicina a soddisfare il relativo comando, utilizzando una serie di espressioni regolari che vengono confrontate con la stringa in ingresso. Se l'espressione regolare è contenuta nella stringa di input, viene aggiunta alla lista insieme al relativo comando e a uno score che esprime quanto un verbo esprime correttamente la funzione del comando. Lo score viene calcolato in base al numero di espressioni regolari, derivate dal verbo associato ad un comando, successivamente normalizzato nel range 0-1. Ad ogni iterazione lo score viene decrementato di 1. Un esempio rende più semplice la comprensione.

Il verbo che identifica il comando *Add Task* è *"ricordami"*. Per il verbo ricordami vengono create le seguenti espressioni regolari:

ricordami – ricordam – ricorda – ricord – ricor – rico – ric – ri – r

alle quali inizialmente vengono associati i seguenti score:

9 – 8 – 7 – 6 – 5 – 4 – 3 – 2 – 1

Siccome non tutti i verbi hanno la stessa lunghezza e pertanto le espressioni regolari associate avrebbero score diversi, si normalizza lo score nell'intervallo 0-1 utilizzando la formula

$$score = \frac{score}{nr.regex}$$

in questo modo gli score precedentemente indicati vengono normalizzati in:

1 – 0,88 – 0.77 – 0,66 – 0,55 – 0,44 – 0,33 – 0,22 – 0,11

Infine viene mantenuto, per ogni comando, il verbo che riporta score maggiore, che è il verbo che identifica nel modo migliore il comando da eseguire e vengono memorizzati quindi i relativi argomenti.

Step 2

Step2: estrazione argomenti

Input: dati estratti allo Step 1

Output: Lista di oggetti PossibleParses (comando, lista argomenti, score)

Per ogni coppia comando-verbo estratta al punto precedente, viene controllata la presenza di ciascun delimitatore (che identifica il relativo argomento come indicato nel file `xx.lang`). Se presente un determinato delimitatore, significa che la parte che segue è un potenziale argomento relativo al verbo estratto in precedenza. Se il potenziale argomento contiene a sua volta altri delimitatori relativi ad altri argomenti, questi vengono rianalizzati per l'estrazione dei successivi argomenti. Anche in questo caso un esempio esemplifica meglio la procedura.

Frase in input: *ricordami di prendere il latte prima delle 22*

Allo Step 1 vene estratta la potenziale coppia "*add task"* - *"di prendere il latte prima delle 22"*.

Allo Step 2 viene riconosciuto il delimitatore "`di`", che identifica l'oggetto, il cosa si vuole fare, per l'appunto prendere il latte, ma viene riconosciuto anche il delimitatore "`prima delle`" che identifica un argomento temporale, cioè il quando.

Lo step2 produce una lista di oggetti che contengono il comando (add task o add event), lo score, e la lista degli argomenti:

- ruolo – OBJECT o TIME;
- contenuto – l'argomento estratto precedentemente;
- tipo – sostantivo o time.

Il possibile comando da eseguire viene scelto tra i due presenti in base allo score. Tale punteggio viene eventualmente aumentato di un punto qualora gli argomenti estratti siano sia uno di tipo OBJECT che uno di tipo TIME. Il comando che ha score più alto è quello che si avvicina maggiormente alla tipologia di azione alla quale l'utente si riferisce.

Modifica per l'italiano

Diversamente da quanto succede per l'inglese, in italiano la struttura della frase può omettere dei delimitatori. Nella frase *"devo prendere il latte prima delle 20"* non compare nessun delimitatore che possa identificare l'oggetto "prendere il latte". La corrispettiva frase inglese è *"I have to buy milk before 8pm"* che contiene il delimitatore *to* che permette di riconoscere correttamente l'oggetto *"buy milk"*.

Per ovviare a questo problema, lo Step 1 è stato modificato rispetto al parser di Ubiquity. Una volta estratta la sottostringa che contiene i vari argomenti della frase, viene controllato che all'inizio di tale sottostringa sia presente almeno uno dei delimitatori indicati nel file `xx.lang`. In caso negativo, si otterrebbe una sottostringa senza delimitatori, perciò allo Step 2 non si riuscirebbe ad estrarre correttamente gli argomenti. Viene pertanto reinserito, all'inizio della sottostringa, il verbo che è stato identificato. Tale verbo funge esso stesso da delimitatore e l'argomento alla sua destra viene riconosciuto correttamente. Nell'esempio indicato poco sopra, dalla frase *"devo prendere il latte prima delle 20"* sarebbe stato estratto allo Step 1 *"prendere il latte prima delle 20"*. Lo step2 non potrebbe riconoscere correttamente l'argomento (di tipo OBJECT) *"prendere il latte"* ma solamente l'argomento (di tipo TIME) *"20"*.

Per far riconoscere correttamente tutti gli argomenti, è necessario reinserire il verbo *"devo"* all'inizio della frase. Essendo il verbo *"devo"* esso stesso un delimitatore (riportato nella lista dei delimitatori presenti nel file `it.lang`), il sistema riconosce correttamente anche questa tipologia di frase. Tale modifica al parser si rende necessaria anche per altre lingue diverse dall'italiano.

Ad esempio anche in francese la situazione è simile all'italiano. Le frasi che potrebbero essere dette sono *"n'oublier pas de prendre le lait"* (che letteralamente significa *"non dimenticare di prendere il latte"*) è *"je dois prendre le lait"* (che letteralmente significa *"devo prendere il latte"*). Nei casi citati, la prima frase possiede il delimitatore *"de"*, pertanto viene correttamente analizzata mediante il riconoscimento del delimitatore, la seconda frase invece non contiene nessun delimitatore, quindi come per l'italiano l'analisi viene fatta correttamente solo se si utilizza come delimitatore il verbo stesso (*"dois"*).

Capitolo 5 . Porte aperte

5.1 . Adaptive preferences

Le opzioni impostabili dalla schermata Preferences del terminale permettono all'utente di personalizzare il comportamento dell'applicazione in modo da adattarlo alle personali esigenze. Nessuno chiaramente vorrebbe trovarsi nella situazione di ricevere delle notifiche nel momento o nel luogo sbagliato. Ad esempio, nel marzo 2005, il giudice Robert Restaino nell'aula giudiziaria delle cascate del Niagara ha fatto arrestare 46 persone perché durante una causa penale un cellulare suonò in aula[19]. Si vorrebbe evitare tale situazione e pertanto adattare automaticamente il modo d'uso dell'applicazione in base alla posizione e situazione del terminale.

Come per il cellulare, del quale ogni utente può personalizzarne il funzionamento adattando volume della suoneria o livello di vibrazione in base al posto in cui si trova, anche il sistema analizzato potrebbe adattare il funzionamento in un modo simile. Si è pensato, pertanto, di cambiare le impostazioni dell'applicazione (per quanto riguarda la tipologia di notifiche) intercettando i cambi di modo d'uso del terminale e adattandosi alle impostazioni del terminale. Utilizzando un tale sistema di adattamento, è possibile modificare le preferenze dell'applicazione: ad esempio se un utente entra in ufficio, è probabile che metta il telefono in modalità silenziosa attivando la vibrazione al posto della suoneria, in modo da non disturbare l'ambiente lavorativo; allo stesso modo si vorrebbe evitare di notificare (se fosse disponibile) qualche task utilizzando metodi che potrebbero disturbare.

Nelle preferenze dell'applicazione è possibile selezionare, oltre alle modalità di notifica, anche le distanze di aggiornamento e di notifica. Tale parametro potrebbe anch'esso essere adattato automaticamente in base al mezzo con il quale ci si muove. Ad esempio, una possibile coppia di valori (per il raggio di notifica e la distanza di

19 http://news.cnet.com/8301-10784_3-9824710-7.html

aggiornamento) per l'utilizzo dell'applicazione a piedi, potrebbe essere 200mt e 250mt (come utilizzato nei test descritti nel Capitolo 6 .), due distanze che permettono di non effettuare troppe richieste verso il server e di notificare dei task facilmente raggiungibili a piedi senza deviare troppo da altri impegni. Sarebbe infatti estremamente poco produttivo notificare task in un raggio di 5 o 10 km dalla posizione dell'utente, distanze non facilmente raggiungibili a piedi. È altrettanto insensato utilizzare come distanza di aggiornamento 250mt se lo spostamento avviene in auto, in quanto il server verrebbe inutilmente inondato di richieste che porterebbero ad un possibile crash del sistema o eventualmente ad un inserimento in una black list da parte del servizio Google Maps per aver inoltrato troppe richieste in poco tempo.

È pertanto auspicabile un adattamento di tali parametri calcolando ad esempio la velocità media di spostamento del terminale, facendo aumentare tale parametri proporzionalmente alla velocità rilevata.

5.2 . Sistema di caching lato server

Come evidenziato nel paragrafo precedente, potrebbero verificarsi dei problemi di black list dei servizi ai quali ci si appoggia per il reperimento dei punti di interesse se vengono effettuate troppe richieste.

Per quanto riguarda il servizio Google Maps, l'utilizzo delle API viene concesso solo a patto di rispettare delle regole[20]:

- Richieste automatizzate proibite: tutte le richieste devono essere fatte in seguito ad una richiesta di un utente finale;
- Tutti i siti o applicazioni che utilizzano le API Google devono essere gratuiti;
- Il caching o la memorizzazione dei risultati sono permessi solo con certe API (si rimanda alla documentazione relativa per le condizioni di utilizzo);
- Nel caso di utilizzo delle API da parte di un RESTful Web Service, richiede l'inserimento di un valido HTTP Referer per garantire la rintracciabilità delle richieste pervenute.

Oltre a sottostare a tali regole, Google dà una serie di accorgimenti da adottare per evitare alcuni problemi comuni:

- Registrarsi per ottenere una API Key che permette a Google di contattare il “proprietario” della API Key in caso di problemi e fornisce un ulteriore metodo di rintracciabilità;
- Assicurarsi di non inoltrare richieste senza parametri che potrebbero comportare l'invio di molte più richieste di quante effettivamente necessarie (alcune volte inoltrate da cicli infiniti) che potrebbero essere valutate come “traffico abusivo”;
- Utilizzando un RESTful Web Service, utilizzare il parametro opzionale `userip` che permette di indicare l'ip dell'utente finale che richiede la risorsa

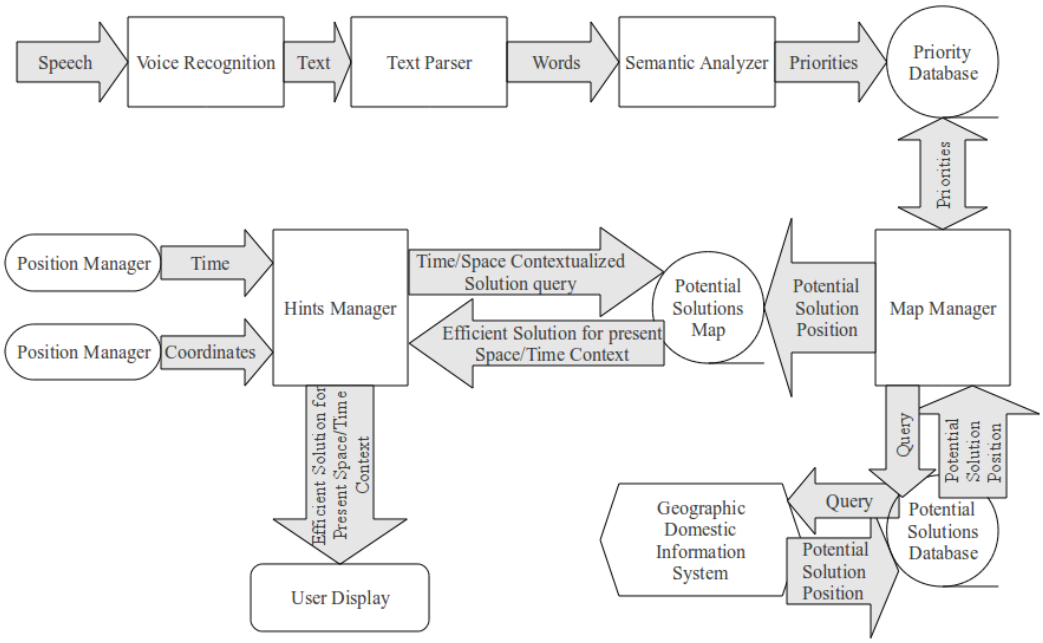
20 Vedere i termini di utilizzo e documentaizone delle API Google: http://code.google.com/intl/it-IT/apis/maps/documentation/staticmaps/

e garantisce pertanto che non si tratta di query automatizzate;

- A volte delle query sono effettuate a causa di crawler di motori di ricerca che visitano le pagine dove sono offerti i servizi Google. Tali situazioni possono essere evitate utilizzando un file di configurazione (ad esempio robots.txt) che evita che i robot inoltrino richieste alle API di Google.

Nel sistema sono rispettate le regole per l'utilizzo delle Google API e sono stati utilizzati dei parametri opzionali come la API Key. Risulta semplice adattare il client per l'utilizzo anche del secondo parametro opzionale (`userip`), facendo in modo che nelle richieste effettuate venga inserito come parametro anche l'ip del terminale.

Al fine di ridurre notevolmente le richieste inviate al servizio Google (ed eventualmente ad altri servizi aggiunti in futuro), si potrebbe optare per un sistema di caching (a seconda dei termini d'uso del singolo servizio) che permetta la memorizzazione dei dati ricevuti dai servizi e il soddisfacimento delle richieste dei client utilizzando il database. Attualmente il client, in base alla distanza di aggiornamento impostata, richiede al server una determinata risorsa. Il server esaudisce tale richiesta inoltrando al servizio Google una query, specificando le parole chiave e la posizione geografica dell'utente. Un sistema di caching si potrebbe frapporre tra il client ed il motore del server. Le richieste potrebbero essere esaudite



effettuando (lato server) una query direttamente al database. Il server dovrebbe pertanto essere modificato in modo da poter aggiornare periodicamente (inoltrando query ai servizi esterni) il database degli hints in base alle parole chiave che identificano i task dei vari utenti, salvando tutte le informazioni che si rendono necessarie. Lo schema del progetto proposto inizialmente (confronta Figura 2.5 a pagina 17) potrebbe essere modificato come indicato in Figura 5.1, dove compare una sezione database dedicata alla memorizzazione dei dati provenienti dai servizi esterni.

Come evidenziato, la richiesta dei potenziali suggerimenti non verrebbe effettuata più direttamente al servizio esterno, ma verso il database. Quest'ultimo verrebbe periodicamente popolato di nuove righe inoltrando, in background, le richieste verso l'esterno.

