UNIVERSITÀ DI PADOVA  FACOLTÀ DI INGEGNERIA

TESI DI LAUREA

Sistema di supporto mnemonico: un approccio "sociale" alla generazione di data-source

---

Relatore: Chia.mo Prof. Mauro Migliardi

Correlatore: Ing. Marco Gaudina

Laureando: Mirco Furlan - matr. 601050

Corso di Laurea MAGISTRALE in INGEGNERIA INFORMATICA

A.A. 2010/2011

Questo testo è stato scritto con il software freeware LaTeX

L'autore può essere contattato all'indirizzo di posta elettronica:
mirco.furlan@tiscalinet.it

Alla mia famiglia

Sommario

Il presente lavoro si prefigge l'obiettivo di realizzare un sistema che, con l'ausilio di strumenti portatili sempre più evoluti e sempre più diffusi, agevoli l'utilizzatore a portare a termine i propri impegni quotidiani in modo completo e soddisfacente affidando ad esso il compito di ricordarli.

Questa tesi è la prosecuzione di un progetto (vedi bibliografia [1, 2, 3, 4, 5, 6, 7, 8]) su cui hanno lavorato in passato altri studenti sia dell'Università di Padova che dell'Università di Genova. Il software progettato permette all'utilizzatore con smatphone Android di memorizzare impegni, compiti e attività da eseguire, individuate come task o eventi. Grazie all'utilizzo del sistema GPS, quando l'utente si trova vicino ai punti di interesse (negozi, ristoranti, uffici, ecc.) che possono soddisfare uno o più dei task memorizzati, il sistema ne segnala attraverso notifiche, siano esse audio, vocali, visuali o vibrazioni.

Il lavoro prende avvio dall'illustrazione delle caratteristiche del sistema di partenza (client e server) da un punto di vista tecnico e con particolare attenzione agli strumenti utilizzati; per passare poi alla descrizione dell'organizzazione del lavoro e di quali strumenti di comunicazione sono stati impiegati.

Sono state eseguite l'analisi del software di partenza e il riconoscimento delle esigenze finalizzate al miglioramento della stabilità dell'applicazione e all'individuazione dei possibili sviluppi delle sue funzioni.

Successivamente si descrivono le funzionalità supportate dal sistema al momento in cui è iniziato il presente lavoro.

Vengono descritte alcune limitazioni del sistema: limitata base di conoscenza dell'applicativo (ontologia) e uso esclusivo di Google Maps Web Service per il reperimento di luoghi finalizzati all'attuazione dei task. Le soluzioni adottate hanno portato l'applicazione ad un livello soddisfacente, tale da poter essere ora utilizzato dall'utente finale.

Come nello stadio precedente del progetto, i test effettuati hanno confermato una ottimizzazione dello spazio percorso, del tempo impiegato e del numero di task ricordati. Infine sono stati apportati miglioramenti nell'usabilità del software migliorandone l'intuitività.

Ringraziamenti

Questo piccolo spazio rappresenta l'occasione più grande che ho per poter esprimere i miei più sinceri ringraziamenti a tutte le persone che mi sono state vicine in questi anni. Lasciando i loro nomi in queste poche righe voglio così coronare il mio successo, che ho potuto raggiungere anche grazie a loro.

Ringrazio i miei genitori, Maria e Luigi, che in questi anni mi hanno sempre supportato, anche nei momenti più difficili. Hanno saputo guidarmi con utili consigli e rimanendo spettatori attenti, proprio come avrei voluto.

Ringrazio tutti i miei parenti.

Un grazie speciale a mio fratello Fabio, per avermi sempre sostenuto in tutti questi anni.

Un grazie ad Anuska, che in questi anni di università, mi ha sopportato, anche nei momenti difficili.

Un grazie sincero a tutti gli Amici, con la A maiuscola, perché sono pienamente convinto che risultati come questo siano frutto, oltre che di tanto impegno, di momenti di spensierata e sana allegria. Grazie davvero.

Ringrazio il Prof. Migliardi, perché i suoi consigli e la sua piena disponibilità si sono rivelati davvero preziosi.

Ringrazio, inoltre, tutti quelli che hanno collaborato nella fase di test del sistema descritto in questo testo.

Onè di Fonte, il 11/10/2011

M.F.

Indice

Elenco delle figure

Capitolo 1

Introduzione

1.1 Definizione del problema

In campo psico-fisiologico recenti studi correlano lo stress con la difficoltà trasferire informazioni dalla memoria a breve termine a quella a medio termine. Questa difficoltà spesso ingenera una riduzione dell'efficienza personale poichè le cose da fare vengono in mente in modo disordinato e non pianificato.

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.

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.

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, agevoli l'utilizzatore ad affrontare quotidianamente le situazioni descritte. Con il termine 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, per il sistema in questione, sono particolarmente interessanti la presenza di un'antenna GPS che permette di geolocalizzare in qualsiasi momento il terminale, la connessione a banda larga ed i sistemi di acquisizione vocale che consentono di comandare il cellulare attraverso il semplice utilizzo della voce.

1.2 Obiettivo del progetto

Il presente lavoro di tesi è il proseguimento di un progetto (vedi bibliografia [1, 2, 3, 4, 5, 6, 7, 8]) su cui hanno lavorato in passato altri studenti sia dell'Università di Padova che dell'Università di Genova, cioè: Giorgio Ravera[1][1, 2] e da Petrus Prasetyo Anggono[2][3, 4] e continuato poi da Guido Geloso[3][7], Alessio Toso[4][8]e Lorenzo Andretta[5].

L'obiettivo comune a tutti noi è quello 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 o eventi). Grazie all'utilizzo del sistema GPS, quando l'utente si troverà ad una certa distanza dai 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. All'utente è data la possibilità di definire, secondo le prorprie esigenze, la tipologia delle notifiche. Esse possono essere audio, vocali, visuali o vibrazioni.

Da un lato il sistema si inserisce quindi nella quotidianità 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

La tesi è strutturata come segue: nella prima parte illustreremo le caratteristiche del sistema di partenza (client e server) da un punto di vista tecnico e con particolare attenzione agli strumenti utilizzati (Capitolo 2 a pagina 5); si descrive anche come il presente lavoro è stato organizzato e quali strumenti di comunicazione sono stati usati.

Successivamente, nel Capitolo 3 a pagina 19, si descrivono l'architettura del sistema e le funzionalità supportate sia dal terminale mobile che dal server nel momento in cui è inziato il presente lavoro di tesi.

Il Capitolo 4 (pag.37), invece, è dedicato alle proposte di lavoro maturate 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.

[1]Giorgio Ravera è stato uno studente dell'Università degli studi di Genova, ha ottenuto la laurea specialistica in Ingegneria Informatica il 31/10/2008.

[2]Petrus Prasetyo Anggono è stato uno studente dell'Università degli studi di Genova, ha ottenuto la laurea specialistica in Ingegneria Informatica il 23/07/2010.

[3]Guido Geloso è stato uno studente dell'Università degli studi di Genova, ha ottenuto la laurea specialistica in Ingegneria Informatica il 11/03/2011.

[4]Alessio Toso è stato uno studente dell'Università degli studi di Padova, ha ottenuto la laurea specialistica in Ingegneria Informatica il 15/03/2011.

[5]Lorenzo Andretta è uno studente dell'Università degli studi di Genova, ha ottenuto la laurea triennale in Ingegneria Informatica il 11/03/2011.

Al fine di testare i diversi aspetti del sistema, sono stati effettuati diversi test dell'applicazione, con l'ausilio di alcuni 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 sia pregressi che futuri. Sono state coinvolte in questa fase persone esterne al progetto per valutarne la facilità di utilizzo, l'intuitività, l'efficienza, la percezione di utilità e di gradimento da parte di potenziali utenti finali, il qui punto di vista era neutrale in quanto, per loro, era la prima volt ache utilizzavano l'applicazione. Le modalità di svolgimento dei test e i risultati ottenuti sono descritti nel Capitolo 5 a pagina 127.

Nel Capitolo 6 (pag.155), vengono individuati alcuni problemi del sistema attuale da risolvere e si descrivono dei punti di sviluppo futuri.

Capitolo 2

Ambiente di sviluppo, tecnologie utilizzate e organizzazione del lavoro

2.1 Ambiente di lavoro: l'hardware e la virtualizzazione

2.1.1 Ambiente server: l'hardware

Inizialmente al progetto erano stati assegnati due server, uno presente nella sede di Padova e un'altro nella sede di Genova. Ciò era permesso dato che il gruppo precedente, che seguiva il progetto, era formato da uno studente dell'univeristà di Genova e da un'altro dell'università di Padova. Dal momento che il presente gruppo di lavoro è interamente di Padova, il server di Genova è stato rimosso. Perciò, l'unico server sopravissuto è quello di Padova avente caratteristiche: Intel(R) Celeron(R) D CPU 3.33GHz, 2Gb di ram e hardisk da 70Gb. Il sistema operativo installato era Linux Ubuntu 10.04[11].

Successivamente, avendo a disposizione un PC più performante, si è deciso di sostituire anche l'unica macchina di Padova. Il server che, tuttora viene utilizzato, ha le seguenti caratteristiche: Intel(R) Core(TM)2 Duo CPU E4600 2.40GHz, ram 4GB e hardisk di capacità 447 GB.

2.1.2 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 una macchine a tale scopo, installando un sistema operativo e tutti i componenti necessari all'utilizzo dell'applicazione.

Ciò che ha spinto la 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.

Appurata la decisione di continuare con la virtualizzazione, si decise di utilizzare come sistema operativo qualcosa che non occupasse molte risorse, qualcosa di leggero, sprovvisto di interfaccia grafica, che permettesse maggiori prestazioni.

Le soluzioni individuate erano due:

1. utilizzare VMware vSphere Hypervisor (Based on ESxi)[10];
2. Ubuntu server 10.04 LTS[11].

I pro e contro delle due possibili soluzioni verrano analizzati nei successive due paragrafi.

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.

2.1.2.1 VMware vSphere Hypervisor (Based on ESXi)

Le possibilità offerte da questa tecnologia sono straordinarie ed esistono numerosi software gratuiti che possono essere sfruttati per creare ed eseguire sul proprio computer una o più macchine virtuali (Vm, virtual machines) con cui provare sistemi operativi e valutare nuovo software senza danneggiare o rallentare Windows. Anche se l'uso della virtualizzazione permette di lavorare in un ambiente di fatto separato e indipendente, questi software sfruttano comunque le risorse hardware della macchina fisica, intaccando dunque, a volte anche in modo significativo, le prestazioni dell'intero sistema. Si tratta di un aspetto fastidioso, soprattutto per chi dispone di un portatile o di un desktop non troppo performante. È possibile però superarlo adottando un approccio diverso. L'idea è quella di assegnare a un computer dedicato il compito di far girare le macchine virtuali, e di sfruttare i Pc ad esso collegati via Lan per visualizzare le interfacce delle Vm e interagire con esse tramite mouse e tastiera. In questo modo l'esperienza utente è praticamente la stessa che si avrebbe se le macchine virtuali fossero in esecuzione localmente, ma l'impatto sulle prestazioni dei computer usati come client è quasi inavvertibile. Esistono numerosi pacchetti che consentono di implementare un'architettura di questo tipo: possono essere installati direttamente sull'hardware (quindi non richiedono, a differenza di prodotti di virtualizzazione come VMware Server[12] o Virtual Box[13], un sistema operativo "ospite") e permettono proprio di allestire un server di macchine virtuali. Purtroppo sono di norma piuttosto cari e concepiti principalmente per l'uso su hardware di fascia server, dunque più costoso rispetto a quello di fascia desktop. Fortunatamente quasi tutti sono disponibili anche in una versione gratuita. Appartengono a questa categoria i software hypervisor come VMware Hypervisor (chiamato anche ESXi), Citrix XenServer[14] e Microsoft Hyper-V Server[15].

La soluzione di VMware è molto diffusa e utilizzata, oltre che scalabile, e supporta moltissimi sistemi operativi: da tutti i sistemi Windows alla maggior parte delle distribuzioni Linux. Hypervisor è solo il primo tassello della piattaforma del colosso californiano ma è proprio la base su cui tutta la sua offerta è costruita. Nonostante Hypervisor sia un pacchetto gratuito, in unione con il modulo (anch'esso gratuito)

vSphere Client offre una serie di funzionalità davvero notevole: permette infatti di creare, gestire, controllare e persino condividere una o più macchine virtuali rapidamente. Grazie a VMware Hypervisor e a vSphere Client si può creare agevolmente – e con un investimento assai contenuto sul fronte dell'hardware – un vero e proprio laboratorio informatico virtuale, in cui il principale limite al numero di Vm che si possono creare ed eseguire contemporaneamente è dato dalle risorse hardware a disposizione. VMware Hypervisor tuttavia può essere usato non solo in uno scenario di questo tipo, ma anche per consolidare uno o più server di produzione – ad esempio un application server e il server di posta elettronica su una sola macchina fisica. Non importa se le macchine da consolidare hanno hardware o persino sistemi operativi differenti: un singolo server fisico con VMware Hypervisor può gestirle senza problemi, con un notevole vantaggio economico dato che in questo modo si riducono sia la quantità di hardware da acquistare e mantenere sia il consumo di energia.

Purtroppo i vantaggi nell'utilizzare VMware vSphere Hypervisor non possono essere apprezzati dal momento che le caratteristiche hardware del nostro server non sono compatibili con la lista dei requisiti, disponibile all'indirizzo `http://www.vmware.com/resources/compatibility/search.php`, per utilizzare questo prodotto vmware.

2.1.2.2 Ubuntu Server 10.04 LTS con Vmware server 2

Accantonata la possibilità di installare VMware vSphere Hypervisor, si è scelto di puntare su un sistema operativo che non occupasse molte risorse, senza interfaccia grafica, come Linux Ubuntu 10.04 LTS. Su esso è stato installato anche il server OpenSSH[16], così da permettere le connessioni cifrate.

Si è scelto di utilizzare il software di virtualizzazione VMWare server versione 2.0.2[17]. 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 trasportando i file necessari al suo funzionamento: i file .vmdk e .vmx contenuti solitamente in /var/lib/vmware/Virtual Machines/.

Per consultare la guida d'installazione del sistema operativo si consulti l'Appendice A oppure il wiki all'indirizzo `http://serverpd.dyndns.org/thesisug`.

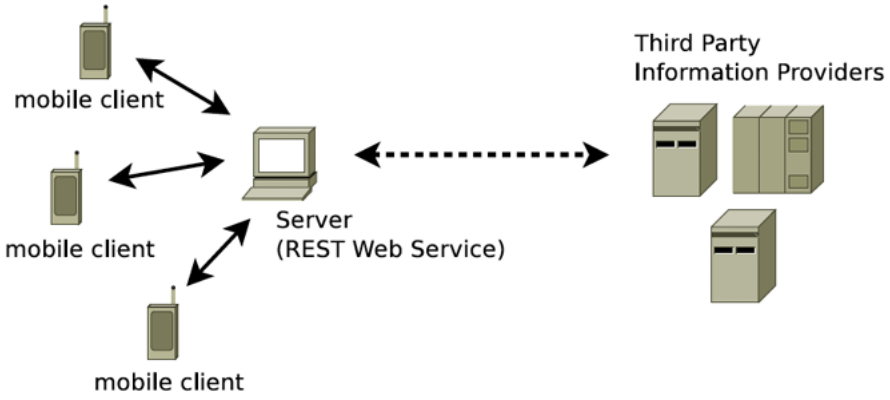
2.2 Tecnologie utilizzate lato server

Il nostro sistema adotta un'architettura client-server. Il server, a cui i vari client possono connettersi via HTTP[18] è responsabile delle attività computazionalmente onerose. Per questo svolge un ruolo fondamentale nel sistema.

Ogni mobile client, d'altra parte, è responsabile di attività computazionalmente più leggere destinate alla generazione dell'interfaccia grafica verso l'utente.

Il server è anche connesso a un provider (correntemente Google) per fornire servizi nella nostra applicazione, come reperire le mappe e trovare posti di interesse dove

poter soddisfare i propri task grazie a Google Maps[19] .



In questa sezione vedremo quali tecnologie, intese come strumenti di lavoro, tool e utility di sviluppo, utilizzate durante il nostro progetto per costruire il più agevolmente possibile il sistema in questione. Per una guida dettagliata sull'installazione dell'ambiente di sviluppo consultare l'Appendice B a pagina 164 o il wiki del progetto all'indirizzo `http://serverpd.dyndns.org/thesisug`.

2.2.1 Eclipse

Eclipse[20] è un ambiente di sviluppo integrato opensource, multi-linguaggio e multi-piattaforma. Ideato da un consorzio di grandi società quali Ericsson, HP, IBM, Intel, MontaVista Software, QNX, SAP e Serena Software, chiamato Eclipse Foundation, viene sviluppato da una comunità strutturata sullo stile dell'open source.

Eclipse può essere utilizzato per la produzione di software di vario genere, si passa infatti da un completo IDE[1] per il linguaggio Java (JDT, "Java Development Tools") a un ambiente di sviluppo per il linguaggio C++ (CDT, "C/C++ Development Tools") e a plug-in che permettono di gestire XML[21], Javascript, PHP e persino di progettare graficamente una GUI[2] per un'applicazione JAVA (Eclipse VE, "Visual Editor"), rendendo di fatto Eclipse un ambiente RAD[3].

Nel nostro caso è stato usato come IDE per il linguaggio JAVA[22].

L'utilizzo di Eclipse è incentrata sull'uso di plug-in. I plug-in che sono stati utilizzati sono:

1. Subclipse: plugin per l'accesso ed il controllo direttamente da Eclipse di SVN[23];

[1] Integrated develpment environment o ambiente di sviluppo integrato è un software che aiuta i programmatori nello sviluppo del codice.

[2] Graphical User Interface, interfaccia grafica per l'utente.

[3] E' una metodologia di sviluppo del software nata negli anni ottanta. Questa metodologia coinvolge modelli di sviluppo iterativi, la costruzione di prototipi e l'utilizzo di strumenti CASE (Computer-Aided Software Engineering, strumenti che supportano lo sviluppo attraverso interfacce grafiche). Solitamente questo approccio allo sviluppo comporta compromessi tra usabilità, funzionalità e velocità d'esecuzione.

2. Maven Integration for Eclipse[24]: plugin per l'integrazione in Eclipse di Apache Maven[25], 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;

3. Eclipse Web Developer Tool[26]: pacchetto che comprende tutti i componenti necessari alla programmazione Web, in particolare con Java EE ed il supporto per la creazione di server locali;

4. Android Developer Tools(ADT)[27]: plugin per l'integrazione dell'Android SDK[28] in Eclipse, per poter gestire l'emulatore Android e una serie di altri componenti (ad esempio DDMS, Dalvik Debug Monitor Server[29]) utili nello sviluppo di applicazioni per Android.

2.2.2 Java Platform

Uno dei punti di forza di Java è l'indipendenza di tale linguaggio dalla piattaforma utilizzata. La Java Virtual Machine[30] non è altro che un software che consente al codice Java precedentemente compilato (detto byte code) di essere, a sua volta, interpretato ed eseguito sul sistema operativo che si sta utilizzando. Essendo il liguaggio utilizzato lato server Java, è neccessaria la Java Virtual Machine. Esistono svariate versioni di JVM (Java Virtual Machine), una per ogni tipo di sistema operativo supportato da Java. Dal sito ufficiale[4] è possibile scaricare JDK (Java Development Kit), un ambiente integrato di lavoro contenente la Virtual Machine, il compilatore javac per i sorgenti e un debugger.

2.2.3 Tomcat e sistemi

Tomcat[31] è un server engine o anche servlet container per pagine JSP[5][32]; esattamente come Asp[33] con IIS[34] e PHP con Apache, anche la tecnologia JSP ha bisogno di un "motore" per la gestione delle chiamate provenienti dagli utenti e il conseguente invio delle risposte sotto forma di output. Tomcat è un progetto sviluppatto dalle stessa Apache Software Foundation[35], `http://www.apache.org/` che ha dato vita al celebre Web Server (Apache Web Server[36]). E' completamente open source e può essere scaricato gratuitamente dal sito ufficiale in versioni differenti a seconda del file system (Windows, Linux, ecc.) in cui intendiamo utilizzarlo.

Tomcat è utilizzato all'interno del nostro progetto come motore di esecuzione lato server.

Di fondamentale importanza sono i file di log generati da tomcat. Si possono trovare nella cartella log/ dell' installazione di tomcat. Sono più file: alcuni si riferiscono

[4] `http://www.java.com/it/download/index.jsp`

[5] JavaServer Page (JSP) è una tecnologia Java per lo sviluppo di applicazioni Web. Nel contesto della piattaforma Java, la tecnologia JSP è correlata con quella dei servlet. All'atto della prima invocazione, le pagine JSP vengono infatti tradotte automaticamente da un compilatore JSP in servlet. Una pagina JSP può quindi essere vista come una rappresentazione ad alto livello di un servlet. Per maggiori informazioni consultare `http://it.wikipedia.org/wiki/JavaServer_Pages`.

alle webapp installate, altre si riferiscono alle stampe che si possono fare nelle servlet/jsp. Essi indicano eventuali crash del sistema, informazioni sul funzionamneto attuale, ecc.

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.

Dato che il nostro sistema si appoggia a Tomcat per poter funzionare, si è potuto scegliere se utilizzare il metodo di logging nativo o se invece utilizzarne un altro. La scelta fatta è quella di utilizzare Log4J[37], che va ad intercettare i log generati da Tomcat e da tutti i sistemi ivi inseriti. Dopo di che abbiamo configurato il modo in cui questi messaggi, appartenenti ai diversi sistemi, vengono visualizzati in modo da avere un output completo per quel che riguarda il nostro lavoro, limitando però quello relativo a Tomcat o altri servizi, in quanto non è di nostro interesse conoscerne i messaggi ad un livello molto dettagliato.

2.2.4 Apache 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[6], è ospitato e fa parte dell'ex progetto Jakarta[38]. 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 archetipi di Maven, definendo pertanto una struttura fissa del progetto stesso.

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.

[6] `http://www.apache.org/`

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.

2.2.5 Web service

Un Web Service (o servizio web)[39, 40], 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[41], 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[42]).

La caratteristica dei Web Service di fornire dei servizi indipendentemente dalla piattaforma utilizzata, li rende dei candidati ideali per il progetto in esame. Un Web Service permette 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)[43] Web Service.

RESTful Web Service

Secondo la definizione data dal World Wide Web Consortium[7][44] (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).

[7] Il W3C sviluppa tecnologie che garantiscono l'interoperabilità (specifiche, guidelines, software e applicazioni) per portare il World Wide Web al massimo del suo potenziale agendo da forum di informazioni, comunicazioni e attività comuni. Sito di riferimento: `http://www.w3.org/`.

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.

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 Roy Fielding[8][45] 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;
- statelessness, cioè ogni ciclo di request/response deve rappresentare un'interazione completa del client con il server;
- essere 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.

[8]Roy Thomas Fielding è uno dei principali architetti delle specifiche per l'HTTP (RFC 2616).Nato a Laguna Beach, in California, ha conseguito il Dottorato dall'Università della California - Irvine - nel 2000. La sua tesi per il dottorato si è basata sulla costruzione di REST (Representational State Transfer), componente fondamentale del WWW di nuova generazione, evoluzione dei primi protocolli dei Web Services, a partire da SOAP e CORBA.

RESTeasy Framework

Dal momento che le librerie Java non presentano il supporto per REST, esistono diversi framework che permettono di implementare la specifica JAX-RS[9][46] in un servizio Restful.

RESTeasy[10][47, 48] è uno di questi; prodotto da JBOSS[49], può essere usato per il deploy in Tomcat. E' stato scelto per facilitare l'implementazione del server dal momento che mette a disposizione delle procedure di invio/ricezione dei dati.

JAX-RS

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, 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[50].

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[11], 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

[9] JAX-RS è un framework che si concentra sull'applicare le annotazioni java agli oggetti piani. Questa annotazione collega una specifica URI e una operazione HTTP a un singolo metodo per una classe java. Ha una iniezione di parametri attraverso l'annotazione che prende i parametri dalla richiesta http. Ha un lettore per il message body, e uno scrittore che permette di disaccopiare il formato dei dati serializzazione e deserializzazione dai tuoi oggetti java. Ha anche un mappatore di eccezioni per i codici di risposta http e i messaggi. Inolre ha diverse altre facility per http content negotiazion.

[10] `http://www.jboss.org/resteasy`

[11] 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.

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 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[12][52] 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.

2.2.6 Ontologia

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[13] e nomologiche[14], mentre non comprende le conoscenze fattuali[15]. 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, l'ontologia è stata usata per mantenere una base di conoscenza e per poter inferire concetti più complessi. Ad esempio, se l'utente richiede al sistema di ricordargli di "prendere il latte", sarebbe banale utilizzare "latte", come parola chiave per la ricerca di attività commerciali dove possa soddisfare il bisogno. Con l'utilizzo dell'ontologia, invece, è possibile inferire dalla parola "latte", le parole chiavi "latteria, supermercato, alimentari, etc..." estendendo la ricerca del "latte" alle entità così identificate.

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[54, 55, 56, 57]. Il linguaggio OWL[16] (Web

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

[13]conoscenza concettuale: è la conoscenza dei concetti che si utilizzano per interpretare la realtà (es. una madre è donna con almeno un figlio).

[14]conoscenze nomologiche: è la conoscenza di regolarità o leggi generali che regolano il mondo (es. una madre ama i propri figli);

[15]conoscenze fattuali: si riferiscono a particolari fatti esistenti (es. Alice è la madre di Bob).

[16]`http://www.w3.org/TR/owl-ref/`

Ontology Language) è lo standard raccomandato nel 2004 dal W3C per la definizione di ontologie per il web semantico.

Nel progetto in questione, viene utilizzata la rappresentazione XML del linguaggio OWL per memorizzare relazioni importanti per il funzionamento del sistema. Per la creazione iniziale del file ci siamo riferiti a Protégé[53]. Quest'ultimo è un editor di ontologie open source. L'abbiamo utilizzato per modellare relazioni tra oggetti/azioni e luoghi e per una iniziale popolazione. Ci sono diversi plugin in Protégé che permettono di verificare la consistenza dell'ontologia e di compiere delle inferenze. Ciò che produce Protégé può esserre processato con le OWL API[55].

Le OWL API sono utilizzate principalemente per leggere il file in formato owl e per la sua modifica. Durante l'esecuzione, le usiamo insieme con Hermit Reasoner[58] API per compiere il resoning sull'ontologia.

HermiT reasoner è utilizzato nel nostro progetto per inferire delle relazioni ragionando sulle asserzioni presenti nel file owl.

2.2.7 MySQL

Il Database utilizzato nel nostro progetto viene definito in SQL. E' stato scelto a questo proposito MySQL[59] per gestire la base di dati dell'intero sistema. MySQL è un Relational database management system (RDBMS), composto da un client con interfaccia a caratteri e un server, entrambi disponibili sia per sistemi Unix come GNU/Linux che per Windows, anche se prevale un suo utilizzo in ambito Unix. Dal 1996 supporta la maggior parte della sintassi SQL e si prevede in futuro il pieno rispetto dello standard ANSI.

MySQL svolge il compito di DBMS nella piattaforma LAMP[60, 61], una delle più usate e installate su Internet per lo sviluppo di siti e applicazioni web dinamiche.

2.2.8 Apache Web Server

Apache HTTP Server, o più comunemente Apache, è il nome dato alla piattaforma server Web modulare più diffusa (ma anche al gruppo di lavoro open source che ha creato, sviluppato e aggiornato il software server), in grado di operare da sistemi operativi UNIX/Linux e Microsoft.

Apache è un software che realizza le funzioni di trasporto delle informazioni, di internetwork e di collegamento, ha il vantaggio di offrire anche funzioni di controllo per la sicurezza come quelli che compie il proxy.

2.2.9 PHPmyAdmin

In Apache è stato installato uno dei più famosi MySQL Manager. Esistono diversi tipi di MySQL Manager, ovvero di strumenti per l'amministrazione di MySQL. Ciò che abbiamo utilizzato per amministrare il database MySQL è phpMyAdmin[62] (richiede un server web come Apache HTTP Server ed il supporto del linguaggio PHP). Si può utilizzare facilmente tramite un qualsiasi browser.

2.3 Tecnologie utilizzate lato client

2.3.1 Sistema Android

La presente applicazione è stata sviluppata per una serie di terminali dotati di caratteristiche avanzate: 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[63], 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[64], ma soprattutto superando il sistema operativo dell'azienda di Cupertino, l'iOS[65] per gli Apple iPhone[66], grazie alle sue caratteristiche di openness. Il sistema operativo targato Google continua ad essere fortemente diffuso. Oltre 600000 nuove attivazioni al giorno.

2.4 Organizzazione del lavoro

Il presente progetto rappresenta una prolungata attività di collaborazione tra l'Università di Padova e l'Università 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[67], con il quale tutti i componenti comunicavano, e l'utilizzo del software di VoIP Skype[68], 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.

Il lavoro è stato organizzato in modo da includere meeting telematici con scadenza circa settimanle per monitorare assiduamente le problematiche e guidare efficacemente gli sviluppi.

Gli aspetti trattati durante le periodiche riunioni telematiche hanno permesso di migliorare continuamente il software generando nuovi componenti e facendo percepire, ad esempio, la necessità di:

- una gestione social del sistema, includendo delle segnalazioni di punti di interesse personali, non presenti nei servizi esterni (ad esempio in Google Maps[69, 70, 71]) 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 dal sistema;
- valutare altre sorgenti esterne di informazione da integrare nel sistema (ad esempio Yahoo! Local Search Web Service[72], PagineGialle[73, 74], 2spaghi[75], ecc.);
- ampliare la portata del reasoner, includendo sempre più collegamenti semantici a parole chiave, estendendo agli utenti la possibilità di popolare l'ontologia;
- utilizzare un navigatore turn-by-turn[17], come guida al raggiungimento del luogo;
- implementare un meccanismo di cache per velocizzare i tempi di risposta del server rispetto al client e per diminuire il numero di richieste ai servizi di Google;
- rendere il sistema il più reattivo possibile includendo dei sistemi di notifica che richiamino l'attenzione degli utenti quando vengono notificati gli hints;
- 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;
- far interagire l'utente con il sistema in modo da personalizzarlo il più possibile in base alle propri esigenze e gusti.

Molti dei punti sopra indicati sono stati sviscerati e verranno 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 di due limiti: tempo e risorse coinvolte nel progetto. Sono stati riportati però degli spunti per dei futuri sviluppi in merito.

Durante tutto il periodo di lavoro al progetto, inoltre, è stato sviluppato un wiki, disponibile all'indirizzo `http://serverpd.dyndns.org/thesisug`. Si è rivelato uno strumento molto utile, in quanto, disponeva sempre degli ultimi aggiornamenti man mano che creavamo nuove funzionalità. Inoltre, contiene una guida utente utile da leggere, soprattutto per i nuovi utenti che si prestano per la prima volta all'utilizzo del sistema.

[17] Per modalità "turn-by-turn" si intendono tutti quei navigatori attivi che generano l'indicazione stradale in base alla continua rilevazione della posizione raggiunta tramite segnale GPS.

2.4.1 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[79], SVN[23], Mercurial[80] 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 del progetto fanno parte più sviluppatori, 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 membri del gruppo di sviluppo;
- confrontare velocemente le varie versioni, poichè 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)[81]. Per questo progetto si è scelto di ospitare il codice scritto nei repository online di Google Code, più precisamente in `http://code.google.com/p/thesis-ug/`.

Capitolo 3

Architettura e funzionalità del sistema di partenza

3.1 Architettura multilayer del server

Il server è stato progettato seguendo il modello di architettura Three-Tier[83]. Nel caso di Web Service questo tipo di struttura è intesa nel seguente modo:

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

Nel nostro sistema abbiamo seguito queste linee guida definendo appunto i tre livelli, che comunicano tra di loro al fine di generare la risposta alla richiesta pervenuta dal client. I tre livelli, che sono mostrati anche in figura 3.1, sono:

1. Livello Web: livello più esterno che accoglie le richieste pervenute dal client e le invia al livello sottostante (Livello Management);
2. Livello Management: effettua delle operazioni basilari, se neccessario, aggrega i dati arrivati dall'esterno dal livello soprastante e interagisce con lo strato sottostante (Livello Database); oltre a ciò restituisce le risposte che torneranno dal livello database, in modo che possano essere reinviate al client in una forma adeguata;
3. Livello Database: è il livello più basso del sistema e 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 al database e il database vero e proprio, siano residenti fisicamente sulla stessa macchina.

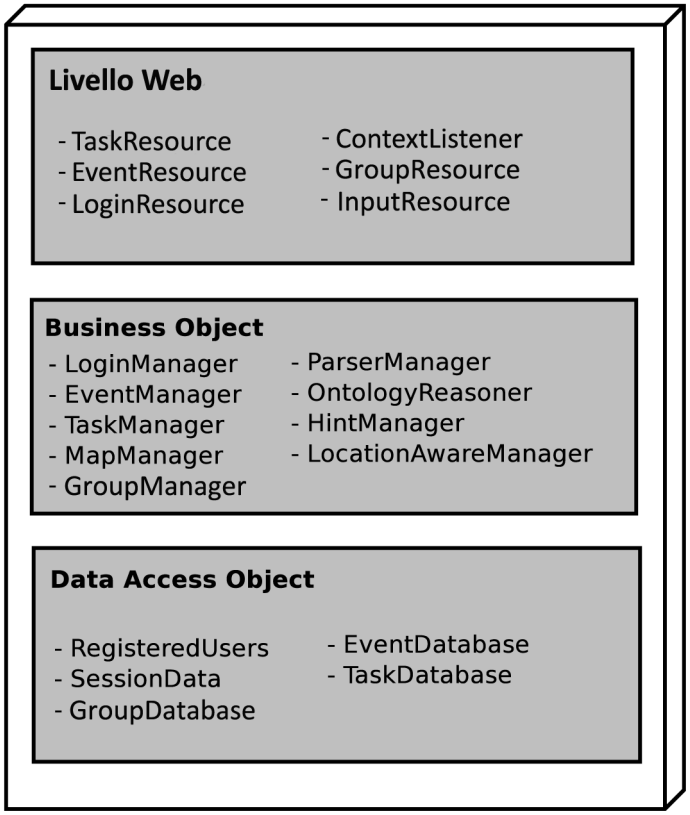


Figura 3.1: Architettura multilayer.

L'architettura realizzata è di tipo client/server e non si discosta molto dal paradigma RMI (Remote Method Invocation) la cui struttura è esemplificata nella figura 3.2. 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.

Il client, come dispoditivo mobile, effettua una richiesta http. Questa richiesta può contenere dei dati da inviare al server o meno. Nel primo caso i dati vengono serializzati in XML.

Il server (REST web service), ricevuta la richiesta dal client, si preoccupa di soddisfarla eseguendo le operazioni neccessarie. La richiesta inizialmente viene accolta dal livello Web, che richiamerà un metodo del livello Manager Object. Questo eseguendo la sua logica, se neccessario, contatterà il livello DAO, che effettua operazioni sui dati. I dati restituiti dal DAO prenderanno la strada inversa raggiungendo il Livello Managment. Qui i risultati verranno convertiti in un oggetto (value object), che verrà serializzato dal livello soprastante Web.

In seguito esamineremo, una per una, le funzionalità del server che erano presenti all'inizio del presente lavoro di tesi.

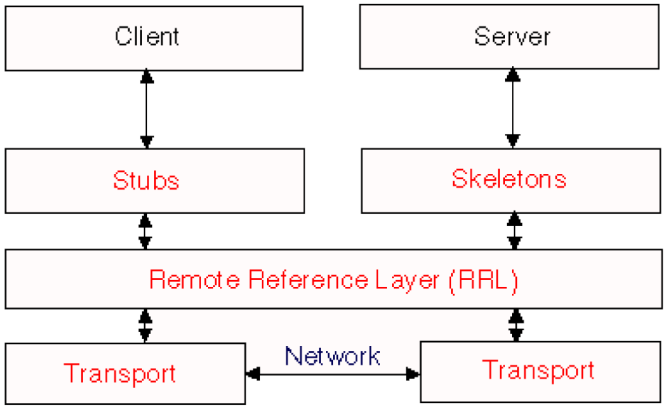


Figura 3.2: Struttura RMI.

3.2 Funzionalità del server

3.2.1 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/ephemere/{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.

3.2.2 Richieste Task

Le richieste da parte del client per ottenere le informazioni necessarie ai Task, avvengono all'url `http://indirizzoserver:8080/ephemere/{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.

3.2.3 Richieste Event

Le richieste da parte del client per ottenere le informazioni necessarie agli Eventi, avvengono all'url `http://indirizzoserver:8080/ephemere/{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.

3.2.4 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/ephemere/{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.

3.2.5 Richieste Context

La sezione reperibile all'url `http://indirizzoserver:8080/ephemer/{username}/location` mette a disposizione due risorse disponibili rispettivamente in /all e /single che provvedono alla ricerca nel servizio utilizzato (Google Maps per ora, poi vedremo anche l'aggiunta di un altro datasource proprio) 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, per inferire dalle parole chiave presenti in un task una serie di parole chiave correlate ed effettuare la ricerca, e le Google Maps API[70], per la ricerca nel database di Google delle possibili attività commerciali che possono soddisfare un task.

3.2.6 Richieste Group

Questa sezione mette a disposizione risorse disponibili per creare un nuovo gruppo, per unirsi o lasciare il gruppo.

Le risorse messe a disposizione dal server, riguardo i gruppi, si possono trovare all'url `http://indirizzoserver:8080/ephemer/{username}/group`. Le varie risorse sono accessibili in alcune sottocartelle, in particolare:

- /create: utilizzata per creare un nuovo gruppo. /invite: utilizzata per invitare un membro ad un gruppo. Per invitare un membro al tuo gruppo bisogna conoscere lo username dell'utente da invitare;
- /list: utilizzata per recuperare la lista di tutti i gruppi di cui l'utente fa parte;
- /getJoinGroupRequest: utilizzata per recuperare la lista delle richieste di join ad un gruppo per l'utente ;
- /getGroupMember: utilizzata per vedere chi è iscritto ad un gruppo;
- /deleteFromGroup: utilizzata per uscire da un gruppo;
- /accept: utilizzata per accettare la richiesta di join ad un gruppo;
- /refuse: utilizzata per rifiutare la richiesta di join ad un gruppo;

3.2.7 Particolarità server

3.2.7.1 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 (o pseudo tale) in modo automatico.

Il funzionamento del parser implementato ricalca sotto qualche aspetto il funzionamento del parser implementato dal Mozilla Labs[76], per Ubiquity[77], add-ons per il famoso browser web Mozilla Firefox[78].

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 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 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

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

per il comando "aggiungi evento".

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 realtivo. 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 puntaggio 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.

Al fine di facilitare l'internazionalizzazione del parser, erano 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-218 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

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, LOCATION, 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

Quindi, riassumendo, 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.

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 e "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 era stato modificato. 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 in seguito 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.

3.2.7.2 Ontologie

Al fine di supportare l'utente, bisogna poter capire, a partire dal testo con il quale si definisce il task, quali siano le categorie di esercizi commerciali in grado di poter soddisfare la richiesta fatta. Per poter raggiungere questo obiettivo intermedio, si possono adottare numerose strategie, al fine di poter legare le diverse parole chiave con i nomi dei luoghi da cercare. Nel sistema di base questa scelta è ricaduta sull'uso di un file di ontologie, nel quale, in pratica vengono salvate delle coppie che definiscono l'appartenenza di un oggetto ad un dato luogo, dopo di che l'azione di un reasoner ci permette di ottenere la lista dei luoghi di interesse.

Nel presente progetto, perciò, 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[70]) 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.

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.

3.2.7.3 Hermit Reasoner

Nel presente progetto è stato utilizzato il reasoner HermiT[58] per rispondere alle interrogazioni.

I reasoner, ovvero un'applicazione in grado di operare deduzioni a partire dalle conoscenze, sono molto complessi e nello stabilire se un insieme di enunciati sia o non sia soddisfacibile utilizzano tutta una serie di euristiche di ottimizzazione, senza le quali i tempi di risposta sarebbero inaccettabili. I reasoner si basano su procedure di decisione per la soddisfacibilità di un insieme di enunciate.

La definizione delle relazioni logiche avviene tramite enunciati contenuti nel file HintsOntologyNew.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).