5.3 . Gestione del silenziamento

Al fine di rendere meno invasiva l'applicazione, è stata pensata una soluzione che permettesse di silenziare i task notificati rimandandone così il soddisfacimento. Con questo sistema, si va a modificare la priorità con la quale un Task viene notificato.

L'implementazione di questa soluzione coinvolge sia la struttura del client che del server e avviene attraverso l'inserimento di un ulteriore parametro che caratterizza un task: la sensibilità. Tale parametro indica un livello sopra il quale i task vengono notificati e sotto il quale no. La sensibilità viene impostata dall'utente oppure può avere un adattamento automatico come quanto indicato nel paragrafo precedente per le tipologie di notifica (si può pensare che anche la quantità di task notificati, e quindi il livello di intrusività del sistema, possa variare in funzione del posto in cui l'utente si trova e in base alla modalità di utilizzo impostata per il cellulare).

Impostando, ad esempio, un range di valori da 1 a 10 (in step di 0.5) per la sensibilità, l'adattamento automatico dovrebbe variare tale parametro utilizzando un fattore moltiplicativo che abbia dei valori decrescenti nel range 0-1, in modo tale da far decrescere la sensibilità proporzionalmente a tale moltiplicatore. Il problema si riconduce ora nella definizione di una legge matematica che permetta di ottenere, partendo da un range di valori 1-10, un moltiplicatore appropriato. È facile notare che il logaritmo in base 10 è un candidato ideale per tale situazione. Un moltiplicatore ottenuto con la seguente formula

$$multiplier = \log(sensibility)$$

permette di ottenere un moltiplicatore nel range (0,1]. Si può pertanto utilizzare tale moltiplicatore per adattare la sensibilità ad ogni silenziamento di un task. Ad ogni silenziamento di un task, la sensibilità relativa a tale task diminuisce e confrontato con il livello impostato dall'utente (manualmente oppure in modo automatico rispetto al modo d'uso impostato nel terminale) il task può venir notificato o meno.

La legge che governa il decrescere della sensibilità è allora definita da:

$$sensibility = sensibility * \log(sensibility)$$

Tale formula viene utilizzata per calcolare il nuovo livello di sensibilità impostato per un task ogni volta che viene silenziato. L'andamento del valore calcolato della sensibilità ad ogni silenziamento di un task è visibile in Figura 5.2.

Livello di sensibilità del task	Nr. di silenzimenti del task											
	1	2	3	4	5	6	7	8	9	10	11
1,0	0,00										
1,5	0,26										
2,0	0,60										
2,5	0,99										
3,0	1,43	0,22									
3,5	1,90	0,53									
4,0	2,41	0,92									
4,5	2,94	1,38	0,19								
5,0	3,49	1,90	0,53								
5,5	4,07	2,48	0,98								
6,0	4,67	3,12	1,55	0,29							
6,5	5,28	3,82	2,22	0,77							
7,0	5,92	4,57	3,01	1,44	0,23						
7,5	6,56	5,36	3,91	2,32	0,85						
8,0	7,22	6,20	4,92	3,40	1,81	0,47					
8,5	7,90	7,09	6,03	4,71	3,17	1,59	0,32				
9,0	8,59	8,02	7,25	6,24	4,96	3,45	1,86	0,50			
9,5	9,29	8,99	8,58	8,00	7,23	6,21	4,92	3,41	1,82	0,47	
10,0	10,00	10,00	10,00	10,00	10,00	10,00	10,00	10,00	10,00	10,00	10,00

In seguito alla modifica del parametro sensibilità nelle preferenze dell'applicazione, un task viene nuovamente notificato se il suo livello di sensibilità risulta essere superiore al livello impostato. Onde evitare che un task inserito non venga più notificato (situazione nella quale il livello di sensibilità impostato è piuttosto alto), si deve prevedere un sistema per l'aumento del parametro relativo al task. L'aumento del valore di sensibilità di un task, potrebbe avvenire dopo un certo tempo che il task non viene notificato. In questo modo anche task che non vengono notificati perché il loro livello di sensibilità è stato diminuito troppo, dopo un certo periodo vengono resi nuovamente visibili all'utente e quest'ultimo decide se soddisfarli, eliminarli oppure continuare a silenziarli.

Infine, quando un utente decide di silenziare un task, potrebbero essere proposte delle azioni da intraprendere nei confronti del task stesso, ad esempio:

- rimandare il task ad un orario differito;
- silenziamento incondizionato;
- eliminazione del task.

L'utente ha così la piena padronanza della gestione dei task e del livello di intrusività dell'applicazione in base alle esigenze.

Capitolo 6 . Test

6.1 . Offline test

Durante lo sviluppo delle nuove funzionalità aggiunte al server ed al client e la correzione di tutti i bug presenti e che si possono verificare, l'attività deve essere costantemente supportata da test del codice per verificare che il funzionamento effettivo corrisponda con quello atteso dai vari componenti.

In tutti gli ambienti di sviluppo, qualsiasi sia il processo di produzione del codice adottato, uno degli aspetti considerati poco importanti è la fase di testing. Spesso però tale aspetto è sottovalutato, per vari motivi. Ultimamente sono stati creati dei framework che servono principalmente ad agevolare la procedura di testing. In particolare buona norma dovrebbe essere quella di eseguire test di unità per assicurarsi che la singola unità di sviluppo assolva le sue funzioni seguendo i requisiti prestabiliti in fase di progettazione. Questo è uno dei più importanti passaggi per poter avere un prodotto finale di buona qualità grazie ad un processo rapido di integrazione del software. Basta riflettere sull'idea di stratificazione del software per immaginare quanto sia complesso il processo di debug se qualcosa va storto nelle unità di base.

Sicuramente la cosa più banale che possa esistere è il test della singola classe con un main che valuti la bontà dei metodi. Questa pratica ha delle evidenti limitazioni, come quella di non poter essere ripetuta nel tempo (ad esempio quando cambia un modulo del software) per i cosiddetti test di regressione (regression tests).

Può essere di aiuto un framework (per la programmazione Java) creato da Eric Gamma e Kent Beck: JUnit. Lo scopo che ha portato alla creazione di questo framework è quello di avere una piattaforma standardizzata per sviluppare test di unità basati su un semplice modello.

A partire dalla versione 4, JUnit si avvale delle novità apportate da Java 5 (annotations) per rendere lo sviluppo di test suite estremamente semplice ed immediato. Nelle vecchie versioni creare uno unit test era comunque semplice ma bisognava conoscere la struttura della classe da estendere e per convenzione chiamare i metodi di test (i test case) `testXXX` in modo che potessero essere eseguiti automaticamente (per reflection). Con le annotazioni tutto ciò si può facilmente evitare in quanto si può annotare ogni singolo metodo che si desidera faccia parte dei test case e, a runtime, le annotazioni verranno valutate.

Il processo di esecuzione quindi è guidato dalle annotazioni riassunte nella Tabella 6.1:

Annotazione	Utilizzo
`@Test`	Annotare i metodi di test
`@Before`	Annotare un metodo che dev'essere eseguito prima dell'esecuzione di un test case (ad esempio l'apertura di una connessione ad un database)
`@After`	Annotare un metodo che dev'essere eseguito dopo l'esecuzione di un test case (ad esempio la chiusura della connessione ad un database)
`@BeforeClass`	Stesso funzionamento di `@Before`, ma eseguito all'inizio dell'esecuzione del test
`@AfterClass`	Stesso funzionamento di `@After`, ma eseguito alla fine dell'esecuzione del test
`@Ignore`	Utilizzato per evitare l'esecuzione di un test (per evitare di commentare il codice)

Tabella 6.1: Annotazioni utilizzate da JUnit

Nell'applicazione sono state create tutte le classi che permettono il test per unità, al fine di testare tutti i metodi scritti e verificarne il funzionamento atteso per quanto riguarda la sezione server. I metodi test (annotati opportunamente dall'annotazione `@Test`) vengono eseguiti senza un ordine preciso, quindi non necessariamente nell'ordine in cui compaiono nelle classi di test. Un test fallisce se fallisce almeno uno dei metodi assert contenuti in esso. I metodi assert sono metodi statici contenuti nella classe `org.junit.Assert` (è stato effettuato un import statico in testa alla classe) che effettuano una semplice comparazione tra il risultato atteso ed il risultato dell'esecuzione. In questo caso non si sta testando nulla di concreto, semplicemente utilizziamo gli `assertTrue` e `assertFalse` (messi a disposizione dalla classe sopra indicata) su dei valori statici.

Utilizzando Eclipse, si ha direttamente a disposizione un plugin con cui poter effettuare direttamente i test dalla console, con una visualizzazione grafica dei test andati a buon fine e dei test falliti.

Per la sezione server, essendo disponibile un RESTful Web Service (che utilizza come protocollo di trasmissione HTTP), è possibile testare la bontà delle classi e dei metodi realizzati tramite un browser web, inserendo come url il path al quale è disponibile una determinata risorsa aggiungendo eventuali parametri richiesti.

Per quanto riguarda la sezione client, i test sono stati effettuati provando direttamente l'applicazione o su emulatore oppure direttamente sul terminale Android e verificando, tramite il LogCat di Android, l'efficace esecuzione dell'applicazione. Analizzando il log dell'applicazione, è possibile intercettare eventuali errori ed eccezioni sollevate e non gestite e passare alla loro correzione.

6.2 . Test con terminali Android

Avendo a disposizione alcuni terminali con sistema operativo Android, si è pensato di realizzare dei test con degli utenti estranei al progetto per avere una valutazione sul sistema realizzato.

E' stato definito un protocollo sperimentale come segue:

- Generazione di un percorso da rispettare (punto di partenza, punto di arrivo ed alcuni punti intermedi da rispettare);
- Creazione di due gruppi di 3 persone ciascuno: il primo gruppo deve fare il percorso facendo affidamento solo sulla memoria, il secondo gruppo deve fare il percorso con l'ausilio del terminale Android;
- Valutazione del comportamento dei vari utenti.

Le locazioni scelte sono state Treviso e Genova. I percorsi sono stati strutturati in modo da non avere troppi punti dove soddisfare i Task lungo il percorso, in modo tale che fosse necessario effettuare delle deviazioni per portarli a termine.

I Task scelti sono (attività quotidiane):

- Comprare il latte
- Recarsi all'ufficio postale
- Comprare il pane
- Fare Benzina
- Andare a prenotare il ristorante per una cena in una particolare data
- Andare a teatro a prendere i biglietti per uno spettacolo
- Comprare l'olio motore
- Passare a vedere la vetrina di un determinato negozio
- Comprare il giornale
- Andare a visitare un determinato monumento

Per il soddisfacimento dei Task non sono stati imposti particolari vincoli di ordine o di tempo, solo una richiesta di rispettare, per quanto possibile, il percorso stabilito ed un tempo massimo di 3 ore. La necessità di soddisfare i task comporta inevitabilmente delle deviazioni, ammissibili ma con la richiesta di non discostarsi troppo da quanto stabilito.

I parametri deputati alla valutazione della bontà dell'applicazione si possono riassumere in:

- *Numero di task soddisfatti*: il risultato principale atteso dall'applicazione è quello di portare a termine un maggior numero di task rispetto alla situazione nella quale si fa uso solamente della memoria;
- *Numero di task soddisfatti in una deviazione*: la possibilità di fare delle deviazioni rispetto al percorso stabilito dovrebbe essere ottimizzata in modo tale che una deviazione serva a portare a termine più task possibili. Se tale situazione venisse a ricrearsi, sarà possibile ottimizzare le deviazioni in termini di tempo e spazio risparmiati (se un utente riesce a soddisfare più task a distanza ravvicinata, risparmia tempo e spazio percorso nel complessivo del percorso).

Si tratta comunque di test effettuati con degli utenti in stile “caccia al tesoro”, quindi non è possibile fare una valutazione strettamente oggettiva sui risultati. I test sono stati effettuati in un tempo piuttosto limitato ed in situazioni che non ricalcano effettivamente una situazione reale. In una realtà quotidiana un utente dovrebbe ricordare altri impegni oltre a quelli indicati che, chiaramente porterebbero inevitabilmente a ricordare meno task di quanti effettivamente preventivati. Per la conduzione di un test con dei risultati che possano avere una valutazione scientifica oggettiva, bisognerebbe impostarli in un periodo di tempo decisamente più lungo e con una serie di impegni da far coincidere con altri che si rendono necessari a livello lavorativo o personale. In questo caso la valutazione sarebbe decisamente più approfondita e reale, ma richiederebbe risorse e tempo decisamente superiori. Una possibile realizzazione potrebbe essere quella, dopo aver sistemato i problemi (soprattutto lato server) che tuttora si presentano, di distribuire l'applicazione ad un gruppo di test e raccogliere in questo modo delle statistiche sull'usabilità del sistema (anche in modo anonimo). Avendo a disposizione una serie di test molto più ampia, ma soprattutto in un utilizzo reale, si può avere a disposizione un campionario decisamente più ampio di valori sui quali trarre delle conclusioni.

Per quanto riguarda però i test effettuati, si può comunque estrapolarne dei dati significativi che portano ad una valutazione decisamente positiva del progetto che si può considerare ancora ad un livello prototipale.

I test sono stati condotti con un cellulare HTC Desire HD con le seguenti caratteristiche:

- Piattaforma: Android 2.2 Froyo;
- Processore: 1GHz;
- RAM: 768MB;
- Memoria interna: 1,5GB;
- Connessione di rete: 3G (fino a 14.4Mbps in download e 5.76Mbps in upload), GPRS, EDGE, WiFi;

Gestore telefonico per connessione a internet: Wind SpA.

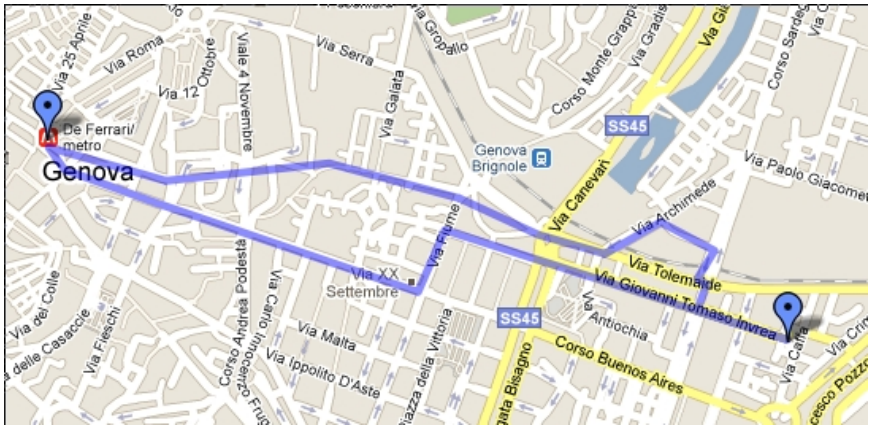
6.2.1 . Analisi dei risultati dei test

Nell'esecuzione dei test sono stati raccolti dei dati con i quali poter analizzare la bontà del sistema utilizzato. A tal proposito verrà effettuata una analisi comparativa dei dati raccolti con e senza l'ausilio del supporto informatico in esame.

Sono state scelte due locazioni per l'esecuzione dei test, Genova e Treviso. In entrambe le locazioni sono stati svolti i test sia nel caso di ausilio del sistema che senza. Si è scelto inoltre di non aggregare i dati registrati nelle due locazioni, poiché a fini statistici non sono confrontabili in quanto cambiano diverse variabili, quali ad esempio distanza complessiva e conformazione del percorso. Verranno pertanto analizzati i dati separatamente. Le conclusioni saranno però tratte da entrambe le esperienze, esplicitando punti a favore o meno dell'una e dell'altra.

6.2.2 . Test Genova

6.2.2.1 . Introduzione



In Figura 6.1 è riportata la mappa del percorso seguito a Genova. Tale mappa è utile per contestualizzare percorso e valutazioni fatte sui dati analizzati.

Nella scelta del percorso si è cercato di stabilire un buon compromesso tra la quantità dei punti di interesse rilevabili e la centralità delle vie scelte, il tutto per rendere il percorso il più vicino ad una possibile situazione reale. Da notare come il Task relativo alla visita del monumento scelto, la Casa di Colombo, pur non trovandosi sul percorso e comportando, per essere soddisfatto, una deviazione risulti essere abbastanza vicino da poter essere rilevato tramite il sistema.

6.2.2.2 . Analisi dei valori medi

Per ottenere dei grafici comparativi, sono state dapprima calcolate le medie per la distanza totale (riferita cioè al punto di partenza del percorso) e parziale (riferita alla distanza tra il soddisfacimento di due Task); successivamente sono stati confrontati i

dati derivanti dall'analisi del percorso effettuato con l'utilizzo del sistema e senza. Dalla Figura 6.2 (analisi medie sulle distanze totali) da una prima veloce analisi se ne ricava che, con l'utilizzo del sistema, si verifica una riduzione abbastanza marcata della distanza media per il soddisfacimento di un Task. La situazione è positiva in quanto effettivamente c'è un guadagno sia in termini di spazio percorso che, conseguentemente, di tempo impiegato per portare a termine gli impegni.

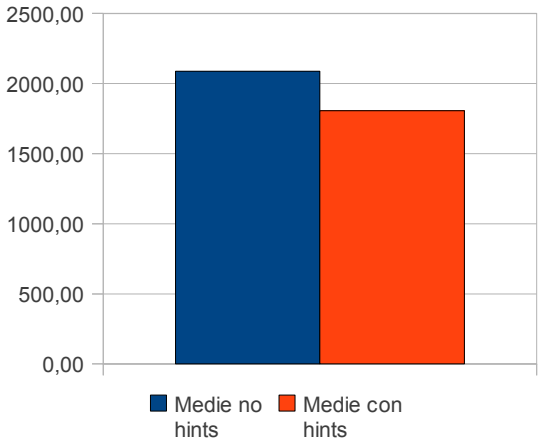


Figura 6.2: Media delle distanze totali medie per i Task

Se si analizzano, invece, le distanze medie parziali (in Figura 6.3), la situazione non cambia. Anche in questo caso si ottiene un riduzione delle distanze (e conseguentemente dei tempi). Questo sta a significare che in un raggio ristretto è possibile portare a termine più obiettivi. A supporto di questa conclusione vengono in seguito riportati altri grafici che analizzano gli inter-spazi (spazio tra due task soddisfatti) oppure la quantità di task che è possibile soddisfare entro un determinato raggio.

I grafici relativi ai tempi (siano essi totali o parziali) vengono omessi, in quanto è

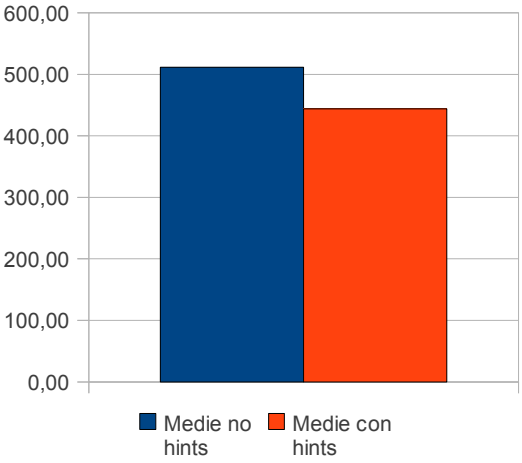
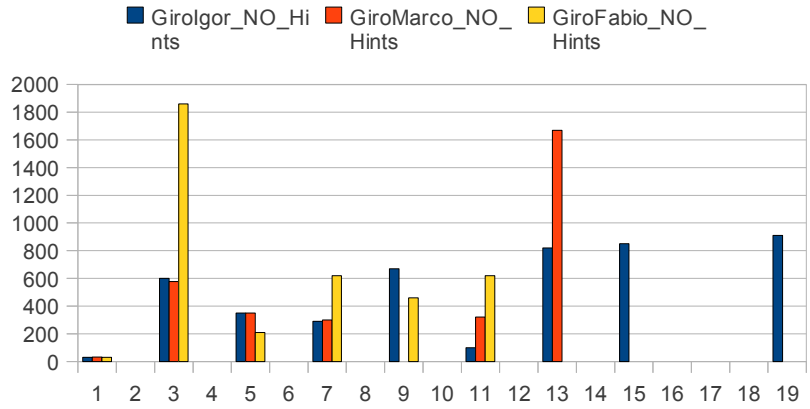


Figura 6.3: Media delle distanze parziali medie per i Task

conseguenza diretta che da una riduzione degli spazi corrisponda una proporzionale riduzione dei tempi.

6.2.2.3 . Analisi dei dati delle singole prove

Una analisi direttamente sui dati raccolti nei due test può portare ad una valutazione generale sul sistema ed inoltre ad una conferma delle valutazioni riportate nel paragrafo precedente. I due grafici presenti in Figura 6.4 e in Figura 6.5 mostrano le correlazioni tra le inter-distanze tra i vari task, suddivise per tipologia di test (con e senza l'ausilio del sistema in oggetto) e all'interno dei rispettivi grafici, divise per utente.



Dal confronto tra i due grafici, si nota palesemente che gli utenti che non potevano contare sul supporto del terminale mediamente hanno impiegato distanze maggiori per portare a termine i vari task. Si può facilmente notare che non tutti gli utenti hanno portato a termine lo stesso numero di task. Questo è dipeso dalla priorità che l'utente ha dato ad un determinato task e dalla sequenza scelta per portarli a termine (nel caso senza l'utilizzo del cellulare) oppure dalla quantità e tipologie di notifiche ricevute (nel caso di utilizzo del terminale).

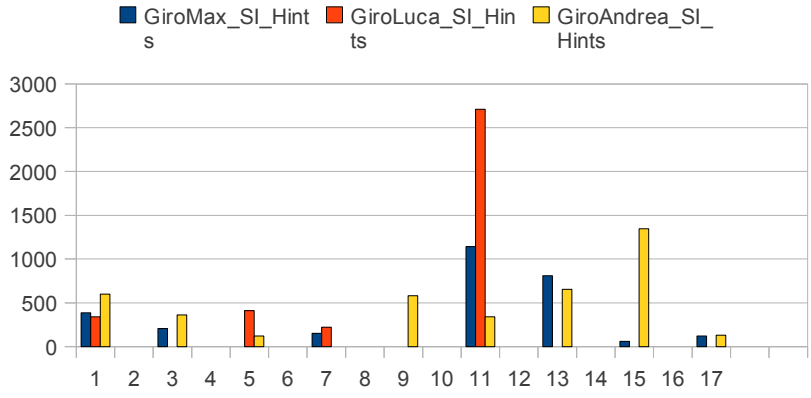


Figura 6.5: Hints - Andamento inter-distanze per ogni task

6.2.3 . Test Treviso

6.2.3.1 . Introduzione

La seconda location scelta per i test sul campo con il terminale Android è stata Treviso. Come caratteristiche il percorso differisce leggermente dal percorso scelto per Genova. La differenza risiede nel fatto che tale percorso è più circolare, quindi durante lo spostamento si verificano degli avvicinamenti alternati a degli allontanamenti rispetto ai punti di interesse. Tale caratteristica è stata appositamente scelta per valutare anche la qualità e quantità di reminder che vengono notificati dal sistema. Il test sotto questo punto di vista si può considerare pienamente riuscito, in quanto all'allontanarsi dai punti dove un task può essere soddisfatto la notifica non viene più fornita, al successivo riavvicinarsi la notifica ricompare come atteso.

Il percorso per i test in Treviso è riportato in Figura 6.6. Come indicato sopra per il percorso genovese, si è cercato di stabilire un buon compromesso tra quantità e posizione dei punti di interesse. Nello stabilire i parametri per il test e le modalità di svolgimento, si è cercato di riprodurre, per quanto possibile, una situazione reale, nella quale l'utente utilizzatore del sistema si trova ad avere una serie di impegni prefissati (ad esempio appuntamenti di lavoro) e quindi una sorta di percorso già prestabilito. Nell'espletare gli impegni già schedulati, il presente sistema tenta di ricordare altri impegni memorizzati, per i quali non è ad esempio stata fissata una precisa collocazione geografica e/o temporale. In questo modo il sistema cerca di massimizzare la redditività di una giornata, ottimizzando i tempi ricordando all'utilizzatore le attività che normalmente potrebbero venir dimenticate.

6.2.3.2 . Analisi dei valori medi

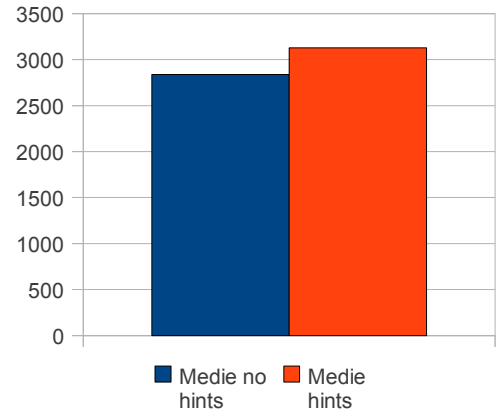


Figura 6.7: Media delle distanze totali medie per i Task

Mentre a Genova i dati ottenuti indicano chiaramente che vi è una riduzione dello spazio percorso e, di conseguenza, del tempo utilizzato, qui ci troviamo di fronte alla situazione opposta: la distanza media percorsa per task (riferita al punto di partenza) aumenta se si utilizza il sistema. Tale situazione non è da considerasi negativa, anzi potrebbe essere una ovvia interpretazione in base alla tipologia di percorso organizzato. Come indicato precedentemente, il percorso di Treviso risulta essere più circolare e quindi il dato sopra ottenuto è esplicativo del fatto che la distribuzione dei Task avviene in maniera più "ragionata". Hanno contribuito a tale risultato anche le preferenze impostate nel sistema, che prevedono un raggio di notifica dei task pari a 200mt e una distanza di aggiornamento hints pari a 250mt. Tali soglie, che si possono

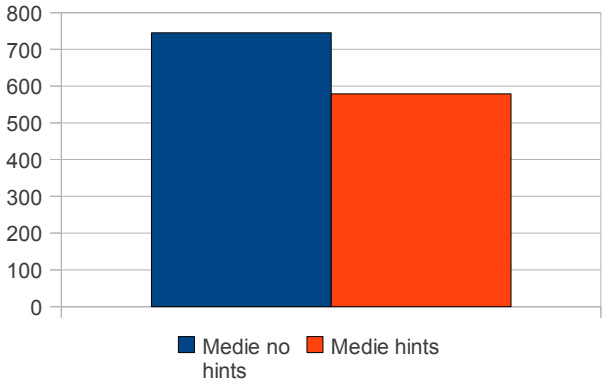


Figura 6.8: Media delle distanze parziali medie per i Task

considerare abbastanza limitate, fanno si che in un percorso come quello scelto per Treviso, l'utente abbia l'opportunità di scegliere i task da soddisfare nella minima distanza dalla propria posizione e quindi "rimandi" altri task a successive notifiche (presumibilmente più avanti nel percorso). Effettivamente questo è stato l'atteggiamento osservato. La tendenza degli utenti sprovvisti di terminale, è infatti quella di cercare di soddisfare inizialmente il maggior numero possibile di task, lasciando a momenti successivi solo quei task la cui locazione non è precisamente

conosciuta oppure che si riconosce da subito poter soddisfare in locazioni vicine alla parte finale del percorso. Tale approccio porta inevitabilmente al dimenticare alcuni task, come effettivamente è stato osservato. L'ausilio del sistema di supporto mnemonico permette dapprima di portare a termine, mediamente, più task e poi di minimizzare le deviazioni rispetto al percorso originale. Con l'utilizzo del sistema, le deviazioni sono inoltre mediamente più produttive (in termini di numero di task soddisfatti) rispetto alle deviazioni fatte senza. Da tutte queste considerazioni, se ne deduce una migliore distribuzione delle attività lungo tutto il percorso, cosa molto positiva perché porta, mediamente, ad un risparmio di spazio e tempo nelle attività quotidiane.

Per avvalorare ulteriormente questa tesi, sono stati calcolati degli ulteriori indicatori statistici per valutare se effettivamente la giustificazione data sia coerente con i dati raccolti. Dapprima si è cercato di valutare in che porzione di percorso sono stati portati a termine una certa quantità di task. Come è possibile notare da Tabella 6.2 e da Tabella 6.3, senza l'ausilio dell'applicazione, la concentrazione di task soddisfatti è maggiore nella prima parte del percorso (ad esempio il 70% dei task soddisfatti avviene nel primo 47% del percorso, contro il 57% con l'utilizzo del terminale). Si verifica pertanto un clustering di task nella parte iniziale del percorso, mentre la seconda parte viene sfruttata solamente per chiudere il tour.

Task	60%	70%	80%
Percorso	37,05%	47,25%	71,66%

Tabella 6.2: Treviso No Hints - Percentuale percorso su percentuale Task soddisfatti

Task	60%	70%	80%
Percorso	31,32%	57,51%	78,75%

Tabella 6.3: Treviso Hints - Percentuale percorso su percentuale Task soddisfatti

La situazione inversa avviene nel percorso di Genova come visibile da Tabella 6.4 e da Tabella 6.5. In questo caso, essendo il percorso più lineare, non c'è la possibilità di “rimandare” un task, quindi può essere soddisfatto solo quando effettivamente l'utente ci passa vicino.