3.2.7.4 Google Maps API

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[71]. Il risultato della query al servizio Maps di Google, viene restituito in formato JSON[84]. 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=ABQIAAAAWuzz0Iu2TplTMtzxmJOuUBQusdbP4Fg6uj%kVg6Gs4Mes9VMUQBRE-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[88] (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:

Listing 3.1: Esempio di risposta JSON servizio Google Local Search

```
1  {"responseData":
   {"results"
3  [{"GsearchResultClass":"GlocalSearch", "viewportmode":"computed", "listingType":
   "local", "lat":"45.670359","lng":"12.235108", "accuracy":"8",
5  "title":"Fradis␣Sas␣Di␣Bernardi␣A.␣\u0026amp;␣C.",
   "titleNoFormatting":"Fradis␣Sas␣Di␣Bernardi␣A.␣\u0026␣C.",
7  "ddUrl":"http://www.google.com/maps?source\u003duds\u0026daddr
   \u003dViale+Monfenera,+11,+Treviso,+Veneto+(Fradis+Sas+Di+Bernardi+A.+%26+C.)
9  ␣␣␣␣␣␣␣␣␣␣␣␣␣␣+@45.670359,12.235108\u0026saddr\u003d45.666096,12.22924",
   "ddUrlToHere":"http://www.google.com/maps?source\u003duds\u0026daddr
11 \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", "content":"", "maxAge":604800,
"phoneNumbers":[{"type":"","number":"0422␣542331"}],
"addressLines":["Via␣Jacopo␣Riccati,␣31","31100␣Treviso,␣Italia"] }, ],
"resultAttribution":"Schede␣attività␣commerciali␣fornite␣da␣\u003ca
"staticMapUrl":"http://maps.google.com/maps/api/staticmap?maptype\u003droadmap
        \u0026format\u003dgif\u0026sensor\u003dfalse\u0026size\u003d150x100
        \u0026zoom\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":"ViaJacopoRiccati,31","city":"Treviso","region":"Veneto",
"country":"Italy","staticMapUrl":"http://maps.google.com/maps/api/staticmap?maptype
        \u003droadmap\u0026format\u003dgif\u0026sensor\u003dfalse\u0026size
        \u003d150x100\u0026zoom\u003d13\u0026markers\u003d45.666188,12.241609",
␣␣␣␣␣␣␣␣"url":"http://www.google.com/maps/place?source\u003duds
        \u0026q\u003dpanetteria\u0026cid\u003d11339606163680129896",
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.

3.3 Funzionalità del client

3.3.1 Registrazione utente

Tutti i sistemi basati su architettura client/server, necessitano di una autenticazione per l'accesso alle risorse, tramite la quale è possibile identificare univocamente un determinato utente e garantirgli l'accesso solo alle sue 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 per i nuovi utenti, disponibile direttamente nel terminale mobile, svincolando così la necessità di centralizzare le registrazioni tramite ad esempio una interfaccia web, che in quel caso si sarebbe resa disponibile ad esempio nel sito di riferimento dell'applicazione.

La procedura di registrazione realizzata (come si vede in figura 3.3), richiede l'inserimento di alcuni dati da parte dell'utente, successivamente tali dati sono trasmessi 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.

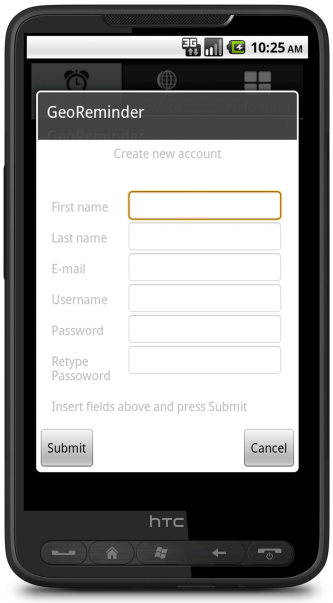


Figura 3.3: Registrazione utente.

Dopo aver portato a termine la registrazione è data la possibilità all'utente di usufruire di 5 login di prova, secondo il modello del trial login, con i quali l'utente può provare l'applicazione e, se questa risulta di suo gradimento, passare all'attivazione del proprio account.

3.3.2 Login

La schermata di Login (come si vede in figura 3.4) 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.

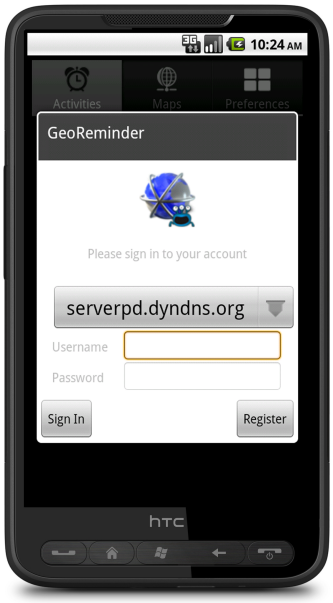


Figura 3.4: Login.

3.3.3 Task e 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", eccetera. 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.

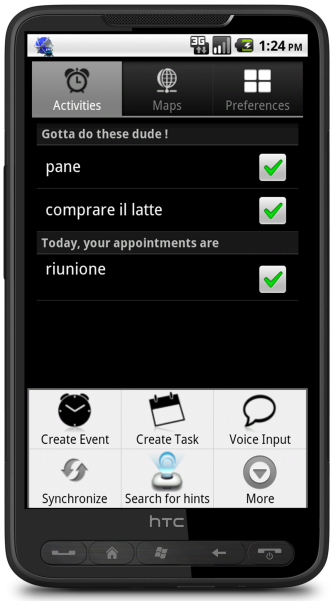


Figura 3.5: Task ed eventi.

3.3.4 Mappa

Questa visualizzazione (figura 3.6) 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.

3.3.5 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 (come si vede in figura 3.7) 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;

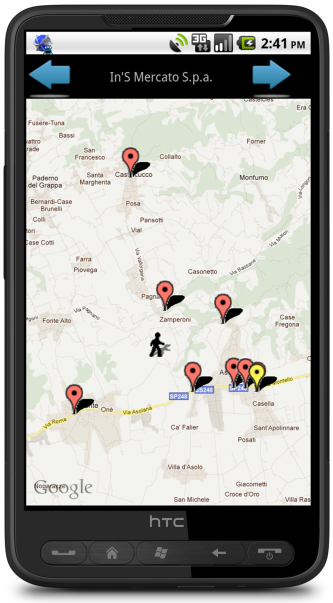


Figura 3.6: Mappa.

- *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);
- *Parser Language*: questa impostazione serve ad indicare la lingua del parser vocale;
- *Navigation Preference*: questo parametro consente di stabilire se le indicazioni stradali per raggiungere un hint vengono indicate in formato lista di google oppure tramite un navigatore turn-by-turn installato nel terminale. Solitamente viene utilizzato anche qui Google Navigator;
- *Notification*: questa impostazione consente di stabilire la forma delle notifiche (audio, vibrazioni, parlato);
- *Altre impostazioni avanzate*: impostazioni avanzate che normalmente non vengono utilizzate da un "normale" utilizzatore; servono per impostare un nuovo indirizzo del server.

3.3.6 Task di gruppo

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

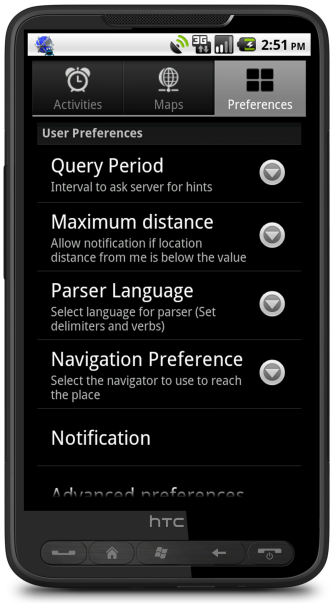


Figura 3.7: Preferences.

- un mezzo attraverso il quale creare il gruppo stesso;

- il metodo usato dagli utenti per venire a conoscenza del gruppo;

- le procedure da seguire per unirsi o lasciare il gruppo.

In un'applicazione di questo tipo il gruppo è in genere formato da utenti che hanno in comune il raggiungimento di un dato obiettivo entro un certo limite di tempo, in modo da trarre un vantaggio comune. È quindi molto probabile che i componenti del gruppo virtuale siano anche componenti di un qualche gruppo reale, ad esempio una famiglia o un gruppo di amici e amiche. Per questa ragione la modalità di iscrizione al gruppo è basata sugli inviti che i soli partecipanti al gruppo possono inviare ad altri utenti, purchè ne conoscano lo username; tale condizione è facile da soddisfare dati i presupposti sui quali ci siamo basati per concepire il gruppo.

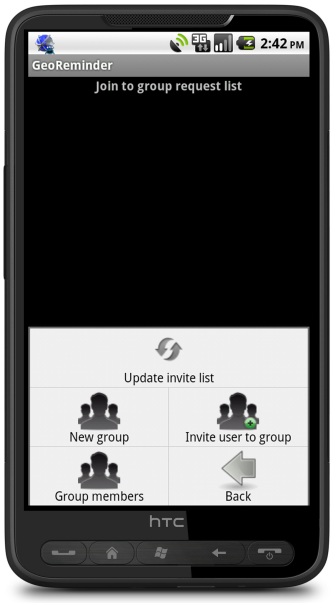


Figura 3.8: Task di gruppo.

Al fine di non stravolgere il modello di creazione e gestione dei task, si è scelto di far si che ogni oggetto creato avesse un gruppo di appartenenza e che tale valore venisse poi inserito nella colonna UserGroup della tabella Task. Dato che il valore da inserire altro non era se non l'id del gruppo a cui appartiene il task, sapendo che tale id è dato da un campo int con attiva la proprietà autoincrement e che tale valore parte da 1, i task personali avevano come gruppo lo zero.

In questo modo risulta subito chiaro quali task siano relativi ad un qualche gruppo e quali invece siano da considerare personali.

La schermata principale che permette all'utente di gestire i gruppi è raggiungibile dal menù principale dell'applicazione e si può vedere in figura 3.8.

3.3.7 Il sistema di geolocalizzazione

L'utente utilizza il sistema GPS per poter essere localizzato.

Il sistema GPS, però, a fronte della sua precisione, presenta alcuni inconvenienti, tra cui:

- Alto consumo in termini di batteria
- Impossibilità d'uso in ambienti in-door
- Possibilità d'uso fortemente legata agli eventi atmosferici

Nonostante questi problemi non siano meno importanti di altri da risolvere, durante il periodo di tesi sono state considerate e risolte altre problematiche.

3.3.8 Cambio di rete e notifiche

Una tipologia di comportamento del sistema che si vuole assolutamente evitare è quella in cui vengono generate così tante notifiche così di frequente da far diventare l'informazione puro rumore. A questo scopo bisogna stabilire delle soglie minime entro le quali non sia possibile notificare nuovamente all'utente eventuali punti di interesse, in quanto si presume che tali notifiche risultino frutto di errori di geolocalizzazione.

Una delle impostazioni del sistema è proprio la distanza tra due query al server, quindi il primo approccio prevede che, ad ogni cambio di provider, dopo aver recuperato la nuova posizione, invece di effettuare direttamente una notifica, si controlli se la distanza percorsa dall'ultima segnalazione sia superiore o meno al limite minimo richiesto.

Un'ulteriore meccanismo volto ad evitare il sovraccarico di notifiche è stato basato su un'intervallo temporale minimo tra le notifiche. Il problema di limitare il numero di notifiche senza eliminare dell'informazione gradita dall'utente non è comunque di facile soluzione e richiede di essere adattato ai singoli utenti. L'utilizzo dei comportamenti degli utenti per adattare alle loro preferenze il comportamento del sistema è un problema aperto che non è stato trattato all'interno di questa tesi.

Capitolo 4

Il sistema: limiti, problemi e soluzioni

4.1 Panoramica dei problemi indivuati

Da un'analisi del software e dai numerosi meeting settimanali fatti, sono emersi diversi problemi e limitazioni del sistema:

- scarsa capacità del sistema nel comprendere il task dell'utente dovuta ad un'ontologia limitata;
- uso esclusivo di una sorgente dati per il reperimento di luoghi finalizzati all'attuazione dei task;
- eccessiva vulnerabilità del sistema ai tempi di risposta dei servizi esterni;
- limitato supporto all'utente nel guidarlo a destinazione.

Analizzeremo in dettaglio, nei prossimi paragrafi, ognuno dei problemi appena nominati. Descriveremo per ognuno di essi, il motivo che ha spinto ad individuarli come problemi, le possibili soluzioni individuate ed infine, l'implementazione scelta.

4.2 Limite del sistema: scarsa capacità del sistema nel comprendere il task dell'utente

Al fine di supportare l'utente, bisogna poter capire, a partire dal testo con il quale si definisce il task, quali siano le categorie di esercizi commerciali in grado di poter soddisfare la richiesta fatta.

Per poter raggiungere questo obiettivo intermedio, si possono adottare numerose strategie, trattate nel paragrafo 4.2.2, al fine di poter legare le diverse parole chiave con i nomi dei luoghi da cercare. Nel sistema di base viene utilizzata un'ontologia, la quale, contiene coppie che definiscono l'appartenenza di un oggetto ad un dato luogo, dopo di che l'azione di un reasoner permette di ottenere la lista dei luoghi di interesse. Infatti, un'ontologia è una rappresentazione formale, condivisa ed esplicita

di una concettualizzazione di un dominio di interesse, che permette di modellare queste relazioni esplicitamente e di svolgere al reasoner il suo lavoro nell'inferire le relazioni implicite. Nel nostro caso si tratta di relazioni tra cose (item o action) e luoghi in cui si possono trovare/svolgere queste cose (location).

Come si è potuto notare nel Capitolo 5, nella figura 3.1, la parte centrale del meccanismo di reasoning è presente a livello Manager ed è rappresentata dalla classe OntologyReasoner. Questa provvede di relazionare i task dell'utente con i luoghi nei d'intorni dell'utente. Il nostro corrente uso dell'ontologia, al momento, è molto semplificato. Essa è contenuta in un file con estensione owl in formato xml. Contiene solamente tre entità: item, action e location. Il compito pricipale dell'OntologyReasoner è inferire la locazione dove soddisfare l'esigenza dell'utente dal task inserito, andando a reperire le informazioni dal file con estensione owl.

La scarsità di informazioni nell'ontologia, porta ad una inneficienza nella comprensione del task. Ciò che si descrive nei prossimi paragrafi è la descrizione dettagliata di questa limitazione e la soluzione al problema.

4.2.1 Descrizione dettagliata del problema

Nel sistema di base, come menzionato nel precedente paragrafo, si utilizza un'ontologia per modellare relazioni tra entità di diversi domini: cose, azioni e luoghi. Sono state definite relazioni nella forma oggetto–>luogo e azione–>luogo. Nel primo caso, si vuole specificare che un determinato oggetto si può trovare in un determinato luogo; nel secondo caso, si stabilisce che un'azione si può svolgere in una data locazione. Vediamo, ora, come si possono creare queste semplici relazioni in formato owl-xml.

Listing 4.1: Intestazione e classi item, action e location del file OWL

```
<?xml version="1.0"?>
<Ontology xmlns=http://www.w3.org/2002/07/owl#
xml:base=http://www.semanticweb.org/ontologies/2011/3/Ontology1302856896322.owl
xmlns:rdfs=http://www.w3.org/2000/01/rdf-schema#
xmlns:xsd=http://www.w3.org/2001/XMLSchema#
xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
xmlns:xml=http://www.w3.org/XML/1998/namespace
ontologyIRI=http://www.semanticweb.org/ontologie/2011/3/Ontology1302856896322.owl>
<Prefix
        name="xsd"
        IRI="http://www.w3.org/2001/XMLSchema#"/>
<Prefix
        name="owl"
        IRI="http://www.w3.org/2002/07/owl#"/>
<Prefix
        name=""
        IRI="http://www.w3.org/2002/07/owl#"/><Prefix
        name="rdf"
        IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
<Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>
<Declaration> <Class IRI="#Action"/> </Declaration>
<Declaration> <Class IRI="#Item"/> </Declaration>
```

```
<Declaration> <Class IRI="#Location"/> </Declaration>

<Declaration>
        <Class IRI="#Item"/>
</Declaration>

<Declaration>
        <Class IRI="#Action"/>
</Declaration>

<Declaration>
        <Class IRI="#Location"/>
</Declaration>
```

Come si vede nell'ultima parte dal codice sopra, utilizziamo un linguaggio a meta-tag, come l'xml[21] per definire classi di oggetti. In questo caso, definiamo l'esistenza di 3 classi: item, action e location. L'item rappresenta la classe che rappresenta gli oggetti fisici, un qualcosa di esistente, ad esempio pane, latte, pantaloni etc. La classe action rappresenta un'azione, un verbo, che si intende svolgere, ad esempio camminare, correre, ballare etc. Location, invece, rappresenta una categoria di luoghi. Ad esempio, supermarket, farmacia, scuola etc. Le tre classi, devono essere disgiunte, ossia un elemento appartenente ad una classe nn può appartenere ad un'altra. Lo specifichiamo usando la seguente sintassi in xml:

Listing 4.2: DisjointClasses: item, action, location

```
<DisjointClasses>
        <Class IRI="#Item"/>
        <Class IRI="#Action"/>
        <Class IRI="#Location"/>
</DisjointClasses>
```

Ora che sono state definite le classi, si possono definire le relazioni che intercorrono tra elementi delle stesse. Per farlo, bisogna prima definire una proprietà, una ObjectProperty, in questo modo:

Listing 4.3: Definizione proprietà CanBeFoundIn nel file OWL

```
<Declaration>
        <ObjectProperty IRI="#CanBeFoundIn"/>
</Declaration>
```

Con tale sintassi si definisce l'esistenza di una proprietà che verrà utilizzata per indicare che un entità, sia essa un item o action, può essere trovata/svolta in una location. E' una proprietà antisimmetrica, irriflessiva in relazione al dominio location e range item/action. Ricordiamo che una relazione è antisimmetrica se, presi comunque due elementi a e b in X, se a è in relazione con b e b è in relazione con a, allora a = b. In simboli:

$$\forall a, b \in X,\ aRb \wedge bRa \Rightarrow a = b$$

Figura 4.1: Relazione antisimmetrica.

Inoltre, una relazione è irriflessiva o antiriflessivase se nessun elemento è in relazione con sé stesso. In simboli:

$$\forall a \in X,\ \neg(aRa)$$

Figura 4.2: Relazione irriflessiva.

Il modello della nostra ontologia a cui noi facciamo riferimento, schemattizato è (figura 4.3)

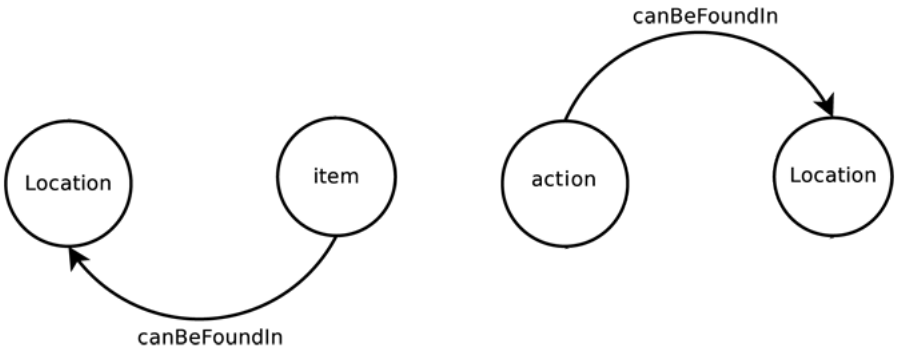


Figura 4.3: Modello Ontologia.

Come esempio, si immagini di voler definire che il riso può essere trovato al supermercato. Inizialmente bisogna definire che il riso è un item (=oggetto) e supermercato è una location.

Listing 4.4: Definizione dell'Item riso

```
<ClassAssertion>
        <Class IRI="#Item"/>
                <NamedIndividual IRI="#riso"/>
</ClassAssertion>
```

e

Listing 4.5: Definizione della location supermercato

```
<ClassAssertion>
        <Class IRI="#Location"/>
                <NamedIndividual IRI="#supermercato"/>
</ClassAssertion>
```

Successivamente relazioniamo i due concetti attraverso la proprietà "CanBeFoundIn".

Listing 4.6: Definizione della relazione riso->supermercato

```
<ObjectPropertyAssertion>
    <ObjectProperty IRI="#CanBeFoundIn"/>
        <NamedIndividual IRI="#riso"/>
            <NamedIndividual IRI="#supermercato"/>
</ObjectPropertyAssertion>
```

In questo esempio si è definito che il riso lo si può trovare al supermercato.

Per popolare l'ontologia, nel modo appena descritto, esiste un tool molto semplice da usare: Protègè[53]. Questa applicazione permette di definire queste semplici regole molto facilemente.

In ogni caso, l'ontologia può essere popolata anche con Java, usando le OWL API v3[55]. Vediamo, a questo proposito, un esempio di utilizzo di tali API, in Java. Dato un item o action, descrive come sia abbastanza semplice accedere al file owl e tramite HermiT reasoner identificare tutte le relative location che soddisfano la proprietà "canBeFoundIn".

Listing 4.7: Funzione getSearchQuery()

```
public static List<String> getSearchQuery(String need){
        log.info("2File name:"+ONTOLOGY_FILE);
        log.info("getSearchQuery with need:"+need);
        String base = ontology.getOntologyID().getOntologyIRI().toString();
        OWLDataFactory dataFactory = manager.getOWLDataFactory();
        log.info(IRI.create(base + "#" + need).toString());
        OWLNamedIndividual item = dataFactory.getOWLNamedIndividual(
                IRI.create(base + "#" + need));
        log.info(IRI.create(base + "#CanBeFoundIn").toString());
        OWLObjectProperty location = dataFactory.getOWLObjectProperty(
                IRI.create(base + "#CanBeFoundIn"));
        // runs classification with the HermiT reasoner
        Reasoner hermit = new Reasoner(manager, ontology);
        hermit.classify(); hermit.classifyObjectProperties();
        NodeSet<OWLNamedIndividual> inferredlocation =
                hermit.getObjectPropertyValues(item, location);
        Set<OWLNamedIndividual> locationvalues = inferredlocation.getFlattened();
        List<String> result = new LinkedList<String>();
        for (OWLNamedIndividual o : locationvalues) {
                result.add(o.toStringID().replace('_', ' ').replaceFirst(base + "#", ""));
        }
        return result;
}
```

In questa funzione, il parametro in ingresso è il bisogno ("need", nell'esempio di prima è il riso). Attraverso le OWL API e HermiT reasoner, si inferiscono le relazioni definite nel file owl, ottenendo così una lista di locazioni in cui soddisfarlo.

Le ontologie hanno guadagnato popolarità nella comunità, soprattutto negli ultimi anni. Esse, infatti, permettono di modellare relazioni esplicitamente e di cocesentire al reasoner di inferire altre relazioni implicite.

Dati i semplici requisiti, cioè porre in relazione le coppie item o action con le location, era possibile utilizzare anche soluzioni alternative, ad esempio una semplice base di dati relazionale. Attraverso una semplice database relazionale. Tuttavia, un database relazionale non avrebbe dato un senso, o meglio una semantica ai dati inseriti. Inoltre, non sarebbe stata una struttura dati consigliabile per definire delle ralazioni tra concetti.

Premesso questo, il problema che ci si è posto è quello dell'inserimento dei dati nella nostra ontologia. E' noto che l'estensione automatica delle ontologie[85, 86] è un problema di difficile soluzione, e, ad oggi, ancora aperto seppure l'uso di sotto-tassonomie tratte da wordnet[87] abba dato alcuni promettenti risultati.

All'inizio utilizzando Protégé 4.1.0 abbiamo popolato l'ontologia manualmente e abbiamo salvato il file nel formato OWL. Solo pochi concetti sono stati inseriti. Il problema principale consiste nel fatto che il file owl non contiene tutti i potenziali concetti che possono caratterizzare la realtà. Da questa limitazione scaturisce la neccessità di ideare un sistema per rendere l'ontologia esauriente e che contenga il massimo delle relazioni possibili. Se l'ontologia è tale la probabilità di ottenere delle locazioni per soddisfare il bisogno dell'utente espresso nel task diventa molto alta.

4.2.2 Possibili soluzioni

Inizialmente, abbiamo usato il tool Protégé per popolare la nostra ontologia, inserendo le coppie item-location o action-location, una per una, manualmente. Questo approccio andrebbe bene, ma per rendere il file owl esauriente (contenente tutti i termini esistenti), come si può intuire, risulta essere decisamente inefficiente. Perciò, al fine di risolvere questo problema si possono seguire 2 strade:

1. Popolare l'ontologia attraverso basi di dati strutturate esistenti. Ci sono diversi servizi disponibili nel web, come ad esempio Pagine Gialle. Essi di solito dispongono l'informazione organizzata per categorie. In ogni caso a noi serve strutturare delle relazioni tra item o action e location e per questo, dopo una discreta ricerca nel web, si può affermare che non esiste alcun servizio disponibile ideale per creare queste coppie in maniera automatica. Anche se la nostra analisi ha portato a questa conclusione uno dei pricipali progetti del MIT (MIT Research Project: ConceptNet 5[89]) ha come scopo la generazione di dati strutturati attraverso una modalità di riversamento automatico di informazioni presenti in altre basi di dati.

2. Popolare l'ontologia in maniera cooperativa inserendo gli utenti in una sorta di progetto di "socially networked ontology". Visto il successo del momento di tutte le applicazioni social, la domanda risulta spontanea: perchè non utilizziamo proprio la conoscenza degli utenti per arrichire l'ontologia? Ovviamente all'inizio, gli utenti dovranno intervenire frequentemente per aiutare il sistema a crescere, a inglobare sempre più informazioni (nella forma item→location o action→location), ma una volta raggiunta la maturità, tutto risulterà più efficiente.

Proprio per i motivi appena citati, si è deciso di seguire un approccio "sociale" alla generazione di informazioni da utilizzare per arricchire la base di conoscenza.

4.2.3 Verso un approccio social: consenso e votazione

Al fine di costruire un sistema social per ampliare la base di conoscenza del sistema è utile conoscere alcuni termini fondamentali per costruire un meccanismo utilizzabile da tutti.

Consenso distribuito

- In sintesi, il consenso distribuito permette a più processi di giungere ad una decisione comune, dipendente dai loro input iniziali. Più formalmente il problema del consenso viene definito nei termini delle seguenti proprietà: Termination: ogni processo corretto alla fine decide per un dato valore;
- Uniform integrity: ogni processo decide al massimo una volta sola;
- Agreement: due processi corretti non decidono in modo differente;
- Uniform validity: se un processo decide per un dato valore v, allora ci vuol dire che v è stato proposto da qualche processo.

Voting

In un sistema social, più votanti indipendenti calcolano i loro risultati e votano per determinare una maggioranza; il risultato che ha ottenuto la maggioranza di voti sarà poi committed (mandato agli utenti). Un esempio di algoritmo è il seguente:

1. un client manda una richiesta ad uno dei votanti;
2. il votante manda una richiesta multi-cast agli altri votanti;
3. i votanti eseguono la richiesta e mandano una risposta al client;
4. il client aspetta per f+1 risposte con lo stesso risultato da diversi votanti, dove f è il numero di fault (guasti) che può essere tollerato; questo è il risultato finale.

Vote to promote

E' il caso in cui un utente vuole promuovere una particolare informazione in una comunità. La promozione di cui si parla prende la forma di voto per quell'informazione e quelle con più voti salgono nel ranking.

E' un meccanismo che sarà utile nel progetto in questione, quindi lo tratteremo più approfonditamente successivamente, ma ora lo descriveremo molto brevemente.

Gli utenti nella comunità possono inserire contenuti in un "pool" di risorse. E' richiesta qualche forma di giudizio democratico per permettere alla comunità di confrontare la qualità soggettiva di un contenuto rispetto ad un altro. Inoltre, e' richiesta una comunità consistente. Idealmente, contenuti popolari nel pool dovrebbero

ricevere significativamente più voti rispetto a quelli non popolari, allo scopo di fare confronti più significativi.

In questo senso è utile fornire un meccanismo di voto, per informazione candidata nel pool della comunità. Nel momento in cui viene aggiunto un voto a favore della promozione di una coppia bisogna:

- Permettere ad un utente un solo voto per ogni informazione;
- Mostrare all'utente il voto fatto, così che sappia per che cosa ha votato;
- L'utente deve essere in grado di cambiare il voto dopo che l'ha mandato;

Le informazioni con meno voti potrebbero venire punite per la loro mancanza di popolarità e quindi cancellate oppure si limitano a cadere nel dimenticatoio e finiscono alla fine della lunga coda del pool della classifica di popolarità.

Si potrebbe voler considerare un controllo di moderazione che permetta alla community di decidere di rimuovere un' informazione completamente, ma non bisogna rendere questo controllo prominente. L'enfasi deve essere più nel promuovere le cose giuste e non di punire quelle sbagliate.

Rating[1]

Un' altra aspetto da considerare nella progettazione di un sistema social è il rating. Un utente desidera lasciare velocemente un' opinione su un oggetto, con la minima interruzione di flusso da parte di qualsiasi altro task coinvolto.

In un rating system, il valore individuale sale o scende in concomitanza con il giudizio soggettivo degli altri utenti. I rating sono raccolti insieme per presentare un rating medio di un oggetto. Gli utenti dovrebbero poter cambiare il loro voto in caso cambiassero opinione.

Statistical rating

Il miglior modo per rendere il sistema di rating statistico è avere un gran numero di voti per ogni informazione (migliaia o decine di migliaia), in modo che ogni anomalia venga considerata rumore. Comunque meno valutazioni si hanno, più probabile è che i rating siano non accurati in relazione al database dei voti nel suo complesso. Idealmente, quello che si deve fare è dare agli utenti con minor numero di voti un peso minore e a quelli con un maggior numero di voti peso maggiore.

Comparison ranking system

In un ranking system, gli utenti (più frequentemente persone) in una gerarchia salgono o scendono secondo regole specifiche oggettive e ben conosciute. Da questo si può prendere spunto per l'implementare di un meccanismo di ranking degli utenti, che faccia salire gli utenti che cercano di inserire nell'ontologia cose giudicate buone da molti.

Il ranking system è il cuore di molti giochi multiutente, un esempio è dato dall' ELO System[90] che è un sistema di ranking per giochi con due giocatori ed è usato nella U.S. Chess Federation[91]. "Days of Wonder" è una variante multigiocatore

[1] Il rating è un punteggio.

dell'ELO System per i propri giochi online. Ogni sistema costruisce una semplice distribuzione sui rating del giocatore attorno a una norma (tipicamente 1500 punti), poi vengono aggiunti o detratti punti in base a vincite o perdite, facendo rimanere costante la somma totale dei punti nel sistema. I giocatori sono poi classificati in base al confronto tra i loro punteggi.

Ci sono comunque diversi altri sistemi di ranking, molti creano una gerarchia di ranking positivi. Comunque, può anche essere usata una gerarchia di ranking negativi. Inoltre, entrambe le direzioni di classifica possono usare sistemi di soglia in modo da soddisfare certi criteri, ad es. se si superano i 2700 punti si è "Grand Master".

I sistemi di ranking, non sono un sistema di decisione collettiva ma sono un risultato collettivo.

Reputation System

Reputation System è molto simile al Ranking system, gli utenti (più frequentemente persone) in una gerarchia salgono o scendono in base a regole specifiche e ben conosciute. Comunque, diversamente dal vero ranking system i sistemi di reputazione, basano la loro modalità di salita e discesa su feedback di altri utenti. L'obiettivo di un sistema di reputazione è, in ultima analisi, creare una metrica di fiducia che consenta a diversi utenti di accedere a diversi poteri.

I sistemi di community voting presentano diversi problemi. Tra questi, la possibilità che i membri della comunità tentino di ingannare il sistema, per diverse motivazioni:

- malizia: forse contro un altro membro della comunità e i suoi contributi;
- guadagno: per ricevere una ricompensa, monetaria o no, influenzando il piazzamento di alcuni item nel pool o per un ordine del giorno generale: promuovendo sempre alcuni punti di vista o dichiarazioni politiche, con poco riguardo per l'effettiva qualità dei contenuti per i quali si sta votando.

Ci sono delle vie per minimizzare questi tipi di abuso o ostacolare coloro che compiono questi abusi:

- Votare per le cose, non per le persone;
- Considerare un rate-limiting di voti;
- Permettere all'utente un certo numero di voti in un dato periodo;
- Limitare il numero di volte che l'utente può votare negativamente per un contenuto particolare di un utente (per prevenire un attacco ad-hominem).;
- Pesare gli altri fattori oltre il numero di voti (es. user history);
- Se informazioni sulle relazioni tra utenti sono disponibili, pesare i voti degli utenti di conseguenza. Vietare agli utenti che hanno, tra loro, relazioni formali di votare per i contenuti l'uno dell'altro.

4.2.4 Soluzione implementata: sistema social per l'espansione dell'ontologia

Nel presente progetto è stato implementato un meccanismo collaborativo che permette di arricchire la base di conoscenza tramite l'inserimento di informazioni da parte degli utenti. Le informazioni saranno del tipo: un oggetto lo posso trovare in un dato luogo o una determinata azione la posso compiere in una locazione. D'ora in poi, indicheremo l'informazione che un utente inserisce con il termine "asserzione".

Perciò esistono due tipi di asserzioni:

- item-location: l'oggetto item si può trovare nel posto location (es. pane-panetteria)
- action-location : l'azione action può essere svolta nel posto location (es. pattinare-palaghiaccio)

Al fine di mantenere le informazioni degli utenti, sarà progettato un database. Inizialmente le asserzioni dei partecipanti saranno inserite nella base dati. Verrano quindi visualizzate dagli altri utenti e sottoposte al giudizio collettivo tramite una votazione.

Una asserzione può risiedere in ontologia (e quindi essere utilizzata da tutti gli utenti), oppure nel database (e quindi essere usata solo dagli utenti che l'hanno votata, in quanto in attesa di promozione). Descriveremo in seguito il meccanismo di promozione di un'asserzione. Per ora, possiamo supporre che se un'asserzione, dopo essere stata introdotta da un utente, ottiene molti consensi positivi, sarà promossa in ontologia. Una volta entrata in ontologia, non sarà più possibile rimuoverla, perché di fatto la maggioranza netta che utilizza il sistema ritiene che quell'informazione sia valida.

Per un'asserzione nel database si tiene traccia del numero di visualizzazioni, del numero di utenti che l'hanno votata positivamente(N_votes), del numero di utenti che l'hanno votata negativamente(N_votes_neg), del voto positivo, o semplicemente voto (che è dato dalla somma dei rank degli utenti che hanno votato positivamente) e del voto negativo (dato dalla somma dei rank degli utenti che hanno votato negativamente).

Vediamo ora, come è stato progettato il database per mantenere le informazioni relative all'ontologia, elencando le tabelle utilizzate e il conseguente uso.

4.2.4.1 Ontology Database

Per gestire l'ontologia del sistema si è reso neccesssario progettare un database. Sono state create diverse tabelle al fine di memorizzare i dati neccessari. Di seguito elenchiamo le tabelle specificandone tramite codice SQL la struttura.

Tabella Item_founIn_loc

Item	Location	Username	N_views	N_votes_neg	Vote	VoteNegative	AddedDate	Promotion	PromotionDate

Figura 4.4: Db Ontology: Item_founIn_Loc.

Struttura della tabella “Item_foundIn_Loc”:

```
CREATE TABLE IF NOT EXISTS ‘Item_foundIn_Loc’ (
‘Item‘ varchar(30) NOT NULL,
‘Location‘ varchar(30) NOT NULL,
‘Username‘ varchar(30) NOT NULL,
‘N_views‘ int(11) NOT NULL, ‘N_votes‘ int(11) NOT NULL,
‘Vote‘ double NOT NULL DEFAULT ’0’,
‘AddedDate‘ timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
‘Cancel‘ int(1) NOT NULL DEFAULT ’0’,
‘CancellationDate‘ timestamp NOT NULL DEFAULT ’0000-00-00 00:00:00’,
‘Promotion‘ int(1) NOT NULL DEFAULT ’0’,
‘PromotionDate‘ timestamp NOT NULL DEFAULT ’0000-00-00 00:00:00’,
PRIMARY KEY (‘Item‘,‘Location‘) )ENGINE=InnoDB DEFAULT CHAR-
SET=latin1;
```

Utilizzo: questa tabella viene utilizzata per memorizzare tutti i valori relativi ad una coppia item → location inserite, in un determinato istante. Per una coppia si memorizza il numero di volte che un utente poteva votarla, il numero di voti positivi e negativi che ha ricevuto, il voto effettivo, la data in cui è stata aggiunta e eventualmente la data di promozione.

Tabella Item_voted

Figura 4.5: Db Ontology: Item_voted.

Struttura della tabella ‘Item_voted‘

```
CREATE TABLE IF NOT EXISTS ‘Item_voted‘ (
‘Item‘ varchar(30) NOT NULL,
‘Location‘ varchar(30) NOT NULL,
‘Username‘ varchar(30) NOT NULL COMMENT ’nome dell”utente che ha votato’,
‘Vote‘ int(1) NOT NULL DEFAULT ’1’ COMMENT ’1-voto, 0-voto cancellato’,
‘VoteDate‘ timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
‘CancellationDate‘ timestamp NOT NULL DEFAULT ’0000-00-00 00:00:00’,
PRIMARY KEY (‘Item‘,‘Location‘,‘Username‘) ) ENGINE=InnoDB DEFAULT
CHARSET=latin1;
```

Utilizzo: questa tabella viene utilizzata per memorizzare le coppie item→location inserite da un utente in un determinato istante. Si inserisce lo username di chi ha votato, la coppia votata, una variabile il cui valore è 0 o 1 a seconda se il voto è ancora attivo o è stato cancellato dall’utente e la data della votazione.

Tabella Item_voted_historical

Figura 4.6: Db Ontology: Item_voted_historical.

Struttura della tabella 'Item_voted_historical'

```
CREATE TABLE IF NOT EXISTS 'Item_voted_historical' (
'Item' varchar(30) NOT NULL,
'Location' varchar(30) NOT NULL,
'Username' varchar(50) NOT NULL,
'Vote' int(1) NOT NULL COMMENT '0:voto cancellato, 1:votato',
'Date' timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY ('Item','Location','Username','Vote','Date') ) ENGINE=InnoDB
DEFAULT CHARSET=latin1;
```

Utilizzo: tabella che rappresenta uno storico delle votazioni fatte relativa alle coppie item→location.

Tabella Action_foundIn_Loc

Figura 4.7: Db Ontology: Action_foundIn_loc.

Struttura della tabella 'Action_foundIn_Loc'

```
CREATE TABLE IF NOT EXISTS 'Action_foundIn_Loc' (
'Action' varchar(30) NOT NULL,
'Location' varchar(30) NOT NULL,
'Username' varchar(30) NOT NULL, 'N_views' int(11) NOT NULL,
'N_votes' int(11) NOT NULL, 'Vote' double NOT NULL DEFAULT '0',
'AddedDate' timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
'Cancel' int(1) NOT NULL DEFAULT '0',
'CancellationDate' timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
'Promotion' int(1) NOT NULL DEFAULT '0',
'PromotionDate' timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
PRIMARY KEY ('Action','Location') ) ENGINE=InnoDB DEFAULT CHAR-
SET=latin1;
```

Utilizzo: questa tabella viene utilizzata per memorizzare tutti i valori relativi ad una coppia action → location inserite, in un determinato istante. Per una coppia si memorizza il numero di volte che un utente poteva votarla, il numero di voti positivi e negativi che ha ricevuto, il voto effettivo, la data in cui è stata aggiunta e eventualmente la data di promozione.

Tabella Action_voted

Figura 4.8: Db Ontology: Action_voted.

Struttura della tabella 'Action_voted'

```
CREATE TABLE IF NOT EXISTS ‘Action_voted‘ (
‘Action‘ varchar(30) NOT NULL,
‘Location‘ varchar(30) NOT NULL,
‘Username‘ varchar(30) NOT NULL COMMENT ’nome dell”utente che ha votato’,
‘Vote‘ int(1) NOT NULL DEFAULT ’1’ COMMENT ’1-voto, 0-voto cancellato’,
‘VoteDate‘ timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
‘CancellationDate‘ timestamp NOT NULL DEFAULT ’0000-00-00 00:00:00’,
PRIMARY KEY (‘Action‘,‘Location‘,‘Username‘) ) ENGINE=InnoDB DEFAULT
CHARSET=latin1;
```

Utilizzo: questa tabella viene utilizzata per memorizzare le coppie action →l ocation inserite da un utente in un determinato istante. Si inserisce lo username di chi ha votato, la coppia votata, una variabile il cui valore è 0 o 1 a seconda se il voto è ancora attivo o è stato cancellato dall'utente e la data della votazione.

Tabella Action_voted_historical

Figura 4.9: Db Ontology: Action_voted_historical.

Struttura della tabella ‘Action_voted_historical‘

```
CREATE TABLE IF NOT EXISTS ‘Action_voted_historical’ (
‘Action‘ varchar(30) NOT NULL,
‘Location‘ varchar(30) NOT NULL,
‘Username‘ varchar(50) NOT NULL,
‘Vote‘ int(1) NOT NULL,
‘Date‘ timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (‘Action‘,‘Location‘,‘Username‘,‘Vote‘,‘Date‘) ) ENGINE=InnoDB
DEFAULT CHARSET=latin1;
```

Utilizzo: tabella che rappresenta uno storico delle votazioni fatte realtiva alle coppie action→location.

Tabella Location

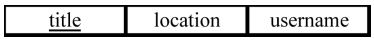


Figura 4.10: Db Ontology: Location.

Struttura della tabella ‘Location’

```
CREATE TABLE IF NOT EXISTS ‘Location‘ (
‘title‘ varchar(200) NOT NULL,
‘location‘ varchar(30) NOT NULL,
‘username‘ varchar(50) NOT NULL,
PRIMARY KEY (‘title‘) ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Utilizzo: tabella che associa ad un titolo del task la location da raggiungere, nel caso il task intenda visitare/raggiungere/vedere un determinato luogo.

Tabella Voted

Figura 4.11: Db Ontology: Voted.