Task	60%	70%	80%
Percorso	49,38%	73,23%	84,16%

Tabella 6.4: Genova No Hints - Percentuale percorso su percentuale Task soddisfatti

Task	60%	70%	80%
Percorso	37,06%	47,99%	81,05%

Tabella 6.5: Genova Hints - Percentuale percorso su percentuale Task soddisfatti

Una ulteriore conferma dei dati sopra indicati deriva dall'analisi di ulteriori indicatori statistici, quali ad esempio il campo di variabilità[21], la varianza[22] e la deviazione standard[23] sulle inter-distanze (le distanze tra due task soddisfatti) e sulle distanze totali (riferite al punto di partenza).

Per quanto riguarda lo scenario di Treviso i dati ottenuti sono riportati in Tabella 6.6 e in Tabella 6.7.

	Media	Campo di variabilità	Varianza	Deviazione standard
Parziali	758,38	1966,33	364324,72	594,04
Totali	2939,47	5549,33	3849771,51	1953,89

Tabella 6.6: Treviso No Hints - Media, Campo di Variabilità, Varianza, Deviazione standard per distanze totali e parziali dei task

	Media	Campo di variabilità	Varianza	Deviazione standard
Parziali	650,92	1390	224962,6	473,76
Totali	3394,52	5016	3397623,23	1842,43

Tabella 6.7: Treviso Hints - Media, Campo di Variabilità, Varianza, Deviazione standard per distanze totali e parziali dei task

Su tali indicatori si possono formulare le seguenti considerazioni:

- Campo di variabilità: tale indicatore risulta essere più alto nel caso senza hints, questo sta a significare che c'è una peggiore distribuzione dei task lungo tutto il percorso. Si può verificare che è possibile soddisfare un task molto vicino come uno molto lontano (riferito al punto di partenza). Visto che la situazione nel caso di utilizzo del terminale è diversa, significa che esiste effettivamente la possibilità di distribuire i task in modo più omogeneo nel percorso, evitando clustering;

21 Campo di variabilità: indicatore di dispersione pari alla differenza tra il valore massimo ed il valore minimo dei campioni della popolazione.

22 Varianza: indicatore di variabilità dei campioni. Indica quanto tali valori si discostano dalla media. Viene calcolato come media aritmetica del quadrato dello scarto di ogni campione rispetto alla media dei campioni.

23 Deviazione standard: o scarto quadratico medio, è un indice di dispersione dei valori dei campioni della popolazione. È strettamente legato alla varianza (il suo valore è la radice quadrata della varianza). Preferibile rispetto alla varianza in quanto ha la stessa unità di misura dei campioni.

- Varianza e deviazione standard: i valori calcolati per le prove senza hints e con rivelano che c'è una maggior dispersione dei campioni nel primo caso. Da questo si può dedurre che lo spostamento medio per soddisfare i task è maggiore senza hints.

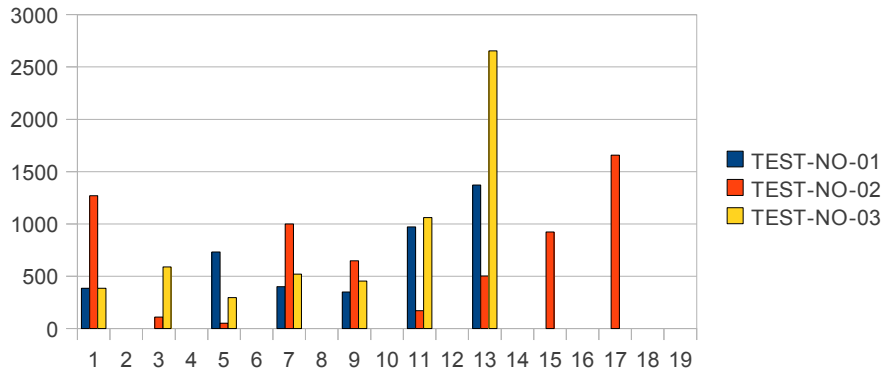
Da queste analisi, è possibile trarre le seguenti conclusioni:

- senza hints si hanno una serie di distanze abbastanza contenute nella prima parte del percorso, il resto del percorso è solo per ritornare al punto di partenza (dimenticando anche di portare a termine alcuni task). Questa situazione porta ad un abbassamento della media (tante distanze brevi ed una unica lunga);
- con hints si ha una maggiore distribuzione nel percorso dei task, perciò sono presenti distanze contenute come distanze più dilatate. Questa situazione porta ad un aumento della media (distanze brevi e distanze lunghe).

Come si vedrà anche dai grafici successivi, oltre che dalla lettura dei singoli dati, anche i test di Treviso evidenziano una riduzione delle inter-distanze utili ad eseguire i task, il tutto sempre per il fatto che in generale l'applicazione ricorda tutti i luoghi utili nel raggio specificato, cosa che in generale la memoria non riesce a fare.

Anche in questo caso, come per i test tenutisi a Genova, si nota una riduzione degli inter-spazi percorsi, perciò l'applicazione porta ad una maggiore probabilità di completare più task in uno spazio ridotto; di conseguenza, si ha una conseguente migliore gestione del tempo.

6.2.3.3 . Analisi dei dati delle singole prove



Le considerazioni che è possibile fare per questi grafici sono simili a quelle fatte per Genova. Da Figura 6.9 e da Figura 6.10 si nota che le inter-distanze si riducono, aumentando la capacità dell'utente di poter soddisfare più task in un raggio ristretto.

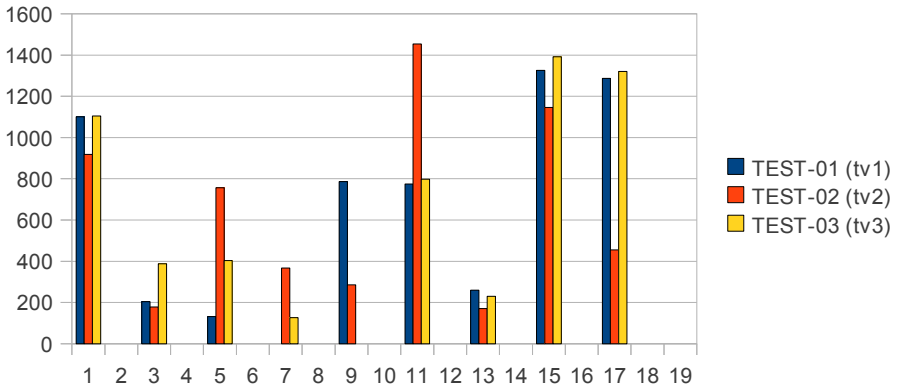
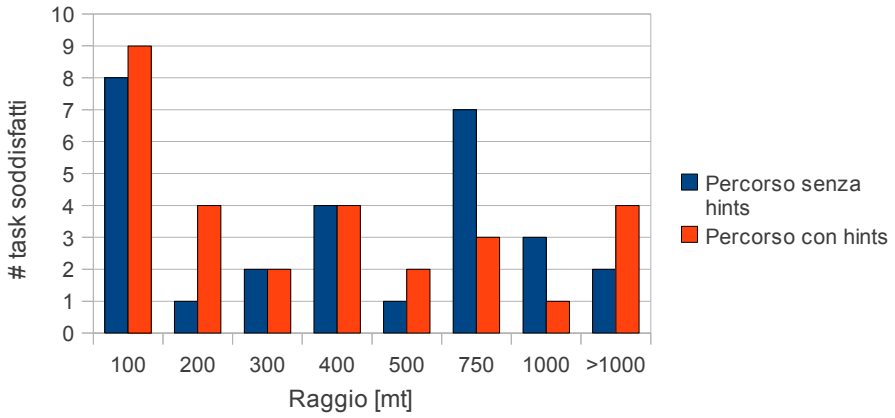


Figura 6.10: Hints - Andamento inter-distanze per ogni task

6.2.3.4 . Considerazioni finali

Oltre ad una analisi diretta delle inter-distanze (valori medi), si è passati ad una analisi, di tipo più qualitativo rispetto alla quantità di task servibili entro una determinata distanza. Questo permette di capire con che modalità i task vengono notificati. Si vorrebbe ottenere una situazione nella quale le inter-distanze siano ridotte al minimo, il che significherebbe che l'utente, con l'ausilio del terminale, riesce a portare a termine più task in uno spazio più limitato. Osservando i grafici in Figura 6.11 (riferita ai test di Genova) e in Figura 6.12 (riferita ai test di Treviso), è possibile notare che, mediante l'utilizzo del sistema, vengono soddisfatti più task in un raggio più limitato. La situazione è più marcata nel test di Treviso, ma anche in quello di Genova è possibile notare un variazione, seppur inferiore.



Tale osservazione porta a trarre sostanzialmente due conclusioni:

- indica che è possibile soddisfare un task effettuando mediamente una deviazione inferiore;
- considerando le distanze abbastanza corte, può verificarsi la situazione nella quale facendo una deviazione dal percorso stabilito, il sistema riesce a notificarmi un altro task che posso soddisfare a distanza ravvicinata.

Entrambe le considerazioni sopra indicate portano ad un soddisfacimento dei task più efficiente, in particolare ad una più efficiente gestione dello spazio (che chiaramente si riconduce ad una migliore gestione del tempo).

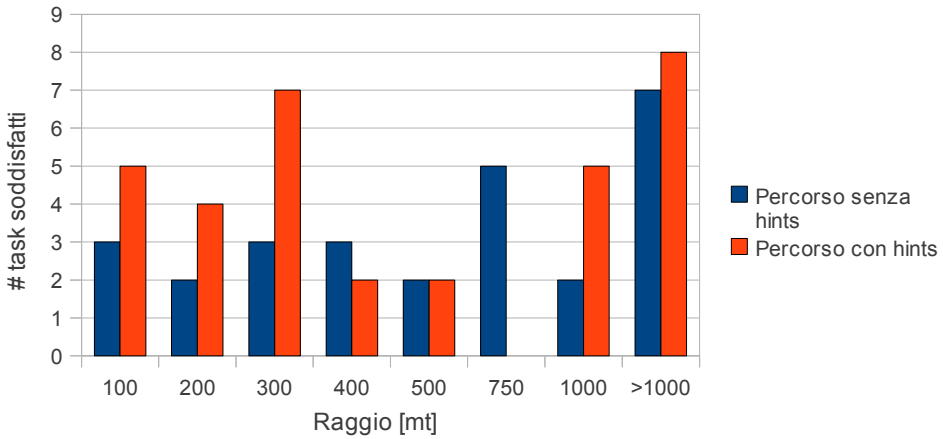


Figura 6.12: Treviso - Numero di task soddisfabili sul raggio

Concludendo, oltre alle considerazioni precedenti, con l'utilizzo del sistema si ha mediamente un aumento del 20% del numero di task ricordati. Per Treviso tale dato è diretto, mentre per Genova si sono verificati un caso particolarmente buono senza l'utilizzo del sistema e un caso particolarmente sfortunato con l'utilizzo del sistema dovuto a problemi imputabili ad un malfunzionamento del server; eliminando questi due casi limite, la percentuale risulta essere uguale.

Alla luce di quanto esposto, il sistema utilizzato può essere considerato effettivamente funzionante e può portare realmente un beneficio nell'utilizzo quotidiano per ricordare i vari impegni.

6.2.4 . Questionari di gradimento

Al fine di valutare il gradimento dell'applicazione da parte degli utenti che hanno svolto i test, è stato fatto compilare, alla fine di ogni test, un questionario per dar modo all'utente di esprimere dei giudizi qualitativi sull'esperienza con l'applicazione.

Oltre ad una valutazione oggettiva sui dati raccolti, è utile anche un feedback dell'utente sull'usabilità del sistema. Tale valutazione permette di scegliere le strategie di sviluppo future e di correggere eventuali implementazioni che potrebbero non

venire apprezzate.

6.2.4.1 . Struttura dei questionari

Il questionario è stato diviso in più sezioni, ognuna delle quali è stata creata per avere le impressioni dell'utente su un diverso aspetto del progetto, nella fattispecie le sezioni create sono:

- Generale: in questa sezione vengono poste alcune domande sull'esperienza d'uso in generale, chiedendo se l'utente ritenga utile l'applicazione;
- Notifiche: dato che sono possibili più modalità di notifica (vibrazione, audio, visuale, vocale) per ognuna di esse viene chiesto di dare un voto relativo all'utilità. Questa sezione è piuttosto articolata in quanto le notifiche sono uno dei punti centrali del progetto, quindi, oltre a questo viene poi chiesto quale sia il grado di congruenza e qualità delle notifiche rispetto al soddisfacimento dei task;
- Funzionalità: oltre a chiedere informazioni circa eventuali malfunzionamenti, viene chiesto quale sia il punto di vista a proposito della possibilità di integrare nel sistema usato delle ulteriori funzionalità relative alla condivisione di punti di interesse con altri utenti e all'utilizzo collaborativo del software;
- Conclusioni: in seguito all'esperienza avuta, si richiede una valutazione globale e una ipotetica disponibilità all'acquisto del sistema.

6.2.4.2 . Valutazione dei questionari

I risultati ottenuti dai questionari compilati dagli utenti di Genova e Treviso sono pressoché simili, a meno di qualche valutazione diversa tra le due locazioni; questo atteggiamento sta a significare che l'esperienza avuta dagli utenti può essere considerata positiva e che, pur trattandosi di una versione ancora prototipale, alla quale bisogna certamente apportare delle modifiche in fatto di stabilità (sia lato client che server) per prevenire qualsiasi tipo di problematica, l'applicazione è stata considerata potenzialmente utile.

In Appendice D sono riportati copia del questionario dato agli utenti e tutte le risposte.

Si passa ora ad una valutazione delle varie sezioni dei questionari, per capire quali siano stati i punti di forza e i punti dolenti del sistema (aspetti nei quali focalizzare future modifiche).

Generale

È emersa una valutazione più che buona dell'applicazione, gli utenti la reputano utile e piuttosto ben sviluppata.

Notifiche

La parte certamente più interessante della valutazione è quella delle notifiche. Essendo un punto cruciale (le notifiche sono il modo per richiamare l'attenzione dell'utente sulla possibilità di compiere qualche azione), l'interazione e l'utilizzo da parte dell'utente dev'essere preso in seria considerazione. La notifica che è stata decisamente preferita è la vibrazione. Meno apprezzate le notifiche audio, visuali e vocali, presumibilmente perché il terminale durante il percorso è stato prevalentemente tenuto in tasca, pertanto tali notifiche risultano meno fruibili.

Per quanto riguarda la congruenza delle notifiche, grossa importanza riveste la tipologia di registrazione che ha effettuato l'attività commerciale sul servizio mappe. In particolare, essendo supportato solamente il database Google Maps, dipende dalle parole chiave che l'attività commerciale utilizza per la propria identificazione. Possono pertanto verificarsi delle incongruenze, in quanto una parola chiave può essere a volte utilizzata per categorie merceologiche differenti e non attinenti. Ciò capita spesso quando si utilizzano termini generali, quali ad esempio supermercato. Tale parola chiave può identificare svariate tipologie di prodotti che non necessariamente vengono vendute in un medesimo posto. Tuttavia la qualità delle notifiche in fatto di possibilità o meno di svolgere una determinata attività nel posto selezionato, è risultata essere piuttosto alta. Un punto a favore è anche il fatto che vengono notificati dei punti di interesse non particolarmente conosciuti, così che l'utente, anche non conoscendo tali posti, possa comunque soddisfare le proprie esigenze. Tale prerogativa è tanto più marcata quanto meno l'utente conosce il posto in cui si trova. Sotto questo aspetto, l'applicazione potrebbe essere vista anche come una sorta di navigatore che porta al compimento di particolari necessità.

La presenza della mappa sulla quale poter posizionare i punti di interesse trovati, si è resa quasi indispensabile per un utilizzo produttivo. Senza poter localizzare le notifiche è estremamente difficile, anche per l'utente che conosce il posto, trovare il punto esatto dove poter soddisfare un Task. Tutti gli utenti hanno infatti reputato estremamente utile la presenza della mappa e la possibilità di vedere posizionate tutte le notifiche del sistema.

Funzionalità

Il fine di questa sezione, oltre ad una verifica sul funzionamento dell'applicazione, è principalmente quello di saggiare quali siano le possibili nuove implementazioni che l'utente potrebbe gradire e poter indirizzare i futuri sviluppi in un verso piuttosto che in un altro. Sono state proposte delle possibili funzionalità aggiuntive del software, richiedendone una valutazione di gradimento:

- possibilità di aggiungere ai punti di interesse memorizzati in servizi esterni

(ad esempio Google Maps) anche punti di interesse personali;
- possibilità di condividere i proprio punti di interesse con gli altri utenti, un utilizzo “social” dell'applicazione basato sulla condivisione delle esperienze;
- possibilità di utilizzo di gruppo del sistema, il poter pertanto inserire dei Task che possano essere visualizzati anche da altri utenti appartenenti ad un gruppo.

Queste nuove funzionalità sono state viste come nuove possibilità, ma anche con un po' di diffidenza. Tale comportamento presumibilmente è legato al fatto che non sempre si raggiunge l'aspettativa dell'utente senza dargli la possibilità di provare personalmente.

Conclusioni

La media dei voti registrati è abbastanza alta, quindi la valutazione complessiva dell'utente è piuttosto buona. Una eventuale commercializzazione non ne riduce l'attrattiva, ulteriore punto a favore nei confronti dell'applicazione sperimentata.

Capitolo 7 . Conclusioni

Trattandosi di un refactoring di una versione già esistente dell'applicazione, si è cercato di correggere, per quanto possibile, eventuali errori causati da parti di codice errate o che si comportavano in modo anomalo. La versione del software di partenza si poteva considerare non più che un prototipo e, in quanto tale, soffriva di errori e incongruenze che lo portavano a non poter essere utilizzato correttamente (o come il sistema promette di funzionare).

La versione attuale può invece essere considerata una beta funzionante in tutte le sue parti. I test effettuati e l'impressione dichiarata dagli utenti che l'hanno provata, portano certamente a delle conclusioni positive in merito. In particolare, i dati raccolti evidenziano che un utilizzo quotidiano dell'applicazione potrebbe portare a dei benefici principalmente sotto due aspetti:

- numero e qualità degli impegni ricordati;
- qualità del percorso (tempo speso per il soddisfacimento degli impegni al di fuori delle attività programmate).

Come indicato, pur essendo una versione beta del sistema, ha dimostrato una certa stabilità nell'utilizzo. Questo tuttavia non esclude, oggettivamente, dei necessari aggiustamenti in particolare lato server. I file di log registrati, durante l'esecuzione dei test, sono stati fondamentali per definire i punti critici del sistema, e hanno dimostrato, mediamente, un funzionamento aderente a quello atteso. Tuttavia, purtroppo, si sono verificati degli errori che non hanno però pregiudicato il funzionamento complessivo in quanto non hanno portato all'arresto dell'applicazione lato server.

Complessivamente le soluzioni implementate hanno apportato un valore aggiunto non indifferente e le funzionalità ora supportate sono sicuramente un sostanzioso passo avanti per la struttura globale del sistema. Tali soluzioni aumentano la stabilità (in primo luogo con il cambiamento del DBMS, secondariamente con la correzione di

molti bug e funzionalità) e l'usabilità da parte dell'utente (ad esempio con l'introduzione di notifiche di diversa natura come vibrazione e sonore) la possibilità di registrazione direttamente dal terminale e l'aggiunta nella mappa dei markers ad identificare i luoghi dove i Task possono venire soddisfatti. Da non sottovalutare inoltre tutti i feedback, in termini di messaggi sul display, creati per l'utente, che permettono di avere una chiara idea di cosa il sistema stia facendo. Sono state implementate inoltre funzionalità che permettono una drastica riduzione degli errori prevedibili sul funzionamento, adottando, ad esempio, procedure per il test della connessione, della versione e della disponibilità di un sistema di posizionamento geografico.

Come inoltre evidenziato dai test effettuati e dai questionari compilati, gli utenti ritengono l'applicazione utile e ben fatta, manifestando anche la propensione ad un acquisto. Questo non esclude pertanto una possibile futura commercializzazione.

Appendice A

Di seguito riportati diagrammi UML di sequenza per:

- Creazione nuovo Task;
- Location query (ricerca Hints);
- Registrazione nuovo utente.

Diagramma di sequenza creazione nuovo Task

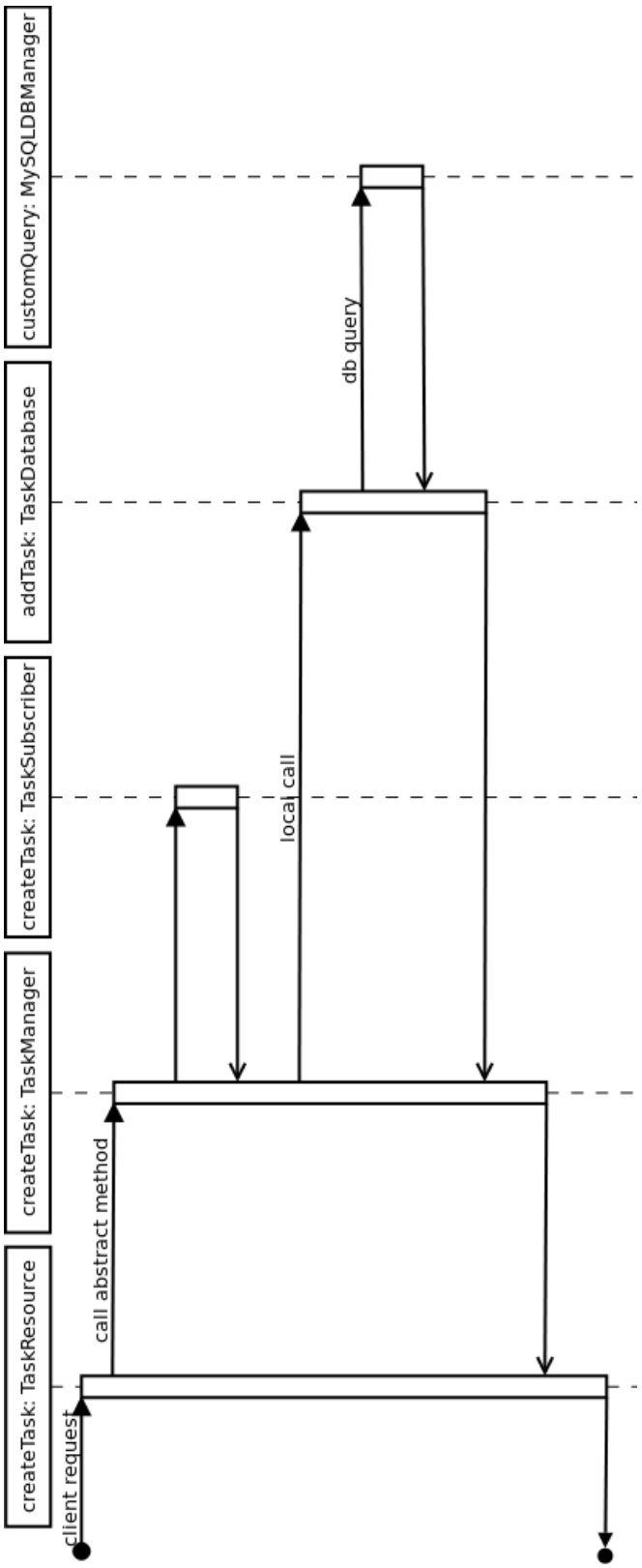


Diagramma di sequenza Location Query (ricerca Hints)

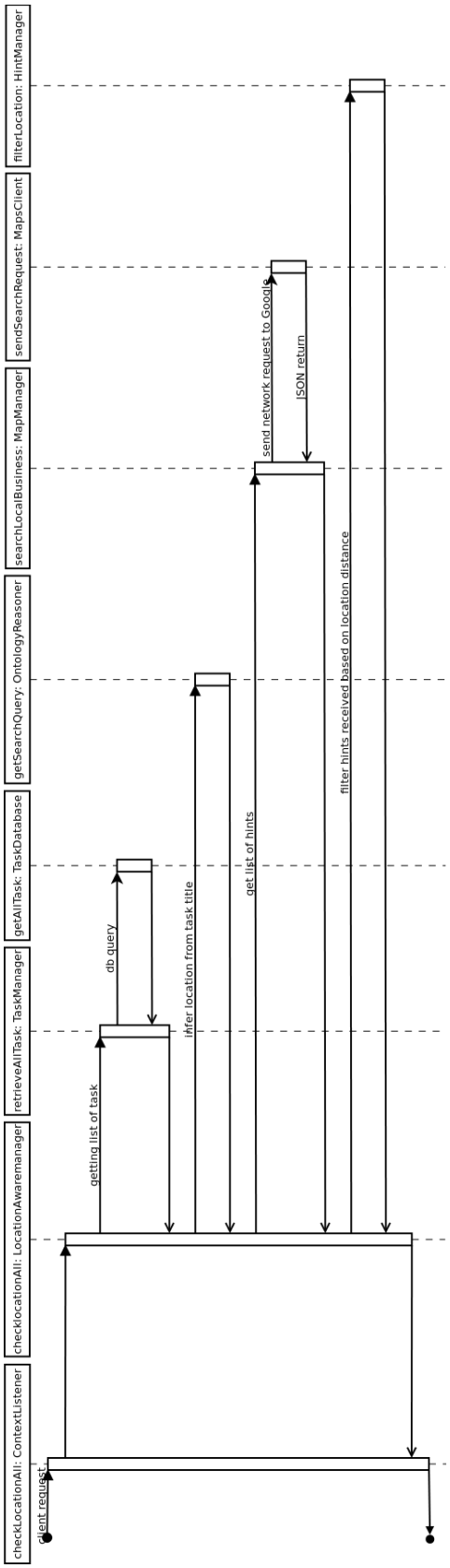
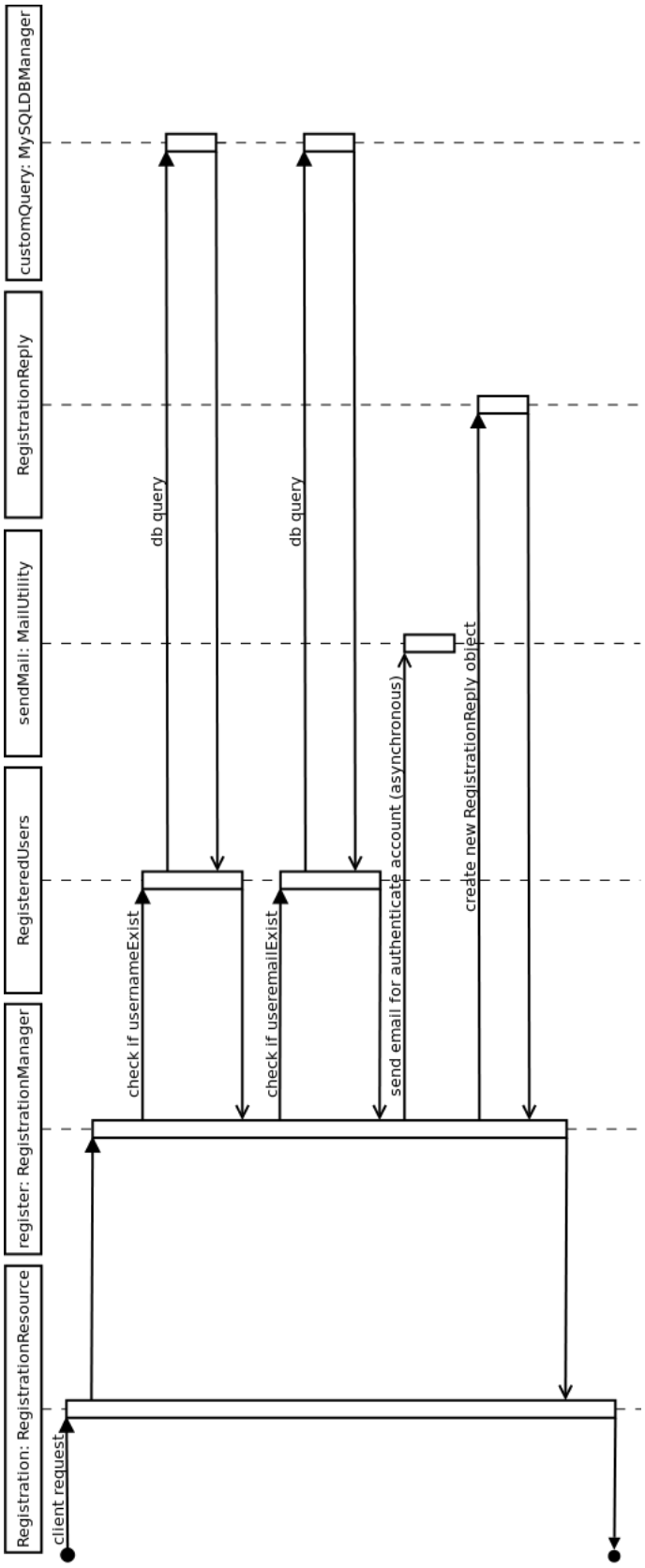
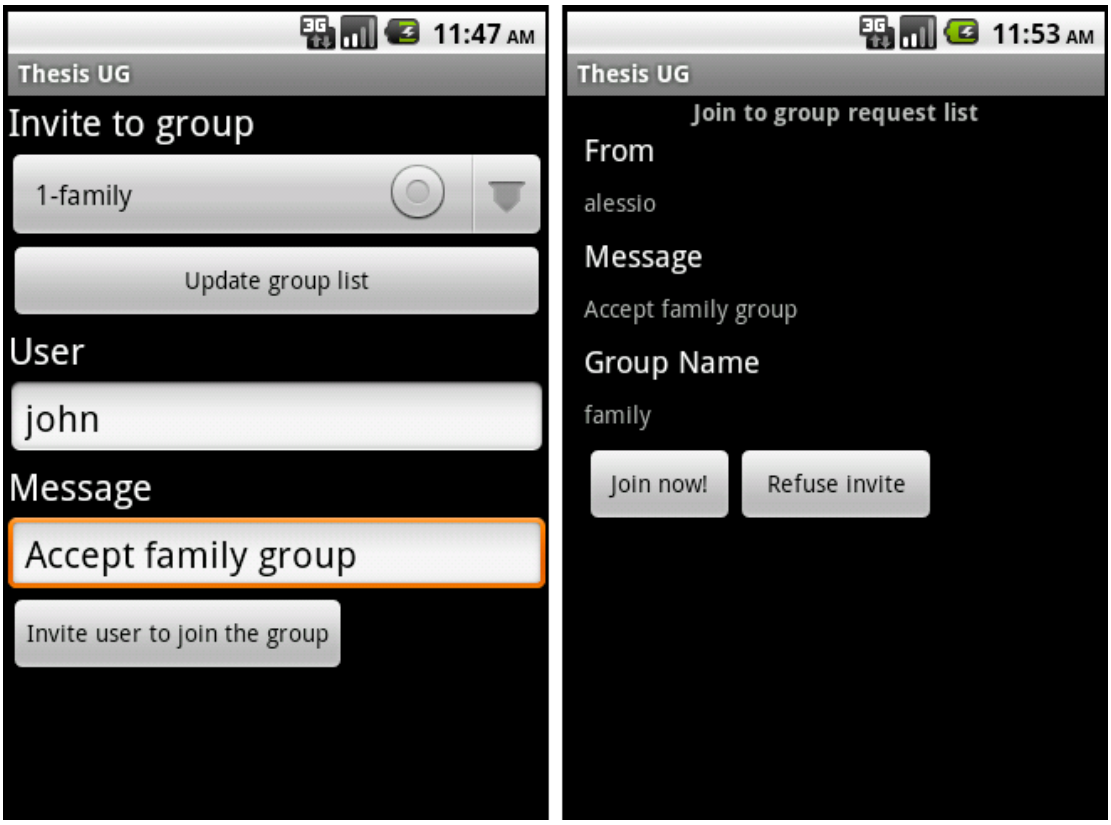