Struttura della tabella 'Voted'

```
CREATE TABLE IF NOT EXISTS 'Voted' (
'object' varchar(30) NOT NULL,
'username' varchar(30) NOT NULL,
PRIMARY KEY ('object','username') ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Utilizzo: tabella che tiene conto se un utente ha votato o meno un item/action. Se un utente ha già votato per una data cosa, a livello interfaccia client non gli viene più richiesta la votazione. Tutto ciò al fine di non risultare troppo stressanti nell'utilizzo del sistema.

Tabella 'user'

Figura 4.12: Db Ontology: Users.

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',
'rank' double NOT NULL DEFAULT '0.1' COMMENT 'rank per la proporzione del voto dell"utente nell"ontologia'
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Utilizzo: tabella che memorizza gli utenti registrati. La tebella esisteva già, è stata modificata in seguito. E' stato aggiunto il campo 'rank' di un utente. Inizilamente il valore di ingresso è 0,1. Successivamente con l'utilizzo del sistema di votazioni, l'utente aumenta il rank o lo diminuisce; in ogni caso il valore minimo è stato fissato a 0,1. Perciò 0,1 è il livello peggiore di autorevolezza di un utente, mentre 1 è il livello massimo.

Il database è stato utilizzato per gestire nuove asserzioni inserite dagli utenti (votate) nella forma item-location o action-location. Ancora una volta, e' neccessario tener presente che un'asserzione è inserita inizialmente nel database; dopo aver ottenuto un numero di voti positivi verrà promossa in maniera definitiva in ontologia; nel caso, invece, abbia ottnuto un numero alto di voti negativi verrà cancellata dalla base di dati.

4.2.4.2 Asserzioni: inserimento e cancellazione

Seguendo l'architettura multilayer definita nel paragrafo 3.1, descriveremo l'implementazione della soluzione lato server partendo dal livello Web e via via scendendo verso il livello più protetto, il DAO (Data Access Object). Al fine di facilitare la comprensione, utilizzeremo anche le varie interfacce client create appositamente; descriveremo il modo in cui client e server comunicano per permettere ad un normale utente l'inserimento di asserzioni, la visualizzazione di tutte quelle inserite e la votazione.

Nel client è stato dapprima creata un'interfaccia che permettesse all'utente di gestire le proprie asserzioni, di visualizzare i propri voti e di inserirli.

A questa sezione si ha accesso, attraverso il menù pricipale dell'applicazione, dopo aver effettuato il login (figura 4.13).

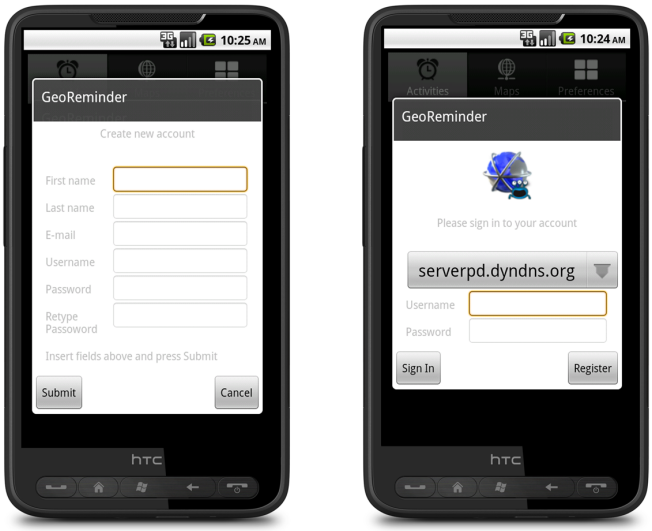


Figura 4.13: Login e registrazione.

Dopo aver effettuato l'acceso, si raggiungono le schermate di figura 4.14.

Figura 4.14: Schermate pricipali.

La prima immagine rappresenta la schermata iniziale, la seconda illustra la prima parte del menù e la terza la seconda parte, che si raggiunge premendo il tasto "More".

Premendo successivamente il tasto "View Assertions", si ha accesso alla sezione (figura 4.15) che permette appunto di gestire le proprie asserzioni.

Figura 4.15: Asserzioni item/action –> Location.

La prima schermata è stata creata solo a scopo informativo; avvisa l'utente di ciò che deve tenere conto nell'utilizzo di questa funzionalità. Essa ricorda che la presente sezione viene utilizzata per aiutare il sistema che si sta utilizzando per comprendere

meglio le esigenze dell'utente espresse in forma di "reminder"; inoltre, che nel primo tab della prima schermata è possibile inserire asserzioni nella forma item → location, mentre nel secondo tab asserzioni del tipo action → location. Premendo il tasto "Go on!", si accede ad una schermata che da la possibilità di inserire asserzioni. Nel primo tab, "item-location", è possibile visualizzare la lista di asserzioni votate dall'utente. Come si può vedere dalla seconda immagine, le tre asserzioni votate sostengono che il pallone e il pepe si possono trovare al supermarket. Per ottenere la lista, il client contatta il server, creando un thread nel seguente modo:

Listing 4.8: Thread getAssertions

```
public static Thread getAssertions(final Handler handler, final Context context) {
        final Runnable runnable = new Runnable()
        {
                public void run() {
                Log.i(TAG, "request␣assertionsList_itemlocation␣for␣the␣user");
                List<SingleItemLocation> result =
                runHttpGetUserSingleItemLocation("/ontology/viewActionLocationVoted"
                        ,null, context);
                sendResult(result, handler, context);
                }
        };
        // start group list request return
                //NetworkUtilities.startBackgroundThread(runnable);
}
```

Tale thread, utilizza la funzione runHttpGetUserSingleItemLocation() per inviare la richiesta al server di restituzione della lista di asserzioni votate dall'utente.

Listing 4.9: Funzione runHttpGetUserSingleItemLocation()

```
private static List<SingleItemLocation> runHttpGetUserSingleItemLocation(
        final String method,final ArrayList<NameValuePair> params, Context c){
        List<SingleItemLocation> result = new LinkedList<SingleItemLocation>();
        DefaultHttpClient newClient = NetworkUtilities.createClient();
        String query = (params == null) ? "" : URLEncodedUtils.format(params,
                        "UTF-8");
        AccountUtil util = new AccountUtil();
        HttpGet request = new HttpGet(NetworkUtilities.SERVER_URI + "/"+
                util.getUsername(c) + method + query);
        request.addHeader("Cookie", "sessionid="+util.getToken(c));
        // send the request to network
        HttpResponse response=NetworkUtilities.sendRequest(newClient,request);
        // if we cannot connect to the server
        if (!HttpResponseStatusCodeValidator.isValidRequest(response.
                        getStatusLine().getStatusCode())){
                Log.i(TAG, "Cannot␣connect␣to␣server␣with␣code␣"+ response
                        .getStatusLine().getStatusCode());
                return null;
                // return null if there's problem with connection
        }
        try {
```

```
22                  // parsing XML message
                    result = AssertionsHandler.parseUserItemsInLocation(response.getEntity()
24                          .getContent()); return result;
            } catch (IllegalStateException e) {
26                  Log.i(TAG, "Illegal State");
                    e.printStackTrace();
28                  return result;
            } catch (IOException e) {
30                  Log.i(TAG, "IOException");
                    e.printStackTrace();
32                  return result;
            } catch (SAXException e) {
34                  Log.i(TAG, "SAXException");
                    e.printStackTrace();
36                  return result;
            }
38  }
```

Questa funzione, invierà una richiesta http al server, contattando la risorsa /ontology/ viewActionLocationVoted . Vediamo ora come gestisce la richiesta lato server tale risorsa del pacchetto Web.

Listing 4.10: Risorsa del server viewItemLocationVoted()

```
    @GET
2   @Path("/viewItemLocationVoted")
    @Consumes("application/xml")
4   @Produces("application/xml")
    public List<SingleItemLocation> viewItemLocationVoted(
6           @PathParam("username") String userid,@CookieParam("sessionid")
                    String sessionid)
8   {
            log.info("Request to view item-location voted from user " + userid +
10                  ", session "+ sessionid);
            System.out.println(" Sono in OntologyListener /viewItemLocationVoted");
12          return OntologyManager.getInstance().retrieveAllItemLocationVoted(userid);
    }
```

A sua volta, questa risorsa del pacchetto Web, seguendo il modello architetturale multilayer del server deve preoccuparsi di comunicare con il livello Manager. A questo livello viene quindi inoltrata la richiesta alla funzione retrieveAllItemLocationVoted(), descritta sotto.

Listing 4.11: Funzione retrieveAllItemLocationVoted()

```
1   /**
    * retrieve all item-location voted by this user
3   * @param userid unique UUID of the user
    * @return list that contains all item-location entered by the user
5   */ public List<SingleItemLocation> retrieveAllItemLocationVoted(String userid)
    {
7           System.out.println(" Sono in Ontology manager retrieveAllItemLocationVoted");
```

```
        return OntologyDatabase.istance.getAllItemLocationVoted(userid);
}
```

Come si vede, tale risorsa non fa altro che deviare richiesta alla risorsa getAllItemLocationVoted() del livello DAO, il più interno, che si preoccuperà di interrogare il DB al fine di recuperare la lista di asserizioni item-action dell'utente.

Listing 4.12: Funzione getAllItemLocationVoted()

```
 /**
* Get all item location voted by a specific userID
* @param userID unique UUID of the user
* @return list containing all ItemLocation voted from the user
*/
public static List<SingleItemLocation> getAllItemLocationVoted(final String userID){
        ArrayList<SingleItemLocation> itemLocationList=
                new ArrayList<SingleItemLocation>();
        Connection conn= (Connection) dbManager.dbConnect();
        String selectQuery="Select * from Item_foundIn_Loc join Item_voted on "
                + "Item_foundIn_Loc.Item = Item_voted.Item and " +
                                "Item_foundIn_Loc.Location = Item_voted.Location"+
                                        "where Item_voted.Username='"+userID+"'" +
                                                " and Item_voted.vote=1";
        System.out.println(" Sono in OntologyDatabase");
        System.out.println(selectQuery);
        QueryStatus qs=dbManager.customSelect(conn, selectQuery);
        ResultSet rs=(ResultSet)qs.customQueryOutput;
        try{
                while(rs.next()){
                String item = rs.getString("Item_foundIn_Loc.Item");
                String location = rs.getString("Item_foundIn_Loc.Location");
                double voto = voteUpToDateItemLocation(item,location);
                itemLocationList.add(new SingleItemLocation(item,location ,
                        rs.getString("Item_foundIn_Loc.Username") ,
                        rs.getInt("Item_foundIn_Loc.N_views") ,
                        rs.getInt("Item_foundIn_Loc.N_votes"),voto));
                }
        }catch(SQLException sqlE){
                        //TODO
        }finally{
                dbManager.dbDisconnect(conn);
        }
        return itemLocationList;
}
```

Questa funzione restituisce una lista di oggetti SingleItemLocation. Ognuno di essi rappresenta una singola asserzione votata dall'utente. L'oggetto viene definito dalla seguente classe:

Listing 4.13: Classe SingleItemLocation

```
@XmlRootElement
```

```
public class SingleItemLocation {
        public String item;
        public String location;
        public String username;
        public int n_views;
        public int n_votes;
        public double vote;

        public SingleItemLocation(){
                super();
        }
        /**
         * Constructor for this class
         * @param item
         * @param location :location where the item can be found in
         * @param username :username of the user that has entered the
                *Item-Location couple
         * @param n_visualizzazioni :number of time that the Item-Location
                *has been visualized
         * @param n_voti :number of people that have voted for this Item-Location
        */

        public SingleItemLocation(String item,String location, String username,
                        int n_views,int n_votes,double vote) {
                this.item = item;
                this.location = location;
                this.username = username;
                this.n_views = n_views;
                this.n_votes = n_votes;
                this.vote = vote;
        }
}
```

La comunicazione tra client e server avviene tramite messaggi in formato XML. Per questo motivo la lista di oggetti SingleItemLocation verrà successivamente serializzata in formato XML e quindi al client verrà spedito un messaggio di una forma simile a

Listing 4.14: Lista di SingleItemLocation serializzata

```
<collection>
        <singleItemLocation>
                <item>pallone</item>
                <location>supermarket</location>
                <username>mirco</username>
                <n_views>1</n_views>
                <n_votes>1</n_votes>
                <vote>0.11</vote>
        </singleItemLocation>
        <singleItemLocation>
                <item>pepe</item>
                <location>supermarket</location>
```

```
13                  <username>mirco</username>
                    <n_views>1</n_views>
15                  <n_votes>1</n_votes>
                    <vote>0.11</vote>
17          </singleItemLocation>
            <singleItemLocation>
19                  <item>sale</item>
                    <location>supermarket</location>
21                  <username>mirco</username>
                    <n_views>1</n_views>
23                  <n_votes>1</n_votes>
                    <vote>0.11</vote>
25          </singleItemLocation>
</collection>
```

Il client ottenuto il risultato, esegue il parser del file XML ottenuto, estrae le informazioni neccessarie e le visualizza a schermo.

Nel caso, consederassimo, invece, le asserzioni nella forma action-location allora al client ritornerà un XML diverso, simile a:

Listing 4.15: Oggetto SingleActionLocation serializzato

```
<collection>
2           <singleActionLocation>
                    <action>pranzare</action>
4                   <location>ristorante</location>
                    <username>mirco</username>
6                   <n_views>1</n_views>
                    <n_votes>1</n_votes>
8                   <vote>0.1</vote>
            </singleActionLocation>
10 </collection>
```

In questo caso l'oggetto SingleActionLocation è un oggetto diverso da SingleItemLocation ed è definto dalla seguente classe,

Listing 4.16: Classe SingleActionLocation

```
@XmlRootElement
2  public class SingleActionLocation {
            public String action;
4           public String location;
            public String username;
6           public int n_views;
            public int n_votes;
8           public double vote;

10          public SingleActionLocation(){
                    super();
12          }

14          /**
```

```
        * Constructor for this class
16      * @param action
        * @param location :location where the action can be made
18      * @param username :username of the user that has entered
                *the Action-Location couple
20      * @param n_visualizzazioni :number of time that the Action-Location
                *has been visualized
22      * @param n_voti :number of people that have voted for this Action-Location
        */
24      public SingleActionLocation(String action,String location, String username,
                int n_views,int n_votes,double vote)
26      {
                this.action = action;
28              this.location = location;
                this.username = username;
30      this.n_views = n_views;
        this.n_votes = n_votes;
32      this.vote = vote;
        }
34  }
```

Vediamo ora, cosa succede nel momento in cui inseriamo un'asserzione. Tratteremo solamente il caso dell'inserimento di item-location, tralasciando la forma action-location, dal momento che il meccanismo è molto simile.

Nel tab "item-location", come si può notare dalla figura, attraverso il menù, si ha la possibilità di inserire un' asserzione, semplicemente premendo il tasto "Create Assertion". Possiamo aggiornare la lista visualizzata nel caso ci siano dei problemi di sincronizzazione tra server e client oppure ritornare alla schermata iniziale con il tasto "Back".

Nel caso si voglia inserire un'asserzione, si deve preme il tasto "Create Assertion". Comparirà di conseguenza la schermata mostrata in figura 4.16.

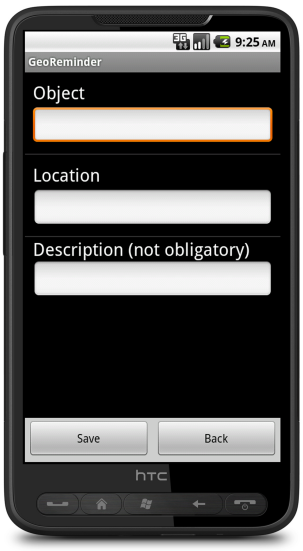


Figura 4.16: Inserimento di un asserzione item–>location.

Dalla figura 4.16 si notano il campo Object, in cui si inserisce l'item/oggetto, il campo location destinato a contenere la locazione e Description, opzionale, per contenere una breve descrizione dellinformazione che l'utente intende dare.

In questo caso l'utente, dopo aver compilato tutti i campi obbligatori, premendo il tasto "Save", a sua insaputa, crea un oggetto SingleItemLocation (visto in precedenza), lo serializza e successivamente crea un thread al fine di comunicare al server (questa volta la richiesta è di tipo POST) la volontà di inserire quell'asserzione.

Listing 4.17: Thread createItemLocation

```
public static Thread createItemLocation(
        final SingleItemLocation toAdd, final Handler handler, final Context context){
                final Runnable runnable = new Runnable(){
        public void run(){
                String body = AssertionsHandler.formatSingleItemLocation(toAdd);
                Log.i(TAG,body);
                final boolean result=runHttpPost("/ontology/addItemInLocationObject"
                        ,null,body, context);
                if (handler == null || context == null) {
                        return;
                }
                handler.post(new Runnable() {
                public void run() {
                if (context instanceof Create_Assertion_item)
                        ((Create_Assertion_item) context).finishSave(result);
                else if (context instanceof Create_Assertion_item_NoDb)
                        ((Create_Assertion_item_NoDb) context).finishSave(result);
                else
                        ((Vote_ont_db) context).finishSave(result);
```

```
20                  }
                    });
22          }};
            return NetworkUtilities.startBackgroundThread(runnable);
24  }
```

Il server, pervenuta la richiesta dal client, risponde con la risorsa addItemInLocationObject(), recuperando il file inviato in formato XML, contentente l'asserzione:

Listing 4.18: Livello Web: addItemInLocationObject()

```
@POST
2 @Path("/addItemInLocationObject")
@Consumes("application/xml")
4 //@Produces("application/xml")
/* @return null if the couple item-location is already
6           /* entered in ontology file or in the db, a SingleItemLocation otherwise
*/
8
public void addItemInLocationObject(@PathParam("username") String userid,
10          @CookieParam("sessionid") String sessionid, SingleItemLocation itemLocation){
            log.info("Request␣to␣add␣item-location␣from␣user␣" + userid + "session␣"+
12                  sessionid);
            log.info("Add"+ itemLocation.item
14                  + "in␣location" + itemLocation.location);
            OntologyManager.getInstance().addItemInLocation(userid, itemLocation.item
16                  ,itemLocation.location);
}
```

che contatterà a sua volta addItemInLocation() del livello Manager. Essa controllerà se la coppia item-location è già presente in ontologia. In caso positivo, l'asserzione non viene inserita nell'ontology database; altrimenti si procede con l'inserimento nella base dati inoltrando la richiesta al livello DAO.

Listing 4.19: Livello Manager: addItemInLocationObject()

```
1 /**
* Enter in the database the couple item-location (this item can be
3 * found in this location) * @param user username of the user that enter
* the couple item-location
5 *
* @param item the item that has been entered
7 * @param location the location in wich the item can be found in
* @return an object SingleItemLocation if the couple action-location
9 * has been entered with success
*/
11 public SingleItemLocation addItemInLocation(String user, String item,String location)
{
13          String ite = item.toLowerCase();
            String loc = location.toLowerCase();
15          ite.replaceAll("␣", "_");
            loc.replaceAll("␣", "_");
```

```
17          List<String> queryList = new ArrayList<String>();
            // list of inferred search query string
19          queryList.addAll(OntologyReasoner.getInstance().getSearchQuery(ite));
            List<Hint> result = new LinkedList<Hint>();
21          // list of search result
            for (String query : queryList){
23                  //se c'è già nell'ontologia la coppia item-location torno null,
                    //in quanto non lo inserisco nel db
25                  if (query.compareToIgnoreCase(loc)==0){
                            log.info("item-location: " + item+"-"+location
27                                  +"già in ontologia");
                            System.out.println("item-location già dentro l'ontologia");
29                          return null;
                    }
31          }
            // Se arriva qua significa che la coppia item-location non è già
33          // presente nell'ontologia, allora la inserisco nel db
            return OntologyDatabase.istance.addItemInLocation(user,item,location);
35  }
```

Questa funzione addItemInLocation(), richiamerà addItemInLocation() del livello DAO, che inserirà l'asserzione nel db.

Listing 4.20: Livello Dao: addItemInLocation()

```
1   /**
    * Enter in the database the couple item-location (this item can be
3   * found in this location) * @param user username of the user that enter
    * the couple item-location
5   * @param item the item that has been entered
    * @param location the location in wich the item can be found in
7   * @return
    */
9   public static SingleItemLocation addItemInLocation(String user,
            String item,String location) {
11          SingleItemLocation itemLocationToReturn=null;
            item = item.toLowerCase();
13          location = location.toLowerCase();
            Connection conn= (Connection) dbManager.dbConnect();
15          log.info("Connected to the db");
            //Starting transaction
17          QueryStatus qs=dbManager.startTransaction(conn);
            if(qs.execError){
19                  //TODO decide what to do in this case (transaction not started)
                    log.error(qs.explainError());
21                  qs.occourtedErrorException.printStackTrace();
                    System.out.println("Error during transaction,item-location not added");
23                  log.error("Error during transaction,item-location not added");
                    dbManager.dbDisconnect(conn);
25                  return null;
            }
27          String insertQuery="Insert into Item_foundIn_Loc(Item,Location,Username,
```

```
N_views,N_votes) values ('"+item+"','"+location +"','"+user+"',1,0)";
        System.out.println(insertQuery);
        qs=dbManager.customQuery(conn, insertQuery);
        if(qs.execError){
                log.error(qs.explainError());
                qs.occourtedErrorException.printStackTrace();
                System.out.println("ERRORE: item-location già inseriti nel db");
                //Rolling back
                dbManager.rollbackTransaction(conn);
                log.error("Error during Item-Location adding... Assertion not added");
                dbManager.dbDisconnect(conn);
                // Essendo già inserite do solo il mio voto
                System.out.println("// Essendo già inserite do solo il mio voto");
                itemLocationToReturn = OntologyDatabase.istance
                        .voteItemByAdding(user,item,location);
                return itemLocationToReturn;
        }
        log.info("Assertion added!");
        dbManager.commitTransaction(conn);
        dbManager.dbDisconnect(conn);
        itemLocationToReturn = OntologyDatabase.istance.voteItem(user,item,location);
        return itemLocationToReturn;
}
```

Analizzando il codice si può vedere l'utilizzo di voteItemByAdding(). E' una funzione molto importante e svolge un ruole centrale nell'inserimento di un'asserzione. Postiamo, innanzitutto il codice; subito dopo descriveremo a parole il suo funzionamento.

Listing 4.21: Funzione voteItemByAdding()

```
public SingleItemLocation voteItemByAdding(String user,String item
        , String location)
{
        Connection conn= (Connection) dbManager.dbConnect();
        SingleItemLocation itemLocationToReturn = null;
        DateUtils date = new DateUtils();
        String insertDate = date.now();
        //Controllo che l'utente non abbia già votato (true ha già votato,
        //il voto è 1, false non ha votato o ha voto 2 )
        if (!hasVotedItemLocation1(user,item,location))  {
        //l'utente non ha votato positivamente
        /*Seleziono il rank dell'utente */
        //String rank= userRank(user);
        //System.out.println("rank utente: "+ rank);
        double rank = userRankDouble(user);
        System.out.println("rank utente: "+ rank);
        //mi trovo il voto per l'item-location
        double voto = voteUpToDateItemLocation(item,location);
        voto = voto+rank;
        if (!hasVotedItemLocation2(user,item,location))   {
                //l'utente non ha votato per quest'asserzione
```

```
            //Starting transaction
            QueryStatus qs=dbManager.startTransaction(conn);
            if(qs.execError){
                //TODO decide what to do in this
                //case (transaction not started)
                log.error(qs.explainError());
                qs.occourtedErrorException.printStackTrace();
                System.out.println("Error during transaction
                                ,Vote_for_item_not_added");
                log.error("Error during transaction starting
                                ,Vote for item not added");
                dbManager.dbDisconnect(conn);
                return null;
            }
            // Inserisco nella tabella Item_Voted che l'utente
            //user ha votato per una item-Location
            String insertQuery="Insert into Item_voted(Item,Location
                        ,Username,Vote,Date)
                values('"+item+"','"+location+"','"+user+"',1,'"
                +insertDate+"')";
            System.out.println(insertQuery);
            qs=dbManager.customQuery(conn, insertQuery);
            if(qs.execError){
                log.error(qs.explainError());
                qs.occourtedErrorException.printStackTrace();
                //Rolling back
                dbManager.rollbackTransaction(conn);
                System.out.println("Error during vote item '"+item+"'
                                ,'"+location+"' "+"... not added in Item_voted");
                log.error("Error during vote item '"+item+"','"+location+"'"
                        +"... not added in Item_voted");
                dbManager.dbDisconnect(conn);
                return null;
            }
            // Inserisco nella tabella Item_voted_historical che
            //l'utente user ha votato per una item-Location
            insertQuery="Insert into Item_voted_historical(Item,Location
                        ,Username,Vote,Date) values ('"+item+"','"+location+"'
                                ,'"+user+"',1,'"+insertDate+"') ";
            System.out.println(insertQuery);
            qs=dbManager.customQuery(conn, insertQuery);
            if(qs.execError){
                log.error(qs.explainError());
                qs.occourtedErrorException.printStackTrace();
                //Rolling back
                dbManager.rollbackTransaction(conn);
                System.out.println("Error during vote item '"+item+"','"
                +location+"'"+"... not added in Item_voted_historical");
                log.error("Error during vote item '"+item+"','"+location+"'"
                +"... not added in Item_voted_historical");
                dbManager.dbDisconnect(conn);
```

```
                        return null;
                }
                String updateQuery="update␣Item_foundIn_Loc␣set␣Vote=("+voto+"),
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣N_votes=(N_votes␣+␣1),N_views=(N_views␣+1)
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣where␣Item='"+item+"'␣and␣Location='"+location+"'";
                System.out.println(updateQuery);
                qs=dbManager.customQuery(conn, updateQuery);
                if(qs.execError){
                        log.error(qs.explainError());
                        System.out.println("Error␣during␣update␣Vote
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣in␣Item_foundIn_Loc␣operation,aborting␣operation");
                        qs.occourtedErrorException.printStackTrace();
                        //Rolling back
                        dbManager.rollbackTransaction(conn);
                        log.error("Error␣during␣update␣Vote␣in␣Item_foundIn_Loc
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣operation,aborting␣operation");
                        dbManager.dbDisconnect(conn);
                        return null;
                }
                log.info("Vote␣from␣"+user+"␣for␣(item:␣"+item+"␣location:␣"
                        +location+")...␣added!");
                String selectQuery= "select␣*␣from␣Item_foundIn_Loc␣where␣Item='"
                        +item+"'and␣Location='"+location+"'";
                System.out.println(selectQuery);
                qs=dbManager.customQuery(conn,selectQuery);
                ResultSet rs=(ResultSet)qs.customQueryOutput;
                try{
                        //Creating SingleItemLocation object from data inserted
                        //into the database
                        if(rs.next()){
                                itemLocationToReturn = new SingleItemLocation(
                                        rs.getString("Item"),rs.getString("Location"),
                                        rs.getString("Username"),rs.getInt("N_views"),
                                        rs.getInt("N_votes"),rs.getDouble("Vote"));
                        }else{
                                //Rolling back
                                dbManager.rollbackTransaction(conn);
                                dbManager.dbDisconnect(conn);
                                return null;
                        }
                }catch(SQLException sqlE){
                        //TODO manage exception
                        sqlE.printStackTrace();
                        //Rolling back
                        dbManager.rollbackTransaction(conn);
                        dbManager.dbDisconnect(conn);
                }
                dbManager.commitTransaction(conn);
                dbManager.dbDisconnect(conn);
                promotionItem(item,location);
        }
```

```
124         else{
                    //l'utente ha votato negativamente per l'asserzione
126                 //mi trovo il voto negativo per l'item-location
                    double votoNeg = voteNegUpToDateItemLocation(item,location);
128                 votoNeg = votoNeg - rank;
                    //Starting transaction
130                 QueryStatus qs=dbManager.startTransaction(conn);
                    if(qs.execError){
132                         //TODO decide what to do in this case (transaction not started
                            log.error(qs.explainError());
134                         qs.occourtedErrorException.printStackTrace();
                            System.out.println("Error during transaction starting,
136 ____________________________________Vote for item not added");
                            log.error("Error during transaction starting
138 ____________________________________,Vote for item not added");
                            dbManager.dbDisconnect(conn);
140                         return null;
                    }
142                 String updateQuery= "update Item_voted set Vote = 1,Date='"
                            +insertDate+"'where Item ='"+item+"' and Location='"
144                         +location+"'and Username='"+user+"'";
                    System.out.println(updateQuery);
146                 qs=dbManager.customQuery(conn, updateQuery);
                    if(qs.execError){
148                         log.error(qs.explainError());
                            System.out.println("Error during update Vote in Item_voted
150 ____________________________________operation,aborting operation");
                            qs.occourtedErrorException.printStackTrace();
152                         //Rolling back
                            dbManager.rollbackTransaction(conn);
154                         log.error("Error during update Vote in Item_voted operation
____________________________________,aborting operation");
156                         dbManager.dbDisconnect(conn);
                            return null;
158                 }
                    // Inserisco nella tabella Item_voted_historical che l'utente user
160                 //ha votato per una item-Location
                    String insertQuery="Insert into Item_voted_historical(Item,Location
162 _________________________,Username,Vote,Date)
_________________________values('"+item+"','"+location+"','"+user+"',1
164 ____________________________________,'"+insertDate+"')";
                    System.out.println(insertQuery);
166                 qs=dbManager.customQuery(conn, insertQuery);
                    if(qs.execError){
168                         log.error(qs.explainError());
                            qs.occourtedErrorException.printStackTrace();
170                         //Rolling back
                            dbManager.rollbackTransaction(conn);
172                         System.out.println("Error during vote item '"+item+"','"
                            +location+"'"+",not added in Item_voted_historical");
174                         log.error("Error during vote item '"+item+"','"
```

```
                                        +location+"' "+",not added in Item voted historical");
176                         dbManager.dbDisconnect(conn);
                            return null;
178                 }
                    //tolgo il voto negativo e aggiungo quello positivo
180                 //non cambio il numero di visualizzazioni
                    updateQuery= "update Item foundIn Loc set Vote = ("+voto+"),
182 ____________________N_votes=(N_votes + 1),N_votes_neg=(N_votes_neg-1),
____________________VoteNegative=("+votoNeg+") where Item='"+item+"'
184 ____________________________________________and Location='"+location+"'";
                    System.out.println(updateQuery);
186                 qs=dbManager.customQuery(conn, updateQuery);
                    if(qs.execError){
188                         log.error(qs.explainError());
                            System.out.println("Error during update Vote
190 ____________________in Item foundIn Loc operation,aborting operation");
                            qs.occourtedErrorException.printStackTrace();
192                         //Rolling back
                            dbManager.rollbackTransaction(conn);
194                         log.error("Error during update Vote in Item foundIn Loc
____________________________________operation,aborting operation");
196                         dbManager.dbDisconnect(conn);
                            return null;
198                 }
                    log.info("Vote from "+user+" for (item: "+item+" location: "
200                         +location,added!");
____________________String selectQuery="select*from Item foundIn Loc where Item='"+
202 ____________________+")item+"' and Location='"+location+"'";
____________________System.out.println(selectQuery);
204 ____________________qs=dbManager.customQuery(conn,selectQuery);
____________________ResultSet rs=(ResultSet)qs.customQueryOutput;
206 ____________________try{
____________________//Creating SingleItemLocation object from data inserted
208 ____________________//into the database
____________________if(rs.next()){
210 ____________________________________itemLocationToReturn = new SingleItemLocation(
____________________________________rs.getString("Item"),rs.getString("Location"),
212 ____________________________________rs.getString("Username"),rs.getInt("N_views"),
____________________________________rs.getInt("N_votes"),rs.getDouble("Vote"));
214 ____________________}else{
____________________________________//Rolling back
216 ____________________________________dbManager.rollbackTransaction(conn);
____________________________________dbManager.dbDisconnect(conn);
218 ____________________________________return null;
____________________}
220 ____________________}catch(SQLException sqlE){
____________________//TODO manage exception
222 ____________________sqlE.printStackTrace();
____________________//Rolling back
224 ____________________dbManager.rollbackTransaction(conn);
____________________dbManager.dbDisconnect(conn);
```

```
226 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣}
    ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣dbManager.commitTransaction(conn);
228 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣dbManager.dbDisconnect(conn);
    ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣promotionItem(item,location);
230 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣}
    ␣␣␣␣␣␣␣␣}
232 return␣itemLocationToReturn;
    }
```

Questa funzione svolge un compito molto importante. Precisamente controlla se l'utente non ha mai votato, oppure se ha votato negativamente o positivamente. Di seguito specificheremo cosa succede in tutti e tre i casi.

1. L'UTENTE NON HA MAI VOTATO Se l'utente non ha mai votato, come si può vedere nel primo ramo dell'if allora si recupera il rank attuale dell'utente attraverso userRankDouble() e il rank attuale dell'asserzione attraverso la funzione VoteUpToDateItemLocation(). Calcolati i due valori, si registra il voto dell'utente nella tabella Item_voted inserendo (Item, Location, Username, Vote, Date). Il campo Vote può assumere valore 1 in caso sia una votazione positiva o 2 in caso sia negativa. Essendo questa funzione richiamata nel momento in cui si vuole inserire un'asserzione assegnerò alla variabile Vote il valore 1. Si memorizza il fatto nello storico, ossia nella tabella Item_voted_historical inserendo (Item, Location, Username, Vote, Date). Ovviamente Date è il campo contente la data in cui è stata fatta la votazione. Successivamente, si aggiorna la tabella Item_foundIn_location, in particolare il record relativo alla coppia item → location votata. Si aggiorna il rank_positivo (sommandogli il rank dell'utente), si incrementa di 1 N_views (numero di volte in cui viene visualizzata l'asserzione) e N_votes (numero di voti positivi). Infine si utilizza la funzione promoteItem() che verifica se tale asserzione ha superato la soglia sufficiente per essere promossa in ontologia.

2. L'UTENTE HA GIA' VOTATO POSITIVAMENTE Se l'utente ha già votato positivamente, la funzione non fa nulla, in quanto un utente può esprimere solamente una preferenza a riguardo di una coppia item/action → location. Grazie a questo si garantisce che una asserzione venga promossa in maniera "onesta". Se questa caratteristica non esistesse un'utente invaliderebbe il sistema votando più volte un'asserzione falsa, come ad esempio "pane → farmacia". Quest'ultima entrerebbe in ontologia e verrebbe utilizzata da tutti gli utenti del sistema.

3. L'UTENTE HA GIA' VOTATO NEGATIVAMENTE Se l'utente ha già votato negativamente, allora si toglie il voto negativo e si terrà conto di quello positivo. Per cui, si calcola il nuovo rank_negativo dell'asserzione recuperando il rank_negativo attuale e si sottrae il rank dell'utente. Successivamente si calcola il rank_positvo recuperando il rank_positivo attuale dell'asserzione e sommando il rank dell'utente. Dopodichè, si aggiorna Item_voted memorizzando che l'utente non ha più votato negativamente ma positivamente (il campo vote ora è uguale a 1 e non più a 2). Si aggiorna, inoltre,

Item_voted_historical registrando la votazione. Successivamente, si aggiorna la tabella Item_foundIn_location, in particolare il record relativo alla coppia item → location votata. In particolare si modifica il rank_negativo e il rank positivo, incremento di 1 N_votes (numero di voti positivi) e decrementando di 1 N_votes_neg. Infine si utilizza la funzione promoteItem() che verifica se tale asserzione ha superato la soglia sufficiente per essere promossa in ontologia.

In modo schematico, attraverso diagrammi di sequenza (figura 4.17 e 4.18), vediamo i vari passaggi appena descritti.

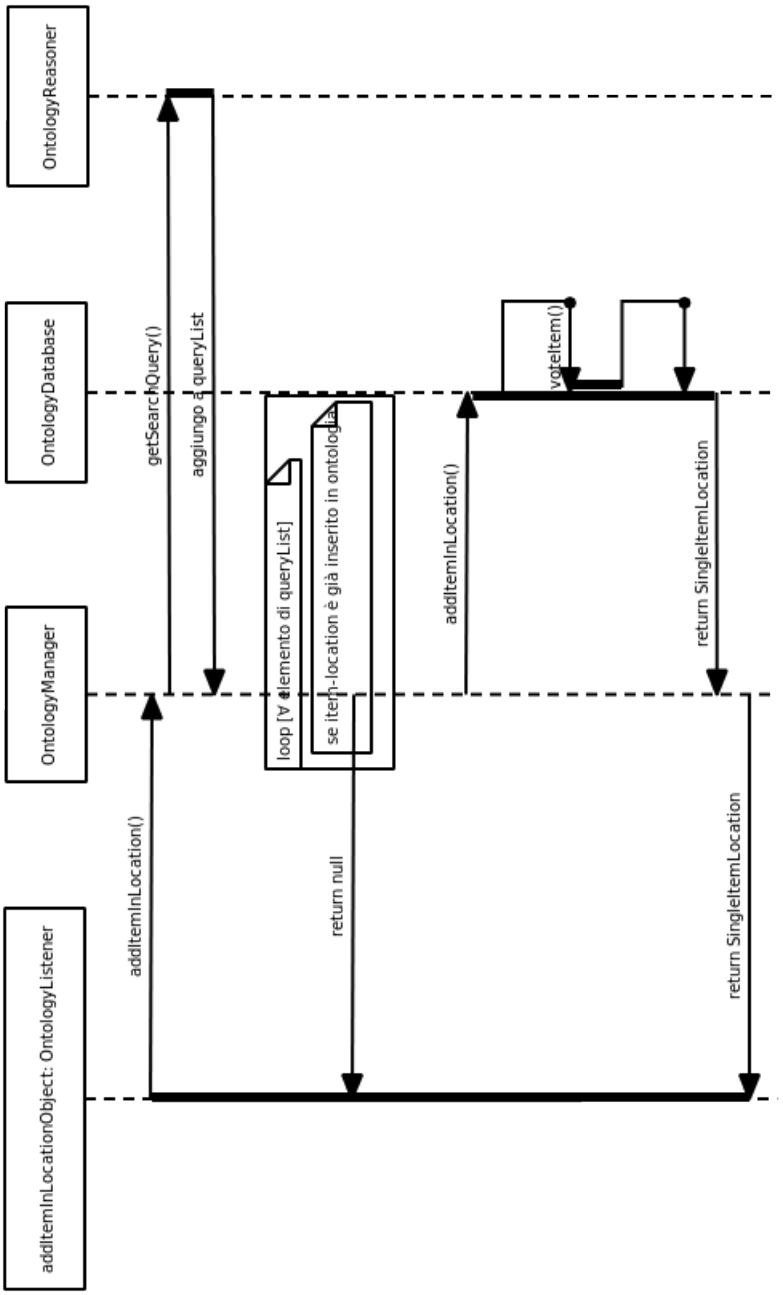


Figura 4.17: Diagramma UML di addItemInLocation().

Mentre per il caso di voteItem(), il diagramma di sequenza (figura 4.18) è

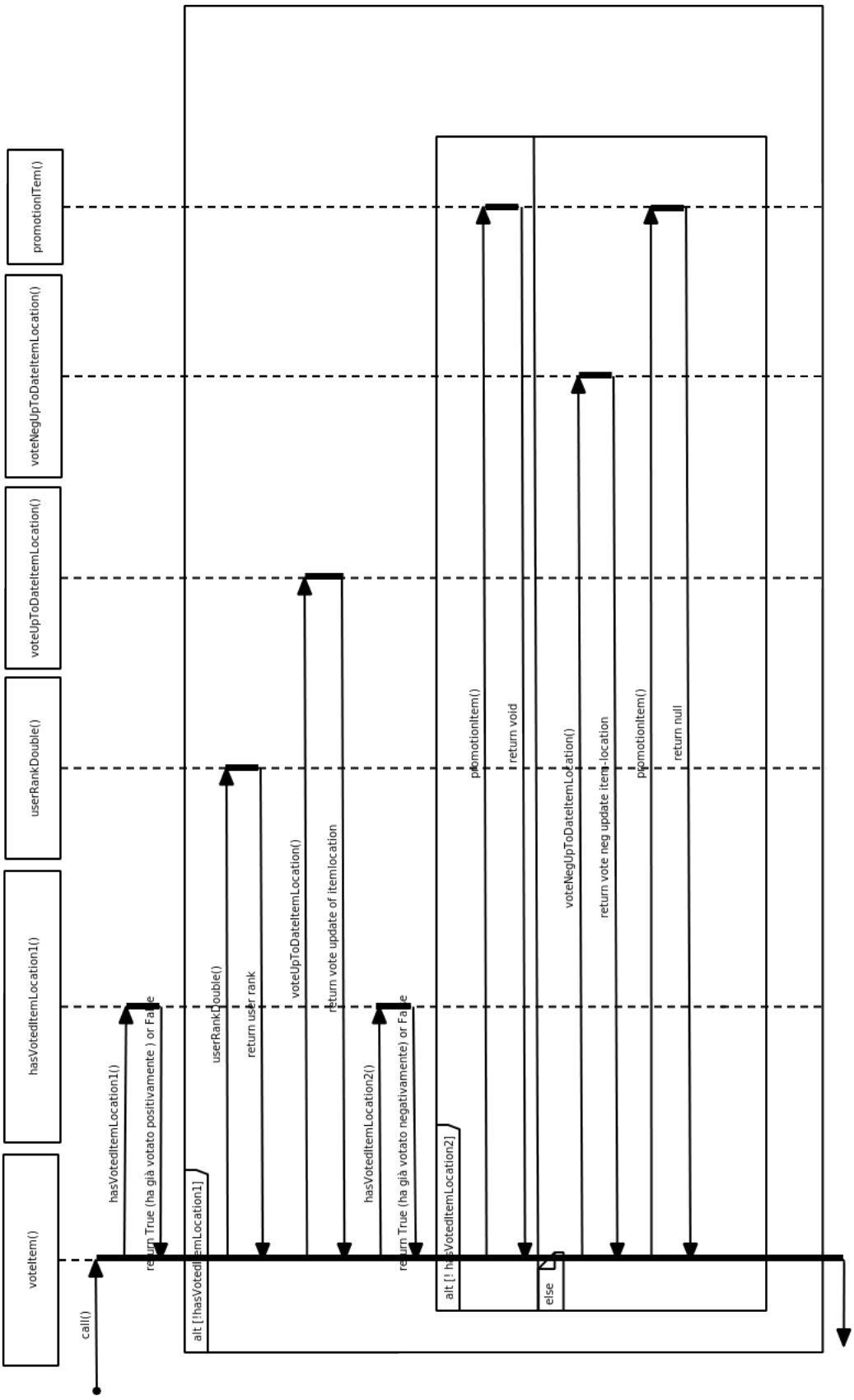


Figura 4.18: Diagramma UML di voteItem().

Vediamo ora, cosa succede, nel caso di un'eliminazione, o meglio della cancellazione del voto di una regola, da parte di un partecipante.

Se l'utente decide di eliminare un'asserzione dalla sezione View_Assertions, causerà una votazione negativa per quella coppia item → location, in quanto è come se l'utente esprimesse un parere negativo.

Per cui nel momento in cui preme il tasto "DeleteAssertions" nella schermata "Details Assertion" (figura 4.19), come si vede dalla figura sottostante

Figura 4.19: Dettagli di un'asserzione.

il client richiemarà nel server la risorsa deleteVoteForItemLocationObject() del pacchetto Web. Vediamo, seguendo il codice Java, cosa succede.

Listing 4.22: Livello Web: deleteVoteForItemLocationObject()