Diagramma di sequenza Registrazione nuovo utente



Appendice B


11:47 AM
Thesis UG
Invite to group
1-family
Update group list
User
john
Message
Accept family group
Invite user to join the group
11:53 AM
Thesis UG
Join to group request list
From
alessio
Message
Accept family group
Group Name
family
Join now!
Refuse invite

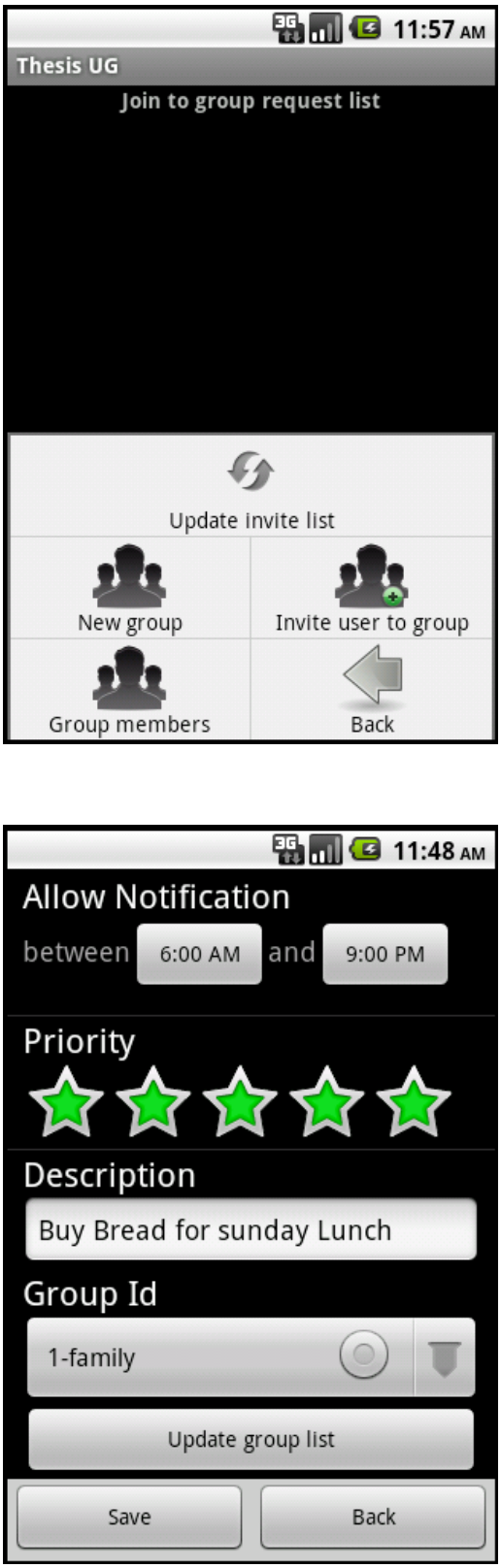


Figura 7.3: Gruppi: Campo Group Id nella schermata Add Task

Appendice C

Genova – percorso 1 – Hints

GiroLuca_SI_Hints		Anni	23								
Inizio		Olio		Benzina		Teatro		Giornale		Poste	
21.10		21.22		21.23		21.37		21.43		21.43	
	340		0		410		220		0		2710
	340		340		750		970		970		3680
	00.12.00		00.01.00		00.14.00		00.06.00		00.00.00		00.01.00
	00.12		00.13		00.27		00.33		00.33		00.34
T. Totale			01.45	6300	sec			Task Totali			10
T. Medio per Task su parzial			00.15								
T. Medio per Task su totale			00.36					Task soddisfatti			7
Dist. Totale			4140	mt				Percentuale			70%
Dist. Media per Task			1859	mt							
V. Media			0,65714	m/sec							

Latte		Pane		Ristorante		Negozio		Monumento		Fine
21.44		21.45		X		X		X		22.55
	0		-		-		-		460	
	3680		-		-		-		4140	
	00.01.00		-		-		-		01.10.00	
	00.35		-		-		-		01.45	

Genova – percorso 2 – Hints

GiroAndrea_SI_Hints		Anni	25								
Inizio		Teatro		Poste		Pane		Latte		Negozio	
21.05		21.10		21.13.00		21.15		21.16		21.23	
	600		360		120		0		580		340
	600		960		1080		1080		1660		2000
	00.05.00		00.03.00		00.02.00		00.01.00		00.07.00		00.09.00
	00.05		00.08		00.10		00.11		00.18		00.27

T. Totale	01.10	4200	sec	Task Totali 10
T. Medio per Task su parzial	00.07			
T. Medio per Task su totale	00.32			Task soddisfatti 10
Dist. Totale	4145	mt		Percentuale 100%
Dist. Media per Task	2404	mt		
V. Media	0,98690	m/sec		

Monumento		Ristorante		Giornale		Olio		Benzina		Fine
21.32		21.54		21.55		21.59		21.59		22.15
	654		1346		130		0	0	15	
	2654		4000		4130		4130		4145	
	00.22.00		00.01.00		00.04.00		00.00.00		00.16.00	
	00.49		00.50		00.54		00.54		01.10	

Genova – percorso 3 – Hints

Girolgor_NO_Hints		Anni	29								
Inizio		Giornale		Teatro		Poste		Pane		Monumento	
15.50		15.52		16.00		16.08		16.17		16.20	
	30		600		350		290		670		100
	30		630		980		1270		1940		2040
	00.02.00		00.08.00		00.08.00		00.09.00		00.03.00		00.12.00
	00.02		00.10		00.18		00.27		00.30		00.42

T. Totale	01.25	5100	sec	Task Totali 10
T. Medio per Task su parzial	00.08			
T. Medio per Task su totale	00.45			Task soddisfatti 10
Dist. Totale	4630	mt		Percentuale 100%
Dist. Media per Task	2402	mt		
V. Media	0,90784	m/sec		

Negozio		Ristorante		Olio		Benzina		Latte		Fine
16.32		16.48		17.00		17.00		17.14		17.15
	820		850		0		910	0	10	
	2860		3710		3710		4620		4630	
	00.16.00		00.12.00		00.00.00		00.14.00		00.01.00	
	00.58		01.10		01.10		01.24		01.25	

Genova – percorso 1 – No Hints

Girolgor_NO_Hints		Anni	29								
Inizio		Giornale		Teatro		Poste		Pane		Monumento	
15.50		15.52		16.00		16.08		16.17		16.20	
	30		600		350		290		670		100
	30		630		980		1270		1940		2040
	00.02.00		00.08.00		00.08.00		00.09.00		00.03.00		00.12.00
	00.02		00.10		00.18		00.27		00.30		00.42
T. Totale			01.25		5100	sec		Task Totali			10
T. Medio per Task su parzial			00.08								
T. Medio per Task su totale			00.45					Task soddisfatti			10
Dist. Totale			4630		mt			Percentuale			100%
Dist. Media per Task			2402		mt						
V. Media			0,90784		m/sec						

Negozio		Ristorante		Olio		Benzina		Latte		Fine
16.32		16.48		17.00		17.00		17.14		17.15
	820		850		0		910		10	
	2860		3710		3710		4620		4630	
	00.16.00		00.12.00		00.00.00		00.14.00		00.01.00	
	00.58		01.10		01.10		01.24		01.25	

Genova – percorso 2 – No Hints

GiroMarco_NO_Hints		Anni	22								
Inizio		Giornale		Teatro		Poste		Pane		Latte	
14.32		14.35		14.55		14.58		15.05		15.15	
	32		578		350		300		0		320
	32		610		960		1260		1260		1580
	00.03.00		00.20.00		00.03.00		00.07.00		00.10.00		00.10.00
	00.03		00.23		00.26		00.33		00.43		00.53
T. Totale			01.43		6180	sec		Task Totali			10
T. Medio per Task su parzial			00.11								
T. Medio per Task su totale			00.43					Task soddisfatti			8
Dist. Totale			3870		mt			Percentuale			80%
Dist. Media per Task			1786		mt						
V. Media			0,62621		m/sec						

Negozio		Olio		Benzina		Ristorante		Monumento		Fine
15.25		15.55		15.55		X		X		16.15
	1670		0		-		-		620	
	3250		3250		-		-		3870	
	00.30.00		00.00.00		-		-		00.20.00	
	01.23		01.23		-		-		01.43	

Genova – percorso 3 – No Hints

GiroFabio_NO_Hints		Anni	23								
Inizio		Giornale		**Negozio**		Teatro		Ristorante		Benzina	
18.10		18.13		18.43		18.51		19.04		19.21	
	30		1860		210		620		460		620
	30		1890		2100		2720		3180		3800
	00.03.00		00.30.00		00.08.00		00.13.00		00.17.00		00.10.00
	00.03		00.33		00.41		00.54		01.11		01.21

T. Totale	01.50	6600	sec	Task Totali 10
T. Medio per Task su parzial	00.13			
T. Medio per Task su totale	01.00			Task soddisfatti 7
Dist. Totale	4300	mt		Percentuale 70%
Dist. Media per Task	2728	mt		
V. Media	0,65152	m/sec		

Latte		Pane		Poste		Olio		Monumento		Fine
19.31		19.40		X		X		X		20.00
	0		-		-		-		500	
	3800		-		-		-		4300	
	00.09.00		-		-		-		00.20.00	
	01.30		-		-		-		01.50	

Genova - Confronto fra distanze e tempi medi

Dist. Media per task senza hints	2305	mt	
T. Medio per Task senza hints	00.47.15		
Dist. Totale media senza hints	4267	mt	
T. Totale medio senza hints	01.39.20		
Dist. Media per task con hints	2021	mt	-12,32%
T. Medio per Task con hints	00.33.29		-29,14%
Dist. Totale media con hints	4348	mt	1,90%
T. Totale medio con hints	01.26.00		-13,42%

Treviso – percorso 1 – Hints

TEST-01 (tv1)

Inizio	Poste	Teatro	Pane	Latte	Giornale
18.30	18.49	18.53	18.56	18.56	19.09
1100	204	131	0	786	775
1100	1304	1435	1435	2221	2996
00.19	00.04	00.03	00.00	00.13	00.11
00.19	00.23	00.26	00.26	00.39	00.50

T. Totale	01.30	5400	sec	Task Totali 10
T. Medio per Task	00.10			Task soddisfatti 10
Dist. Totale	6162	mt		Percentuale 100%
Dist. Media per Task	685	mt		
V. Media	1,14111	m/sec		

Monum.	Negozio	Rist.	Olio	Benzina	Fine
19.20	19.26	19.42	19.57	19.57	20.00
260	1325	1287	0	294	
3256	4581	5868	5868	6162	
00.06	00.16	00.15	00.00	00.03	
00.56	01.12	01.27	01.27	01.30	

Treviso – percorso 2 – Hints

TEST-02 (tv2)

Inizio	Teatro	Pane	Giornale	Latte	Monum.
13.24	13.37	13.40	13.49	13.54	13.57
919	178	756	367	286	1454
919	1097	1853	2220	2506	3960
00.13	00.03	00.09	00.05	00.03	00.20
00.13	00.16	00.25	00.30	00.33	00.53

T. Totale	01.16	4560	sec	Task Totali 10
T. Medio per Task	00.07			Task soddisfatti 10
Dist. Totale	5962	mt		Percentuale 100%
Dist. Media per Task	596	mt		
V. Media	1,30746	m/sec		

Negozio	Rist.	Poste	Olio	Benzina	Fine
14.17	14.19	14.31	14.35	14.35	14.40
170	1146	455	0	231	
4130	5276	5731	5731	5962	
00.02	00.12	00.04	00.00	00.05	
00.55	01.07	01.11	01.11	01.16	

Treviso – percorso 3 – Hints

TEST-03 (tv3)

Inizio	Poste	Giornale	Teatro	Latte	Pane	
21.41	22.00	22.05	22.09	22.11	22.11	
1104	388	403	126	0		798
1104	1492	1895	2021	2021		2819
00.19	00.05	00.04	00.02	00.00		00.11
00.19	00.24	00.28	00.30	00.30		00.41

T. Totale	01.19	4740 sec	Task Totali	10
T. Medio per Task	00.08		Task soddisfatti	10
Dist. Totale	6047	mt	Percentuale	100%
Dist. Media per Task	672	mt		
V. Media	1,27574	m/sec		

Monum.	Negozio	Rist.	Olio	Benzina	Fine
22.22	22.25	22.42	22.57	22.57	23.00
230	1391	1320	0	287	
3049	4440	5760	5760	6047	
00.03	00.17	00.15	00.00	00.03	
00.44	01.01	01.16	01.16	01.19	

Treviso – percorso 1 – No Hints

TEST-NO-01

Inizio	Latte	Pane	Poste	Giornale	Teatro	
08.55	08.58	08.58	09.07	09.15	09.23	
385	0	732	401	350		971
385	385	1117	1518	1868		2839
00.03	00.00	00.09	00.08	00.08		00.18
00.03	00.03	00.12	00.20	00.28		00.46

T. Totale	01.40	6000 sec	Task Totali	10
T. Medio per Task	00.14		Task soddisfatti	7
Dist. Totale	5848	mt	Percentuale	70%
Dist. Media per Task	835	mt		
V. Media	0,97467	m/sec		

Negozio	Rist.	Monum.	Benzina	Olio	Fine
09.41	10.06	X	X	X	10.35
1371	-	-	-	1638	
4210	-	-	-	5848	
00.25	-	-	-	00.29	
01.11	-	-	-	01.40	

Treviso – percorso 2 – No Hints

TEST-NO-02						
Inizio	Teatro	Giornale	Poste	Latte	Monum.	
14.50	15.06	15.09	15.12	15.25	15.35	
1271	111	50	1000	649	171	
1271	1382	1432	2432	3081	3252	
00.16	00.03	00.03	00.13	00.10	00.05	
00.16	00.19	00.22	00.35	00.45	00.50	

T. Totale	01.45	6300 sec	Task Totali	10
T. Medio per Task	00.10		Task soddisfatti	9
Dist. Totale	6621	mt	Percentuale	90%
Dist. Media per Task	662	mt		
V. Media	1,05095	m/sec		

Rist.	Pane	Olio	Benzina	Negozio	Fine
15.40	15.50	16.05	16.30	X	16.35
501	924	1657	-	287	
3753	4677	6334	-	6621	
00.10	00.15	00.25	-	00.05	
01.00	01.15	01.40	-	01.45	

Treviso – percorso 3 – No Hints

TEST-NO-03						
Inizio	Latte	Pane	Poste	Giornale	Teatro	
09.45	09.50	09.59	10.05	10.13	10.22	
386	590	296	519	453	1061	
386	976	1272	1791	2244	3305	
00.05	00.09	00.06	00.08	00.09	00.15	
00.05	00.14	00.20	00.28	00.37	00.52	

T. Totale	01.32	5520 sec	Task Totali	10
T. Medio per Task	00.11		Task soddisfatti	8
Dist. Totale	6221	mt	Percentuale	80%
Dist. Media per Task	778	mt		
V. Media	1,12699	m/sec		

Rist.	Olio	Benzina	Negozio	Monum.	Fine
10.37	11.10	11.10	X	X	11.17
2654	0	-	-	262	
5959	5959	-	-	6221	
00.33	00.00	-	-	00.07	
01.25	01.25	-	-	01.32	

Treviso - Confronto fra distanze e tempi medi

Dist. Media per task senza hints	758	mt	
T. Medio per Task senza hints	00.12.06		
Dist. Totale media senza hints	6230	mt	
T. Totale medio senza hints	01.39.00		
Dist. Media per task con hints	651	mt	-14,17%
T. Medio per Task con hints	00.08.48		-27,31%
Dist. Totale media con hints	6057	mt	-2,78%
T. Totale medio con hints	01.21.40		-17,51%

Appendice D

Risposte questionari

	GENOVA			
Domanda	ge1	ge2	ge2	Media
1	8	9	8	8,33
2	9	8	9	8,67
3	7	10	6	7,67
4	Tasca	Tasca	Tasca	Tasca
Visuali	7	9	8	8,00
Vibrazione	8	10	10	9,33
Audio	7	4	5	5,33
Vocali	8	5	5	6,00
6	8	10	9	9,00
7	Più no che si	In alcuni casi si in altri no	Più si che no	In alcuni casi si in altri no
8	3	2	3	2,67
9	1	1	1	1,00
10	1	1	1	1,00
11	0	1	1	0,67
12	10	3	9	7,33
13	10	9	8	9,00
14	7	2	8	5,67
15	8	9	7	8,00
16	1	1	1	1,00
17	2 – 4	5 – 10	<1	2,33 – 5

	TREVISO			
Domanda	tv1	tv2	tv3	Media
1	8	8	7	7,67
2	8	10	8	8,67
3	7	7	7	7,00
4	tasca	tasca	tasca	
Visuali	6	8	1	5,00
Vibrazione	9	10	9	9,33
Audio	9	9	9	9,00
Vocali	6	6	1	4,33
6	8	9	9	8,67
7	più no che si	no	più no che si	
8	2	2	2	2,00
9	1	1	1	1,00
10	1	1	1	1,00
11	0	1	1	0,67
12	10	9	9	9,33
13	6	9	5	6,67
14	10	7	8	8,33
15	8	9	6	7,67
16	1	1	1	1,00
17	2 – 4	5 – 10	5 – 10	4 – 8

Questionario

Questionario di gradimento

Generale

1) Come giudichi l'esperienza di utilizzo dell'applicazione? (1: pessima – 10: molto buona)
1 ❑ . 2 ❑ . 3 ❑ . 4 ❑ . 5 ❑ . 6 ❑ . 7 ❑ . 8 ❑ . 9 ❑ . 10 ❑

2) Ti sembra un'applicazione utile? (1: poco o per niente – 10: molto)
1 ❑ . 2 ❑ . 3 ❑ . 4 ❑ . 5 ❑ . 6 ❑ . 7 ❑ . 8 ❑ . 9 ❑ . 10 ❑

3) L'interfaccia è semplice da utilizzare? (1: poco o per niente – 10: molto)
1 ❑ . 2 ❑ . 3 ❑ . 4 ❑ . 5 ❑ . 6 ❑ . 7 ❑ . 8 ❑ . 9 ❑ . 10 ❑

4) Dove hai tenuto il cellulare durante il percorso?
Prevalentemente in tasca ❑ . Prevalentemente in mano ❑

Notifiche

5) Ti sono servite le notifiche date dal programma? (1: poco o per niente – 10: molto)
Visuali (notifica in alto sullo schermo):
1 ❑ . 2 ❑ . 3 ❑ . 4 ❑ . 5 ❑ . 6 ❑ . 7 ❑ . 8 ❑ . 9 ❑ . 10 ❑
Vibrazione:
1 ❑ . 2 ❑ . 3 ❑ . 4 ❑ . 5 ❑ . 6 ❑ . 7 ❑ . 8 ❑ . 9 ❑ . 10 ❑
Audio (notifiche con suoneria, tipo sms):
1 ❑ . 2 ❑ . 3 ❑ . 4 ❑ . 5 ❑ . 6 ❑ . 7 ❑ . 8 ❑ . 9 ❑ . 10 ❑
Vocali (notifiche parlare, il sistema legge la notifica):
1 ❑ . 2 ❑ . 3 ❑ . 4 ❑ . 5 ❑ . 6 ❑ . 7 ❑ . 8 ❑ . 9 ❑ . 10 ❑

6) Le notifiche ricevute sono congruenti al soddisfacimento dei task? (1: poco o per niente – 10: molto)
1 ❑ . 2 ❑ . 3 ❑ . 4 ❑ . 5 ❑ . 6 ❑ . 7 ❑ . 8 ❑ . 9 ❑ . 10 ❑

7) Saresti andato/a negli stessi posti indicati dall'applicazione?
No ❑ . Più no che si ❑ . In alcuni casi si in altri no ❑ . Più si che no ❑ . Si ❑

8) Hai dovuto deviare di tanto per soddisfare una notifica segnalata dal sistema? (1: di poco o per niente – 10: di molto)
1 ❑ . 2 ❑ . 3 ❑ . 4 ❑ . 5 ❑ . 6 ❑ . 7 ❑ . 8 ❑ . 9 ❑ . 10 ❑

9) Hai utilizzato la mappa?
SI ❑ . NO ❑

10) Se si, ti è stata utile?
SI ❑ . NO ❑ . ND ❑

continua sul retro

Funzionalità

11) Si sono verificati crash o malfunzionamenti dell'applicazione durante l'utilizzo?
SI ❑ . NO ❑

12) Riterresti utile poter aggiungere dei punti di interesse/reminder personali (da aggiungere a quelli esistenti)? (1: poco o per niente – 10: molto)
1 ❑ . 2 ❑ . 3 ❑ . 4 ❑ . 5 ❑ . 6 ❑ . 7 ❑ . 8 ❑ . 9 ❑ . 10 ❑

13) Riterresti utile poter condividere con altri utenti i tuoi punti di interesse (p.e. le altre persone possono conoscere e utilizzare i posti dove ti sei trovato bene)? (1: poco o per niente – 10: molto)
1 ❑ . 2 ❑ . 3 ❑ . 4 ❑ . 5 ❑ . 6 ❑ . 7 ❑ . 8 ❑ . 9 ❑ . 10 ❑

14) Riterresti utile poter inserire dei task che possano essere visualizzati anche da altre persone? (1: poco o per niente – 10: molto)
1 ❑ . 2 ❑ . 3 ❑ . 4 ❑ . 5 ❑ . 6 ❑ . 7 ❑ . 8 ❑ . 9 ❑ . 10 ❑

Concludendo...

15) Complessivamente come valuteresti l'applicazione utilizzata?
1 ❑ . 2 ❑ . 3 ❑ . 4 ❑ . 5 ❑ . 6 ❑ . 7 ❑ . 8 ❑ . 9 ❑ . 10 ❑

16) Utilizzeresti l'applicazione se non fosse gratuita?
SI ❑ . NO ❑

17) Quanto saresti disposto a spendere per questa applicazione?
1€ o meno ❑ . 2-4€ ❑ . 5-10€ ❑ . 11€ o più ❑

Appendice E

Installazione ambiente di sviluppo

Si è preso come riferimento l'installazione dell'ambiente di sviluppo su sistema operativo Linux Ubuntu 10.04/10.10.

Requisiti:

- Java JDK (v6 Update 22);
- Apache Maven (v3.0);
- Eclipse Helios;
- Maven Integration for Eclipse;
- Apache Tomcat 6.0.29;
- Android SDK;
- ADT Plugin (Android Development Tools Plugin).

1. Installazione JDK da java.sun.com
   Rimuovere la versione OpenJDK eventualmente presente nel sistema con
   `sudo apt-get remove openjdk-6-jre`
   rimuovere tutte le dipendenze orfane utilizzando
   `sudo apt-get autoremove`
   aggiungere il repository Java Sun ai repository conosciuti dal sistema con
   `sudo add-apt-repository "deb http://archive.canonical.com/ lucid partner"`
   aggiornare la lista dei repository con
   `sudo apt-get update`
   installare Java Sun con
   `sudo apt-get install sun-java6-jre sun-java6-jdk sun-java6-plugin sun-java6-fonts`
   per verificare la corretta installazione digitare sul terminale
   `java -version`

la risposta dev'essere

```
java version "1.6.0_22"
Java(TM) SE Runtime Environment (build 1.6.0_22-b04)
Java HotSpot(TM) Server VM (build 17.1-b03, mixed mode)
```

2. Scaricare ed installare Maven per automatizzare il build del progetto al link http://maven.apache.org/download.html
   estrarre il contenuto del file apache-maven-3.0-bin.tar.gz con
   ```
   tar xvzf /cartelladownload/apache-maven-3.0-bin.tar.gz
   ```
   e copiarlo in
   ```
   sudo cp -R apache-maven-3.0 /usr/local/apache-maven
   ```
   aggiungere le seguenti variabili d'ambiente globali al file /etc/profile
   ```
   export M2_HOME=/usr/local/apache-maven/maven
   export M2=$M2_HOME/bin
   export PATH=$M2:$PATH
   ```
   dare i permessi di lettura/scrittura a tutti gli utenti della cartella apache-maven
   ```
   sudo chmod 777 /usr/local/maven
   ```
   controllare la corretta installazione con
   ```
   mvn --version
   ```
3. Installare Eclipse come IDE per lo sviluppo ed installare i seguenti plugin:
   1. Maven Integration for eclipse
      da Eclipse: Help->Install New Software
      http://m2eclipse.sonatype.org/sites/m2e
      Add->OK
      Spuntare Maven Integration for Eclipse
      Next >
      Next >
      Accettare la licenza
      Finish
      Riavviare Eclipse
      Eseguire il comando
      ```
      mvn -Declipse.workspace=/usr/local/maven eclipse:add-maven-repo
      ```
      da Eclipse: Windows->Preferences->Maven->Installation click su Add e puntare alla cartella di installazione di maven (`/usr/local/maven`) e successivamente controllare che in Windows->Preferences->Maven->User Settings si impostato come User Settings il path del file settings.xml di Maven nel sistema (`/usr/local/maven/settings.xml`).
   2. Maven Integration for WTP
      da Eclipse: Help->Install New Software
      http://m2eclipse.sonatype.org/sites/m2e-extras
      Add->OK
      Spuntare Maven Integration for WTP (optional)
      Next >
      Next >
      Accettare la licenza
      Finish

Riavviare Eclipse.