```
@POST
@Path("/deleteVoteForItemLocationObject")
@Consumes("application/xml")
//cancella il voto di un utente
public boolean deleteVoteForItemLocationObject(
        @PathParam("username")
        String userid,@CookieParam("sessionid") String sessionid,
                SingleItemLocation itemLocation) {
        log.info("Request to delete vote for
        item location: " + itemLocation.item);
        return OntologyManager.getInstance()
                .deleteVoteForItemLocation(userid,itemLocation.item
                        ,itemLocation.location);
}
```

Questa funzione non fa altro che richiamere deleteVoteForItemLocation() del livello Manager.

Listing 4.23: Livello Manager: deleteVoteForItemLocationObject()

```
/**
* delete vote for an item-location by the user userid
* @param userid unique UUID of the user
* @param item
* @param location
* @return true if the vote is correct deleted
*/ public boolean deleteVoteForItemLocation(String userid
                ,String item,String location) {
                log.info("Request to delete vote for item-location: "
                + item+" - "+location);
                return OntologyDatabase.istance
                        .deleteVoteForItemLocation(userid,item,location);
}
```

A sua volta verrà inoltrata la richiesta alla funzione deleteVoteForItemLocation() del livello DAO. Come per la funzione voteItemByAdding() vediamo postiamo prima il codice e poi vediamo la sua funzione.

Listing 4.24: Livello Dati: deleteVoteForItemLocationObject()

```
public boolean deleteVoteForItemLocation(String userid,String item,String location)
{
        Connection conn= (Connection) dbManager.dbConnect();
        item = item.toLowerCase();
        location = location.toLowerCase();
        //Starting transaction
        QueryStatus qs=dbManager.startTransaction(conn);
        if(qs.execError){
                //TODO decide what to do in this case (transaction not started)
                log.error(qs.explainError());
                qs.occourtedErrorException.printStackTrace();
                System.out.println("Error during transaction starting,
                Delete vote for item-location not done");
                log.error("Error during transaction starting,Delete vote
                for item-location not done");
                dbManager.dbDisconnect(conn);
                return false;
        }
        DateUtils cancDate = new DateUtils();
        String cancellationDate = cancDate.now();
        String rank= userRank(userid);
        System.out.println("rank utente: "+ rank);
        //String rankVoted = userRankVotedItem(userid,item,location);
        String updateQuery= "update Item_voted set Vote = 2,Date='"+
                cancellationDate+"' where Item ='"+item+"' and Location='"
                +location+"' and Username='"+userid+"'";
        System.out.println(updateQuery);
        qs=dbManager.customQuery(conn, updateQuery);
        if(qs.execError){
                log.error(qs.explainError());
                System.out.println("Error during delete Vote in Item_voted
```

```
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣operation,aborting␣operation");
                    qs.occourtedErrorException.printStackTrace();
                    //Rolling back
                    dbManager.rollbackTransaction(conn);
                    log.error("Error␣during␣delete␣Vote␣in␣Item_voted␣operation
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣,aborting␣operation");
                    dbManager.dbDisconnect(conn);
                    return false;
            }
            //aggiorno la tabella Item_voted_historical inserendo
            //che l'utente ha cancellato il voto
            String insertQuery="Insert␣into␣Item_voted_historical(Item,
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Location,Username,Vote,Date)␣values␣('"+item+"','"
                            +location+"','"+userid+"',2,'"+cancellationDate+"')␣";
            System.out.println(insertQuery);
            qs=dbManager.customQuery(conn, insertQuery);
            if(qs.execError){
                    log.error(qs.explainError());
                    qs.occourtedErrorException.printStackTrace();
                    //Rolling back
                    dbManager.rollbackTransaction(conn);
                    System.out.println("Error␣during␣delete␣vote␣item␣'"
                            +item+"','"+location+"'␣"
                            +"...␣in␣Item_voted_historical");
                    log.error("Error␣during␣delete␣vote␣item␣'"+item+"','"
                            +location+"'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"+"...␣in␣Item_voted_historical");
                    dbManager.dbDisconnect(conn);
                    return false;
            }
            updateQuery= "update␣Item_foundIn_Loc␣set␣N_votes_neg=(N_votes_neg
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+␣1),N_votes=(N_votes−1)␣where␣Item='"+item+"'␣and␣Location='"
                            +location+"'";
            System.out.println(updateQuery);
            qs=dbManager.customQuery(conn, updateQuery);
            if(qs.execError){
                    log.error(qs.explainError());
                    System.out.println("Error␣during␣delete␣Vote␣operation␣in
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Item_foundIn_Loc,aborting␣operation");
                    qs.occourtedErrorException.printStackTrace();
                    //Rolling back
                    dbManager.rollbackTransaction(conn);
                    log.error("Error␣during␣delete␣Vote␣operation␣in␣Item_foundIn_Loc
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣,aborting␣operation");
                    dbManager.dbDisconnect(conn);
                    return false;
            }
            dbManager.commitTransaction(conn);
            dbManager.dbDisconnect(conn);
            cancellationItem(item,location);
            return true;
```

```
83 }
```

Tale funzione, in maniera molto simile alle altre che abbiamo visto, aggiorna la tabella Item_voted. In particolare il campo "Vote" del record relativo all'asserzione item → location viene impostato a 2 (indicando una votazione negativa). Aggiorna la tabella Item_voted_historical e la tabella item_foundInLocation modificando il numero di voti negativi e il numero di voti positivi. Il voto positivo non viene aggiornato in quanto una cancellazione di una asserzione non indurra sicuramente una promozione. Si ricalcolerà il voto effettivo solo in presenza di un inseriemento. Infine, verrà richiamata anche la funzione che controlla se una asserzione abbia ricevuto un numero di voti negativi sufficiente ad essere cancellata dal database (che verrà descritta più tardi quando tratteremo le promozioni e cancellazioni).

Le considerazioni fatte per il tab item-location, ricordiamo, sono valide anche per la sezione action-location, per cui omettiamo di riprodurre gli stessi algoritmi e schemi.

4.2.4.3 Cosa cambia nell'inserimento di un task?

Nell'inserimento di un task (figura 4.20) il sistema contolla se la frase inserita come titolo del task contiene delle parole conosciute dal sistema.

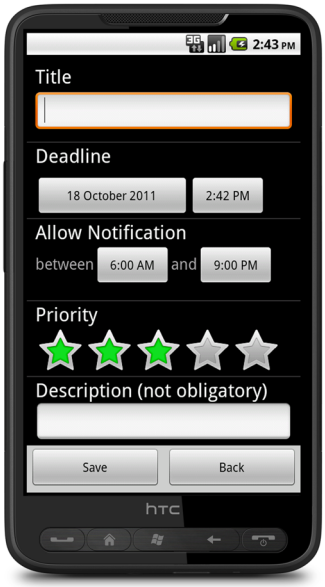


Figura 4.20: Inserimento di un task.

Nel momento in cui, l'utente preme "Save", l'intero task viene inviato al server. Il server controlla, parola per parola, se il titolo contiene delle parole a lui conosciute. Data una parola del task, accede per primo all'ontologia, alla ricerca dell'esistenza di qualche regola che contenga la parola del task. Succesivamente, controlla se qualcuno

ha inserito un asserzione con item o action uguali alla parola del task. A questo punto comunica il risultato della ricerca, sia in ontologia che nell'ontology database, al client. Nel caso abbia riscontrato qualche match, il sistema chiede all'utente di specificare le proprie preferenze.

Per esempio, nel caso l'utente inserisca "prendere il pane", il sistema analizza parola per parola in task. Supponiamo che per le parole "prendere" e "il" non abbia trovato niente. Ora, contolla in ontologia e nel database se esiste un match per "pane". In entrambi i casi recupera le location corrispondenti, presenti nell'ontologia e nel database (dedotte da asserzioni votate votate da utenti) e li comunica nella forma mostrata in figura 4.21.

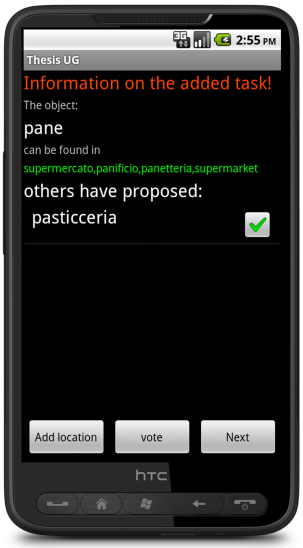


Figura 4.21: Informazioni sul task appena inserito.

Come vediamo dalla figura, con riferimento all'esempio che abbiamo fatto, l'item pane è stata trovata sia nell'ontologia, sia nel database. Infatti si specifica che il pane per l'ontologia attuale può essere trovato nel supermercato, nel panificio, nella panetteria e nel supermarket, mentre per il database può essere trovato anche in pasticceria. Qui, si permette all'utente di effettuare una votazione su delle asserzioni inserite da altri utenti oppure di crearne delle altre. Se l'utente in questa schermata vota la location pasticceria, voterà un'asserzione inserita da un altro utente. Se invece non desidera votarla allora, deselezionerà il checkbox; se vuole inserire altre location e votarle basterà cliccare sul tasto "Add Location" e il client chiederà di inserire la locazione che fino a quel momento non è stata mai inserita.

Vediamo ora più in dettaglio cosa succede tra il client e il server.

Subito dopo aver inserito un task, il client contatta la risorsa checkInOntologyDb() del server per informarsi se un dato item o action è presente in ontologia o/e nel database.

Listing 4.25: Thread checkInOntologyDb()

```
public static Thread checkInOntologyDb(final String title,
        final Handler handler, final Context context)
{
        final Runnable runnable = new Runnable()
        {
                public void run()
                {
                ArrayList<NameValuePair> nameValuePairs =
                        new ArrayList<NameValuePair>();
                nameValuePairs.ad(new BasicNameValuePair("title",title));
                List<Item> result = runHttpGetUsercheckInOntologyDb(
                        "/ontology/checkInLocationDb",
                        nameValuePairs, context);
                sendResult_item(result, handler, context);
                }
        };
        //start group list request
        return NetworkUtilities,startBackgroudThread(runnable);
}
```

Lato server, risponderà la risorsa checkInOntologyDb() presente nel pacchetto Web.

Listing 4.26: Livello Web: checkInOntologyDb()

```
@GET
@Path("/checkInOntologyDb")
@Consumes("application/xml")
@Produces("application/xml")
public List<Item> checkInOntologyDb(@PathParam("username")
                String userid,@QueryParam("title") String title
                ,@CookieParam("sessionid") String sessionid) {
        log.info("Request to check title from user " + userid +
        ", session "+ sessionid);
        List<Item> queryListItem = new ArrayList<Item>();
        // list of inferred search query string
        /*
        * Controllo se c'è corrispondenza nel db Location(cioè mi ritorna
        * una stringa diversa da ""), se si torno un item=5,
        * altrimenti controllo parola per parola
        */
        String location = OntologyManager.getInstance()
                .findLocation(userid, title.toLowerCase());
        if (!location.equalsIgnoreCase("")){
                Item item= new Item();
                item.name=title;
                item.itemActionType=2; //location
                item.ontologyList="";
                item.dbList="";
                item.nScreen=5;
                queryListItem.add(item);
```

```
27          } else {
                List<String> needs = new ArrayList<String>();
29              String[] words = title.split("␣");
                needs.addAll(Arrays.asList(words));
31              // remove duplicates by using HashSet
                HashSet<String> needsfilter = new HashSet<String>(needs);
33              needs.clear();
                needs.addAll(needsfilter);
35              for (String o : needs) {
                    List<String> ontList = new ArrayList<String>();
37                  List<String> dList = new ArrayList<String>();
                    int u;
39                  boolean isLocation=false;
                    Item item= new Item();
41                  item.name=o;
                    List<String> list1 = OntologyManager.getInstance()
43                      .viewLocationForItemVoted(userid,o);
                    System.out.println("list1:"+ list1);
45                  List<String> list2 = OntologyManager.getInstance()
                        .viewLocationForActionVoted(userid,o);
47                  System.out.println("list2:"+ list2);
                    boolean hasVoted = OntologyManager.getInstance()
49                      .hasVoted(userid,o);
                    //l'utente non ha votato niente per quell'item
51                  ontList.addAll(OntologyReasoner.getInstance()
                        .getSearchQuery(o));
53                  for (String l : ontList)
                        item.ontologyList=item.ontologyList+l+",";
55                      if(!item.ontologyList.equals(""))
                        {
57                      u=item.ontologyList.lastIndexOf(",");
                        item.ontologyList=item.ontologyList.substring(0, u);
59                      }
                        if (!ontList.isEmpty())
61                      if(OntologyReasoner.getInstance().isItem(o))
                        {
63                  item.itemActionType=1;//è un Item
                        dList.addAll(OntologyManager.getInstance()
65                          .viewLocationForItem(userid,o));
                        }else{
67                      item.itemActionType=0;//è un Action
                        dList.addAll(OntologyManager.getInstance()
69                          .viewLocationForAction(userid,o));
                        }else{
71                      dList.addAll(OntologyManager
                            .getInstance().viewLocationForItem(userid,o));
73                      if (!dList.isEmpty()){
                            item.itemActionType=1;//è un Item
75                      }else{
                            dList.addAll(OntologyManager.getInstance()
77                          .viewLocationForAction(userid,o));
```

```
                        if (!dList.isEmpty())
79                          item.itemActionType=0;//è un Action
                        }
81                  }
                    for (String l : dList)
83                      item.dbList=item.dbList+l+",";
                    if (!item.dbList.equals(""))
85                  {
                        u=item.dbList.lastIndexOf(",");
87                      item.dbList=item.dbList.substring(0, u);
                    }
89                  if (ontList.isEmpty())
                        if (dList.isEmpty())
91                      {
                            //non ho trovato niente né in
93                          //ontologia né in db
                            //quindi non lo inserisco
95                          //nella lista di ritorno
                            item.nScreen=4;
97                      }else{
                            //non ho trovato niente in
99                          //ontologia ma qualcosa nel db
                            item.nScreen=3;
101                         queryListItem.add(item);
                        }else{
103                         if (OntologyReasoner.getInstance()
                                .isLocation(o))
105                         {
                            item.itemActionType=2;//è una Location
107                         item.ontologyList="";
                            item.dbList="";
109                         item.nScreen=5;
                            queryListItem.add(item);
111                     } else {
                            if (dList.isEmpty())
113                         {
                        //ho trovato qualcosa in ontologia
115                     //, ma niente nel db
                            item.nScreen=2;
117                         queryListItem.add(item);
                            }else{
119                         //ho trovato sia in ontologia sia in db
                                item.nScreen=1;
121                             queryListItem.add(item);
                            }
123                     }
                    }
125             }

127             /*
                * se l'utente ha già votato per quell'item ritorno
```

```
129                     * nScreen=5 che significa
                        * che non compare più la schermata di votazione perchè
131                     * ha già votato
                        */
133                     //se l'utente ha deciso che non vuole più votare per quell'item
                else{
135                     if(!list1.isEmpty()){
                                item.itemActionType=1; //item
137                             item.dbList="";
                                System.out.println("!list1.isEmpty()");
139                     }else{
                                item.itemActionType=0; //action
141                             item.dbList="";
                                System.out.println("item.itemActionType="
143                                     +item.itemActionType);
                                System.out.println("item.dbList="+item.dbList);
145                     }
                        ontList.addAll(OntologyReasoner.getInstance()
147                             .getSearchQuery(o));
                        item.ontologyList="";
149                     item.nScreen=5;
                        queryListItem.add(item);
151                     }
                }
153     }
        return queryListItem;
155 }
```

Tale funzione controllerà se nel task sono presenti degli item che esistono anche nell'ontologia e/o nel database. Se possibile recupera i luoghi corrispondenti ad ogni item rispettivamente nell'ontologia e nel database (dedotti dalle asserzioni votate).

In caso l'item sia presente in ontologia o/e nel database il client si aspetta che il server lo riferisca attraverso un oggetto creato appositamente per questo tipo di messaggi, l'oggetto Item:

Listing 4.27: Classe Item

```
1 public class Item implements Parcelable{

3       public String name;
        public String nScreen;
5       //il tipo di schermata

7       public String itemActionType;
                //1=item,0=action
9               public String ontologyList;
                public String dbList;
11      }
}
```

Omettiamo, per non rendere pesante la trattazione, i metodi di questo oggetto, creati per garantire le regole dell'incapsulameto e per la serializzazione dell'oggetto stesso.

Una volta serializzato l'oggetto item in formato XML assumerà la seguente forma:

Listing 4.28: Oggetto Item serializzato

```
<item>
    <name>pane</name>
    <nScreen>5</nScreen>
    <itemActionType>0</itemActionType>
    <ontologyList>supermercato, panificio, panetteria,
        supermarket</ontologyList>
    <dbList>pasticceria</dbList>
</item>
```

La risposta del server, in realtà, è costituita da ua lista di item. Ciò è dovuto al fatto che in un task possono esserci ad esempio più item; basti pensare ad una task del tipo "comprare il latte e il sale".

Nella risposta, i tag "name" racchiudono il match trovato. All'interno dei tag "nScreen" viene indicato un valore intero da 1 a 5 che indica in che situazione (in che stato) si trova l'utente relativamente a quel'item.

In paricolare:

- il valore 1 significa che esiste una corrispondenza per l'item sia in ontologia, sia nel database e l'utente non hai mai votato per ciò;
- il valore 2 indica che ho trovato una corrispondenza per l'item solo nell'ontologia e l'utente non hai mai votato per ciò;
- il valore 3 indica che ho trovato una corrispondenza per l'item solo nel database e l'utente non hai mai votato per ciò;
- il valore 4 indica che non ho trovato una corrispondenza dell'item né nell'ontologia né nel database;
- il valore 5 indica che l'utente ha già votato per gli item riscontrati nel task per cui non si richiede più una votazione all'utente;

All'interno dei tag "itemActionType" viene indicato 0 se si tratta di un action, 1 se si tratta di un item e 2 se si tratta di una location. All'interno dei tag "ontologyList" vengono indicate le location trovate nell'ontologia separate da una virgola, mentre nei tag "dbList" si trovano le location trovate nel database ricercate in asserzioni votate dagli altri utenti.

Se in nell'oggetto item compare il tag <nScreen>1</nScreen> allora la schermata che comparirà all'utente sarà simile a quella di figura 4.22.

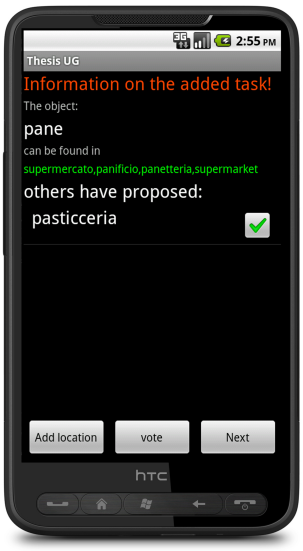


Figura 4.22: Proposta di voto.

Nell'esempio, che si è fatto prima, dell'item "pane", si avverte l'utente che per l'ontologia le asserzioni pane → supermercato, pane → panificio, pane → panetteria e pane → supermarket sono già valide per tutti gli utenti. Al contrario, pane → pasticceria non è ancora entrata in ontologia, ma è stata proposta da uno o più utenti e quindi soggetta ad una votaziona. Per cui l'utente, se ritiene che il pane si possa trovare in pasticceria, voterà tale asserzione selezionando il checkbox a fianco e premendo il tasto vote.

Se in item compare il tag <nScreen>2</nScreen>, allora la schermata che comparirà all'utente è la stessa di prima con la differenza che nessuno ha proposto altre locazioni per l'item.

Se, invece, in item compare il tag <nScreen>3</nScreen>, significa che in ontologia non è stato trovato nulla per cui si avverte l'utente e si visualizza ciò che è stato trovato nel database. Ad esempio per un task "comprare una sveglia", potrebbe comparire la schermata in figura 4.23.

Figura 4.23: Nessun match in ontologia.

Se in item compare il tag <nScreen>4</nScreen>, significa che in ontologia e nel database non è stato trovato nulla. Per cui si richiede all'utente di specificare cosa desidera.

Le possibilità sono tre:

- Ricercare un oggetto da comprare, etc;
- Compiere un azione, come correre, allenarsi, etc;
- Recarsi in un luogo.

In quest'ultimo caso, la schermata, lato client, che comparirà sarà come si vede in figura4.23.

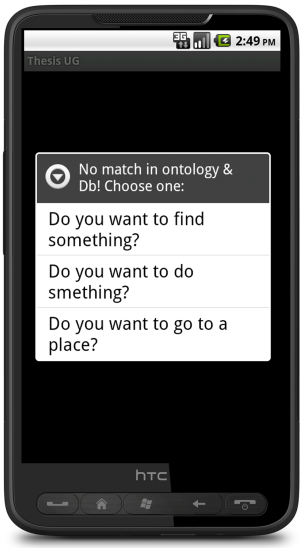


Figura 4.24: Nessun match in ontologia e nel db.

Tutti le domande, daranno origine ad una votazione. Nel primo caso la votazione sarà del tipo item → location, nel secondo action → location e nell'ultimo caso, si specificherà una location per quel task. Per esempio nel caso l'utente scegliesse la prima possibilità, ossia "Do you want to find something?", allora comparirebbe direttamenta la schermata di votazione mostrata in figura 4.25.

Figura 4.25: Votazione nel caso "Do you want to find something)".

La votazione viene effettuata seguendo le linee guida spiegate nel paragrafo "Asserzioni: inserimento e cancellazione".

Infine, se in item compare il tag <nScreen>5 </nScreen>, significa che l'utente ha già espresso un parere riguardo quell'item, per cui non si effettua richiede più una votazione. Ciò è dovuto al fatto che se un utente ha già votata una volta, se desidera modificare il suo voto avrà a disposizione la sezione View Assertion. Nel normale flusso di utilizzo del programma non verrà disturbato.

Vediamo il funzionamento di checkInOntologyDb() attraverso un diagramma di sequenza (Figure 4.26 e 4.27).

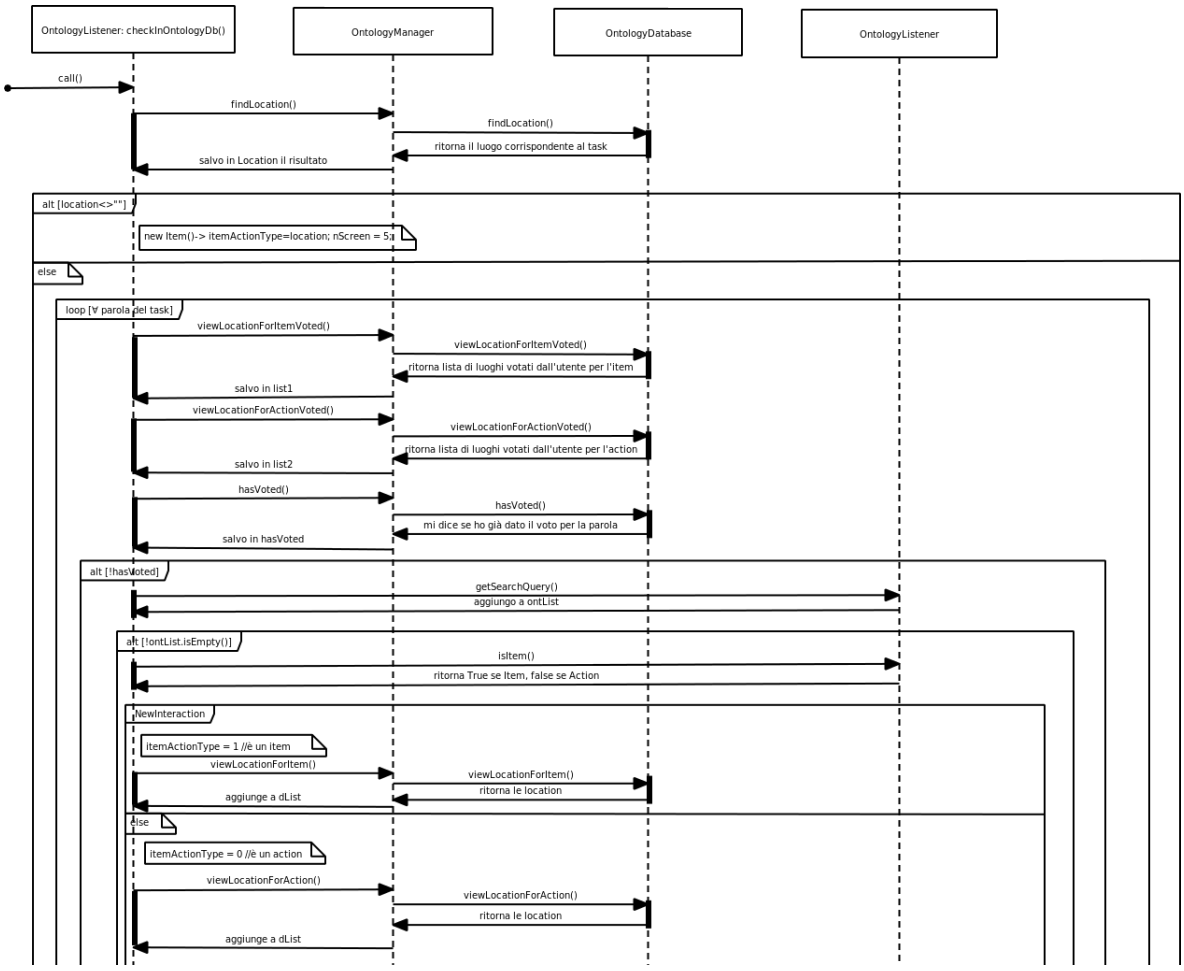


Figura 4.26: Diagramma di sequenza - checkInOntologyDb(), Parte 1..

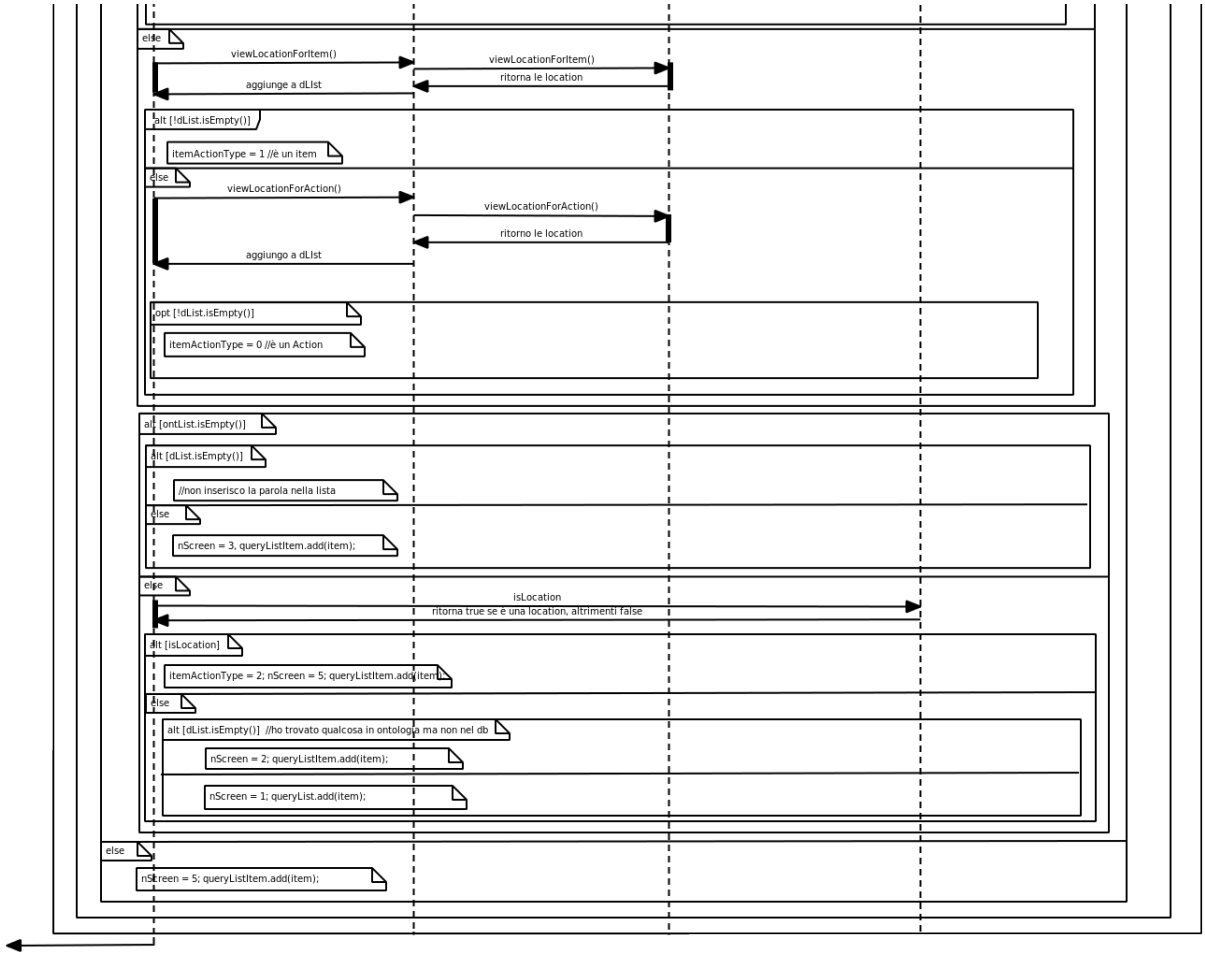


Figura 4.27: Diagramma di sequenza - checkInOntologyDb(), Parte 2.

Nel paragrafo "Asserzioni: inserimento e cancellazione" abbiamo nominato la funzione promotionItem(), che controlla se un'asserizione ha ricevuto un numero di voti positvi sufficiente ad essere promossa. Abbiamo nominato anche cancellationItem() che verifica se ci sono le circostanze per eliminare un'asserzione perchè ha ricevuto un numero di voti negativi elevato. Queste funzioni ovviamente esistono anche per asserzioni del tipo action-locatio, non solo per la forma item-location. Nei prossimi paragrafi, descriveremo il meccanismo di ranking degli utenti, quello di promozione e di cancellazione di un'asserzione in modo dettagliato.

4.2.4.4 Ranking

Il rank, o puntenggio, di un utente rappresenta il grado di affidabilità e autorevolezza di un utente. Al momento dell'iscrizione, gli viene assegnato un rank di default.

Tramite l'inserimento di nuove asserzioni in ontologia o tramite la votazione di quelle inserite da altri si permette all'utente di aumentare il proprio rank.

Il rank di default al momento dell'iscrizione, per convenzione è stato fissato a 0.1, mentre il rank massimo raggiungibile è 1. Un utente che ha un rank di 1 è colui che ha sempre votato e aggiunto asserzione ritenute valide anche per altri utenti e quindi promosse in ontologia. Un suo voto peserà molto nel punteggio finale di un asserzione.

Al fine di detemrinare la promozione di un'asserzione, invece, si considera il numero di visualizzazioni, il numero di utenti che hanno effettivamente votato e il voto (che è dato dalla somma dei rank degli utenti che hanno votato).

Il rank di un utente, nel momento in cui un'asserzione da lui votata viene inserita nell'ontologia (promossa), deve essere aggiornato. Vediamo ora quali meccanismi di aggiornamento rank utente sono stati proposti in sede di progettazione. Alla fine analizzando i pro e contro, ne sceglieremo uno.

Proposta A

Utente che ha inserito (per primo) un'asserzione : aumenta il suo rank di 0.05;

Utente che ha votato per un'asserzione: aumenta il suo rank di 0.025;

Pro: aumento costante per tutti gli utenti.

Contro: non tiene conto del rank dell'utente e si cresce troppo velocemente.

Proposta B

Utente che ha inserito (per primo) un'asserzione : aumenta il suo rank di 1/5 del rank attuale Utente che ha votato per un'asserzione: aumenta il suo rank di 1/10 del suo rank

Pro: l'incremento del rank è proporzionale al rank attuale, (rank + elevato -> incremento + grande).

Contro: non si tiene conto del "resto del mondo" che ha votato.

Proposta C

Utente che ha inserito (per primo) un'asserzione : aumenta il suo rank di 1/5 del rank attuale; Utente che ha votato per un'asserzione: aumenta il suo rank di 1/10 del rank (prima di essere aggiornato) dell'utente che ha inserito l'asserzione; Pro: l'incremento del rank dell'utente che ha inserito l'asserzione è proporzionale al rank attuale, (rank + elevato -> incremento + grande);

Pro/Contro: l'incremento del rank di tutti gli utenti che hanno votato è proporzionale al rank dell'utente che ha inserito (se il rank di quest'ultimo è maggiore di quello che ha votato conviene all'utente che ha votato, se invece è inferiore non conviene all'utente che ha votato);

Contro: non tiene conto della comunità che ha votato;

Proposta D

Sia M = voto_tot/ n_utenti_che_hanno_votato (rank medio degli utenti che hanno votato);Utente che ha inserito(per primo) un'asserzione : aumenta il suo rank di 1/5 di M;Utente che ha votato per un'asserzione : aumenta il suo rank di 1/10 di M.

Pro: incremento in base al rank medio di tutti gli utenti che hanno votato, si tiene conto del peso dei voti del "resto del mondo";

Pro/Contro: se M è maggiore del rank attuale -> incremento maggiore; se M inferiore al rank attuale -> incremento minore;

Al fine di scegliere quale delle proposte implementare, abbiamo creato più di un event-set; ognuno conteneva circa una ventina di eventi casuali. Su questi abbiamo fatto delle simulazioni e analizzando i risultati abbiamo optato per la proposta B.

4.2.4.5 Meccanismi di promozione di un'asserzione

In fase di progettazione abbiamo individuato diversi meccanismi per la promozione di una asserzione. Per meccanismo di promozione si intende la "soglia" che bisogna superare per permettere ad un'asserzione inserita nel database di entrare definitivamente in ontologia e quindi venir inserita nel file owl.

Esaminiamo ora, le varie proposte.

Proposta 1

Promuovere quando l'80% degli utenti ha votato quell'asserzione;

Pro: facile implementazione.

Contro: non tiene conto del rank degli utenti. Un'asserzione per entrare in ontologia deve essere usata e votata dall'80% degli utenti. Non sempre una asserzione è usata da tutti (o quasi), perciò questo meccanismo escluderebbe, dall'inserimento in ontologia, tutte le asserzioni che magari vengono meno usate.

Proposta 2

Promuovere quando almeno l'80% di quelli che hanno visualizzato l'asserzione l'hanno votata e inoltre il numero di quelli che l'hanno votata è >=50% degli utenti.

Pro: tiene conto dell'effettivo utilizzo di una asserzione, l'80% di chi l'ha effettivamente vista deve averla votata.

Contro: non tiene conto del rank degli utenti.

Proposta 3

Promuovere quando il voto >= (50% del numero di utenti) * (media rank utenti(tutti gli utenti anche quelli che non hanno votato));

Pro: tiene conto del rank degli utenti e quindi del voto totale ottenuto.

Contro: non tiene conto del rapporto Nvoti/Nvisualizzazioni.

Proposta 4

Promuovere quando almeno l'80% di quelli che hanno visualizzato l'asserzione l'hanno votata e voto >= (50% del N di utenti)* (media rank utenti);

Pro: tiene conto sia del rank degli utenti che del rapporto Nvoti/Nvisualizzazioni.

Contro: al momento non ne abbiamo individuato.

Anche in questo caso, come per i meccanismi di aggironamento rank utente, al fine di selezionare una delle precedenti proposte abbiamo creato vari event-set contenenti degli eventi del tutto casuali, utili a simulare una situazione reale di un gruppo di utenti che votano, positivamente o negativamente, alcune asserzioni. Dopo alcune simulazioni, abbiamo analizzato i risultati e si è deciso di implmentare la Proposta 4.

4.2.4.6 Promozione di una asserzione e aggiornamento del rank degli utenti

Nel momento in cui un'utente vota positivamente un'asserzione, si deve verificare se il suo voto permette all'asserzione di essere promossa in onologia. A questo scopo esistono le funzioni promotionItem() e promotionAction(), richiamate dopo una votazione di un utente. Le due funzioni sono molto simili, come si può intuire, per questo, come già fatto in precedenza, descriveremo solo quelle relative alle asserzioni nella forma item-location. Vediamo ora cosa succede nel momento in cui viene chiamata la funzione promotionItem().

Listing 4.29: Funzione promotionItem()

```
public void promotionItem(String item,String o){
        int n_views=0;
        int n_votes=0;
        int n_votes_neg=0;
        String username="";
        // username of the user that have insert the assertion
        Connection conn= (Connection) dbManager.dbConnect();
        String selectQuery="Select * from Item_foundIn_Loc
                where Item='"+item+ "'
                and Location='"+o+"' and Promotion=0";
        QueryStatus qs=dbManager.customSelect(conn, selectQuery);
        ResultSet rs=(ResultSet)qs.customQueryOutput;
        try{
                while(rs.next()){
                        n_views=rs.getInt("N_views");
                        n_votes=rs.getInt("N_votes");
                        n_votes_neg=rs.getInt("N_votes_neg");
                        username=rs.getString("Username");
                }
        }catch(SQLException sqlE){
                //TODO
                return;
        }finally{
                dbManager.dbDisconnect(conn);
        }
        System.out.println("N_views="+n_views);
        System.out.println("N_votes="+n_votes);
        //controllo se almeno l'80% degli utenti che hanno
        //visualizzato
        //l'asserzione, l'ha anche votata positivamente
        double rap = (double)n_votes/(double)n_views;
        System.out.println("n_votes/n_views="+rap);
        if (rap >= 0.8)  {
                //trovo il numero degli utenti
                int userNumber = userNumber();
                //trovo la media del rank di tutti gli utenti
                double avgRank = avgRank();
                //trovo il voto della coppia item-location
                //tenendo conto dei rank aggiornati degli
```

```
40                  //utenti in quel preciso istante
                    double vote=voteUpToDateItemLocation(item,o);
42                  //controllo che il
                    //voto>=(50% del N di utenti)*(media rank utenti totali)
44                  double minVote = (userNumber/2)*avgRank;
                    if (vote > minVote) {
46                          OntologyReasoner.updateOntology(item,o,1);//1 significa item
                            //funzione che mi rende evidente nel db la promozione in
48                          //ontologia
                            promoteUpdate(item,o,username);
50                  }
            }
52  }
```

Tale funzione controlla se il numero di voti diviso il numero di viste dell'asserzione è maggiore dell'80%. Se è vero allora si controlla se il voto effettivo dell'asserzione è maggiore del 50% del numero degli utenti per la media rank utenti. In questo caso si avvia l'inserimento della coppia (item, location) in ontologia attraverso il metodo updateOntology() della classe OntologyReasoner e successivamente si aggiorna il rank degli utenti con la funzione promoteUpdate(). Vediamo quest'ultima funzione.

Listing 4.30: Funzione promoteUpdate()

```
public void promoteUpdate(String item,String location
2           ,String username) {
            //salvo in Item_foundIn_Loc che è stato promosso
4           promoteItem(item,location);
            System.out.println("promoteItem(item,location)");
6           //il rank aumeta di 1/5 del proprio rank
            rankUpdateUser(item,location,username);
8           //aggiornare rank degli utenti:tutti gli utenti
            //che hanno voto in
10          //vigore devono aumentare il rank di 1/10 del proprio rank
            rankUpdateItem(item,location);
12          System.out.println("rankUpdate(item,location)");
            //aggiornare rank dell'utente che ha inserito
14          //per primo l'asserzione
            System.out.println("rankUpdateUser(item,location,username)");
16          //elimino da Item_voted il voto di tutti gli utenti per
            //la coppia item-voted(che cmq rimane memorizzato in historical)
18          deleteVoteForItemPromotion(item,location);
            System.out.println("deleteVoteForItemPromotion(item,location)");
20  }
```

PromoteUpdate() aggiorna il rank degli utenti che hanno votato la coppia item-location con rankUpdateUser() e rankUpdateItem(), salva nella tabella Item_foundin_loc che l'asserzione è stata promossa, elimino da Item_voted il voto di tutti gli utenti per la coppia item-voted(che rimane però memorizzato in historical);

Vediamo ora, come rankUpdateUser() e rankUpdateItem() aggiornano il rank degli utenti. Il rank dell'utente, che per primo ha inserito l'asserzione, aumenta il voto di 1/5 del suo rank attuale.

Listing 4.31: Funzione rankUpdateUser()

```
public void rankUpdateUser(String item,String location,
                String username) {
        Connection conn= (Connection) dbManager.dbConnect();
        QueryStatus qs=dbManager.startTransaction(conn);
        if(qs.execError){
                //TODO decide what to do in this case (transaction not started)
                log.error(qs.explainError());
                qs.occourtedErrorException.printStackTrace();
                System.out.println("Error during transaction starting,
                        Delete vote for item-location not done");
                log.error("Error during transaction starting
                        ,Delete vote for item-location not done");
                dbManager.dbDisconnect(conn);
                return;
        }
        String updateQuery = "update User set rank=
                =if (rank+(rank/5)>1, 1, rank+(rank/5) )"
                +System.out.println(updateQuery);
        qs=dbManager.customQuery(conn, updateQuery);
        if(qs.execError){
                log.error(qs.explainError());
                System.out.println("Error during update
                        User rank,aborting operation");
                qs.occourtedErrorException.printStackTrace();
                //Rolling back
                dbManager.rollbackTransaction(conn);
                log.error("Error during update User Rank,aborting operation");
                dbManager.dbDisconnect(conn);
                return;
                }
                dbManager.commitTransaction(conn);
        dbManager.dbDisconnect(conn);
        return;
}
```

Il rank dell'utente, che ha votato quell'asserzione, aumenta il suo voto di 1/10 del suo voto attuale. Vediamo come.

Listing 4.32: Funzione rankUpdateItem()

```
public void rankUpdateItem(String item,String location) {
        Connection conn= (Connection) dbManager.dbConnect();
        //mi salvo i nomi degli utenti che hanno votato per questa asserzione
        List<String> userList= new ArrayList<String>();
        String selectQuery="Select Username from Item_voted where
                Item='"+item+"' and Location='"+location+"' and Vote=1";
        QueryStatus qs=dbManager.customSelect(conn, selectQuery);
        ResultSet rs=(ResultSet)qs.customQueryOutput;
        try{
                while(rs.next()){
                        userList.add(rs.getString("Username"));
```

```
                }
        }catch(SQLException sqlE){
                //TODO
                return;
        }finally{}
        String o=""; ListIterator<String> it = userList.listIterator();
        while(it.hasNext()) {
                o=it.next();
                qs=dbManager.startTransaction(conn);
                if(qs.execError){
                        //TODO decide what to do in this
                        //case (transaction not started)
                        log.error(qs.explainError());
                        qs.occourtedErrorException.printStackTrace();
                        System.out.println("Error␣during␣transaction␣starting
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣,update␣user␣rank␣not␣done");
                        log.error("Error␣during␣transaction␣starting
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣,update␣user␣rank␣not␣done");
                        dbManager.dbDisconnect(conn);
                return;
            }
                //aggiorno il rank
                String updateQuery = "update␣User␣"+"␣set␣rank␣=
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣=if␣(rank+(rank/10)>1,1,rank+(rank/10))" +"␣where
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣username='" +o+"'";
                System.out.println(updateQuery);
                qs=dbManager.customQuery(conn, updateQuery);
                if(qs.execError){
                        log.error(qs.explainError());
                        System.out.println("Error␣during␣update␣User␣rank,aborting
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣operation");
                        qs.occourtedErrorException.printStackTrace();
                        //Rolling back
                        dbManager.rollbackTransaction(conn);
                        log.error("Error␣during␣update␣User␣Rank,
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣aborting␣operation");
                        dbManager.dbDisconnect(conn);
                        return;
                }
        dbManager.commitTransaction(conn);
        }
        dbManager.dbDisconnect(conn);
        return;
}
```

Abbiamo visto che nel caso di una votazione negativa di un'asserzione (o ugualmente nel momento in cui viene eliminata dall'utente dall'interfaccia), viene richiamata la funzione per controllare se il voto negativo consegue una cancellazione dal database. Un'asserzione viene eliminata dal database se almeno l'80% degli utenti che ha visualizzato l'asserzione l'ha anche votata negativamente e se il voto_neg è

maggiore o uguale del 50% degli utenti per media rank utenti totali.

Inoltre il rank dell'utente che l'ha inserita diminuisce di 1/5 del rank attuale, mentre coloro che l'hanno votata diminuiranno il proprio rank di 1/10. Vediamo in dettaglio la funzione cancellationItem(). Vediamo ora dal punto di vista del codice Java cosa succede.

Listing 4.33: Funzione cancellationItem()

```
public void cancellationItem(String item,String o) {
        int n_views=0;
        int n_votes=0;
        int n_votes_neg=0;
        String username="";
        // username of the user that have insert the assertion
        Connection conn= (Connection) dbManager.dbConnect();
        String selectQuery="Select * from Item_foundIn_Loc
                where Item='"+item+"' and Location='"+o+"'
                and Promotion=0";
        QueryStatus qs=dbManager.customSelect(conn,selectQuery);
        ResultSet rs=(ResultSet)qs.customQueryOutput;
        try{
                while(rs.next()){
                        n_views=rs.getInt("N_views");
                        n_votes=rs.getInt("N_votes");
                        n_votes_neg=rs.getInt("N_votes_neg");
                        username=rs.getString("Username");
                }
        }catch(SQLException sqlE){
                //TODO
                return;
        }finally{
                dbManager.dbDisconnect(conn);
        }
        //controllo se almeno l'80%
        //degli utenti che hanno visualizzato
        //l'asserzione, l'ha anche votata negativamente
        double rap = (double)n_votes_neg/(double)n_views;
        System.out.println("n_votes_neg/n_views="+rap);
        if (rap >= 0.8)   {
                //trovo il numero degli utenti
                int userNumber = userNumber();
                //trovo la media del rank di tutti
                //gli utenti
                double avgRank = avgRank();
                //trovo il voto negativo
                double vote_neg=voteNegUpToDateItemLocation(item,o);
                //controllo che il
                //voto>=(50% del N di utenti)*(media rank utenti totali)
                double minVote=(userNumber/2)*avgRank;
                if (vote_neg > minVote) {
                        //funzione che mi rende evidente nel db la cancellazione
                        cancellationItemUpdate(item,o,username);
```

```
45                  }
            }
47  }
```

CancellationItem() utilizza anche cancellationItemUpdate() che aggiorna i rank degli utenti attraverso rankUpdateUserNeg() ed elimina l'asserzione dalla tabella Item_foundInLoc, come vediamo dal seguente codice.

Listing 4.34: Funzioni cancellationItemUpdate() e rankUpdateUserNeg()

```
1   public void cancellationItemUpdate(String item, String o,String username) {

3   //il rank dell'utente che ha inserito l'asserzione
            //diminuisce di 1/5 del proprio rank
5           rankUpdateUserNeg(item,o,username);
            deleteItemFoundInLoc(item,o,username);
7           //elimino da Item_voted il voto di tutti gli utenti per
            //la coppia item-voted(che cmq rimane memorizzato in historical)
9           deleteVoteForItemPromotion(item,o);
    }
11
    //-----------------------------------

13
    public void rankUpdateUserNeg(String item,String location, String username)
15          {
            Connection conn= (Connection) dbManager.dbConnect();
17          QueryStatus qs=dbManager.startTransaction(conn);
            if(qs.execError){
19                  //TODO decide what to do in this case (transaction not started)
                    log.error(qs.explainError());
21                  qs.occourtedErrorException.printStackTrace();
                    System.out.println("Error during transaction starting,
23  ------------------------Delete vote for item-location not done");
                    log.error("Error during transaction starting,
25  ------------------Delete vote for item-location not done");
                    dbManager.dbDisconnect(conn);
27                  return;
            }
29          String updateQuery = "update User set rank=if
    ------------------(rank-(rank/5)<0.1 , 0.1 , rank-(rank/5) )"
31                          + " where username='"+username+"'";
            System.out.println(updateQuery);
33          qs=dbManager.customQuery(conn, updateQuery);
            if(qs.execError){
35                  log.error(qs.explainError());
                    System.out.println("Error during update User rank.. aborting operation");
37                  qs.occourtedErrorException.printStackTrace();
                            //Rolling back
39                          dbManager.rollbackTransaction(conn);
                            log.error("Error during update User Rank.. aborting operation");
41                          dbManager.dbDisconnect(conn);
                            return;
```

```
43          }
            dbManager.commitTransaction(conn);
45          dbManager.dbDisconnect(conn);
            return;
47  }
```

4.3 Limite del sistema: uso esclusivo di una sorgente dati per il reperimento di luoghi finalizzati all'attuazione dei task

Nei prossimi paragrafi, descrveremo dapprima i motivi principali che hanno condotto alla creazione di un sistema social per la generazione di una sorgente dati da cui reperire i luoghi d'interesse, per soddisfare i task degli utente. Successivamente definiremo la soluzione implementata.

4.3.1 Descrizione del problema

Una volta che l'utente abbia inserito i task all'interno del sistema, bisogna avere a disposizione un metodo che permetta di comprendere quali siano i luoghi che possono essere utili per portare a termine i compiti prefissati: una volta individuata la categoria o l'insieme di categorie adatte, serve una o più fonti a cui attingere per poter trovare gli esercizi commerciali da proporre. Qualora si scegliesse una fonte dati nella quale non siano presenti un numero sufficiente di esercizi commerciali, può capitare che l'utente non venga avvertito della possibilità di completare una richiesta nelle vicinanze, rendendo il tutto meno utile di quanto potrebbe.

Da queste considerazioni, si può dedurre che uno dei difetti del sistema è senz'altro il fatto che questi si affidi ad un unico provider esterno per la localizzazione dei luoghi dove poter sodisfare i bisogni richiesti; quindi se sul provider nessuno ha contrassegnato un certo luogo, non è possibile poterlo notificare all'utente, anche se magari potrebbe interessarlo. Per questa ragione si è pensato di realizzare un componente che permettesse di inserire luoghi segnalati dagli utenti.

Un sistema che ben si presta ad essere un candidato per assolvere questo compito è Four Square[92, 93, 94], un software che basa la propria popolarità proprio sul fatto che gli utenti possano segnalare che in un dato posto sia possibile svolgere una certa attività, dando poi anche un giudizio e consigli sullo svolgimento dell'attività in se. Per quanto concerne la nostra applicazione risulta necessario quindi trovare il modo di sfruttare anche altre basi di dati oppure creare un sistema simile a quello di FS in modo da poter avere una propria base informativa sulla quale poter intrapendere attività di datamining.

Nell'attuale implementazione del sistema tali suggerimenti vengono generati a partire da quel che viene ritornato da una query al servizio Maps di Google, quindi l'effettiva utilità dell'applicazione è fortemente legata alla quantità e qualità dei dati ivi inseriti.

È necessaria una riflessione sulla bontà della strategia sin qui utilizzata, in modo da comprendere se e come si possa migliorare la qualità di questo componente vitale del sistema, al fine di migliorare sensibilmente la qualità dell'esperienza d'uso.

4.3.2 Soluzione implementata: sistema social per la generazione del database Places

Per risolvere il problema esposto nel precedente paragrafo, abbiamo sin da subito scartato l'ipotesi "Four Square". Il motivi di tale decisione sono diversi:

- le API[94], che fs mette a disposizione sono ancora in una versione Beta;
- non sarebbe possibile permettere all'utente di inserire dei luoghi private, visibili solo a lui.
- nei termini di utilizzo delle api di fs è espressamente indicato il divieto di usare la loro base dati insieme ad altre sorgenti di informazioni; nel sistema in questione si utilizza Google Maps API, per reperire informazioni dalle basi dati di Google.

Si è per questo, deciso di generare una base dati autonoma, svincolata da ogni altro progetto, ma gestibile direttamente dalla nostra applicazione. Questa base autonoma sarà generata grazie a segnalazioni di luoghi degli utenti. La possibilità di segnalare un luogo è resa possibile dalla sezione "New Places", raggiungibile dal menu principale dell'applicazione (lo vedremo più tardi).

Abbiamo visto che nell'ontologia, un'utente, potrebbe inserire un'asserzione non valida, del tipo, pane → farmacia. Il meccanismo di votazioni, ideato in quel caso, svolgeva quindi una funzione di filtraggio, che cerca di promuovere le asserzioni valide e cancellare quelle false. Anche qui il sistema è molto simile. Un luogo pubblico nel momento in cui viene segnalato da un'utente, diventa visibile per tutti gli utenti. Un utente può votare tale luogo pubblico, dichiarando di volerlo includere nella lista degli hint, per soddisfare il proprio task, di ritorno dal server. Può anche decidere di non votarlo e quindi di non utilizzarlo.

La votazione questa volta viene fatta solamente dalla schermata New Places, ricercando un luogo in base a dei parametri ed infine votandolo. Si è deciso di non permettere la votazione di un luogo dopo l'inserimento di un task, come nel caso delle asserzioni, per non stressare particolarmente l'utente.

Quindi, nel caso dei luoghi pubblici, la votazione è intesa come il "consenso" ad utilizzare anche quel luogo. Diversamente dall'ontologia, qui non si è implementato nessun sistema di ranking dell'utente e nessun sistema di promozione di un luogo. Lo scopo della sezione Places è di poter condividere con altri utenti dei luoghi pubblici, che mancano nelle repository di Google. L'utente può decider di utilizzare questi luoghi inseriri dagli altri utenti o meno.

4.3.2.1 Database Places

Al fine di implementare questa nuova funzionalità, è stato creato un database adatto a mantenere i nuovi luoghi segnalati dagli utenti. Esso deve avere una struttura congrua per tener traccia dei luoghi private e publici con annesse votazioni.

Elenchiamo ora le tabelle create, specificandone la strutture tramite SQL.

Place (public)

title	lat	lng	Street Address	streetNumber	cap	city	user	userGroup	Added Date

Figura 4.28: Db Place: PlacePublic.

Struttura della tabella 'Place'

```
CREATE TABLE IF NOT EXISTS 'Place' (
'title' varchar(100) NOT NULL,
'lat' varchar(20) NOT NULL,
'lng' varchar(20) NOT NULL,
'streetAddress' varchar(100) NOT NULL,
'streetNumber' varchar(10) NOT NULL,
'cap' varchar(6) NOT NULL,
'city' varchar(100) NOT NULL,
'user' varchar(50) NOT NULL,
'userGroup' int(11) NOT NULL DEFAULT '-1' COMMENT '-1 pubblico, 0 privato',
'addedDate' timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY ('title','lat','lng')
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

PlacePrivate

title	lat	lng	Street Address	streetNumber	cap	city	user	userGroup	Added Date

Figura 4.29: Db Place: PlacePrivate.

Struttura della tabella 'PlacePrivate'

```
CREATE TABLE IF NOT EXISTS ‘PlacePrivate‘ (
‘title‘ varchar(100) NOT NULL,
‘lat‘ varchar(20) NOT NULL,
‘lng‘ varchar(20) NOT NULL,
‘streetAddress‘ varchar(100) NOT NULL,
‘streetNumber‘ varchar(10) NOT NULL,
‘cap‘ varchar(6) NOT NULL,
‘city‘ varchar(100) NOT NULL,
‘user‘ varchar(50) NOT NULL,
‘userGroup‘ int(11) NOT NULL,
‘addedDate‘ timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
PRIMARY KEY (‘title‘,‘lat‘,‘lng‘,‘user‘)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Place_private_category

title	lat	lng	category	username

Figura 4.30: Db Place: Place_Private_category.

Struttura della tabella ‘PlacePrivate_category‘

```
CREATE TABLE IF NOT EXISTS ‘PlacePrivate_category‘ (
‘title‘ varchar(100) NOT NULL,
‘lat‘ varchar(20) NOT NULL,
‘lng‘ varchar(20) NOT NULL,
‘category‘ varchar(50) NOT NULL,
‘username‘ varchar(50) NOT NULL, PRIMARY KEY
(‘title‘,‘lat‘,‘lng‘,‘category‘,‘username‘)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Place_category

title	lat	lng	category	username

Figura 4.31: Db Place: Place_Public_category.

Struttura della tabella ‘Place_category‘

```
CREATE TABLE IF NOT EXISTS ‘Place_category‘ (
‘title‘ varchar(100) NOT NULL,
‘lat‘ varchar(20) NOT NULL,
‘lng‘ varchar(20) NOT NULL,
‘category‘ varchar(50) NOT NULL,
‘username‘ varchar(50) NOT NULL COMMENT ’nome dell”utente che ha inserito la categoria’,
PRIMARY KEY (‘title‘,‘lat‘,‘lng‘,‘category‘) ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Place_voted

title	lat	lng	category	username	vote	date

Figura 4.32: Db Place: Place_voted.

Struttura della tabella ‘Place_voted‘

```
CREATE TABLE IF NOT EXISTS ‘Place_voted‘ (
‘title‘ varchar(100) NOT NULL,
‘lat‘ varchar(20) NOT NULL,
‘lng‘ varchar(20) NOT NULL,
‘category‘ varchar(50) NOT NULL,
‘username‘ varchar(50) NOT NULL,
‘vote‘ int(11) NOT NULL COMMENT ’1 voto positivo, 2 voto negativo’,
‘date‘ timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (‘title‘,‘lat‘,‘lng‘,‘category‘,‘username‘) ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Place_voted_historical

title	lat	lng	category	username	vote	date

Figura 4.33: Db Place: Place_voted_historical.

Struttura della tabella ‘Place_voted_historical‘

```
CREATE TABLE IF NOT EXISTS ‘Place_voted_historical‘ (
‘title‘ varchar(100) NOT NULL,
‘lat‘ varchar(20) NOT NULL,
‘lng‘ varchar(20) NOT NULL,
‘category‘ varchar(50) NOT NULL,
‘username‘ varchar(50) NOT NULL,
‘vote‘ int(11) NOT NULL COMMENT ’1 voto positivo, 2 voto negativo’,
‘date‘ timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (‘title‘,‘lat‘,‘lng‘,‘category‘,‘username‘,‘vote‘,‘date‘) ) ENGI-
NE=InnoDB DEFAULT CHARSET=latin1;
```

4.3.2.2 Inserimento di un luogo

Nel momento in cui l'utente ha effettuato l'accesso all'applicazione nel terminale, attraverso la seconda parte del menu può raggiungere la sezione "New Places".

Figura 4.34: Accesso alla sezione "New Place".

A questo punto si trova di fronte ad una schermata divisa in 2 tab, Private Places e Public Places, come si può notare nell'immagine sottostante (figura 4.35).

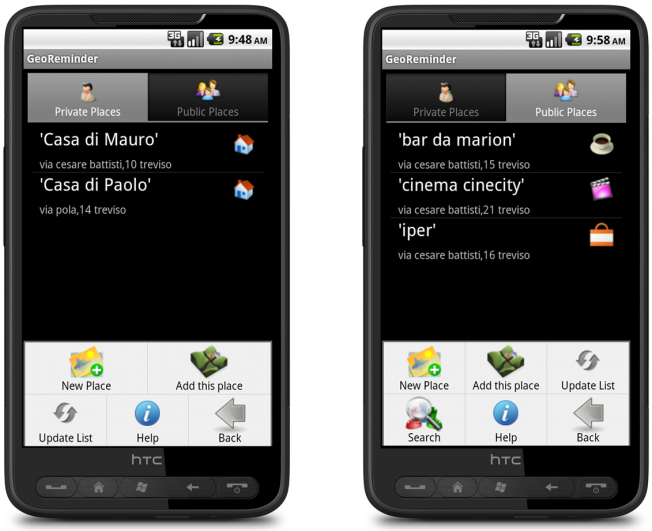


Figura 4.35: Sezione "New Place".

I due tab, distinguono:

- luoghi privati: lista di luoghi privati inseriti dall'utente. Per luogo privato si intende per esempio "casa di Marco", un luogo di interesse puramente personale;

- luoghi publici: lista di luoghi votati o inseriti dall'utente che sono visibili all'interà comunità.

Questa divisione è dovuta al fatto che un utente potrebbe voler inserire luoghi di interesse personale o luoghi che possono risultare utili anche ad altre persone.

Per l'inserimento di un luogo è possibile sfruttare due metodi:

- Inserimento del luogo compilando (figura 4.36) i vari campi neccessari (Nome luogo, indirizzo, CAP, etc.);

Figura 4.36: Inserimento nuovo luogo con textbox.

- Inserimento del luogo sfruttando le coordinate GPS (figura 4.37), aggiungendo il Nome del posto e la cetegoria associate.

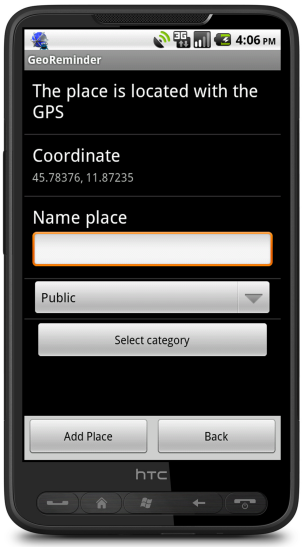


Figura 4.37: Inserimento nuovo luogo con GPS.

In entrambi i casi si sfruttano le api di Google, per convertire un'indirizzo in coordinate o vicerversa.

Nel caso in cui l'utente premi il tasto "New place" appare una schermata con dei textbox da compilare. Questo vale sia per i posti privati che per i posti pubblici.

Nel caso dei luoghi privati una locazione viene sempre aggiunta al database, anche se un'altro utente l'ha aggiunta per se stesso. Nel caso dei pubblici questo non vale. Se un luogo pubblico è già stato inserito da un'altro utente, allora l'eventuale inserimento dello stesso posto da parte di un'altro utente comporta una semplice votazione per quel luogo. Un'altra differenza nell'inserimento dei luoghi pubblici rispetto a quelli privati sta nel fatto, che prima di permetter all'utente di inserire un luogo nel db, si controlla anche che tale non sia presente nelle repository di Google. Ciò viene fatto utilizzando le API di Google Maps; i risultati ritornati vengono analizzati, e se contengono la locazione che l'utente sta cercando di inseirire, il posto non viene aggiunto al database.

Vediamo ora in dettaglio come avviene il processo di inserimento di un luogo, a partire dal client. La risorsa del server, contattata dal terminale, sia nel caso dell'inserimento di un luogo attraverso compilazione dei campi neccessari, sia attraverso coordinate GPS, è sempre la stessa. Lo vedremo meglio quando decriveremo la risorsa addPrivatePlace() del livello DAO.

Descriviamo per comodità solamente il caso dell'insermento di un luogo privato tramite compilazione dei campi.

Dopo aver compilato tutti i campi presenti nell'interfaccia della figura sopra e aver selezionato una o più categorie (figura 4.38)

Figura 4.38: Inserimento categorie nuovo luogo.

il client crea un oggetto PlaceClient, rappresentato dalla seguente classe:

Listing 4.35: Classe PlaceClient

```
1 @XmlRootElement
  public class PlaceClient implements Serializable{
3
        public String title;
```

```
5         public String lat;
          public String lng;
7         public String streetAddress;
          public String streetNumber;
9         public String cap;
          public String city;
11        public String category;
          //stringa che contiene la lista di categorie
13        //separate da una virgola

15        public PlaceClient()
          {
17                super();
          }
19        /**
          * Constructor for this class
21        * @param Place
          */
23
          public PlaceClient(String title,String lat,String lng
25                ,String streetAddress,String streetNumber
                          ,String cap, String city,String category)
27        {
                  this.title = title;
29                this.lat = lat;
                  this.lng = lng;
31                this.streetAddress = streetAddress;
                  this.streetNumber = streetNumber;
33                this.cap = cap;
                  this.city = city;
35                this.category = category;
          }
37        //Omettiamo in questo scritto alcuni metodi
          //di questa classe per non rendere pesante la trattazione
39  }
```

A questo punto il client, invierà l'oggetto PlaceClient, dopo averlo serializzato in XML, al server creando un nuovo thread nel seguente modo

Listing 4.36: Thread createPrivatePlace

```
1   public static Thread createPrivatePlace(final PlaceClient toAdd
          ,final Handler handler, final Context context)
3   {
          final Runnable runnable = new Runnable() {
5                 public void run() {
                          String body = PlacesHandler.formatPlaceClient(toAdd);
7                         Log.i(TAG,body);
                          final boolean result = runHttpPost("/places/privatePlaces"
9                                 ,null,body, context);
                          if (handler == null || context == null) {
11                                return;
```

```
                }
13              handler.post(new Runnable() {
                        public void run() {
15                              ((Create_new_place) context)
                                .finishSaveToAdd(result,toAdd);
17                              }
                        });
19              }};
        return NetworkUtilities.startBackgroundThread(runnable);
21  }
```

L' oggetto serializzato assumerà quindi la seguente forma

Listing 4.37: Oggetto placeClient serializzato

```
1  <placeClient>
        <title></title>
3       <lat></lat>
        <lng></lng>
5       <streetAddress></streetAddress>
        <streetNumber></streetNumber>
7       <city></city>
        <cap></cap>
9       <category></category>
   </placeClient>
```

Ora vediamo cosa succede al server all'arrivo di una richiesta di aggiunta di un luogo. Discutiamo, come abbiamo già detto prima, il caso dei luoghi Privati, dato che quelli pubblici sono molto simili. Le uniche differenze consistono nel controllare che le repository di Google Maps non abbiano già a disposizione tale luogo e che se il posto è già stato inserito da un utente, verrà votato (l'utente consente l'utilizzo di quell luogo nella lista di hint di ritorno).

Lato server a livello web il server risponde con la risorsa PrivatePlace() in Place.

Listing 4.38: Livello Web: addPrivatePlace()

```
   @POST @Path("/addPrivatePlace")
2  @Consumes("application/xml")
   public void addPrivatePlace(@PathParam("username") String userid
4       , @CookieParam("sessionid") String sessionid, PlaceClient place) {
        log.info("Request␣to␣add␣private␣place␣from␣user␣" + userid
6               + ",␣session␣"+ sessionid);
        log.info("Add"+ place.title + "␣" + place.lat + "␣"
8               + place.lng);
        //creo la lista di category, dato che mi arriva una stringa
10      //con le location separate da una virgola
        List<String> categoryList= new LinkedList<String>();
12      String[] words = place.category.split(",");
        categoryList.addAll(Arrays.asList(words));
14      PlacesManager.getInstance().addPrivatePlace(userid,place.title
                ,place.lat,place.lng,place.streetAddress,place.streetNumber
16              ,place.cap,place.city,categoryList)
```

```
}
```

che inotrerà la richiesta alla risorsa addPrivatePlace() al livello Manager

Listing 4.39: Livello Manager: addPrivatePlace()

```
/**
* Enter in the database the private place
*/
public void addPrivatePlace(String user,String lat
        , String lng, String titlePlace,String streetAddressPlace
        , String streetNumberPlace,String capPlace
        , String cityPlace,List<String> category){
        String title = titlePlace.toLowerCase();
        String streetAddress = streetAddressPlace.toLowerCase();
        String streetNumber = streetNumberPlace.toLowerCase();
        String cap = capPlace.toLowerCase();
        String city = cityPlace.toLowerCase();
        title.replaceAll("␣", "_");
        System.out.println("placeManager:"+title+"␣"+streetAddress+"␣"
                +streetNumber+"␣"+cap+""+city);
        PlacesDatabase.istance.addPrivatePlace(user,lat, lng,title
                ,streetAddress,streetNumber,cap,city,category);
}
```

che, di nuovo, inoltrerà la richiesta alla risorsa addPrivatePlace() del livello DAO

Listing 4.40: Livello Dao: addPrivatePlace()

```
public static void addPrivatePlace( String userID, String title1
        ,String lat1, String lng1, String streetAddress1
        ,String streetNumber1, String cap1, String city1
        ,List<String> category )
{
        String title = title1;
        String lat = lat1;
        String lng = lng1;
        String streetAddress=streetAddress1;
        String streetNumber = streetNumber1;
    String cap=cap1;
        String city = city1;
    Connection conn= (Connection) dbManager.dbConnect();
        //Starting transaction
        QueryStatus qs=dbManager.startTransaction(conn);
   if(qs.execError){
                //TODO decide what to do in this case (transaction not started)
                log.error(qs.explainError());
                qs.occourtedErrorException.printStackTrace();
                System.out.println("Error␣during␣transaction␣starting
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣,Add␣private␣place␣not␣done");
                log.error("Error␣during␣transaction␣starting
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣,Add␣private␣place␣not␣done");
                dbManager.dbDisconnect(conn);
```

```
                return;
        }
        Coordinate placeCoordinate = new Coordinate();
        Address add; if (streetAddress.equals(""))
        {
                add = convertCoordinateToAddress(lat,lng);
                streetAddress = add.getStreetAddress();
                streetNumber= add.getStreetNumber();
                cap = add.getCap();
                city = add.getTownHall();
        }
        else{
                placeCoordinate = convertAddressToCoordinate(streetAddress
                        ,streetNumber,cap,city);
                System.out.println("coordinate␣lat:"+placeCoordinate.getLat());
                System.out.println("coordinate␣lng:"+placeCoordinate.getLng());
                lat =Double.toString(placeCoordinate.getLat());
                lng =Double.toString(placeCoordinate.getLng());
        }
        String query="Insert␣into␣PlacePrivate" + "␣(title,lat,lng
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣,streetAddress,streetNumber,cap,city,user,userGroup)
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣values␣('" + title + "','"+ lng + "','" + lat + "','" +
                streetAddress + "','" + streetNumber + "','"+ cap+"','"+city
                        + "','"+userID+"',0)";
        System.out.println(query);
        qs=dbManager.customQuery(conn, query);
        if(qs.execError){
                log.error(qs.explainError());
                System.out.println("Error␣during␣add␣private␣Place,aborting␣operation");
                qs.occourtedErrorException.printStackTrace();
                //Rolling back
                dbManager.rollbackTransaction(conn);
                log.error("Error␣during␣␣Add␣private␣place,aborting␣operation");
                dbManager.dbDisconnect(conn);
                //aggiungi categoria
                for (String s:category)
                {
                        addPrivatePlaceCategory(userID,title,Double.parseDouble(lng)
                        ,Double.parseDouble(lat),s);
                }
                return;
        }
        dbManager.commitTransaction(conn);
        dbManager.dbDisconnect(conn);
        //aggiungi categoria
        for (String s:category){
                addPrivatePlaceCategory(userID,title,Double.parseDouble(lng)
        ,Double.parseDouble(lat),s);
                }
return;
}
```

Uno degli aspetti fondamentali di questa funzione del livello DAO risiede nel seguente pezzo di codice

Listing 4.41: Parte di codice di addPrivatePlace()

```
if (streetAddress.equals())
{
        //codice relativo all'inserimento di un luogo tramite coordinate GPS
        //utilizzo della funzione convertCoordinateToAddress(lat, lng);
}else{
        //codice relativo all'inserimento di un luogo,
        //dati i campi compilati nell'interfaccia
        //utilizzo della funzione convertAddressToCoordinate(streetAddress
                //,streetNumber,cap,city);
}
```

Se entriamo nel prima ramo dell'if significa che l'utente lato client sta inserendo un luogo tramite coordinate GPS; contrariamente se entriamo nel secondo ramo, l'utente sta inserendo un luogo compilando i campi (nome luogo, indirizzo, cap, ecc.).

Le due funzioni, convertCoordinateToAddress() e convertAddressToCoordinate(), svolgono 2 mansioni, una l'inverso dell'altra. La prima date le coordinate GPS, lat e lng, ritorna l'indirizzo corrispondente in forma estesa (Via, numero civico Comune Città Stato).

La seconda dato l'indirizzo, ne ricava le coordinate. Entrambe utilizzano le api di google, e precisamente il servizio "The Google Geocoding API[95]".

Una richiesta di Geocoding, ha la seguente forma

`http://maps.googleapis.com/maps/api/geocode/output?parameters`

dove l'output può assumere due forme distinte:

- json (che è raccomandato) o JavaScript Object Notation (JSON);
- xml.

Noi abbiamo utilizzato il formato JSON, perché si riesce a farne il parser più velocemente.

Per accedere alle Geocoding API tramite HTTPS, abbiamo usato il servizio disponibile all'indirizzo: `https://maps.googleapis.com/maps/api/geocode/output?parameters`

HTTPS è consigliato per applicazioni che includono imporanti dati, come la posizione dell'utente. Alcuni parametri sono opzionali, mentre altri sono obbligatori.

Come nelle normali richieste http, nell'URL i parametri sono separati usando "&"; la lista dei parametri opzionali e non è:

- address (richiesto) — L'indirizzo che vogliamo convertire in coordinate.

Oppure, nel caso volessimo convertire le coordinate GPS in indirizzo:

- latlng (richiesto) — lat e lng del luogo di cui vogliamo ottenere l'indirizzo;

- bounds (opzionale) — Per aiutare google a dare una risposta più precisa. Per maggiori informazioni Viewports[2];
- regione (opzionale) — Il codice della regione. Per maggiori informazioni RegionCodes[3];
- language (opzionale) — Il linguaggio con cui vogliamo che ritornino i risultati;
- sensor (richiesto) — Indica se la richiesta di geocoding proviene da un device con un sensore GPS. I valori possibili sono true o false.

Vediamo in dettaglio, una delle due funzioni, ad esempio convertCoordinateToAddress().

Listing 4.42: Funzione convertCoordinateToAddress()

```
public static final Address convertCoordinateToAddress(String lat
        ,String lng)
{
        System.out.println("placeDatabase- convertCoordinateToAddress
                ,coordinate:"+lat+","+lng);
        MapsClient geocoding = new MapsClient();
        // currently there's only google, so we use direct call
        Address c = geocoding.covertCoordinateToAddress(lat,lng);
        //Coordinate c = new Coordinate("12","45");
        return c;
}
```

Questa funzione, crea un istanza di oggetto MapsClient; quest'ultimo, forma correttamente la richiesta rispettando la sintassi della richiesta e la inoltra a Google.

La funzione covertCoordinateToAddress(), invece, si preoccupa di intercettare il json restituito da google, come risposta alla richiesta di geocoding, e di recuperare i dati. In pratica esegue il parser della risposta, come si può notare sotto.

Listing 4.43: Funzione covertCoordinateToAddress()

```
public Address covertCoordinateToAddress(String lat,String lng)
{
        Map<String, String> params = new LinkedHashMap<String, String>();
        params.put("latlng", lng +","+lat);
        params.put("sensor", "false");
        ResponseC r = sendGeocodingRequest(GOOGLE_GEOCODING, params);
        System.out.println("Fatto!");
        if (r.getStatus().equals("ZERO_RESULTS")){
                System.out.println("Nessuna corrispondenza in COORDINATE!!!");
                log.info("Google result: ZERO_RESULT_:
                        Nessuna corrispondenza in COORDINATE!!!");
        } else{
                Result[] ad= r.getResult();
                System.out.println(ad[0].geometry.location.getLat() + ","
```

[2] http://code.google.com/apis/maps/documentation/geocoding/
[3] http://code.google.com/apis/maps/documentation/geocoding/

```
15                          + ad[0].geometry.location.getLng());
                    Address add = new Address();
17                  int length = ad[0].address_components.length;
                    System.out.println("...Inserimento luogo individuato tramite
19 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣coordinate GPS...");
                    System.out.println("Numero elementi address_components:" + length);
21                  int pos=0;
                    while (pos<length) {
23                          if (ad[0].address_components[pos].types[0]
                                    .equals("street_number"))
25                          {
                                    add.setStreetNumber((ad[0].address_components[pos]
27                                          .long_name));
                            System.out.println("streetnumber:"+add.getStreetNumber());
29                          }
                            if (ad[0].address_components[pos].types[0].equals("route"))
31                          {
                                    add.setStreetAddress((ad[0].address_components[pos]
33                                  .long_name));
                            System.out.println("streetaddress:"+add.getStreetAddress());
35                          }
                            if (ad[0].address_components[pos].types[0].equals("locality"))
37                          {
                            add.setTownHall((ad[0].address_components[pos].long_name));
39                          System.out.println("comune:" + add.getTownHall());
                            }
41                          if (ad[0].address_components[pos].types[0]
                                    .equals("administrative_area_level_2"))
43                          {
                            add.setCity((ad[0].address_components[pos].long_name));
45                          System.out.println("città:" + add.getCity());
                            }
47                          if(ad[0].address_components[pos].types[0]
                                    .equals("administrative_area_level_1"))
49                          {
                            add.setRegion((ad[0].address_components[pos].long_name));
51                          System.out.println("regione: " + add.getRegion());
                            }
53                          if (ad[0].address_components[pos].types[0].equals("country"))
                            {
55                          add.setState((ad[0].address_components[pos].long_name));
                            System.out.println("stato:" + add.getState());
57                          }
                            if (ad[0].address_components[pos].types[0].equals("postal_code"))
59                          {
                            add.setCap((ad[0].address_components[pos].long_name));
61                          System.out.println("CAP:" + add.getCap());
                            }
63                          pos++;
                    }
65          return add;
```

```
        }
67      return null;
}
```

Nel momento in cui i dati, come route, streetnumber, city etc. sono stati recuperati si procede all'inserimento nel db.

4.3.2.3 Cancellazione di un luogo

Ora vediamo cosa comporta la cancellazione per i luoghi privati e pubblici.

Per i luoghi privati, la cancellazione di un luogo è semplice e non va ad influire nessun altro utente. Per i luoghi pubblici, in ogni caso si va ad eliminare il proprio voto.

Per cancellare un luogo si deve cliccare un elemento della lista della schermata principale dei luoghi. Si apre così, la finestra Details (figura 4.39), che mostra i dettagli del luogo. Attraverso il menù l'utente può tornare indietro, attraverso il tasto "Back" oppure eliminare il posto con il pulsante "Delete".

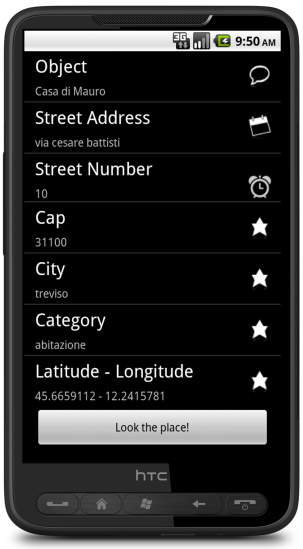


Figura 4.39: Cancellazione di un luogo inserito.

Premendo su "Delete Place", a livello client, si crea un oggetto PlaceClient e quindi un nuovo thread per inviare una richiesta di cancellazione al server. Ovviamente, come al solito, il PlaceClient lato server viene serializzato e poi inviato.

Il server risponde alla richiesta con la risorsa deletePrivatePlace()

Listing 4.44: Livello Web: deletePrivatePlace()

```
@POST
2 @Path("/deletePrivatePlace")
@Consumes("application/xml")
```

```
4  public void deletePrivatePlace(@PathParam("username") String userid
       ,@CookieParam("sessionid") String sessionid,PlaceClient place)
6  {
       log.info("Request to delete private place from
8  ----------------user " + userid + ", session "+ sessionid);
       System.out.println("-PlacesResource metodo deletePrivatePlace");
10     PlacesManager.getInstance().deletePrivatePlace(userid,place.title
           ,place.lat,place.lng);
12 }
```

che a sua volta contatterà la risorsa a livello manager deletePrivatePlace()

Listing 4.45: Livello Manager: deletePrivatePlace()

```
/**
2 * delete private place entered by this user
*/
4 public void deletePrivatePlace(String userid,String title,String lat, String lng)
{
6      PlacesDatabase.istance.deletePrivatePlace(userid, title, lat, lng);
}
```

la quale, rispettando l'architettura three tier, richiamerà la funzione deletePrivatePlace() in PlaceDatabase del livello DAO

Listing 4.46: Livello Dao: deletePrivatePlace()

```
1  public static void deletePrivatePlace(final String userID
       , final String title, final String lat, final String lng) {
3      Connection conn= (Connection) dbManager.dbConnect();
       //Starting transaction
5      QueryStatus qs=dbManager.startTransaction(conn);
       if(qs.execError){
7          //TODO decide what to do in this case (transaction not started)
           log.error(qs.explainError());
9          qs.occourtedErrorException.printStackTrace();
           System.out.println("Error during transaction starting... Delete
11 ------------------------private place not done");
       log.error("Error during transaction starting... Delete
13 ------------------------private place not done");
       dbManager.dbDisconnect(conn);
15         return;
       }
17     String deleteQuery="Delete from PlacePrivate" +"
----------------where user='"+ userID +"'and title='" + title + "'
19 ----------------and lat='" + lat +"' and lng='" + lng + "' and userGroup=0";
       System.out.println(deleteQuery);
21     qs=dbManager.customQuery(conn, deleteQuery);
       if(qs.execError){
23         log.error(qs.explainError());
           System.out.println("Error during delete private place,aborting
25 ------------------------operation");
           qs.occourtedErrorException.printStackTrace();
```

```
27                  //Rolling back
                    dbManager.rollbackTransaction(conn);
29                  log.error("Error during delete private place,aborting operation");
                    dbManager.dbDisconnect(conn);
31                  return;
            }
33          dbManager.commitTransaction(conn);
            dbManager.dbDisconnect(conn);
35          deletePlaceCategory(userID,title,lat,lng);
            return;
37  }
```

Questa funzione cancellerà il luogo dal database. Nel caso dei luoghi pubblici, invece la cencellazione, implica l'eliminazione del voto su un determinato luogo. Il luogo continuerà comunque ad esistere, rimarrà disponibile per la comunità che utilizza l'applicazione.