3. Subclipse (to access SVN)
   da Eclipse: Help->Install New Software
   http://subclipse.tigris.org/update_1.6.x
   Add->OK
   Spuntare Core SVNKit Library, Optional JNA Library, Subclipse
   Next >
   Next >
   Accettare la licenza
   Finish
   Riavviare Eclipse
   Windows->Preferences->Team->SVN
   Alla voce SVN interface: → Client impostare SVNKit.
4. Eclipse Web Developer Tool for Web, XML, and Java EE Develop
   da Eclipse: Help->Install New Software
   http://download.eclipse.org/releases/helios
   Add->OK
   Spuntare Web, XML, and Java EE Develop
   Next >
   Next >
   Accettare la licenza
   Finish
   Riavviare Eclipse

4. Avviare a questo punto Eclipse e: File->Nwe->Other->SVN->CheckOut Project from SVN:
   1. È possibile sincronizzare il workspace locale con il repository Google (in sola lettura) in:
      https://thesis-ug.googlecode.com/svn/trunk
      https://thesis-ug.googlecode.com/svn/branches
      È stato creato un account per il progetto: username: thesisUG, password: checkthesisUG.
      Se necessario la password per l'accesso a SVN in scrittura è: `vV7Kz2FR8nJ6`.
   2. Click Finish.
5. Alcune librerie non vengono importate automaticamente con Maven. Eseguire il file bash installdependencies.sh digitando da terminale
   `./installdependencies.sh`
   dalla cartella che contiene tutte le dipendenze per installarle.
6. Se la connessione del computer avviene tramite proxy, impostare l'indirizzo del proxy nel file `/src/main/resources/system.conf`. In caso contrario commentare le righe relative al proxy nel file indicato.
7. Installazione del servlet container Apache Tomcat
   download di Tomcat da http://tomcat.apache.org/download-60.cgi, scompattare il file con
   `tar xvzf /cartelladownload/apache-tomcat-6.0.29.tar.gz`
   e copiarne il contenuto nella cartella

```
sudo mv apache-tomcat-6.0.29 /usr/local/
```

8. L'avvio del server Tomcat avvine direttamente da Eclipse. Configurare i path per il riconoscimento della cartella di installazione in Windows->Preferences->Server->RuntimeEnvironment->Add, selezionare Apache Tomcat v6.0 e scegliere la cartella di installazione di Tomcat, quindi Finish.
9. Per avviare la sezione server del progetto, basta semplicemente fare clic col pulsante destro del mouse sul nome del progetto e selezionare Run->Run on Server.
10. Il sistema di logging utilizzato è log4j. È sufficiente copiare i file `log4j.properties`, `log4j-1.2.16.jar` e `tomcat-juli-adapters.jar` nella cartella `/usr/local/apache-tomcat-6.0.29/lib`.
11. Installazione del database MySQL.
    Da terminale digitare
    `sudo apt-get install mysql-server`.
12. Installazione dell'SDK Android.
    Download del pacchetto dall'url
    http://developer.android.com/sdk/index.html
    scompattare l'archivio e copiarlo nella directory `/usr/local/`.
    Aggiungere la seguente variabile di ambiente globali al file `/etc/profile`
    ```
    export PATH=$PATH:/usr/local/android-sdk/tools
    ```
13. Installare l'ADT (Android Development Tools) plugin in Eclipse.
    Avviare Eclipse e da Help->Install new Software inserire il repository Google per l'ADT:
    https://dl-ssl.google.com/android/eclipse/
    selezionare la voce Developer Tools e procedere all'installazione.
    Riavviare Eclipse.
14. Dal menù Windows->Preferences cambiare le preferenze dell'ADT installando i target necessari allo sviluppo delle applicaizoni Android (per il presente progetto si è scelto il target Android 2.1 con Google APIs).
15. Sincronizzare il progetto client su repository SVN seguendo le linee guida indicate al precedente punto 4.
16. Inserire nel file /res/layout/map.xml la Google Map Key calcolata come indicato all'url http://code.google.com/intl/it-IT/android/maps-api-signup.html.

Appendice F

Installazione server

L'installazione dell'ambiente server è stata effettuata su macchina vituale VMWare. Si fa riferimento all'installazione dell'ambiente server su sistema operativo Linux Ubuntu 10.04/10.10.

1. Installazione JDK da java.sun.com
   Rimuovere la versione OpenJDK eventualmente presente nel sistema con
   `sudo apt-get remove openjdk-6-jre`
   rimuovere tutte le dipendenze orfane utilizzando
   `sudo apt-get autoremove`
   aggiungere il repository Java Sun ai repository conosciuti dal sistema con
   `sudo add-apt-repository "deb http://archive.canonical.com/ lucid partner"`
   aggiornare la lista dei repository con
   `sudo apt-get update`
   installare Java Sun con
   `sudo apt-get install sun-java6-jre sun-java6-jdk sun-java6-plugin sun-java6-fonts`
   per verificare la corretta installazione digitare sul terminale
   `java -version`
   la risposta dev'essere
   ```
   java version "1.6.0_22"
   Java(TM) SE Runtime Environment (build 1.6.0_22-b04)
   Java HotSpot(TM) Server VM (build 17.1-b03, mixed mode)
   ```
2. Installazione del Web Server Apache.
   Da terminale digitare:
   `sudo apt-get install apache2`

Conclusa l'installazione, controllarne il funzionamento digitando su un browser web l'indirizzo http://localhost/, se l'installazione è avvenuta con successo verrà visualizzata la scritta It Works!

3. Installazione motore PHP.
   Da terminale digitare:
   ```
   sudo apt-get install php5 libapache2-mod-php5
   ```
   riavviare il demone di Apache con il comando
   ```
   sudo /etc/init.d/apache restart
   ```
   e controllare la corretta installazione creando il file `test.php` con il comando
   ```
   sudo vi /var/www/test.php
   ```
   e scrivendoci (successivamente salvare il file)
   ```
   <?php phpinfo(); ?>
   ```
   puntando con un browser all'indirizzo http://localhost/test.php si dovrebbe visualizzare la pagina di informazioni del motore php.
   L'installazione del web server Apache e del motore PHP permettono l'utilizzo degli script citati nel paragrafo 3.1.3.2 . a pagina 32.
4. Installazione del DBMS MySQL.
   Da terminale digitare il comando
   ```
   sudo apt-get install mysql-server.
   ```
5. Installazione di phpmyadmin.
   Digitare da terminale il comando:
   ```
   sudo apt-get install libapache2-mod-auth-mysql php5-mysql phpmyadmin
   ```
   al termine dell'installazione aprire il file
   ```
   sudo vi /etc/php5/apache2/php.ini
   ```
   e modificare la riga con scritto `;extension=msql.so` eliminando il `;` iniziale.
   PHPMyAdmin è uno script in PHP che mette a disposizione una interfaccia grafica per interagire con il DBMS MySQL.
6. Installazione del servlet container Apache Tomcat
   Download di Tomcat da http://tomcat.apache.org/download-60.cgi, scompattare il file con il comando
   ```
   tar xvzf /cartelladownload/apache-tomcat-6.0.29.tar.gz
   ```
   e copiarne il contenuto nella cartella
   ```
   sudo mv apache-tomcat-6.0.29 /usr/local/
   ```
   Creare un link simbolico alla cartella di installazione di tomcat con il comando:
   ```
   ln -s /usr/local/apache-tomcat-6.0.29 tomcat
   ```
   Creare il file `/etc/init.d/tomcat` per avvio, stop e riavvio di Tomcat. Tale file contiene lo script bash:
   ```
   #!/bin/bash
   #
   # Startup script for the Tomcat server
   #
   # chkconfig: - 83 53
   # description: Starts and stops the Tomcat daemon.
   # processname: tomcat
   # pidfile: /var/run/tomcat.pid
   # See how we were called.
   ```

```
case $1 in
   start)
      export JAVA_HOME=/usr/lib/jvm/java-1.5.0-sun/
      export CLASSPATH=/usr/local/tomcat/lib/servlet-api.jar
      export CLASSPATH=/usr/local/tomcat/lib/jsp-api.jar
      export JRE_HOME=/usr/lib/jvm/java-1.5.0-sun/
      echo "Tomcat is started"
      sh /usr/local/tomcat/bin/startup.sh
   ;;
   stop)
      export JRE_HOME=/usr/lib/jvm/java-1.5.0-sun/
      sh /usr/local/tomcat/bin/shutdown.sh
      echo "Tomcat is stopped"
   ;;
   restart)
      export JRE_HOME=/usr/lib/jvm/java-1.5.0-sun/
      sh /usr/local/tomcat/bin/shutdown.sh
      echo "Tomcat is stopped"
      sh /usr/local/tomcat/bin/startup.sh
      echo "Tomcat is started"
   ;;
   *)
      echo "Usage: /etc/init.d/tomcat start|stop|restart"
   ;;
esac
exit 0
```

7. Rilasciare i permessi, rendere eseguibile e registrare il file per l'avvio automatico del server Tomcat all'avvio del computer, con i seguenti comandi:

```
sudo chmod 755 /etc/init.d/tomcat
sudo update-rc.d tomcat defaults
```

Il deploy della sezione server del progetto, può essere effettuata automaticamente accedendo al manager di Tomcat da browser web all'url http://indirizzoserver:8080/ e cliccando sul link Manager App e caricando con l'apposito form il filw war esportato da Eclipse.

Appendice G

Di seguito riportato il file SQL con lo schema del database.

```
SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";

--
-- Database: `thesisug`
--
CREATE DATABASE `thesisug` DEFAULT CHARACTER SET latin1 COLLATE
latin1_swedish_ci;
USE `thesisug`;

-- --------------------------------------------------------

--
-- Struttura della tabella `Event`
--

CREATE TABLE IF NOT EXISTS `Event` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `User` varchar(50) DEFAULT NULL,
  `dueDate` varchar(50) DEFAULT NULL,
  `startTime` varchar(50) DEFAULT NULL,
  `endTime` varchar(50) DEFAULT NULL,
  `location` varchar(200) DEFAULT NULL,
  `ReminderId` int(11) DEFAULT NULL,
  `Done` int(1) DEFAULT '0',
  `DoneTime` datetime DEFAULT NULL,
  `DoneLatitude` varchar(30) DEFAULT NULL,
```

```
  `DoneLongitude` varchar(30) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

-- --------------------------------------------------------

--
-- Struttura della tabella `ExternalService`
--

CREATE TABLE IF NOT EXISTS `ExternalService` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(100) DEFAULT NULL,
  `Description` varchar(250) DEFAULT NULL,
  `ServiceType` enum('EventService','TaskService') DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

-- --------------------------------------------------------

--
-- Struttura della tabella `ExternalServiceCredentials`
--

CREATE TABLE IF NOT EXISTS `ExternalServiceCredentials` (
  `User` varchar(50) DEFAULT NULL,
  `ExternalServiceID` int(11) DEFAULT NULL,
  `username` varchar(50) DEFAULT NULL,
  `password` varchar(50) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Struttura della tabella `GroupMember`
--

CREATE TABLE IF NOT EXISTS `GroupMember` (
  `UserGroup` int(11) NOT NULL,
  `User` varchar(50) NOT NULL,
  `TimeStamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- --------------------------------------------------------
```

```
--
-- Struttura della tabella `GroupRequest`
--

CREATE TABLE IF NOT EXISTS `GroupRequest` (
  `id` int(15) NOT NULL AUTO_INCREMENT,
  `Sender` varchar(50) NOT NULL,
  `UserGroup` int(11) NOT NULL,
  `User` varchar(50) NOT NULL,
  `Message` varchar(150) NOT NULL,
  `TimeStamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=2 ;

-- --------------------------------------------------------

--
-- Struttura della tabella `Reminder`
--

CREATE TABLE IF NOT EXISTS `Reminder` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `priority` int(11) DEFAULT NULL,
  `description` varchar(350) DEFAULT NULL,
  `title` varchar(200) DEFAULT NULL,
  `type` int(11) DEFAULT NULL,
  `latitude` varchar(30) DEFAULT NULL,
  `longitude` varchar(30) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=320 ;

-- --------------------------------------------------------

--
-- Struttura della tabella `Task`
--

CREATE TABLE IF NOT EXISTS `Task` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `User` varchar(50) DEFAULT NULL,
  `dueDate` varchar(50) DEFAULT NULL,
  `notifyTimeStart` varchar(50) DEFAULT NULL,
  `notifyTimeEnd` varchar(50) DEFAULT NULL,
```

```
  `ReminderId` int(11) DEFAULT NULL,
  `UserGroup` int(11) DEFAULT '0',
  `Done` int(1) DEFAULT '0',
  `DoneTime` datetime DEFAULT NULL,
  `DoneLatitude` varchar(30) DEFAULT NULL,
  `DoneLongitude` varchar(30) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=223 ;

-- --------------------------------------------------------

--
-- Struttura della tabella `User`
--

CREATE TABLE IF NOT EXISTS `User` (
  `username` varchar(50) DEFAULT NULL,
  `password` varchar(50) DEFAULT NULL,
  `firstname` varchar(50) NOT NULL,
  `lastname` varchar(50) NOT NULL,
  `email` varchar(100) NOT NULL,
  `sessionKey` varchar(50) DEFAULT NULL,
  `active` int(1) DEFAULT '0',
  `verified` tinyint(1) DEFAULT '0',
  `verification_code` varchar(32) NOT NULL,
  `trial_login` int(11) NOT NULL DEFAULT '5'
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Struttura della tabella `UserGroup`
--

CREATE TABLE IF NOT EXISTS `UserGroup` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `group_name` varchar(50) NOT NULL,
  `owner` varchar(50) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=2 ;
```

Bibliografia

[1] M. Migliardi , G. Ravera – A Support System for Memory Impaired Subjects Master Dissertation – Universita Degli Studi di Genova – 2008.

[2] M. Migliardi, Petrus Prasetyo Anggono – A mobile, context aware system for memory support and planning – Università Degli Studi di Genova – 2010.

[3] Cay S. Horstmann – Concetti di informatica e fondamenti di Java 2 – Apogeo – 2000.

[4] Karsten Samaschke – Java Dai fondamenti alla programmazione avanzata – Apogeo – 2005.

[5] Martin Fowler – UML Distilled Guida rapida al linguaggio di modellaizone standard – 4a edizione – Addison Wesley – 2010.

[6] Massimo Carli – Android Guida per lo sviluppatore – Apogeo – 2010.

[7] Marco Colombetti – Ingegneria della conoscenza 2009-10 – Dipartimento di Elettronica e Informazione, Politecnico di Milano – http://home.dei.polimi.it/colombet/IC/materiale/IC%2009-10%20Parte%20I.pdf - 2010.

[8] “db4objects API reference : native java, .NET and mono open source object database”.

[9] Google Maps APIs documentation – http://code.google.com/intl/it-IT/apis/maps/index.html.

[10] Google corp., Google Maps, service available online at http://maps.google.it/maps

[11] Hotels near me - http://www.blumedialab.com/hotels/, 2010.

[12] Loopt - http://www.loopt.com/, 2010.

[13] Remember the milk - http://www.rememberthemilk.com/services/, 2010.

[14] F. Adelstein, S. Gupta, R. Golden III, L. Schwiebert - Fundamentals of Mobile and Pervasive Computing - McGrawHill - 2004

[15] Ivan Stojmenovic - Handbook of Sensor Networks: Algorithms and Architectures - Adobe E-Book – 2005.

[16] J. E. Bardram, A.Mihailidis, D.Wan - Pervasive Computing in Healthcare - CRC Press – 2006.

[17] R. Grimm - System Support for Pervasive Applications - ACM Trans. Comp. Syst. - 2004.

[19] HermiT Reasoner – Information System Group, University of Oxford – service available online at http://hermit-reasoner.com/.

[20] Ubiquity documentation – Mozilla Labs – service available online at https://wiki.mozilla.org/Labs/Ubiquity.