A livello client il processo di cancellazione dei luoghi publici è del tutto simile. Viene inviato un oggetto PlaceClient (serializzato), che rappresenta l'oggetto da cancellare e il server risponde con la risorsa

Listing 4.47: Livello Web: deleteVotePublicPlace()

```
1   @POST
    @Path("/deleteVotePublicPlace")
3   @Consumes("application/xml")
    public void deleteVotePublicPlace(@PathParam("username") String userid
5           ,PlaceClient place, @CookieParam("sessionid") String sessionid)
    {
7           log.info("Request to delete private place from user "
                    + userid + ", session "+ sessionid);
9           System.out.println("-PlacesResource metodo deletePrivatePlace");
            List<String> categoryList= new LinkedList<String>();
11          String[] words = place.category.split(",");
            categoryList.addAll(Arrays.asList(words));
13          PlacesManager.getInstance().deleteVotePublicPlace(userid,place.title
                    ,place.lat,place.lng,categoryList);
15  }
```

che richiamerà la funzione deleteVotePublicPlace() del livello Manager, che non fa altro che richiamare la funzione deleteVotePublicPlace() del livello DAO

Listing 4.48: Livello Dao: deleteVotePublicPlace()

```
1   public static void deleteVotePublicPlace(String userID,String title
            ,String lat, String lng,List<String> category) {
3           Connection conn= (Connection) dbManager.dbConnect();
            //Starting transaction
5           QueryStatus qs=dbManager.startTransaction(conn);
            if(qs.execError){
7                   //TODO decide what to do in this case (transaction not started)
                    log.error(qs.explainError());
9                   qs.occourtedErrorException.printStackTrace();
```

```
                System.out.println("Error during transaction starting
                        ,Delete vote for public place not done");
                log.error("Error during transaction starting
                        ,Delete vote for public place not done");
                dbManager.dbDisconnect(conn);
                return;
        }
        //cancello il voto in Place_voted
        for (String s:category){
                String deleteQuery="Delete from Place_voted"
                +" where username='"+ userID +"' and title='" + title
                + "' and lat='" + lat +"' and lng='" + lng + "'
                and category='"+s+"'";
                System.out.println(deleteQuery);
                qs=dbManager.customQuery(conn, deleteQuery);
                if(qs.execError){
                        log.error(qs.explainError());
                        System.out.println("Error during delete vote for public
                                place,aborting operation");
                        qs.occourtedErrorException.printStackTrace();
                        //Rolling back
                        dbManager.rollbackTransaction(conn);
                        log.error("Error during delete public place,aborting operation");
                        dbManager.dbDisconnect(conn);
                        return;
                }
                //aggiorno lo storico
                String query="Insert into Place_voted_historical"
                        +" (title,lat,lng,category,username,vote) values ('" + title
                        + "','"+ lat + "','" + lng + "','"
                        + s + "','"+userID+"',2)";
                System.out.println(query);
                qs=dbManager.customQuery(conn, query);
                if(qs.execError){
                        log.error(qs.explainError());
                        System.out.println("Error during adding delete public
                                vote in Place historical,aborting operation");
                                qs.occourtedErrorException.printStackTrace();
                                //Rolling back
                                dbManager.rollbackTransaction(conn);
                                log.error("Error during adding delete public vote
                                        in Place historical,aborting operation");
                                dbManager.dbDisconnect(conn);
                                return;
                }
        }
        dbManager.commitTransaction(conn);
        dbManager.dbDisconnect(conn);
        return;
}
```

4.3.2.4 Ricerca e votazione di luoghi pubblici

Come abbiamo già detto nel paragrafo precedente, nel caso dei luoghi pubblici, la votazione è intesa come "consenso" ad utilizzare quel dato luogo. Non esiste in questo caso una promozione di un luogo o una cancellazione. Non si considera, nemmeno il ranking di un utente. Dato che già, nell'inseriemento di un task, all'utente il sistema potrebbe chiedere di votare, per evitare l'inserimento di un'ulteriore votazione durante l'immissione di un task si è deciso di isolare questa fase in una sezione dedicata dell'applicazione. In particolare, quest'ultimo può essere fatto solamente accedendo alla sezione "New Places" e al tab "Public Place" (figura 4.40).

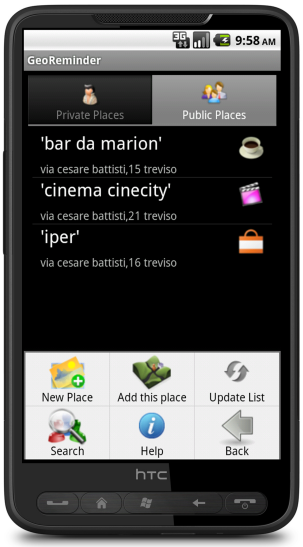


Figura 4.40: Accesso alla funzione "Search".

Da qui si preme sul tasto Search, per cercare dei luoghi inseriti da altri utenti come si vede in figura4.41.

Figura 4.41: Ricerca di un luogo votato da altri utenti.

Premendo su Search, il client invia una richiesta al server con i parametri inseriti dall'utente. Viene creato, a riguardo, nel client, un thread

Listing 4.49: Thread searchPublicPlace()

```
public static Thread searchPublicPlace(final PlaceClient toSearch
        , final Handler handler, final Context context)
{
        final Runnable runnable = new Runnable()
        {
                public void run() {
                        Log.i(TAG, "request_public_search_places_for_the_user");
                        ArrayList<NameValuePair> nameValuePairs =
                                new ArrayList<NameValuePair>();
                        nameValuePairs.add(new BasicNameValuePair("title"
                                ,toSearch.title));
                        nameValuePairs.add(new
                                BasicNameValuePair("streetAddress",toSearch
                                        .streetAddress));
                        nameValuePairs.add(new BasicNameValuePair("streetNumber"
                                ,toSearch.streetNumber));
                        nameValuePairs.add(new BasicNameValuePair("cap"
                                ,toSearch.cap));
                        nameValuePairs.add(new BasicNameValuePair("city"
                                ,toSearch.city));
                        nameValuePairs.add(new BasicNameValuePair("category"
                                ,toSearch.category));
```

```
                        List<PlaceClient> result = runHttpGetPublicPlaces(
                                "/places/searchPublicPlace"
                                , nameValuePairs, context);
                        Log.i(TAG, "request public places for the user 2");
                        sendResult_searchPublicPlaces(result, handler, context);
                }
        };
        // start group list request
        return NetworkUtilities.startBackgroundThread(runnable);
}
```

che utilizza la funzione runHttpGetPublicPlaces() per inoltrare la richiesta di tipo GET al server

Listing 4.50: Client: runHttpGetPublicPlaces()

```
private static List<PlaceClient> runHttpGetPublicPlaces(final String method
        ,final ArrayList<NameValuePair> params, Context c)
{
        List<PlaceClient> result = new LinkedList<PlaceClient>();
        DefaultHttpClient newClient = NetworkUtilities.createClient();
        String query = (params == null) ? "" :
                URLEncodedUtils.format(params,"UTF-8");
        AccountUtil util = new AccountUtil();
        HttpGet request = new HttpGet(NetworkUtilities.SERVER_URI + "/"
                + util.getUsername(c) + method +"?"+ query);
        Log.i(TAG, NetworkUtilities.SERVER_URI + "/"+ util.getUsername(c)
                + method +"?"+ query);
        request.addHeader("Cookie", "sessionid="+util.getToken(c));
        // send the request to network
        HttpResponse response = NetworkUtilities.sendRequest(newClient, request);
        // if we cannot connect to the server
        if (!HttpResponseStatusCodeValidator.isValidRequest(response
                .getStatusLine().getStatusCode()))
        {
                Log.i(TAG, "Cannot connect to server with code "
                        + response.getStatusLine().getStatusCode());
                return null;
                // return null if there's problem with connection
        }
        try { // parsing XML message
                result = PlacesHandler.parseUserPublicPlaces(response
                        .getEntity().getContent());
                Log.i(TAG,"List of search place OK!");
                return result;
        } catch (IllegalStateException e) {
                Log.i(TAG, "Illegal State");
                e.printStackTrace();
                return result;
        } catch (IOException e)
        {
                Log.i(TAG, "IOException");
```

```
                e.printStackTrace();
38              return result;
        } catch (SAXException e) {
40              Log.i(TAG, "SAXException");
                e.printStackTrace();
42              return result;
        }
44  }
```

Il server, come già detto varie volte, risponderà a livello web con la risorsa

Listing 4.51: Livello Web: searchPublicPlace()

```
@GET
2 @Path("/searchPublicPlace")
@Produces("application/xml")
4 public List<PlaceClient> searchPublicPlace(@PathParam("username") String userid
        ,@CookieParam("sessionid") String sessionid
6       ,@QueryParam("title") String title
        ,@QueryParam("streetAddress") String streetAddress
8       ,@QueryParam("streetNumber") String streetNumber
        ,@QueryParam("cap") String cap
10      ,@QueryParam("city") String city
        ,@QueryParam("category") String category)
12 {
        log.info("Request␣to␣add␣public␣place␣from␣user␣" + userid + "
14 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣,␣session␣"+ sessionid);
        System.out.println("placeResource");
16      //creo la lista di category, dato che mi arriva una stringa
                con le location separate da una virgola
18      List<String> categoryList= new LinkedList<String>();
        String[] words = category.split(",");
20      categoryList.addAll(Arrays.asList(words));
        return PlacesManager.getInstance().searchPublicPlace(userid
22              ,title,streetAddress,streetNumber,cap,city,categoryList);
}
```

Quest'ultima inoltra la richiesta a livello Manager,

Listing 4.52: Livello Manager: searchPublicPlace()

```
1 /**
* retrieve all public places with certain constraints
3 * @return list that contains all public places that corrispond to the request
*/
5 public List<PlaceClient> searchPublicPlace(String userid,String title
        ,String streetAddress,String streetNumber
7       ,String cap,String city,List<String> category)
{
9       return PlacesDatabase.istance.searchPublicPlace(userid
                ,title,streetAddress,streetNumber,cap,city,category);
11 }
```

che a sua volta inoltrerà la richiesta a livello DAO alla risorsa searchPublicPlace()

Listing 4.53: Livello Dao: searchPublicPlace()

```
/*
* Ritorna la lista di posti pubblici inseriti da altri utenti o
* dall'utente stesso, * ma non votati da quest'ultimo,
* corrispondenti ai criteri di ricerca
*/
public static List<PlaceClient> searchPublicPlace(
        String userid,String title,String streetAddress
        ,String streetNumber,String cap,String city
        ,List<String> category)
{
        ArrayList<PlaceClient> placeList=new ArrayList<PlaceClient>();
        Connection conn= (Connection) dbManager.dbConnect();
        boolean whereflag = false;
        //creo la query
        String selectQuery = "Select * from Place_category join Place
                on Place_category.title=Place.title and " +
                "Place_category.lat=Place.lat and Place_category.lng=Place.lng";

        if (!title.equalsIgnoreCase(""))
                if (whereflag==false)
                {
                selectQuery = selectQuery + " where (Place.title LIKE '%"
                        +title.toLowerCase()+"%') ";
                whereflag = true;
                }else
                {
                selectQuery = selectQuery + " and (Place.title LIKE '%"
                        +title.toLowerCase()+"%') ";
        }
        }

        if(!streetAddress.equalsIgnoreCase(""))
        {
                if (whereflag==false)
                {
                        selectQuery = selectQuery + " where
                                streetAddress='"+streetAddress.toLowerCase()+"' ";
                        whereflag = true;
                }
                else {
                        selectQuery = selectQuery + " and streetAddress='"
                                +streetAddress.toLowerCase()+"' ";
        }
        }

        if (!streetNumber.equalsIgnoreCase("")) {
        if (whereflag==false)
                {
```

```
                        selectQuery = selectQuery + " where
                                        streetNumber='"+streetNumber.toLowerCase()+"' ";
                        whereflag = true;
                } else {
                        selectQuery = selectQuery + " and
                                        streetNumber='"+streetNumber.toLowerCase()+"' ";
                }
        }

        if (!cap.equalsIgnoreCase("")){
                if (whereflag==false) {
                          selectQuery = selectQuery + " where cap='"
                                          +cap.toLowerCase()+"' ";
                                          whereflag = true;
                } else {
                selectQuery = selectQuery + " and cap='"+cap.toLowerCase()+"' ";
                  }
        }

        if (!city.equalsIgnoreCase("")) {
        if (whereflag==false) {
                        selectQuery = selectQuery + "
                                        where (city LIKE '%"+city.toLowerCase()+"%') ";
                whereflag = true;
                } else {
                selectQuery = selectQuery + " and (city LIKE '%"
                                +city.toLowerCase()+"%') ";
                }
    }

        if ( !category.isEmpty()) {
                if (whereflag==false) {
                boolean first= true; //se è il primo della lista
                for (String s : category)
                        {
                        if (s.equalsIgnoreCase(""))
                                        continue;
                                if (first){
                                        selectQuery = selectQuery + " where (";
                                selectQuery = selectQuery + " category ='"
                                                +s.toLowerCase()+"' ";
                                first = false;
                                }
                        else {
                    selectQuery = selectQuery + " or category ='"
                                        +s.toLowerCase()+"' ";
                        }   }
                        if (!first)
                                        selectQuery = selectQuery + " ) ";
                        }
                        else
```

```
                {
101                     boolean first= true; //se è il primo della lista
                        for (String s : category)
103                             if (s.equalsIgnoreCase(""))
                                        continue;
105                     if (first){
                        selectQuery = selectQuery + "␣and␣(␣";
107                             selectQuery = selectQuery + "␣category␣='"
                                        +s.toLowerCase()+"'␣";
109                             first = false;
                           } else {
111                             selectQuery = selectQuery + "␣or␣category␣='"
                                                +s.toLowerCase()+"'␣";
113                     }
                        }
115             if (!first)
                        selectQuery = selectQuery + "␣)␣";
117             }
        }
119     System.out.println(selectQuery);
        QueryStatus qs=dbManager.customSelect(conn, selectQuery);
121     ResultSet rs=(ResultSet)qs.customQueryOutput;
        try{
123             while(rs.next()){
                        String titleQ = rs.getString("title");
125                     String latQ = rs.getString("lat");
                        String lngQ = rs.getString("lng");
127                     String streetAddressQ = rs.getString("streetAddress");
                        String streetNumberQ = rs.getString("streetNumber");
129                     String capQ = rs.getString("cap");
                        String cityQ = rs.getString("city");
131                     String categoryQ = rs.getString("category");
                        placeList.add(
133                     new PlaceClient(titleQ,latQ,lngQ,streetAddressQ
                                ,streetNumberQ,capQ,cityQ,categoryQ));
135                     }
                }
137             catch(SQLException sqlE){
                }finally{
139                     dbManager.dbDisconnect(conn);
                }
141     rrayList<PlaceClient> placeListToReturn =
                new ArrayList<PlaceClient>();
143     //ritorno solo le cose non votate dall'utente
        for (PlaceClient p: placeList) {
145             boolean hasAlreadyVoted = publicHasAlreadyVoted(userid,p.title
                        ,p.lat,p.lng,p.category);
147             if (!hasAlreadyVoted) {
                        //controllo se l'ho già inserito, se si aggiungo
149                     //solo la categoria
                        Iterator it= placeListToReturn.iterator();
```

```
151                             boolean isInsert = false;
                                while(it.hasNext()) {
153                                     PlaceClient value=(PlaceClient)it.next();
                                        if ( value.title.equals(p.title) && value.lat
155                                             .equals(p.lat) && value.lng.equals(p.lng))
                                        {
157                                             isInsert = true;
                                                value.category = value.category+","
159                                                     +p.category;
                                                break;
161                                     }
                                }
163                             //se non è già stato inserito lo inserisco
                                if (!isInsert){
165                                     placeListToReturn.add(
                                        new PlaceClient(p.title,p.lat,p.lng,p.streetAddress
167                                             ,p.streetNumber,p.cap,p.city,p.category));
                                }
169                     }
                }
171             return placeListToReturn;
}
```

Questa funzione, come si può notare, svolge la ricerca con i parametri passati all'utente. Quindi accede al database Places, per recuperare i luoghi pubblici le cui caratteristiche rispettano i vincoli della ricerca.

Altra caratteristica, è l'utilizzo della funzione publicHasAlreadyVoted() che no fa altro che controllare se l'utente ha già votato o meno per un dato posto. Se ha già votato, la funzione publicHasAlreadyVoted() termina, altrimenti registra il voto dell'utente.

Al client ritornerà una lista dei posti trovati, che renderizzerà a video in questa maniera (figura 4.42)

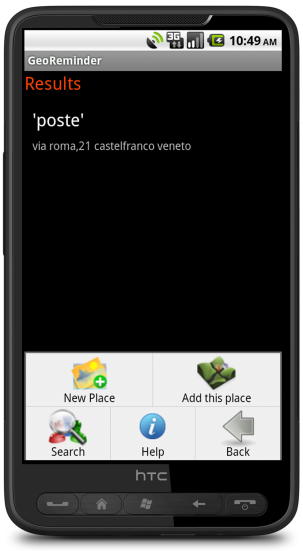


Figura 4.42: Risultati ricerca

Selezionando una voce della lista si potrà vedere i dettagli del posto e in caso si voglia, tramite il menu, votare il luogo (figura 4.43) .

Figura 4.43: Votazione di un luogo già inserito da altri utenti.

Da questo momenti in poi, il luogo votato, verrà incluso nella lista dei luoghi di ritorno, nel caso sia compatibile con il bisogno dell'utente.

Vediamo, come ultimo, la funzione principale votePublicPlaces() del livello DAO

Listing 4.54: Livello Dao: votePublicPlaces()

```
public static void votePublicPlaces(String userid,List<PlaceClient> places)
{
        for (PlaceClient p:places) {
                //creo la lista di category, dato che mi arriva
                //una stringa con le location separate da una virgola
                List<String> categoryList= new LinkedList<String>();
                String[] words = p.category.split(",");
                categoryList.addAll(Arrays.asList(words));
                votePublicPlace(userid, p.title,p.lat,p.lng,categoryList);
        }
}
```

Come vediamo dal codice della funzione e dall'interfaccia client è possibile votare un luogo anche solo relativamente ad una categoria. La funzione votePublicPlace() si preoccupa di registrare le votazioni degli utenti.

4.4 Problema: eccessiva vulnerabilità ai tempi di risposta dei servizi esterni

Al fine di ridurre i tempi di risposta verso il client e per difendersi da possibili ritardi nelle risposte inviate dal 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. 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.

I vantaggi di un sistema di chaching possono essere:

- i tempi di risposta del server verso il client diminuiscono; avendo già dei dati a disposizione nel db cache (se presenti), il server migliora la sua prontezza nell'inviare i dati al client

Gli svantaggi potrebbero essere:

- Se i dati non sono presenti in cache, si perde del tempo inutile a controllare se ci sono; successivamente, non trovando niente, si contatta Google.

A riguardo di questo problema, durante i meeting si è discusso molto. Per una descrizione dettagliata dell'implementazione del sistema di caching consultare il riferimento bibliografico[9].

4.5 Problema: limitato supporto all'utente nel guidarlo a destinazione

Nel momento in cui l'utente seleziona un hint nella mappa che visualizza i possibili luoghi per il soddisfacimento di un task, compare una lista contenente le indicazioni stradali fornite da Google da seguire per arrivare alla destinazione. Dal punto di vista della user-friendliness, ma soprattutto della sicurezza, sarebbe utile avere la possibilità di giungere al luogo di destinazione senza la necessità di osservare il display del terminale e potendo quindi concentrare l'attezione sull'ambiente circostante. A tale scopo, abbiamo sviluppato un sistema di guida basato su segnalazione audio e vibratili atte a suggerire all'utente quale strada prendere per arrivare alla meta.

Comepunto di partenza, abbiamo utilizzato AndNav2[96], un navigatore open-source, e modificato il codice sorgente in maniera che integrasse il sistema di notifica attraverso vibrazioni.

Analizzando il codice, abbiamo individuato le posizioni precise nel codice in cui inserire una chiamata alle seguenti routine. Per posizione precisa si intende il momento in cui il navigator segnala all'utente una svolta a sinistra o a destra.

Listing 4.55: Funzioni vibrateSx() e vibrateDx

```
1  public static void vibrateSx()
   {
3  if (vibrateOnOffValue != VIBRATE_STATUS_OFF) {
           Vibrator v=(Vibrator)getSystemService(Context.VIBRATOR_SERVICE);
5          //Vibrate for 3000 milliseconds
           long milliseconds = 3000;
7          v.vibrate(milliseconds);
           }
9  }

11 //--------------------------------

13 public static void vibrateDx()
   {
15 if (vibrateOnOffValue != VIBRATE_STATUS_OFF) {
           Vibrator v=(Vibrator)getSystemService(Context.VIBRATOR_SERVICE);
17         //Vibrate for 1000 milliseconds
           long milliseconds = 1000;
19         v.vibrate(milliseconds,500,milliseconds,500,milliseconds);
           }
21 }
```

Come si può notare entrambe le routine svolgono una semplice funzione: far vibrare lo smartphone. Per distinguere le indicazioni "gira a sx" e "gira a dx", si

sono usati due diversi pattern di vibrazione. Girare a sinistra è indicato attraverso una vibrazione continua della durata di 3000 millisecondi. Girare a destra è indicato invece con 3 brevi vibrazioni da un secondo ciascuna, intervallate da mezzo secondo.

Il navigatore è stato modificato nel modo appena descritto. Un problema, che tutt'ora è rimasto irrisolto, sta nel fatto che, Andnav2, come tutti i navigator emettono le indicazioni stradali fino al max 50m prima di una svolta, a sinistra o a destra. Per cui l'utente, tenendo il cellulare in tasca non conoscerà mai il momento preciso che dovrà svoltare.

Capitolo 5

Test con Terminali Android

5.1 I test

Lo sviluppo delle nuove funzionalità del server e del client implicano una costante attività di test per verificare che il funzionamento effettivo corrisponda a quello atteso dai vari componenti.

Avendo a disposizione un terminale con sistema operativo Android, si è deciso di realizzare test con utenti estranei al progetto per avere una valutazione neutrale sul sistema realizzato, senza quindi che l'utente fosse influenzato da una conoscenza del software che invece poteva esistere se i test fossero stati eseguiti dai membri del gruppo di lavoro. Le prove così eseguite hanno scansato l'ipotesi di aver falsato i dati, ingaggiando persone che non avevano mai visto il programma e alcune di loro, tra l'altro, con poca dimestichezza con i computer.

E' stato definito un protocollo sperimentale come segue:

- generazione di un percorso da rispettare (punto di partenza, punto di arrivo);
- creazione di due gruppi: il primo gruppo deve fare il percorso facendo affidamento soltanto sulla memoria, il secondo gruppo deve fare il percorso con l'ausilio del terminale Android provvisto dell'applicazione;
- valutazione del comportamento dei vari utenti;
- compilazione di un questionario di gradimento da parte degli utenti.

Le locazioni scelte sono state Bassano del Grappa (Vicenza) e Castelfranco Veneto (Treviso).

Entrambi 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. Ad ogni modo abbiamo scelto due paesi in cui esistono più opportunità di soddisfare un task nel raggio di 200/300m.

I task (attività quotidiane) scelti per Bassano del Grappa sono:

- comprare il latte;
- recarsi all'ufficio postale;

- comprare il pane;
- andare a prenotare un ristorante per una cena
- andare al cinema a prendere i biglietti per un film;
- passare a vedere la vetrina di un determinato negozio (negozio Cenere);
- comprare il giornale;
- andare a visitare un determinato monumento (Ponte Vecchio);
- andare in farmacia;
- andare in Libreria

Nel test di Castelfranco Veneto è stato sostituito il negozio di Cenere con il negozio Bertoldo, il Ponte Vecchio con la Casa del Giorgione e il teatro con il cinema.

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);
- *distanza media del soddisfacimento del task in relazione al punto di partenza*: sono state dapprima calcolate le medie per la distanza totale riferita cioè al punto di partenza del percorso;
- *distanza media del soddisfacimento del task rispetto al task precedente*: sono state calcolate le medie per la distanza parziale riferita cioè alla distanza tra il soddisfacimento di due task;

Non è possibile fare una valutazione strettamente oggettiva sui risultati dal momento che 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 test che possano fornire un'effettiva valutazione dell'efficacia del nostro sistema, sarebbe necessario effettuare un test basato sul comportamento quotidiano naturale dei soggetti e prolungato nel tempo in modo da sovrapporsi effettivamente alla vita quotidiana. Il nostro breve test non ha quindi la valenza di un'effettiva e completa valutazione dell'efficacia della metodologia, ma fornisce una valutazione dell'apprezzamento degli utenti in termini di servizio fornito ed intrusività dell'interfaccia.

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

I test sono stati condotti con un cellulare Vodafone IDEOS[97], con le seguenti caratteristiche:

- piattaforma: OS Android 2.2 Froyo;
- processore: CPU Qualcomm MSM 7225 da 528 MHz;
- RAM: 256MB;
- GPS;
- memoria: memoria interna 200 Mb espansa con micro SD da 1 Gb; connessione di rete: 3G (fino a 14.4Mbps in download e 5.76Mbps in upload), GPRS, EDGE, HSDPA,WiFi;

Gestore telefonico per connessione a internet: Vodafone.

E' stata predisposta per i test una macchina virtuale installata a Padova e allogiata in VMware 2.0.2, installato su sistema operativo Linux Ubuntu Server 10.04 LTS. Nella macchina virtuale, è stato installato e configurato il server web Apache, PHP, PHPMyAdmin (uno script PHP per l'interazione con MySql), MySql, Tomcat (con deploiato il war sviluppato).

Il server è disponibile all'indirizzo: `http://serverpd.dyndns.org:8080/ephemere`.

Nell'esecuzione dei test sono stati raccolti i 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.

In entrambe le locazioni, Castelfranco Veneto e Bassano del Grappa, 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.

5.2 Test a Bassano del Grappa

5.2.1 Il percorso programmato

In figura5.1 è riportata la mappa del percorso seguito a Bassano del Grappa. Tale mappa è utile per contestualizzare percorso e valutazioni fatte sui dati analizzati.

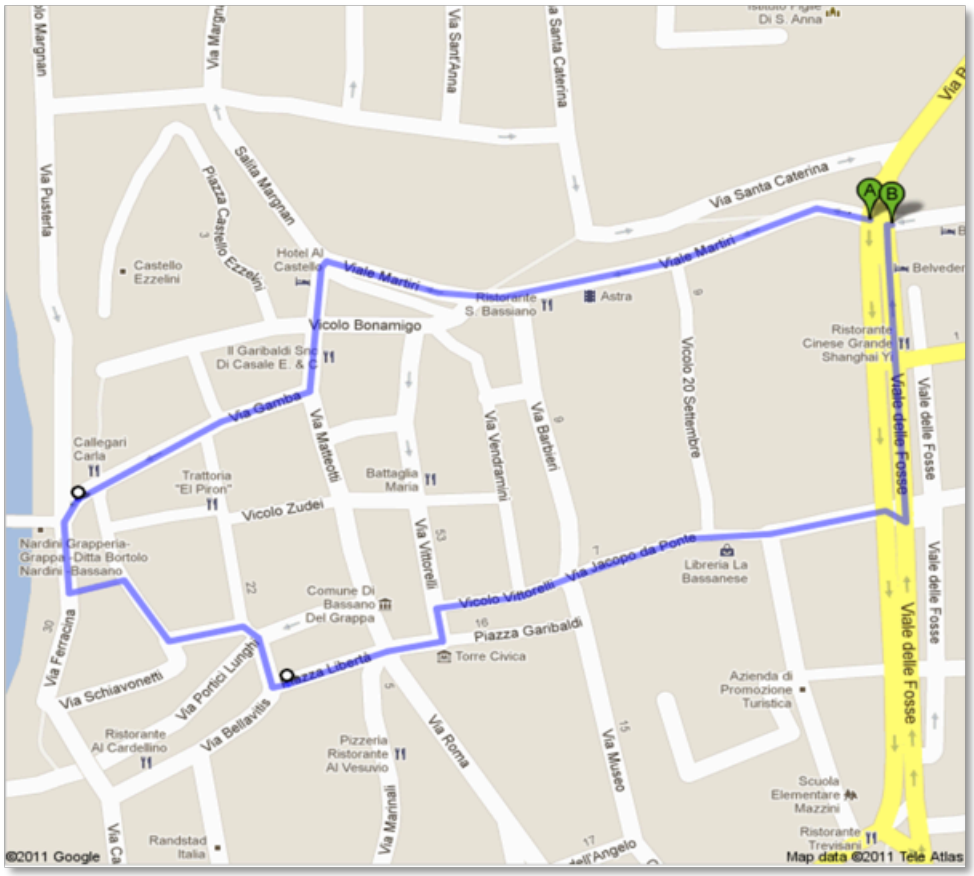


Figura 5.1: Il percorso di Bassano del Grappa.

L'età dei partecipanti è data dalla seguente tabella (figura 5.2):

No_hint	Età	Hint	Età
Andrea	26	Michela	37
Davide	26	Manuel	26
Michela	37	Marco	26
Stefano	26	Fabio	37
Media	28,75	**Media**	31,5

Figura 5.2: Età dei partecipanti dei test a Bassano del Grappa.

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/o compiti di vita quotidiana) 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 col-

locazione geografica e/o temporale. In questo modo il sistema cerca di massimizzare la redditività di una giornata, ottimizzando i tempi e ricordando all'utilizzatore le attività che normalmente potrebbero venir dimenticate.

Il test senza smartphone doveva essere svolto da persone che fossero in grado di poter portare a termine una serie di dieci attività, avvalendosi del solo supporto mnemonico; per questa ragione, il target, a livello di età, non doveva essere troppo elevato, così da garantire che le eventuali dimenticanze non fossero da imputare a deficit mnemonici dovuti all'età del soggetto.

5.2.2 Analisi dei valori medi dal punto iniziale

Per ottenere dei grafici comparativi, sono state dapprima calcolate le medie per la distanza totale (riferite cioè al punto di partenza del percorso) e parziale (riferite 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.

La seguente tabella (figura 5.3) rappresenta i valori ottenuti dalle prove effettuate senza l'utilizzo del terminale. All'incrocio, riga-colonna, il valore è da intendersi come il numero di metri che l'utente della riga ha impiegato per soddisfare il task in colonna dal punto iniziale. La penultima riga rappresenta la media in colonna, ossia quanto mediamente gli utenti hanno impegato (in metri) per soddisfare il task in colonna. L'ultima riga invece riporta il numero dei task soddisfatti ripsettivamente per ogni singolo obiettivo.

	CINEMA	RISTORANTE	LIBRERIA	PONTE VECCHIO	PANE	LATTE	FARMACIA	POSTE	N. CENERE	GIORNALE	MEDIA DISTANZA PER TASK
Andrea	180	210	700	650	895	2195	870	1495	2213	700	1010,80
Davide	180	210	2127	610	1560		1635	960	2065	1550	1210,78
Michela	180	410	710	660	1000	1480	1060		1498	980	886,44
Stefano	180	210	1425	810	460		1075			1030	741,43
MEDIA	**180**	**260**	**1241**	**682,5**	**978,8**	**1838**	**1160**	**1228**	**1925**	**1065**	**1055,71**
Task soddisfatti	*4*	*4*	*4*	*4*	*4*	*2*	*4*	*2*	*3*	*4*	*35*

Figura 5.3: Valori medi delle distanze dal punto iniziale dei test a Bassano del G. senza terminale.

La tabella in figura 5.4, invece, rappresenta i valori ottenuti dalle prove effettuate con l'utilizzo del terminale. All'incrocio, riga-colonna, il valore è da intendersi come il numero di metri che l'utente della riga ha impiegato per soddisfare il task in colonna. La penultima riga rappresenta, come prima, la media in colonna, ossia quanto mediamente gli utenti hanno impegato (in metri) per soddisfare il task in colonna.

	CINEMA	RISTORANTE	LIBRERIA	PONTE VECCHIO	PANE	LATTE	FARMACIA	POSTE	N. CENERE	GIORNALE	MEDIA DISTANZA PER TASK
Michela	180	900	480	660	110	10	1250	1800	2500	1110	900,00
Manuel	180	1747	420	997	507	290	140	1347	2497	2147	1027,20
Marco	180	561	480	651	1601	1837	1657	1001	1857	1667	1149,20
Fabio	180	210	510	790	900	10	1470	2547	1547	840	900,40
MEDIA	**180**	**854,5**	**472,5**	**774,5**	**779,5**	**536,8**	**1129**	**1674**	**2100**	**1441**	**994,20**
Task soddisfatti	*4*	*4*	*4*	*4*	*4*	*4*	*4*	*4*	*4*	*4*	*40*

Figura 5.4: Valori medi delle distanze dal punto iniziale dei test a Bassano del G. con terminale.

Nella tabella sottostante (figura 5.5) da una prima veloce analisi se ne ricava che, la distanza media percorsa per task (riferita al punto di partenza) diminuisce se si utilizza il sistema.

	Totali medie	Task soddisfatti	% di Task eseguiti	Totali Medie teoriche
NO-HINT	1055,7	35	88%	1206,5
HINT	994,2	40	100%	994,2

Figura 5.5: Confronto valori medi, hint, no hint, dal punto inziale di Bassano del Grappa con e senza terminale.

Nella tabella, rispettivamente per ogni riga, senza terminale o con terminale, sono evidenziati i totali delle medie, il numero dei task soddisfatti, la percentuale dei task raggiunti e le totali medie teoriche. Come si può vedere nel grafico di figura 5.6, l'ausilio del terminale, riduce le distenze medie percorse per task dal punto iniziale.

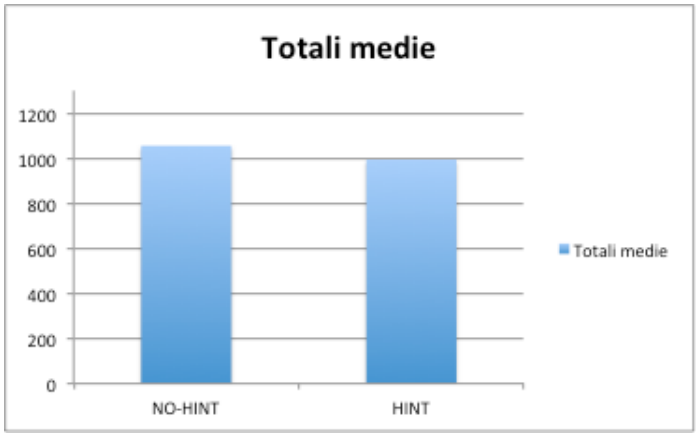


Figura 5.6: Confronto medie (totale) di Bassano del Grappa.

Il numero totale dei task assegnati era 10 per ogni utente, per cui il totale dei task per tutti gli utenti è di 40. Nel primo test, senza l'ausilio del terminale, il totale delle medie è nettamente superiore all'analogo totale del secondo test effettuato con il palmare, ma è da considerare come si vede nella colonna "task soddisfatti" che non tutti i compiti sono stati eseguiti. Infatti, sono 35 su 40 i task eseguiti, pari all'88 per cento. Nell'ultima colonna, è stato ipotizzato quale distanza media sarebbè teoricamente stata raggiunta se tutti i task, anche nel primo test, fossero stati soddisfatti e quindi è stato incrementato di una percentuale di circa il 12% (100% - 88%) il totale della media. Così facendo, abbiamo reso omogeneo i dati di confronto perché in tale stima si è tenuto conto anche del fatto che nel secondo test il terminale guidava l'utente al soddisfacimento di tutti i task, mentre nel primo caso, non utilizzando alcun supporto, l'utente oltre che compiere più strada si dimenticava di svolgere tutti i task, senza completare l'intero test.

Teoricamente, omogeneizzando i dati, come spiegato sopra, si nota come la differenza sia molto marcata. Ciò è stato evidenziato dal grafico in figura 5.7.

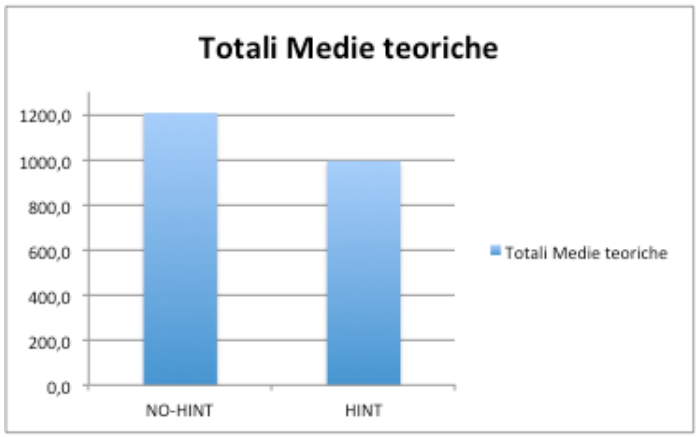


Figura 5.7: Confronto medie (teoriche) di Bassano del Grappa.

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.

5.2.3 Analisi dei valori medi differenziali (dal task precedente)

Qui si analizzano, invece, le distanze medie parziali. In questo caso la situazione non cambia. Si ottiene, infatti, una 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 riportati altri grafici (figure5.8 e 5.9) che analizzano gli inter-spazi (spazio tra due task soddisfatti).

	CINEMA	RISTORANTE	LIBRERIA	PONTE VECCHIO	PANE	LATTE	FARMACIA	POSTE	N. CENERE	GIORNALE	MEDIA DISTANZA PER TASK
Andrea	180	30	50	440	25	700	270	600	18	0	231,30
Davide	180	30	62	400	10		75	350	430	590	236,33
Michela	180	230	50	250	20	420	60		18	270	166,44
Stefano	180	30	350	350	250		45			220	203,57
MEDIA	**180**	**80**	**128**	**360**	**76,25**	**560**	**112,5**	**475**	**155,3**	**270**	**239,71**
Task soddisfatti	*4*	*4*	*4*	*4*	*4*	*2*	*4*	*2*	*3*	*4*	*35*

Figura 5.8: Valori medi delle interdistanze dei test a Bassano del G. senza terminale.

	CINEMA	RISTORANTE	LIBRERIA	PONTE VECCHIO	PANE	LATTE	FARMACIA	POSTE	N. CENERE	GIORNALE	MEDIA DISTANZA PER TASK
Michela	170	240	300	180	110	10	140	550	700	10	241,00
Manuel	160	400	130	350	87	130	140	350	350	400	249,70
Marco	180	81	300	90	600	170	56	350	20	10	185,70
Fabio	170	30	300	280	60	10	570	1000	77	50	254,70
MEDIA	**170**	**187,8**	**257,5**	**225**	**214,3**	**80**	**226,5**	**562,5**	**286,8**	**117,5**	**232,78**
Task soddisfatti	*4*	*4*	*4*	*4*	*4*	*4*	*4*	*4*	*4*	*4*	*40*

Figura 5.9: Valori medi delle interdistanze dei test a Bassano del G. con terminale.

Nella tabella successiva (figura 5.10), rispettivamente per ogni riga, senza terminale o con terminale, sono evidenziati i totali delle medie, il numero dei task soddisfatti, la percentuale dei task raggiunti e le totali medie teoriche. Come si può vedere nel seguente grafico, l'ausilio del terminale, riduce le distenze medie percorse per task dal punto iniziale.

Come nel caso dell'analisi dei valori medi per task dal punto iniziale, abbiamo reso omogeneo i dati di confronto perché in tale stima si è tenuto conto anche del

fatto che nel secondo test il terminale guidava l'utente al soddisfacimento di tutti i task, mentre nel primo caso, non utilizzavano alcun supporto; l'utente oltre che compiere più strada si dimenticava di svolgere tutti i task, senza completare l'intero test.

	Totali medie	Task soddisfatti	% di Task eseguiti	Totali Medie teoriche
NO-HINT	239,7	35	88%	272,4
HINT	232,7	40	100%	232,7

Figura 5.10: Confronto valori medi, hint, no hint, dal task precedente di Bassano del Grappa con e senza terminale.

Non omogeneizzando i dati, si vede (figura 5.11) comunque che il terminale ha conseguito ad una riduzione delle inter-distanze.

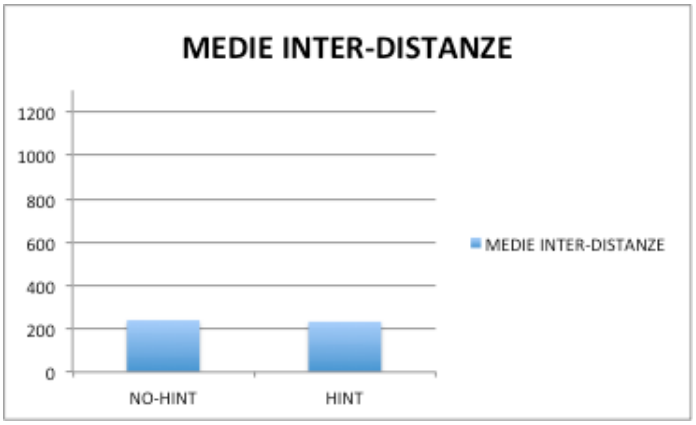


Figura 5.11: Confronto medie (totale) interdistanze cone e senza terminale di Bassano del Grappa.

Teoricamente, omogeneizzando i dati, come spiegato precedentemente, si ottiene il grafico di figura 5.12. Si nota come, anche in questo caso, la differenza sia molto marcata.

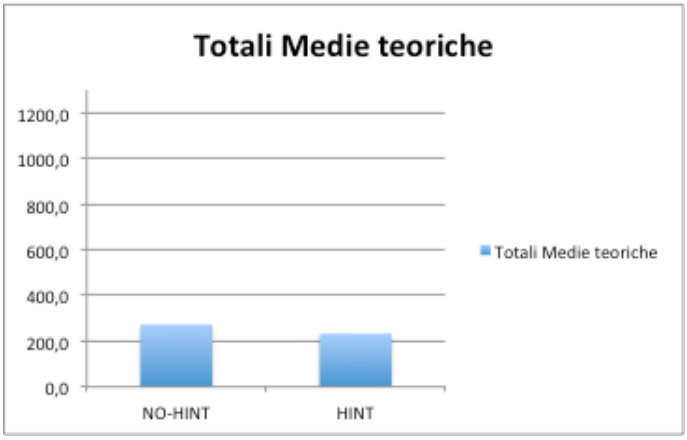


Figura 5.12: fig: Confronto medie (teoriche) interdistanze con e senza terminale di Bassano del Grappa.

5.2.4 Analisi delle singole prove

Una analisi direttamente sui dati raccolti nei due test può portare ad una valutazione generale sul sistema. Ciò è quanto abbiamo visto prima. Ora esaminiamo i dati statistici ottenuti nei test da un punto di vista delle singole prove dei partecipanti. I due grafici seguenti mostrano i metri percorsi da ogni partecipante per ogni task rispetto al punto inziale. Nel primo grafico (figura 5.13) gli utenti non hanno fatto uso del supporto sviluppato.

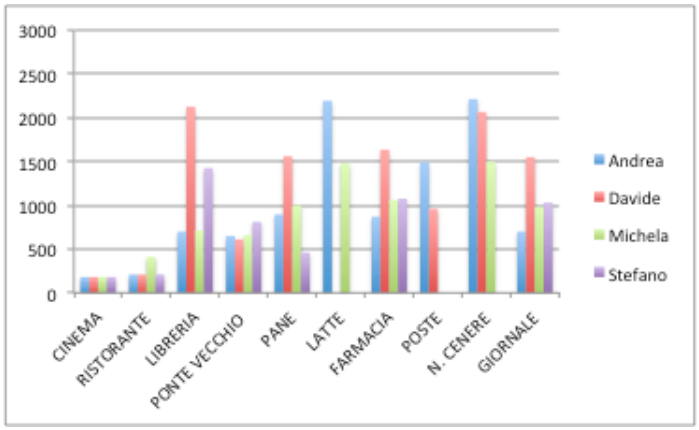


Figura 5.13: Grafico distanze singole prove dal punto inziale senza terminale di Bassano del G..

Nel seguente grafico (figura 5.14), invece, vengono rappresentati i dati che caratterizzano gli utenti che hanno fatto uso del terminale.

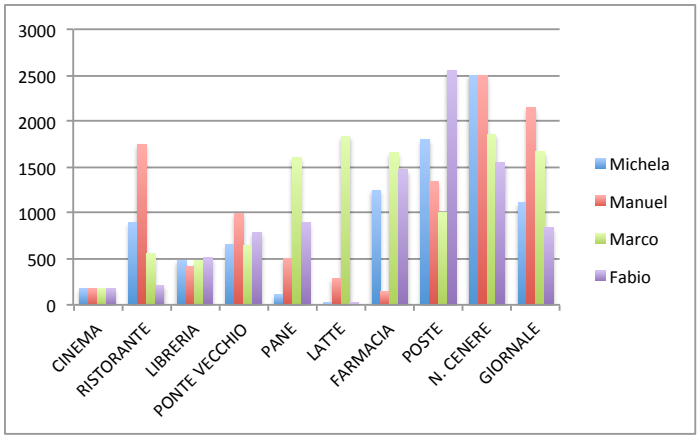


Figura 5.14: Grafico distanze singole prove dal punto inziale con terminale di Bassano del G.

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).

Si deduce perciò che le distanze percorse da ogni utente rispetto al punto iniziale, per soddisfare un task, sono maggiori in assenza del terminale.

Come già detto in precedenza, risulta utile valutare l'incremento o il decremento delle distanze percorse per portare a termine ciascun task.

I due grafici successivi (figure 5.15 e 5.16), invece, mostrano lo svolgimento dei test: ogni linea rappresenta la distanza percorsa per portare a termine ciascun task a partire dal task precedente, si parla quindi di inter-distanze, e quel che è possibile notare è che i tester che non si sono avvalsi dei suggerimenti, hanno percorso mediamente una distanza maggiore per portare a termine i vari task, questo a conferma del fatto che il portare a termine un task implica che si resti concentrati ancora su questa attività nonostante questa sia finita, mentre nel caso in cui si abbia a disposizione il nostro strumento, questi si occupa di ricordarci anche altre attività che si possono eseguire nei dintorni. Si può così parlare di economia di raggio d'azione.

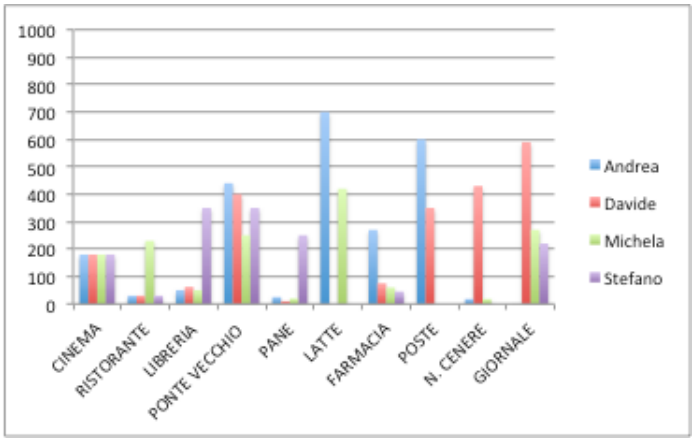


Figura 5.15: Grafico distanze singole prove dal task precedente senza terminale di Bassano del G..

Il fatto che i vari partecipanti abbiano un numero di linee diverse è dato dal fatto che non tutti si sono ricordati di portare a termine tutti i task, inoltre nel caso in cui due task siano stati portati a termine nello stesso punto, l'inter-distanza risulta ovviamente pari a zero. Per distinguere i due casi si può usare la tabella riassuntiva dei dati che, per ogni prova, riporta quanti task sono stati portati a termine.

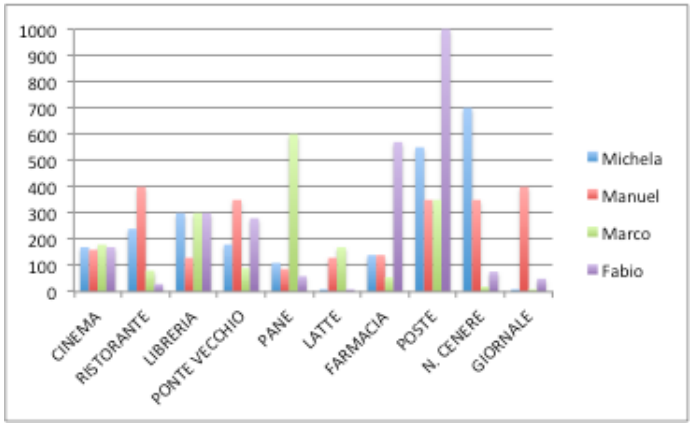


Figura 5.16: Grafico distanze singole prove dal task precedente senza terminale di Bassano del G.

Dal confronto tra i due grafici (figure 5.15 e 5.16), anche in questo caso, si nota palesemente che gli utenti che non potevano contare sul supporto del terminale mediamente hanno impiegato inter-distanze maggiori per portare a termine i vari task.

5.3 Test a Castelfranco Veneto

5.3.1 Il percorso programmato

In figura 5.17 è riportata la mappa del percorso seguito a Castelfranco Veneto. Tale mappa è utile per contestualizzare percorso e valutazioni fatte sui dati analizzati.

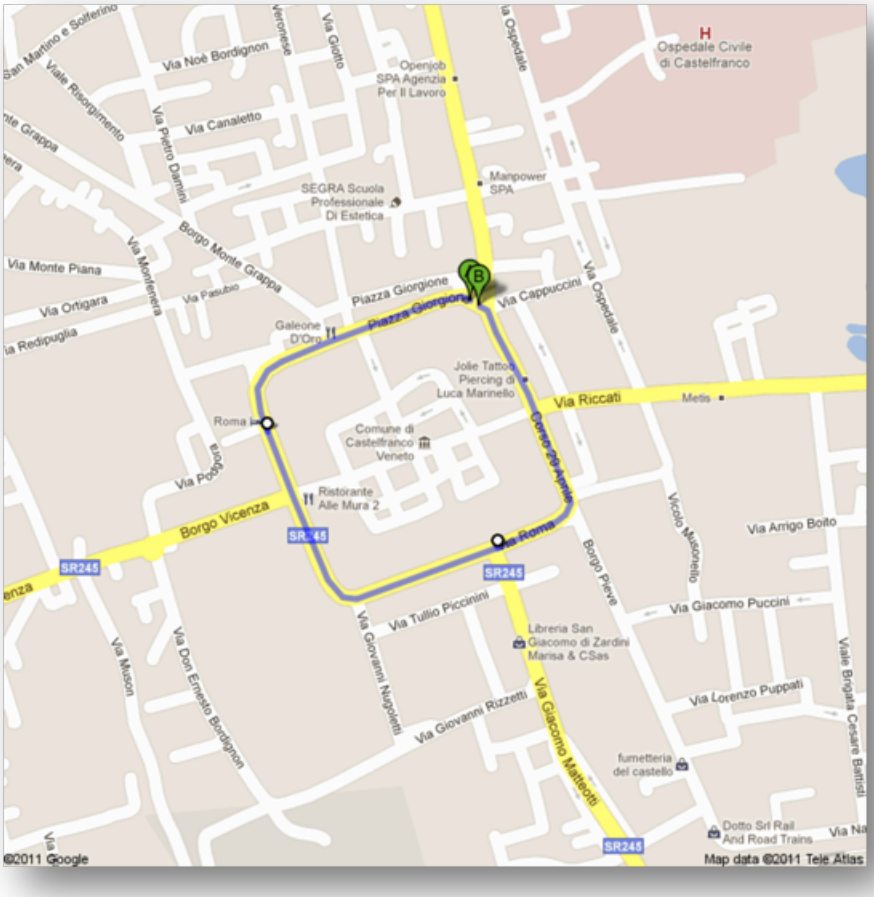


Figura 5.17: Il percorso di Castelfranco Veneto.

Ricordiamo che i task scelti per Castelfranco Veneto sono:

1. comprare il latte;
2. comprare il pane;
3. andare a prendere una medicina in farmacia;
4. andare in libreria;
5. Pprenotare un ristorante;

6. visitare il monumento “Casa del Giorgione”;

7. andare a prendere i biglietti al teatro;

8. andare a vedere la vetrina del negozio “Bertoldo”;

9. recarsi alle poste 10. comprare il giornale

Di seguito (figura 5.18) si riportano le età dei partecipanti in questo test:

No_hint	Età	Hint	Età
Romina	25	Marco	26
Marco	26	Michela	37
Alice	30	Alessio	30
Anna	26	Barbara	29
Media	26,75	**Media**	30,5

Figura 5.18: Età dei partecipanti per i test di Castelfranco Veneto.

Come si può notare sono tutti giovani, quindi eventuali dimenticanze durante il percorso senza cellulare sono da attribuirsi al fatto che magari non si sa dove sia di preciso un posto o a piccole dimenticanze, non alla presenza di un elevato deficit di memoria dovuto all’età.

5.3.2 Analisi dei valori medi dal punto iniziale

Per ottenere dei grafici comparativi, come nel caso dei test a Bassano del Grappa, 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 (figura 5.20) e senza (figura 5.19).

	latte	pane	farmacia	libreria	ristorante	monumento	teatro	negozio	poste	giornale	MEDIA DISTANZA PER TASK
Romina	70	70	1026	770	876	800	871	540	1626	140	678,9
Marco	1993	1993	1676	620	300	705	776	400	1276	10	974,9
Alice	1340	1340	5		300			400	800	10	599,2857
Anna	70	70	145	740		820			1270	140	465
MEDIA	**868,3**	**868,3**	**713**	**710**	**492**	**775**	**823,5**	**446,7**	**1243**	**75**	701,5
Task soddisfatti	*4*	*4*	*4*	*3*	*3*	*3*	*2*	*3*	*4*	*4*	*34*

Figura 5.19: Valori medi delle distanze dal punto iniziale dei test a Castelfranco V. senza terminale.

	latte	pane	farmacia	libreria	ristorante	monumento	teatro	negozio	poste	giornale	MEDIA DISTANZA PER TASK
Marco	155	75	5	1060	425	875	930	525	1410	10	547
Michela	60	60	130	270	641	491	420	791	1191	125	417,9
Alessio	80	80	800	175	955	675	950	1255	1655	425	705
Barbara	135	135	205	95	495	945	1000	595	1500	200	530,5
MEDIA	**107,5**	**87,5**	**285**	**400**	**629**	**746,5**	**825**	**791,5**	**1439**	**190**	550,1
Task soddisfatti	*4*	*4*	*4*	*4*	*4*	*4*	*4*	*4*	*4*	*4*	*40*

Figura 5.20: Valori medi delle distanze dal punto iniziale dei test a Castelfranco V. con terminale.

Facendo la media delle medie dei dati si trova (figura 5.21):

	Totali medie	Task soddisfatti	% di Task eseguiti	Totali Medie teoriche
NO-HINT	701,5	34	85%	825,3
HINT	550,1	40	100%	550,1

Figura 5.21: Confronto valori medi, hint, no hint, dal punto inziale di Castelfranco V. con e senza terminale.

In questa tabella, da una prima veloce analisi se ne ricava che, con l'utilizzo del sistema, si verifica una riduzione della distanza media per il soddisfacimento di un Task (vedi figura 5.22). La situazione è positiva in quanto effettivamente c'e un guadagno sia in termini di spazio percorso che, conseguentemente, di tempo impiegato per portare a termine gli impegni. Come per i test di Bassano del Grappa, anche per Catselfranco Veneto, si è calcolata la media teorica per confrontare dati omogenei, ossia tenendo conto del fatto che per i test senza cellulare sono stati dimenticati il 5% dei task. Il numero totale dei task assegnati era 10 per ogni utente, per cui il totale dei task per tutti gli utenti è di 40. Nel primo test, senza l'ausilio del terminale, il totale delle medie è nettamente superiore all'analogo totale del secondo test effettuato con il palmare, ma è da considerare come si vede nella colonna "task soddisfatti" che non tutti i compiti sono stati eseguiti. Infatti, sono 34 su 40 i task eseguiti, pari all'85 per cento.

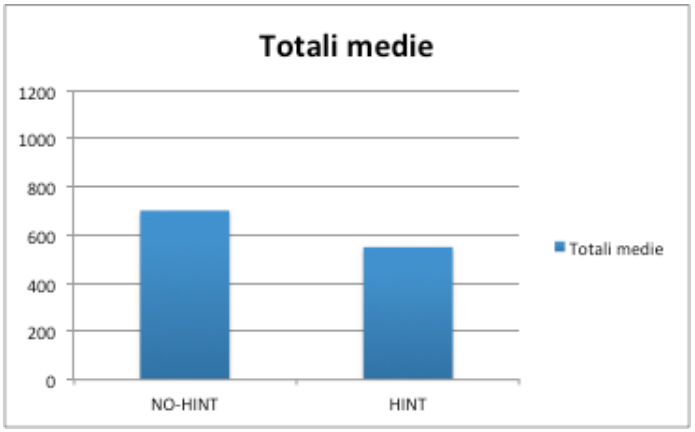


Figura 5.22: Confronto medie (totale) di Castelfranco Veneto.

Teoricamente, omogeneizzando i dati, come spiegato prima, si ottiene il grafico di figura 5.23. Si nota come la differenza sia molto marcata.

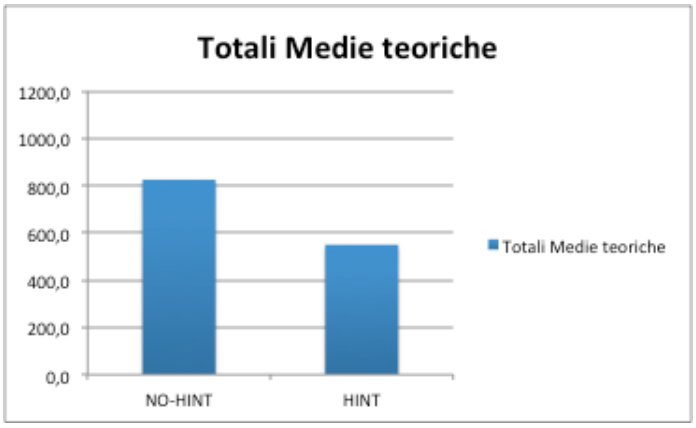


Figura 5.23: Confronto medie (teoriche) di Castelfranco Veneto.

5.3.3 Analisi dei valori medi differenziali (dal task precedente)

Analizzando la media delle distanze medie parziali (figure 5.24 e 5.20), cioè la distanza media tra il soddisfacimento di un task e il precedente, notiamo che anche qui5.21 la media con hint è inferiore rispetto a quella senza. In questo caso si può dedurre che, in generale, con la nostra applicazione, si possono perseguire più obiettivi in un raggio più limitato.

no_hint	latte	pane	farmacia	libreria	ristorante	monumento	teatro	negozio	poste	giornale	MEDIA DISTANZA PER TASK
Romina	0	70	150	230	5	80	71	400	600	70	167,6
Marco	0	317	400	220	290	85	71	100	500	10	199,3
Alice	0	540	5		290			400	400	5	234,3
Anna	0	70	5	595		80			450	70	181,4
MEDIA	**0,0**	**249,3**	**140**	**348,3**	**195**	**81,7**	**71**	**300,0**	**487,5**	**38,75**	191,2
Task soddisfatti	*4*	*4*	*4*	*3*	*3*	*3*	*2*	*3*	*4*	*4*	*34*

Figura 5.24: Valori medi delle interdistanze dei test a Castelfranco V. senza terminale.

hint	latte	pane	farmacia	libreria	ristorante	monumento	teatro	negozio	poste	giornale	MEDIA DISTANZA PER TASK
Marco	80	65	5	130	270	350	55	100	350	5	141
Michela	0	60	5	140	150	71	150	150	400	65	119,1
Alessio	80	0	125	95	5	250	150	300	400	250	165,5
Barbara	0	40	5	95	290	350	55	100	500	65	150,0
MEDIA	**40,0**	**41,3**	**35**	**115**	**178,8**	**255,3**	**102,5**	**162,5**	**412,5**	**96,25**	143,9
Task soddisfatti	*4*	*4*	*4*	*4*	*4*	*4*	*4*	*4*	*4*	*4*	*40*

Figura 5.25: Valori medi delle interdistanze dei test a Castelfranco V. con terminale.

	Totali medie	Task soddisfatti	% di Task eseguiti	Totali Medie teoriche
NO-HINT	191,2	34	85%	224,9
HINT	143,9	40	100%	143,9

Figura 5.26: Confronto valori medi, hint, no hint, dal task precedente di Castelfranco V. con e senza terminale.

Nella tabella (figura 5.26), rispettivamente per ogni riga, senza terminale o con terminale, sono evidenziati i totali delle medie, il numero dei task soddisfatti, la percentuale dei task raggiunti e le totali medie teoriche omogeneizzando i dati. Come si può vedere nel grafico di figura 5.27, l'ausilio del terminale, riduce le distenze medie percorse per task dal punto iniziale.

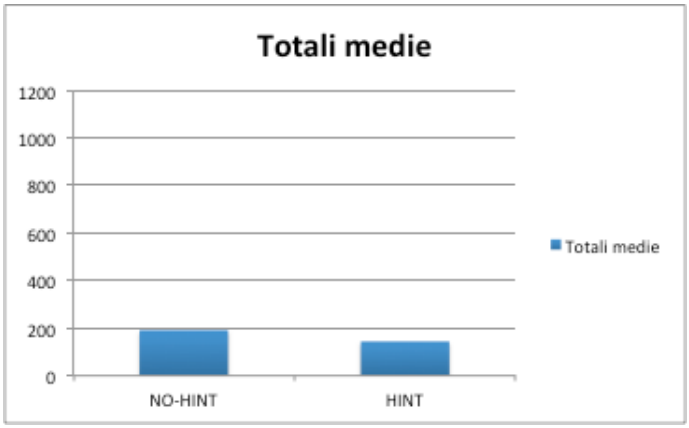


Figura 5.27: Confronto medie (totale) interdistanze con e senza terminale di Castelfranco V.

Omogeneizzando i dati, tenendo conto dei task dimenticati nel caso dei test senza terminale, otteniamo il seguente grafico (figura 5.28).

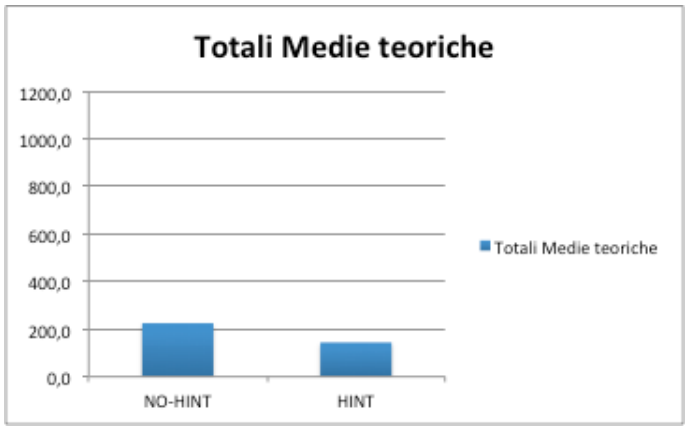


Figura 5.28: Confronto medie (teoriche) interdistanze con e senza terminale di Castelfranco V.

Anche qui, come per Bassano del Grappa, 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.

5.3.4 Analisi delle singole prove

L'analisi diretta dei dati ottenuti durante le due prove può portare ad una valutazione più generale.

I grafici in figura 5.29 e 5.30 mostrano le distanze dal punto iniziale del soddisfacimento dei task da parte di ogni utente (rappresentato da un colore diverso come spiegato nella legenda).

Come si può notare dai grafici e dai dati presentati nella tabella in figura 5.21, si nota che il numero medio di task portati a termine nel caso senza hint è pari al 85%, mentre nel caso con Hint al 100%.

Passando ad analizzare i dati relativi alle inter-distanze (figura 5.27) si ottengono i grafici in figura 5.31 e 5.32 : ogni linea rappresenta la distanza percorsa per portare a termine ciascun task a partire dal task precedente, si parla quindi di inter-distanze.Dal confronto tra i due grafici, si nota che gli utenti che non potevano contare sul supporto del terminale mediamente hanno impiegato distanze maggiori per portare a termine i vari task.

Questo sta a significare che, come detto nel paragrafo precedente, in un raggio più ristretto, usando il terminale, si portano a termine più task.

Il fatto che i vari tester abbiano un numero di linee diverse è dovuto al fatto che non tutti hanno portato a termine tutti i task, oltre a ciò se due task sono stati soddisfatti nello stesso luogo l'inter-distanza risulta essere zero, quindi per distinguere i due casi si vedano i dati nelle Tabelle 5.24 e 5.25.

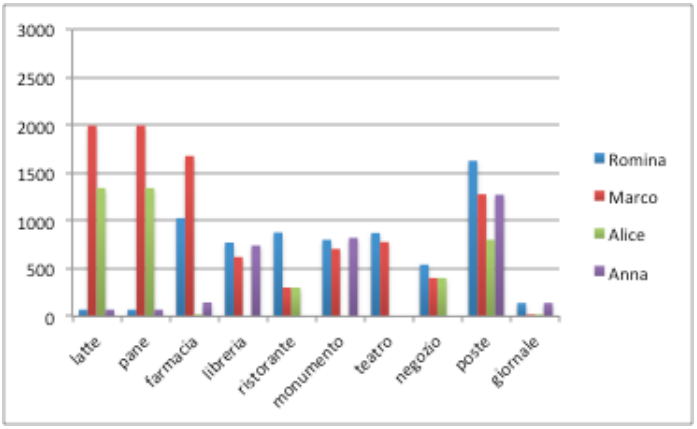


Figura 5.29: Grafico distanze singole prove dal punto iniziale senza terminale di Castelfranco Veneto.

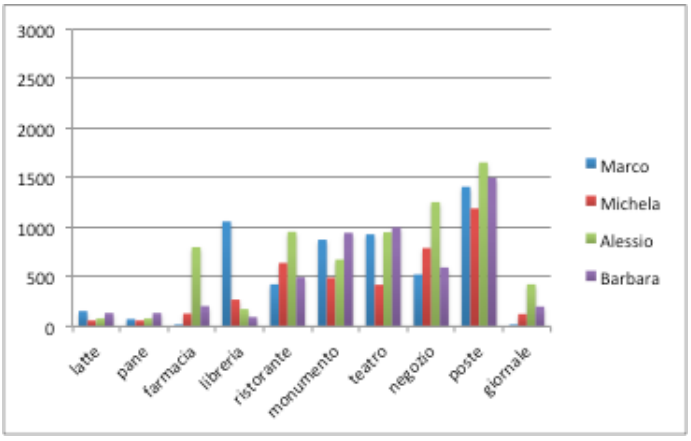


Figura 5.30: Grafico distanze singole prove dal punto iniziale con terminale di Castelfranco Veneto.

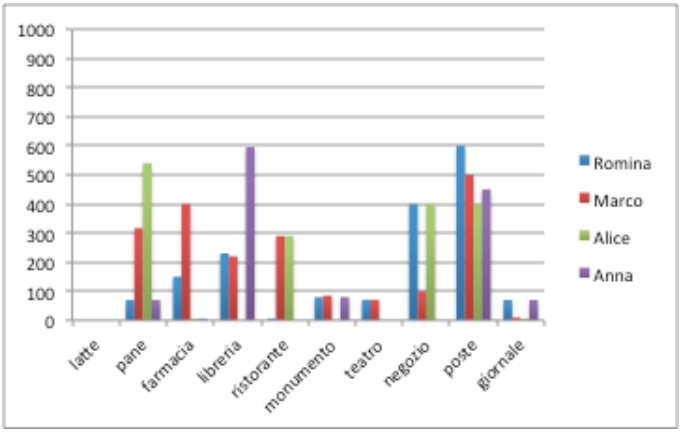


Figura 5.31: Grafico distanze singole prove dal task precedente senza terminale di Castelfranco Veneto.

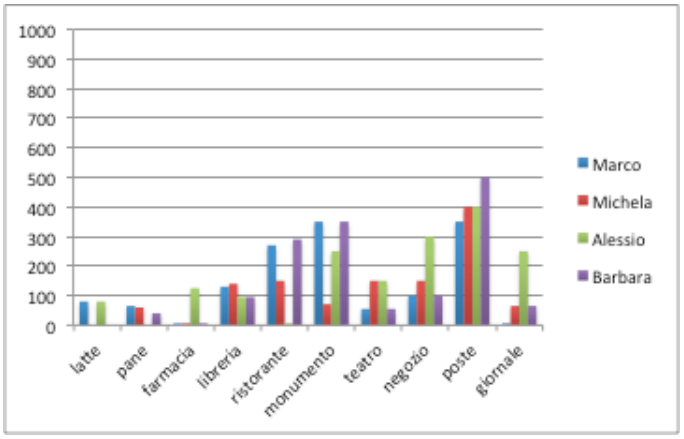


Figura 5.32: Grafico distanze singole prove dal task precedente con terminale di Castelfranco Veneto.

Altro dato rilevante è il numero di task soddisfatti per deviazione effettuata. Considerando i valori medi si ottengono i risultati in figura 5.33:

NO_HINT		
	n° deviazioni medio	2
	n task soddisfatti x dev medio	2,375
HINT		
	n° deviazioni medio	2
	n task soddisfatti x dev medio	2,5

Figura 5.33: Deviazioni a Castelfranco Veneto.

Dalla tabella si evince che a parità di numero di deviazioni medie nel caso senza hint e con hint, il numero medio di task soddisfatti per deviazione è più elevato nel caso con hint. Questo risultato sta a significare che con il nostro sistema vengono suggeriti posti nelle vicinanze che, o per distrazione o perché non si conoscono, senza suggerimenti non verrebbero presi in considerazione per soddisfare un determinato task.

5.4 Confronto tra i due test

I percorsi scelti nelle due locazioni, Bassano del Grappa e Castelfranco Veneto, sono molto simili per quanto riguarda la lunghezza. Anche per questo motivo, si hanno avuto dei comportamenti pressochè uguali. A Bassano del Grappa la percentuale di task soddisfatti degli utenti sprovvisti di cellulare è dell'88%, a Castelfranco Veneto, invece, del 85%. Per gli utenti che usavano lo smartphone, ovviamente, per entrambe le locazioni, tutti i task sono stati soddisfatti.

In generale, i risultati ottenuti, sottolineano nei comportamenti degli utenti muniti di terminale una riduzione delle distanze medie dal punto iniziale. Nel caso di Castelfranco Veneto la riduzione è più marcata, circa 150m, mentre per Bassano del Grappa sono 60m.

Comparando, invece, i grafici di entrambi i test relativi alle interdistanze medie tra un task ed un altro, con e senza cellulare, per Castelfranco la media è di 50m, per Bassano di 7m come si può notare nella tabella delle interdistanze medie (figura 5.21). In entrambi i casi si è notato un risparmio di metri percorsi.

5.5 Analisi dei questionari

Al fine di valutare l'esperienza dell'applicazione da parte degli utenti che hanno svolto i test che abbiamo analizzato sopra, 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.

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 generali, chiedendo se l'utente ritenga utile l'applicazione, se l'interfaccia era semplice da utilizzare, se gli è piaciuta l'applicazione, ecc.;
- *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;
- *Interfaccia*: si chiede all'utente di rispondere a domande relative all'aspetto e alla facilità d'uso dell'interfaccia;
- *Conclusioni*: in seguito all'esperienza avuta nell'utilizzo del sistema di supporto mnemonico, si richiede una valutazione globale e una ipotetica disponibilità all'acquisto del sistema.

I risultati ottenuti dai questionari compilati dagli utenti sono pressoché simili; questo atteggiamento sta a significare che l'esperienza avuta dagli utenti può essere considerata molto positiva e che l'applicazione è potenzialmente utile.

Di seguito riportiamo le domande del questionario creato per valutare l'applicazione. A fianco della domanda, tra parentesi, abbiamo inserito il range dei valori ammissibile per quella domanda.

QUESTIONARIO

1. Ti è piaciuta l'applicazione? (1: poco o per niente - 10: molto)
2. Ti sembra utile? (1: poco o per niente – 10: molto)
3. L'interfaccia è semplice da utilizzare? (1: poco o per niente – 10: molto)
4. Dove hai tenuto il cellulare durante il percorso? (1: prevalentemente in tasca 2: in mano)
5. Ti sono servite le notifiche date dal programma? (1: poco o per niente – 10: molto) - Visuali - Vibrazione - Audio - Vocali
6. Le notifiche ricevute sono congruenti al soddisfacimento dei task? (1: poco o per niente – 10: molto)
7. Saresti andato negli stessi posti indicati dall'applicazione? (1: no – 2: più no che si - 3: metà e metà – 4: più si che no – 5: si)
8. Mediamente hai dovuto deviare di tanto per soddisfare una notifica segnalata dal sistema? (1: poco o per niente – 10: molto)
9. Hai utilizzato la mappa? (1: Si – 2: No)
10. Se sì, ti è stata utile? (1: Si – 2: No)
11. Si sono verificati crash o malfunzionamenti dell'applicazione durante l'utilizzo? (1: Si – 2: No)
12. Ritieni utile utilizzare dei punti d'interesse personali? (1: Poco o per niente – 10: Molto)
13. Ritieni utile condividere con altri utenti i tuoi punti d'interesse? (Le altre persone possono usare i luoghi pubblici segnalati da te) (1: Poco o per niente – 10: Molto)
14. Ritieni utile poter inserire dei task che possono essere visualizzati anche da altre persone? (1: Poco o per niente – 10: Molto)
15. Trovi particolarmente fastidioso specificare il task (istruire il sistema a risolvere un dato task con determinate categorie di luoghi) la prima volta che lo inserisci? (1: Poco o per niente – 10: Molto)
16. Trovi facile inserire nuovi luoghi (Privati e Pubblici)? (1: Poco o per niente – 10: Molto)
17. Trovi sufficientemente semplice l'interfaccia "New Places"? (1: Poco o per niente – 10: Molto)
18. Trovi semplice la ricerca e la votazione di un luogo inserito da altri? (1: Poco o per niente – 10: Molto)

19. Trovi che l'applicazione risponda in tempi adeguati alle tue richieste? (1: Poco o per niente – 10: Molto)

20. Senza una guida utente l'utilizzo dell'applicazione è intuitivo? (1: Poco o per niente – 10: Molto)

21. Se poco o per niente, quale parte del programma risulta poco intuitiva? (1: Inserimento task – 2:Nuovi luoghi – 3: Assertion – 4:Mappa – 5:Done del task)

22. Complessivamente, come valutereste l'applicazione utilizzata? (1: Poco o per niente – 10: Molto)

23. Utilizzereste l'applicazione se non fosse gratuita? (1: Si – 2: No)

24. Quanto saresti disposto a pagare per questa applicazione? (1: 1 euro o meno – 2: 2-4 euro - 3: 5-10 euro – 4: 11 euro o più)

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).

Vediamo ora i risultati ottenuti dagli utenti di Bassano del Grappa e Castelfranco Veneto, riassunti tramite una tabella (figura 5.34).

Dom. n°	Fabio	Manuel	Marco	Michela	Marco	Alessio	Barbara	MEDIA
GENERALE								
1	10	10	8	9	9	8	8	8,86
2	10	8	8	10	7	7	8	8,29
3	9	5	6	8	7	6	8	7,00
4	2	2	2	2	2	2	2	
NOTIFICHE								
5a	10	4	7	10	10	10	8	8,43
5b	9	4	6	10	8	9	9	7,86
5c	9	4	6	9	9	9	4	7,14
5d	6	4	6					5,33
6	9	6	9	10	9	8	8	8,43
7	4	5	5	3	3	2	2	
8	4	2	3	3	3	7	2	3,43
9	SI	SI	SI	SI	SI	SI	SI	
10	SI	SI	SI	NO	SI	SI	SI	
FUNZIONALITA'								
11	NO	NO	NO	NO	NO	NO	NO	
12	9	6	6	10	8	9	9	8,14
13	10	10	8	10	8	7	7	8,57
14	10	8	10	10	9	7	7	8,71
INTERFACCIA								
15	5	4	5	2	2	6	2	3,71
16	9	2	8	10	9	8	7	7,57
17	9	10	7	10	9	8	5	8,29
18	9	9	5		7	9	5	7,33
19	9	5	7	10	9	10	8	8,29
20	8	7	6	6	6	5	6	6,29
21	5	/	5	5	5		2	
CONCLUSIONI								
22	9	8	8	9	9	8	8	8,33
23	SI	SI	NO	NO	SI	NO	SI	
24	2	1	1	2	2	2	2	

Figura 5.34: Risultati Questionario.

Generale

È emersa una valutazione ottima dell'applicazione, gli utenti la reputano utile e non molto diffile da utilizzare.

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. Le notifica più apprezzate sono quelle visuali, quelle audio e le vibrazioni. Meno apprezzata la notifica vocali, presumibilmente perché durante i nostri test, effettuati camminando, sentire il terminale che legge i task è un po' fastidioso, si preferisce pittosto un unico effetto audio che l'utente associa all'arrivo di nuove notifiche e possibilità.

Perciò, tra le modalità di notifica, quella considerata migliore è stata quella visuale e la vibrazione con una media di 7,25, seguita da quella tramite audio con una media di 7. Quella vocale non è stata utilizzata dagli utenti che hanno effettuato i test.

Per quanto riguarda la qualità 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 (gli utenti inzialmente non hanno inserito nuovi luoghi), 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, è far esprimere pareri agli utenti relativamente alle funzionalità del sistema.

Durante i test non si sono verificati malfunzionamenti di nessun tipo e come si può notare dalla tabella la media voti delle domande relative a questa sezione è molto alta.

Ne deduciamo che gli utenti hanno apprezzato le diverse funzioni diponibili. Anche la possibilità di sfruttare il navigatore per ragiungere il luogo dove poter soddisfare un task.

Interfaccia

Gli utenti hanno trovato poco fastidioso aiutare il sistema dopo l'inserimento di un task, nel caso quest'ultimo sia inserito per la prima volta. Quindi non è

fastidiosissimo effettuare le votazioni, anche perché il fatto di comunicare al sistema che un determinnato oggetto lo si desida trovare nei posti a preferiti è considerato un valore aggiunto. L'interfaccia, in base ai pareri degli utenti, non è difficile da comprendere. Dopo qualche minuto, gli utenti riescono ad usare l'applicazione senza alcuna difficoltà.

Conclusioni

La valutazione complessiva dell'utente, in generale, è piuttosto buona. Anche se si tratta comunque di un'applicazione per smartphone, è pur sempre un'aplicazione diversa dal comune. Pochi utenti hanno mai utilizzato applicazioni similari. Anche per questo, molti sono rimasti sorpresi dal suo funzionamento.

5.6 Confronto tra gli ultimi test e quelli passati

Alessio Toso[1] [da pag.67 a 81 di [8]] e Guido Geloso[2] [da pag.46 a 60 di [7]], entrambi studenti di Ingegneria Informatica, che durante lo svolgimento della loro tesi hanno partecipato a questo progetto, hanno effettuato i test a Treviso e Genova. Le modalità con cui sono stati fatti i test, che si sono presentati prima, ricalcano, grosso modo, proprio i test passati. Ciò è neccessario, se si vuole confrontare i risultati ottenuti.

I test eseguiti a Genova con l'utilizzo del terminale avevano riportato una riduzione della distanza media dal punto iniziale e anche tra il soddisfacimento di un task ed il successivo, rispetto al caso senza terminale. Situazione che corrisponde anche ai dati trovati per i test di Castelfranco Veneto e Bassano del Grappa.

Invece il test a Treviso aveva avuto una distanza media del soddisfacimento del task dal punto iniziale superiore nel caso di utilizzo del terminale rispetto al caso senza. Tale risultato era dovuto al fatto che, essendo il percorso di Treviso sommariamente circolare, l'utente decideva di compiere in un secondo momento task che potevano venire notificati anche più avanti e soddisfare prima quelli più vicini. Nonostante questa situazione la media delle interdistanze tra il soddisfacimento di due task risultava comunque inferiore nel caso di utilizzo dell'applicazione.

Passando ora al confronto tra le risposte ottenute nel loro questionario e quelle relative al nostro si è notato che la maggior parte dei risultati era concorde con quanto da noi concluso nel paragrafo precedente. Ci sono state, però, delle piccole divergenze:

- La modalità di notifica preferita nei test di primavera era quella vibratoria, in quanto gli utenti che avevano effettuato il test avevano tenuto prevalentemente il cellulare in tasca. Negli ultimi test la modalità preferita è risultata quella visuale (seguita da quella vibratoria), in questo caso però il terminale è stato tenuto prevalentemente in mano;

[1] Alessio Toso è stato uno studente dell'Università degli studi di Padova, ha ottenuto la laurea specialistica in Ingegneria Informatica il 15/03/2011.

[2] Guido Geloso è stato uno studente dell'Università degli studi di Genova, ha ottenuto la laurea specialistica in Ingegneria Informatica il 11/03/2011.

- Nei test passati durante l'utilizzo del sistema si erano verificati dei crash, invece nei test attuali questa situazione non si è verificata. Questo è senza dubbio un passo in avanti per quanto riguarda la stabilità del sistema.

Concludendo, si può sostenere che l'applicazione si è evoluta con l'ampliamento dell'ontologia e con la possibilità di aggiungere nuovi posti, entrambi aspetti molto aprezzati dagli utenti che si sono prestati nei test. Oltre a ciò, si è ottenuto una versione dell'applicazione che funziona bene e che durante i test non ha riscontrato problemi.

Capitolo 6

Possibili sviluppi futuri

6.1 Miglioramento della comprensione del task/event

Uno degli obiettivi del sistema è rendere la comprensione dei task molto precisa. L'idea che si intende è far uso di un qualche sistema o di instaurare un qualche meccanismo capace di analizzare la frase inserita da un utente, ed estrappolare il bisogno reale. Al momento si cerca di capire il task, cercando dei match sul database ontologico o sul file dell'ontologia. Per migliorare questo aspetto, bisognerebbe spezzettare la frase in soggetto, verbo, complement oggetto, etc. In questo modo, aumentando la complessità del reasoning, si riuscirebbe a comprendere effettivamente quale sia l'esigenza dell'utente.

6.2 Miglioramento interfaccia client

Per la maggior parte delle persone che hanno contribuito a questo progetto è stata la prima vera esperienza nella programmazione in ambiente Android. Inizialmente non tutti avevamo mai avuto a che fare con smartphone di ultima generazione, ne con quelli che erano i problemi che abitualmente insidiano gli utenti di questo genere di sistemi.

Ci siamo però resi conto ben presto di quanto fossero complesse o scomode alcune operazioni che, utilizzando l'emulatore con l'ausilio del mouse e della tastiera, ci risultavano molto semplici. Ad esempio, scrivere una frase molto lunga può diventare difficile ed è spesso funzione del programma usato come tastiera; inoltre anche solo il fatto di trovare un programma all'interno del menù può diventare tedioso quando si installano molte applicazioni, in quanto lo scorrimento veloce del menu rende difficile l'individuazione dell'icona relativa al programma ricercato; esistono poi vari desktop su cui aggiungere le icone più usate, ma lo spazio a disposizione è funzione dell'ampiezza dello schermo del dispositivo e, per questo, può essere molto facile da riempire.

Le interface del client, con cui l'utente instaura un diretto rapporto con l'intero sistema, sono state progettate cercando di rendere intuitiva l'esperienza dell'utente. Per il motivo, descritto all'inizio di questo paragrafo, attraverso uno studio più ap-

profondito sull'usabilità dell'applicazione, si potrebbe rendere le interface ancora più semplici.

Per queste ragioni uno dei prossimi obiettivi dovrebbe essere quello di aggiungere al progetto la realizzazione di un'interfaccia migliore capace di soddisfare maggiormente l'esperienza dell'utente.

6.3 Stabilizzazione del sistema

Il sistema, nei test che si sono fatti (vedi Capitolo 5), si è comportato abbastanza bene. Non ci sono stati malfunzionamenti e il terminale non si è mai bloccato. Le variabili in gioco, nel funzionamento di tale applicativo sono talmente tante e diverse, che non sorprende che ci sia ancora qualche tentativo di stabilizzare il sistema. I problemi potrebbero nascere nel caso in cui il segnale GPS venga meno. In quel caso specifico, bisognerebbe che il terminale si accorgesse della mancanza, comunicasse il problema all'utente e si ponesse in uno stato di standby. Ad intervalli regolari, l'applicativo lato client dovrebbe controllare se il segnale è ritornato accettabile e in tal caso riprrendere il funzionamento.

Tuttavia il risolvere tale problema non dovrebbe presentare difficoltà concettuali, ma solo un esercizio di ricodifica.

6.4 Cassandra nel nostro sistema

Con l'avvento del Web 2.0 la rete ha sperimentato per la prima volta le criticità legate alla gestione infrastrutturale di un ambiente che dovesse sostenere ed accentrare le attività tipiche di una community. Indirizzare correttamente tematiche come lo scaling, il tempo di risposta e la gestione di grossissimi volumi di dati è lentamente diventato essenziale per una cerchia sempre più vasta di player web, tra i quali cito Facebook[100], YouTube[101], Flickr[102] e molti altri ancora.

Per risolvere questa tematica e tra le tante osservazioni che si possono fare una delle più importanti è notare che la maggior parte delle più popolari applicazioni web a sfondo social possono essere sviluppate anche senza un'utilizzo massiccio delle funzionalità relazionali tipiche dei RDBMS. In questo modo nascono tutta una serie di prodotti che, rinunciando a concetti come chiavi esterne, joins, database schemas, consentono di memorizzare in modo rapidissimo e distribuito strutture più o meno simili ad Hash {chiave:valore}.

Cassandra è una database distribuito, fault-tolerant, elastico e con consistenza dei dati regolabile sia in read che in write; questo significa che il server è solitamente installato in una configurazione clustered nella quale più nodi di Cassandra cooperano per ottimizzare e distribuire le informazioni. Nessun nodo è in alcun modo diverso dagli altri, in questo modo non esiste all'interno del cluster una specificità critica che in caso di malfunzionamento possa rendere inoperante il database.

I dati sono automaticamente duplicati su più nodi, questo garantisce che all'eventuale crash di un elemento dell'insieme non debba necessariamente seguitare la perdita di informazioni importanti o lo stallo dell'intera istanza di Cassandra.

Durante le operazioni di read/write, è possibile esplicitare il livello di consistenza che si vuole mantenere: l'impostazione si attua passando alla funzione di lettura/-scrittura due parametri che indicano il comportamento atteso; le scelte disponibili variano per le funzioni di write:

- "esegui tutte le operazioni in asincrono" e conseguentemente non verificare l'effettiva scrittura del dato
- "scrivi e verifica la scrittura almeno sulla metà + 1 del numero di nodi indicati dalle impostazioni di fault-tolerance"
- "scrivi e verifica la scrittura su ogni nodo"

Analogamente esistono parametri simili per le funzioni di read che consentono di scegliere da quanti nodi effettuare la lettura prima di determinare quale sia la versione più fresca del dato richiesto.

Aumentando il numero di nodi da leggere, così come aumentando quelli da scrivere per quanto riguarda le funzioni di write, si ottiene un lineare miglioramento nella consistenza del dato a scapito delle performance: è quindi importante decidere con attenzione quale sia il miglior compromesso per l'applicazione.

La sezione luoghi dell'applicazione descritta nel Capitolo 4.3.2, fa uso di un database SQL, in particolare mySql, a cui gli utenti accedono per un periodo brevissimo, effettuando una semplice transazione. Dal momento che tale base di dati, dovrebbe resistere all'accesso contemporaneo di molti utenti, Cassandra[98, 99] risulta molto indicato per garantire prestazioni elevate.

In base alle premesse fatte, Cassandra potrebbe essere un buon sostituto del database relazionale usato. E', infatti, indicato per gestire grandi quantità di dati dislocati in diversi server, fornendo inoltre un servizio orientato alla disponibilità, senza single point of failure. Bisognerebbe valutare la convenienzza di una tale sostituzione, con particolare riguardo alle performance.

6.5 Miglioramento della sezione Places

Uno degli aspetti più significativi della sezione social del nostro sistema è quello di poter inserire dei luoghi pubblici. Quest'ultimi diventeranno disponibili per tutta la communità di utenti che potranno votarli o meno. Una simpatico add-on a ciò sarebbe quello di poter inserire eventuali opinioni sul posto segnalato, in modo da creare un vero e proprio social network relativo solamente ai luoghi. Nel momento in cui, all'utente arrivano degli Hint (posti dove soddisfare un task), avrebbe la possibilità di consultare le opinioni e pareri inseriti da altri utenti che utilizzano l'applicazione.

6.6 Gestione del silenziamento

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. 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 aggiornamento) per l'utilizzo dell'applicazione a piedi, potrebbe essere 200mt e 250mt, 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.

6.7 Sostituzione delle Local Search API con Maps Api Web Service

Le Api di Google che vengono utilizzate attualmente in questo progetto sono di vecchia generazione, in particolare sono le Local Search Api[1][71]. Esiste già da parecchio tempo, la nuova versione, che effettivamente va a sostituire la vecchia. Conseguenza di ciò, è che Google, da maggior priorità al traffico entrante relativamente alle nuove Api, cercando di limitare il più possibile quelle vecchie. Dati i vincoli imposti sulle api non più di ultima generazione, una delle principali azioni da svolgere per conti-

[1] `http://code.google.com/apis/maps/documentation/localsearch/index.html`

nuare il presente lavoro è sicuramente quella di sostituire le vecchie API con quelle nuove, le Google Maps API Web Services[2][103].

Con l'utilizzo delle Api di ultima generazione, Google mette a disposizione una personale interfaccia grafica, che permette di analizzare le statistiche relative al numero di volte che si utlizzano i servizi messi a disposizione dai web service di Google. Tale caratteristiche, potrebbe essere molto interessante, anche per vedere se stiamo innondando questi servizi di richieste inutili.

6.8 Funzione "Sbriga tutti i task"

Il sistema descritto in questo scritto, mira ad aiutare l'utente a non scordare e risolvere i task inseriti, rispettando eventuali deadline. Quindi l'utente, nel corso della sua giornata, tra tutti i vari impegni, può far uso del sistema in questione. Al momento non è stata predisposta nessuna funzionalità che cerca di risolvere tutti i task inseriti dall'utente definendo il percorso più breve per portarli a termine al momento senza dilazionarli nell'intera giornata, tra un'impegno ed un'altro. Sarebbe interessante estendere l'utilizzo di questa applicazione anche in casi, in cui un utente inserisce una serie di impegni da sbrigare al momento e il sistema attua uno scheduling inteliggente, capace di guidare l'utente al soddisfacimento di tutti i task uno dopo l'altro, ma facendogli percorrere la strada più breve.

[2] `http://code.google.com/apis/maps/documentation/webservices/index.html`

APPENDICE A

Installazione sistema operativo

Per installare il sistema operativo procediamo con:

- Formattare il disco con qualche utility in ext3. Attenzione:non in FAT32[104] altrimenti nascono delle incompatibilità a livello di comandi shell;
- Installare Ubuntu Server facendo attenzione a non installare Virtual Host[105] (altrimenti va in conflitto con la successiva installazione di vmware server). Installare, invece, OpenSSH server, Email Server e Samba[106]. Eventualmente se necessario installare la piattaforma LAMP. Per quello che dobbiamo fare noi servono solo i primi 3.

Configurazione della scheda di rete del server

Nel nostro caso il server ha IP statico, che viene assegnato in base al MAC Address per cui questa sezione può essere saltata. In ogni caso si può scoprire l'ip del server fisico digitando dal terminale

ipconfig

Se invece c'è la necessità di configurare i parametri di rete digitare

sudo nano /etc/network/interfaces

e modificare il file con i parametri adeguati, ad esempio:

```
# This file describes the network interfaces available
#on your system and how to activate them.
#For more information, see interfaces(5).
# The loopback network interface auto lo
iface lo inet loopback
# The primary network interface auto eth0
iface eth0 inet static
address 192.168.0.100
netmask 255.255.255.0
network 192.168.0.0
broadcast 192.168.0.255
gateway 192.168.0.1
```

ora facciamo in modo che la macchina riconosca l'ip con

sudo nano /etc/hosts

e configuriamo i DNS con

modificando il file in questo modo

```
# OpenDNS
nameserver 208.67.220.220
nameserver 208.67.222.222
```

Elementi utili da installare nel server

1. Installare il browser web con sudo apt-get install links2 e testare se la configurazione della rete funziona con ping www.google.it;
2. seguire i seguenti comandi shell sudo apt-get install binutils cpp gcc make automake autoconf debhelper g++ build-essential.

Configurazione del demone SSH

Il server SSH lo abbiamo già installato con l'installazione di Ubuntu Server quindi ora lo configuriamo. Facciamo un backup del file di configurazione:

sudo cp /etc/ssh/sshd_config /etc/ssh/sshd_config.backup

Copiamo anche le chiavi di sicurezza generate

*sudo cp /etc/ssh/ssh_host*_key* /directory/backup*

configuriamo SSH nel file ssh_config

sudo nano /etc/ssh/sshd_config

Nel file che si apre cambiamo la porta in 22222.
Salviamo il file con Maius + ctrl + O.
Chiudiamo il programma con Maiusc + ctrl + X. Avviamo SSH

sudo /etc/init.d/ssh start

oppure lo riavviamo con:

sudo /etc/init.d/ssh restart

Per personalizzare ancora SSH leggere la documentazione di Ubuntu Server con il comando

man sshd_config

Installazione di vmware

Con un'altro pc d'appoggio scaricare il file (con Firefox altrimenti gli altri browser non lo scaricano) con l'estensione tar.gz.

Quindi aprire il firefox e andare nel sezione Prodotti di `www.vmware.com/it`. Scaricare vmware VMware-server-2.0.2-203138.x86_64.tar.gz.

Scaricarlo dopo essersi registrato, dato che è necessario ottenere la licenza (serial number).

Scaricare lo script all'indirizzo `https://nodeload.github.com/raducotescu/vmware-server-linux-2.6.3x-kernel/tarball/master` il cui nome è raducotescu-vmware-server-linux-2-3.6.3x-kernel-release-1.5-1-g71f8b66.tar.

Scaricato il tutto copiarlo in una usb e collegarla al server in cui abbiamo installato Ubuntu Server.

Ora ritorniamo nel pc in cui abbiamo appena installato Ubuntu Server dalla shell digitare

dmesg

le ultime righe indicano il nome dedicato al dispositivo connesso. Creare un directory temporanea con

sudo mkdir /var/tmp/usbpen

e montarla con

sudo mount /dev/sdb1 ("o il nome che abbiamo trovato facendo dmesg") /var/tmp/usbpen

a questo potete leggere il contenuto in /var/tmp/usbpen Una volta entrati nella chiavetta copiare il file di vmware e lo script in home/nomeutente/ dezippare lo script con

tar -xf raducotescu-vmware-server-linux-2-3.6.3x-kernel-release-1.5-1-g71f8b66.tar

e copiare il file VMware-server-2.0.2-203138.x86_64.tar.gz. all'interno della cartella generata dezippando lo script. A questo punto entrare nella cartella generata dezippando lo script ed eseguire

sudo /vmware-server-2.0.x-kernel-2.6.3x-install.sh

Seguire l'installazione rispondendo alle domande.

Migrazione della macchina virtuale

Una volta installato vmware potete copiare il file Thesiug.vmdk e Thesisug.vmx in /var/lib/vmware/Virtual Machines/Thesisug.

A questo punto da un'altro pc munito di browser che supporta javascritp, eseguire il comando ssh creando un tunnel ssh con

ssh -p 22222 -l "nome_utente" -L 8222:127.0.0.1:8222 "indirizzo IP Ubuntu server"

Aprire il browser e accedere alla pagina

http://localhost:8222/

immettere username : root e password:quella dell'utente in Ubuntu Server. Importare la macchina virtuale con

Add Virtual Machine to Inventory

e selezionare il file Thesisug.vmx. Quando facciamo partire la VM, alla domanda la virtual machine è stata copiata o mossa rispondete è STATA MOSSA, altrimenti vmware sconvolge le impostazioni di base della virtual machine e non è più raggiungibile dall'esterno.

Connessione alla Macchina virtuale e Fisica

Per connettersi al server fisico via SSH digitare

ssh -p 22222 -l "nome utente" -L 8222:127.0.0.1:8222 "indirizzo IP Ubuntu server"

in questo modo è disponibile sia l'interfaccia web di vmware all'indirizzo

http://localhost:8222/

che la shell a linea di comando. Per connettersi alla Virtual machine Thesisug creare prima un tunnel SSH con:

ssh -p 22222 -l "nome utente macchina virtuale" -L 10001:127.0.0.1:5900 "indirizzo IP macchina virtuale"

e poi con VNC → KRDC (o altro) aprire l'indirizzo vnc//127.0.0.1:10001/ e immettere la psw di vnc. Se l'utente dalla sua postazione remota non riesce ad accedere alla macchina digitare

cd /home/nomeutente/.ssh/

e rinominare il file

mv known_hosts known_hosts.bak

ed eliminare known_hosts con

sudo rm known_hosts

Riprovare a connettersi via SSH.

APPENDICE B

INSTALLAZIONE AMBIENTE DI SVILUPPO LATO SERVER

Installazione JDK da java.sun.com

Rimuovere la versione OpenJDK[107] 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/lucidpartner` *'*

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)
```

aggiungere le variabili d'ambiente globali in coda al file /etc/profile aprendo il terminale e digitando

sudo gedit /etc/profile

una volta che il file si è aperto aggiungere in coda le seguenti righe

```
JAVA_HOME=/usr/lib/jvm/java-6-sun export JAVA_HOME
PATH=$PATH:$JAVA_HOME/bin export PATH
```

Salvare prima di chiudere il file.

Riavviare il sistema per rendere effettive le modifiche.

Installazione di MAVEN

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

rinominarlo con sudo rm /cartelladownload/apache-maven-3.0-bin apache-maven e copiarlo con

sudo cp -R apache-maven /usr/local/apache-maven

ps. Attenzione al nome della cartella estratta. In questo caso è stato scaricato il file apache-maven-3.0-bin.tar.gz.
Aprire il file /etc/profile con:

sudo gedit /etc/profile

Aggiungere le seguenti variabili d'ambiente globali al file /etc/profile, accodando le seguenti 3 righe al file aperto al passo precedente:

```
export M2_HOME=/usr/local/apache-maven export
M2=$M2_HOME/bin export PATH=$M2:$PATH
```

Salvare prima di chiudere il file. Dare i permessi di lettura/scrittura a tutti gli utenti della cartella apache-maven

sudo chmod 777 /usr/local/apache-maven

Riavviare il sistema. Controllare la corretta installazione con:

mvn –version

Installazione di Eclipse HELIOS come IDE per lo sviluppo e plugin

Scaricare il pacchetto di installazione di eclipse da `http://www.eclipse.org/downloads/`, scompattarlo e copiarne il contenuto in una directory a piacere - es. /home/nomeutente/eclipse/ - creando una cartella - es. /home/nomeutente/workspace/ - per il workspace. Installare i seguenti plugin:

- 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/apache-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/apache-maven/conf/settings.xml).

- 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;
- 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;
- 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.

Importare il codice della parte server del progetto Avviare a questo punto Eclipse e: File -> New -> Other -> SVN -> CheckOut Project from SVN: Click Create a new repository Location, in Location URL inserire il seguente: `https://thesis-ug.googlecode.com/svn/branches` Click su server e poi selezionare mysqlSupport. Click su Finish.

Librerie mancanti Alcune librerie non vengono importate automaticamente con Maven

Scaricando il progetto del server il file si torva nella cartella resources. Bisogna togliere # all’inizio delle righe commentate, ultimamente lo si è utilizzato solo per l’ultima riga (quella relativa a mysql). Quindi, spostarsi all’interno della cartella del server (nel nostro caso ephemere) e copiare i seguenti .jar:

1. owlapi-3.0.0.jar (application/java-archive) 1876K;
2. mysql-connector-java-5.1.13-bin.jar (application/java-archive) 750K;
3. HermiT-1.2.3.jar (application/java-archive) 1535;
4. gdata-core-1.41.1.jar (application/java-archive) 1015K;
5. gdata-client-1.41.1.jar (application/java-archive) 125K;
6. gdata-calendar-2.0.jar (application/java-archive) 49K;
7. db4o-7.12.132.jar (application/java-archive) 2388K.

Copiare anche installdependencies.sh in ephemere(cartella del progetto lato server contenuta in workspace dopo la sincronizzazione con la repository di google).

Aprire il terminale e spostarsi nella cartella ephemere. Dare al file installdependencies.sh i permessi per l'esecuzione digitando da terminale

sudo chmod +x installdependencies.sh

Eseguire il file bash installdependencies.sh digitando

./installdependencies.sh

Successivamente è possibile cancellare tutti i file e cartelle indicate sopra.

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.

Installazione del servlet container Apache Tomcat

Download di Tomcat da http://tomcat.apache.org/download-60.cgi (se possibile la versione 6), scompattare il file con

tar xvzf /cartelladownload/apache-tomcat-6.0.29.tar.gz (Attenzione alla versione di tomcat nel nome del file)

Rinominare il file in apache-tomcat digitando da terminale

sudo /cartelladownload/apache-tomcat-6.0.29 apache-tomcat

e copiarne il contenuto nella cartella

sudo mv apache-tomcat /usr/local/

Per raggiungere più facilmente tomcat da terminale, creare un link simbolico alla cartella di tomcat con il comando sudo ln -s /usr/local/apache-tomcat tomcat in questo modo il percorso sarà /usr/local/tomcat.

Aumentare la memoria riservata dalla JVM a tomcat

aprire il file catalina.sh che si trova nella cartella dove è installato apache-tomcat nella cartella bin. - spostarsi in questa posizione:

```
- if [ -z "$LOGGING_MANAGER" ]; then JAVA_OPTS="$JAVA_OPTS
-server -Xms1024m -Xmx1024m -Djava.util.logging.manager=
org.apache.juli.ClassLoaderLogManager"
else JAVA_OPTS="$JAVA_OPTS $LOGGING_MANAGER" Fi
```

e modificare lo statment della prima parte dell’if a come appena scritto!

Ovviamente perché le modifiche abbiano effetto, riavviare tomcat!

```
Xmx (massima memoria utilizzabile)
Xms (minima memoria utilizzabile)
```

Configurare eclipse per avviare direttamente il server Tomcat

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.

Avvio del server da eclipse

Basta semplicemente fare clic col pulsante destro del mouse sul nome del progetto e selezionare Run->Run on Server. (Per ora non lo avviamo...aspettiamo la fine di tutta l’installazione!!!).

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/lib.

Verranno creati dei file di log giorno per giorno, per evitare che un unico file di log diventi troppo grosso e difficile da trattare. Il file di log del giorno corrente è sempre tomcat.log, al cambio di data del server verrà creato un file tomcat.AAAA-MM-DD.log ed il file tomcat.log verrà svuotato.

Installazione mysql server

Da terminale digitare

sudo apt-get install mysql-server

Creazione utente mysql Scegliete un nome utente ed una password ed usate il comando

create user ’NOME_UTENTE’@’localhost’ identified by ’PASSWORD’

il nome utente di default per il nostro sistema è tesi.

Utilizzarlo se non si vogliono andare a modificare i file di configurazione, altrimenti, qualora si scelga di cambiare questo parametro, modificare il file src/main/-resources/system.conf nella sezione MysqlDatabaseManagement.

Creazione del database

Se si vogliono usare i dati di default per il nome del database, sono quelli riportati qui sotto, altrimenti sarà poi necessario andare a modificarei il file src/-main/resources/system.conf nella sezione MysqlDatabaseManagement, con i dati di *default:*

Database_name= Database_user=* Database_password=**

Assegnazione di tutti i privilegi all'utente creato grant all privileges on *.* to NOME_UTENTE@'localhost';

In realtà non è necessario utilizzare *.* piuttosto usare NOME_DATABASE.*

Esecuzione script per la creazione automatica del database Entrate in mysql da bash usando la seguente riga:

mysql -u tesi –p

alla richiesta digitate la password "tesi" senza virgolette.

Scaricate il file con lo script di creazione del database presente nel branch del codice del server, il nome del file è databaseCreation.sql e si trova nel package dao.management.mysql Eseguite la seguente riga di comando da mysql:

source percorso_file/databaseCreation.sql

Se ad esempio avete scaricato il file nella vostra home dovrete eseguire

source /home/username/databaseCreation.sql

Verifica esecuzione script database Dopo essere entrati in MySQL eseguite:

show databases;

Se compare il database thesisug allora tutto è andato a buon fine, altrimenti provate a seguire nuovamente i passi precedenti e, quando eseguite il comando source, verificate l'output generato in quanto autoesplicativo. Ora si può eseguire il server. Entrare in eclipse e fare clic col pulsante destro del mouse sul nome del progetto e selezionare Run->Run on Server. Se il tutto funziona non dovrebbero esserci errori!

INSTALLAZIONE AMBIENTE DI SVILUPPO LATO CLIENT

Installazione dell’SDK Android.

Download del pacchetto dall’url `http://developer.android.com/sdk/index.html`.

Scompattare l’archivio con

tar xvzf /cartelladownload/android-sdk-tools

Successivamente andare nella cartella dove è stato appena scompattato e rinominare la cartella in android-sdk e copiarla nella directory /usr/local/ eseguendo da terminale:

sudo mv android-sdk /usr/local/

Aggiungere la seguente variabile di ambiente globali al file /etc/profile

export PATH=$PATH:/usr/local/android-sdk/tools

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.

Installare i target necessari allo sviluppo in Android 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). Da available Package selezionare Android repository selezionare SDK Platform Android 2.1, API 7, revision 2 selezionare Third party Add-ons selezionare tutti i Google APIs by Google Inc., Android API xx, revision x (per velocizzare si possono installare tutti i componenti dall’Android Repository e tutti i componenti Google APIs, tale operazione richiede però più tempo per scaricare tutti i file necessari). Clic su Install selected Accettare le licenze Install. Creare un virtual device per l’esecuzione del client Creare un Virtual Device. Da Eclipse Windows -> Android SDK and AVD Manager -> Virtual Device New inserire quindi i seguenti dati: Name: Nome device (a scelta) Target: scegliere dall’elenco Google APIs (Google Inc.) - API Level 7 (oppure un qualche livello superiore al 7) Size: 512 MiB Clic su Create AVD.

Sincronizzazione del progetto client su repository SVN

Avviare a questo punto Eclipse e: File -> New -> Other -> SVN -> CheckOut Project from SVN: Click Create a new repository Location, in Location URL inserire il seguente: `https://thesis-ug.googlecode.com/svn/trunk` Click su client. Click

su Finish.

Ottenere la Google Map Key

Avviare Eclipse e copiare il path relativo al debug.keystore andando in Window -> Preferences -> Android -> Build

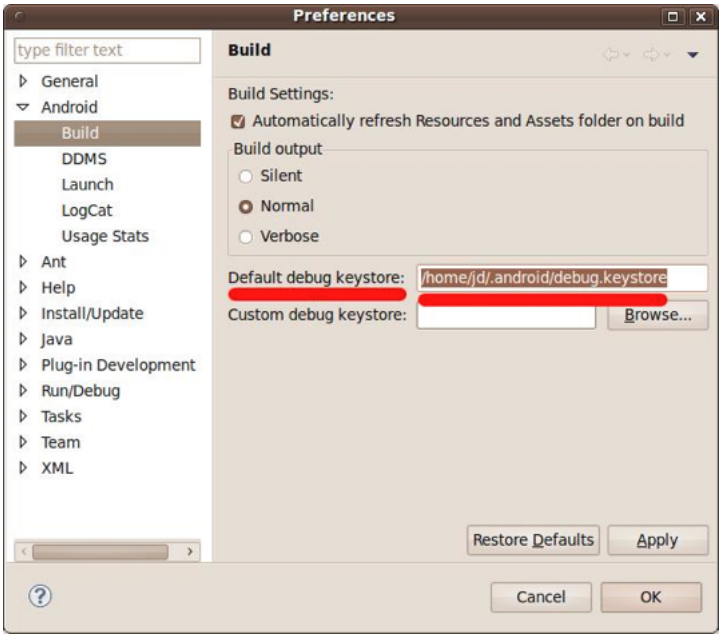


Figura 6.1: Configurazione Google Api Key.

Utilizzare la stringa copiata al posto di <path> nel seguente comando:

keytool -list -alias androiddebugkey -keystore <path> -storepass android -keypass android

Copiare il codice MD5 generato.

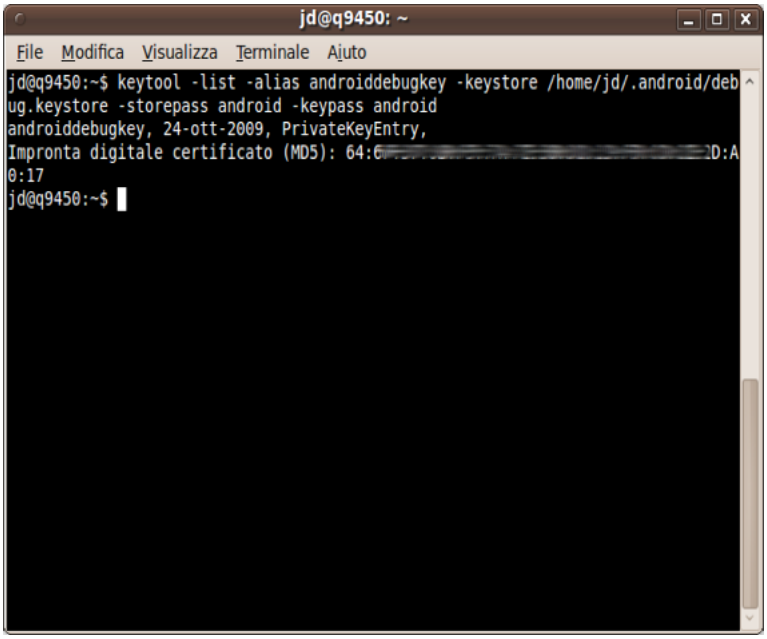


Figura 6.2: Configurazione Google Api Key.

Utilizzare l’MD5 sul sito Sign Up for the Android Maps API - Android Maps API - Google Code per ricevere la map-key. `http://code.google.com/intl/it-IT/android/maps-api-signup.html`. Successivamente andare a modificare il file /res/-layout/map.xml inserendo la Google Map Key ottenuta.

In particolare inserire la chiave ottenuta modificando questa riga android: api-Key="##" />

APPENDICE C

Di seguito sono riportati i diagrammi UML di sequenza per:

- Creazione nuovo Task;

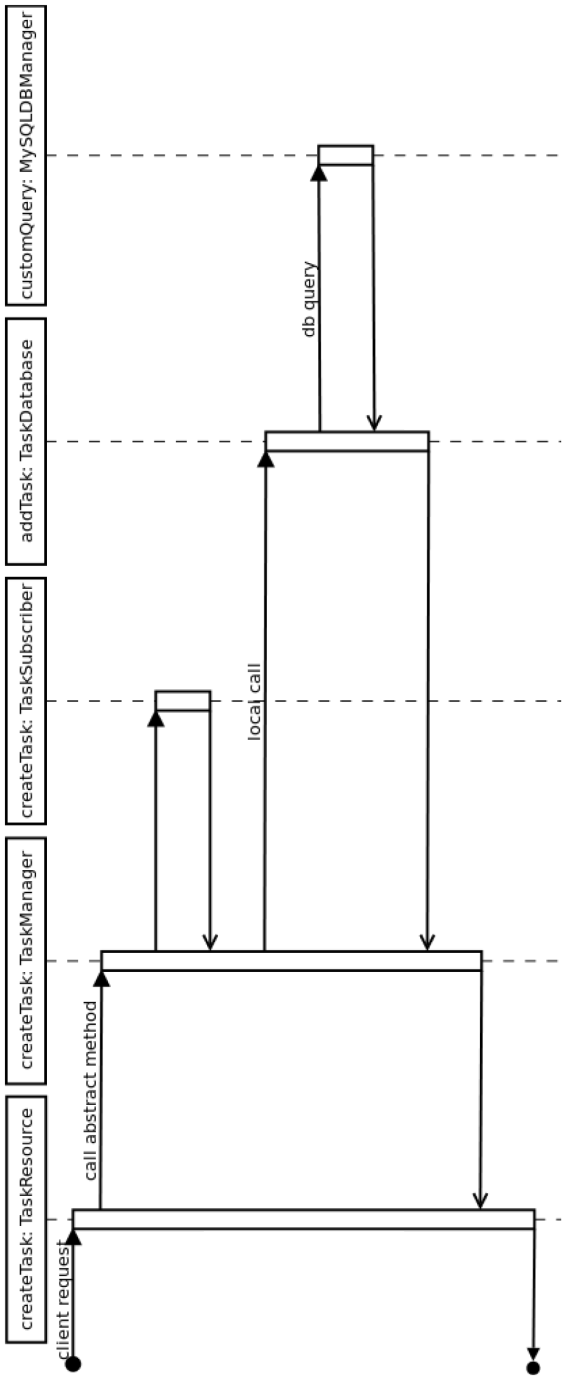


Figura 6.3: Diagramma UML di createTask.

- Registrazione nuovo utente.

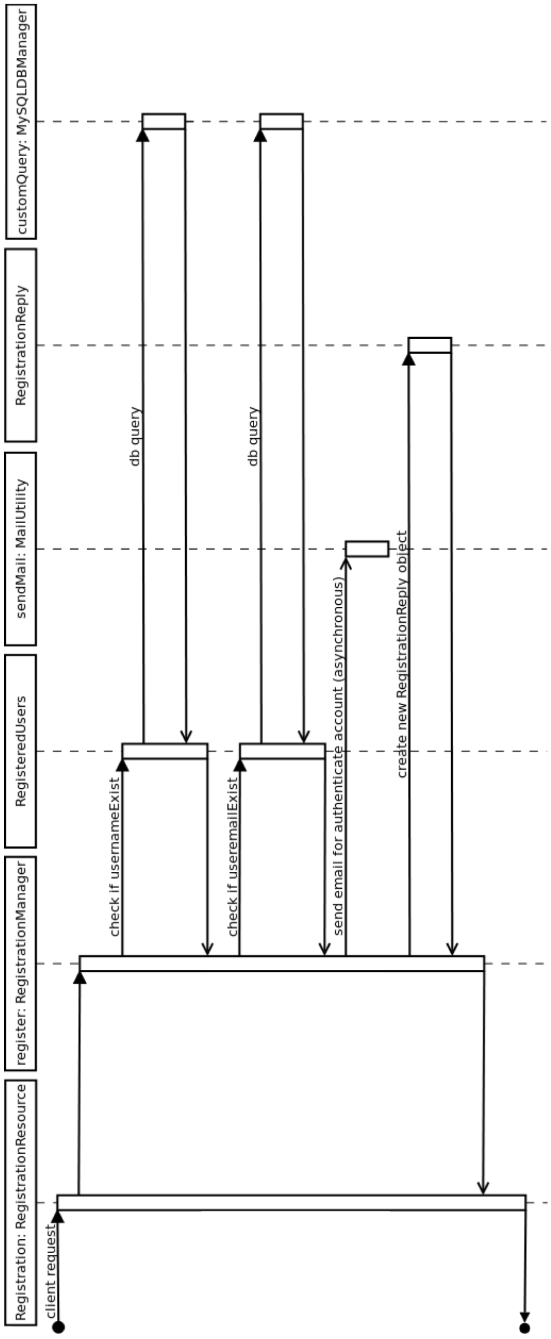


Figura 6.4: Diagramma UML di registration().

APPENDICE D

Il codice creato lato client e lato server sono disponibili nella directory “codice” di questo cd-rom.

Bibliografia

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

[2] Mauro Migliardi, Giorgio Ravera, A Support System for Memory Impaired Subjects, Proc. of the 2009 International Conference on Information and Knowledge Engineering, Las Vegas, Nevada, USA, July 13-16, 2009.

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

[4] Petrus Prasetyo Anggono and Mauro Migliardi, A Mobile, Context Aware System For Memory Support And Planning, Proc. of the 2010 International Conference on Information and Knowledge Engineering, Las Vegas, Nevada, USA, July 12-15, 2010.

[5] Mauro Migliardi, Marco Gaudina, A mobile platform for the improvement of personal efficiency, Proc. of the IEEE Conference on Intelligent Systems, Opatija (HR), 23-27 May 2011.

[6] Mauro Migliardi, Marco Gaudina, Active Personal Information Manager: a System for Human Memory Support, Proc. of the 4th International Conference on Intelligent Interfaces for Human-Computer Interaction, Korean Bible University, Seoul (S. Korea), June 30th - July 2nd, 2011.

[7] G. Geloso, M. Migliardi - "Active PIM: dalla lista della spesa ai suggerimenti geolocalizzati" – Università degli studi di Genova, 2011

[8] A. Toso, M. Migliardi - "Sistema Informativo Mobile Geolocalizzato per Supporto Mnemonico Basato su Architettura Android e RESTful Web Service" – Università degli studi di Padova, 2011.

[9] Anuska Benacchio, M. Migliardi, Marco Gaudina - "Sistema di supporto mnemonico: architettura multi-layer per la gestione di data-source multipli" – Università degli studi di Padova, 2011.

[10] VMware vSpere Hypervisor, `http://www.vmware.com/products/vsphere-hypervisor/overview.html`

[11] Ubuntu Server 10.04 LTS, `http://releases.ubuntu.com/lucid/`

[12] VMware 2.0.2, http://www.vmware.com/support/server2/doc/releasenotes_vmserver202.html

[13] Virtual Box, https://www.virtualbox.org/

[14] Citrix XenServer, http://www.citrix.com/English/ps2/products/product.asp?contentID=683148

[15] Microsoft Hyper-V Server, http://www.microsoft.com/en-us/server-cloud/hyper-v-server/default.aspx

[16] OpenSSH, http://www.openssh.com/

[17] VMware Server 2.0.2 Release Notes. Online: http://www.vmware.com/support/server2/doc/releasenotes_vmserver202.html

[18] HTTP, http://www.w3.org/Protocols/

[19] Google Maps API, http://code.google.com/apis/maps/index.html

[20] Eclipse, http://www.eclipse.org/

[21] XML, http://www.w3.org/XML/

[22] Java, http://www.java.com/it/

[23] Apache Subversion (SVN), http://subversion.apache.org/

[24] Eclipse plugin per Maven, http://maven.apache.org/eclipse-plugin.html

[25] Apache Maven Project, http://maven.apache.org/

[26] Eclipse Web Tools, http://www.eclipse.org/webtools/

[27] ADT plugin per Eclipse, http://developer.android.com/sdk/eclipse-adt.html

[28] Android SDK, http://developer.android.com/index.html

[29] DDMS, Dalvik Debug Monitor Server, http://developer.android.com/guide/developing/debugging/index.html

[30] Java virtual machine Wikipedia. http://it.wikipedia.org/wiki/Macchina_virtuale_Java

[31] Apache Tomcat, http://tomcat.apache.org/

[32] JavaServer Pages, http://www.oracle.com/technetwork/java/javaee/jsp/index.html

[33] ASP, http://www.asp.net/

[34] IIS, http://www.iis.net/

[35] Apache Foundation, http://www.apache.org/

[36] Apache Web Server, http://www.apache.org/

[37] Log4j, http://logging.apache.org/log4j/1.2/

[38] Jacarta Project, http://jakarta.apache.org/

[39] Web service wikipedia, http://it.wikipedia.org/wiki/Web_service

[40] Web Services Architecture, W3C Working Group Note 11 February 2004 online: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[41] WSDL, http://www.w3.org/TR/wsdl

[42] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), W3C Recommendation 27 April 2007 - Online: http://www.w3.org/TR/soap12-part1/

[43] "RESTful Web Services", Sameer Tyagi, Agosto 2006 - http://www.oracle.com/technetwork/articles/javase/index-137171.html

[44] W3C, http://www.w3.org/

[45] "Architectural Styles and the Design of Network-based Software Architectures",capitolo 5, Roy Thomas Fielding, 2000. Online: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[46] JAX-RS, http://jcp.org/en/jsr/detail?id=311

[47] Framework RESTEasy, http://www.jboss.org/resteasy

[48] "RESTEasy Reference Guide for Use with JBoss Enterprise Web Platform", Edition 5.1.1, Chapter 17 JAXB Provider. Online: http://docs.redhat.com/docs/en-US/JBoss_Enterprise_Web_Platform/5/html-single/RESTEasy_Reference_Guide/index.html#Built_in_JAXB_providers

[49] JBOSS, http://www.jboss.org/

[50] JAXB, http://www.oracle.com/technetwork/articles/javase/index-140168.html

[51] Java EE, http://www.oracle.com/technetwork/java/javaee/overview/index.html

[52] SAX, http://www.saxproject.org/

[53] Protégé, http://protege.stanford.edu/

[54] OWL, http://www.w3.org/TR/owl-ref/

[55] OWL API, http://owlapi.sourceforge.net/

[56] "OWL Web Ontology Language Overview", W3C Recommendation, 10 February 2004. Online: http://www.w3.org/TR/owl-features/

[57] "OWL 2 Web Ontology Language Document Overview", W3C Recommendation, 27 October 2009. Online: http://www.w3.org/TR/owl2-overview/

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

[59] MySQL, http://www.mysql.com/

[60] Piattaforma LAMP, http://it.wikipedia.org/wiki/LAMP_(piattaforma)

[61] Installazione LAMP in Ubuntu, https://help.ubuntu.com/community/ApacheMySQLPHP

[62] phpMyAdmin, http://www.phpmyadmin.net/home_page/index.php

[63] Android. Online: http://www.android.com/

[64] Symbian, http://it.wikipedia.org/wiki/Symbian_OS

[65] iOS di Apple, http://www.apple.com/it/ios/

[66] Apple iPhone, http://www.apple.com/it/iphone/

[67] Google Groups, http://groups.google.com/

[68] Skype, http://www.skype.com

[69] Google corp., http://maps.google.it/maps

[70] Google Maps APIs documentation – http://code.google.com/apis/maps/index.html

[71] Google Local Search API, http://code.google.com/intl/it-IT/apis/maps/documentation/localsearch/jsondevguide.html

[72] Yahoo! Local Search Web Service. http://developer.yahoo.com/search/local/V3/localSearch.html

[73] Pagine gialle. http://www.paginegialle.it/

[74] Paginegialle.it Maps API. http://api.visual.paginegialle.it/tcolnew/mapsapi/pgMapsV1.html#theTop

[75] 2Spaghi.it. http://www.2spaghi.it/

[76] Mozila Labs, http://mozillalabs.com/

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

[78] Mozilla Firefox, http://www.mozilla.org/it/firefox/new/

[79] CVS – Current Version System, http://cvs.nongnu.org/

[80] Mercurial, http://mercurial.selenic.com/

[81] OSI - Open Source Initiative, http://www.opensource.org/

[82] Google Code, http://code.google.com/

[83] Architettura three-tier wikipedia, http://it.wikipedia.org/wiki/Architettura_three-tier

[84] JSON, http://www.w3.org/MarkUp/Forms/wiki/Json

[85] Navigli, R.; "Automatically extending, pruning and trimming general purpose ontologies," Systems, Man and Cybernetics, 2002 IEEE International Conference on, vol.1, no., pp. 631- 635, 6-9 Oct. 2002 doi: 10.1109/ICSMC.2002.1168049 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1168049&isnumber=26342

[86] Yoo Jung An; Geller, J.; Yi-Ta Wu; Soon Ae Chun; "Automatic Generation of Ontology from the Deep Web," Database and Expert Systems Applications, 2007. DEXA '07. 18th International Workshop on , vol., no., pp.470-474, 3-7 Sept. 2007 doi: 10.1109/DEXA.2007.43 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4312938&isnumber=4312839

[87] Wordnet, http://wordnet.princeton.edu

[88] Sign Up for the Google Maps API. http://code.google.com/intl/it-IT/apis/maps/signup.html

[89] MIT Research Project: ConceptNet 5. http://conceptnet5.media.mit.edu/

[90] Elo System, http://en.wikipedia.org/wiki/Elo_rating_system

[91] U.S. Chess Federation, http://www.uschess.org/

[92] Foursquare sito web. https://it.foursquare.com/

[93] FourSquare wikipedia. http://it.wikipedia.org/wiki/Foursquare

[94] Foursquare developers – Venues Project – API Documentation https://developer.foursquare.com/venues/

[95] Geocoding Google Api, http://code.google.com/apis/maps/documentation/geocoding/

[96] AndNav2, http://www.andnav.org/

[97] Vodafone IDEOS, http://lab.vodafone.it/wiki/Vodafone_IDEOS

[98] Apache Cassandra.http://cassandra.apache.org/

[99] Cassandra, http://database.html.it/articoli/leggi/3321/introduzione-a-apache-cassandra/

[100] Facebook, http://www.facebook.com

[101] Youtube, http://www.youtube.com

[102] Flickr, http://www.flickr.com/

[103] Google Places Api, http://code.google.com/apis/maps/documentation/places/

[104] FAT, http://msdn.microsoft.com/en-us/windows/hardware/gg463080

[105] Virtual Host, http://www.ubuntu.com/business/server/virtualisation

[106] Samba, http://www.samba.org/

[107] Java SE, http://www.oracle.com/technetwork/java/javase/downloads/index.html