ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

SCUOLA DI INGEGNERIA E ARCHITETTURA
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA E SCIENZE INFORMATICHE

Notifiche push in applicativi Android: uso di GCM ed una soluzione alternativa basata su Long-Polling HTTP

Relazione finale in:
Tecnologie Web/Internet

Relatore:

Prof. Mario Bravetti

Correlatore:

Dott. Cesare Gregori

Presentata da:

Mattia Torelli

Sessione II

Anno Accademico 2013/2014

INDICE DEI CONTENUTI

CAPITOLO 1 - INTRODUZIONE **1**

CAPITOLO 2 - CONCETTI DI BASE **5**

2.1 SERVIZI WEB 5
2.1.1 REST *6*
2.1.2 RESTFUL WEB SERVICE *9*
2.1.3 AMBIENTE DI SVILUPPO *12*
2.2 ANDROID 19
2.2.1 ARCHITETTURA ANDROID *20*
2.2.1.1 LINUX KERNEL 21
2.2.1.2 LIBRARIES 22
2.2.1.3 APPLICATION FRAMEWORK 23
2.2.1.4 APPLICATIONS 24
2.2.2 APP ANDROID *24*
2.2.3 AMBIENTE DI SVILUPPO *26*
2.3 MAPPE 30
2.3.1 GOOGLE MAPS *30*
2.3.2 OPENSTREETMAP *31*
2.4 LIBRERIE 33
2.4.1 OSMDROID *33*
2.4.2 OSMBONUSPACK *36*
2.4.3 JAVAAPIFORKML *38*
2.5 NOTIFICHE 40
2.5.1 TECNOLOGIE *41*
2.5.1.1 SOCKET 42
2.5.1.2 BOSH 43
2.5.1.3 WEBSOCKET 45
2.5.1.4 SSE 47
2.5.2 PUBLISH-SUBSCRIBE *50*
2.5.3 XMPP *52*
2.5.4 MQTT *54*
2.5.5 GCM *56*
2.5.5.1 CARATTERISTICHE 57
2.5.5.2 ARCHITETTURA 57
2.5.5.3 FLUSSO CICLO DI VITA 58
2.5.5.4 INVIO NOTIFICA 60
2.5.5.5 RICEZIONE NOTIFICA 60
2.5.5.6 COMUNICAZIONE GCM 61
2.5.6 FILE KML *65*

CAPITOLO 3 - PROGETTAZIONE **67**

3.1 ANALISI DEL PROBLEMA 67
3.2 ARCHITETTURA 69
3.2.1 APPLICAZIONE ANDROID CON GCM *69*
3.2.1.1 FUNZIONALITÀ DEL WEB SERVICE 72
3.2.1.2 FUNZIONALITÀ DEL CLIENT 73
3.2.1.3 FUNZIONALITÀ DEL GCM 74
3.2.2 APPLICAZIONE ANDROID CON BOSH *75*
3.2.2.1 FUNZIONALITÀ DEL WEB SERVICE 77
3.2.2.2 FUNZIONALITÀ DEL CLIENT 78
3.2.3 INTERFACCIA AMMINISTRATIVA *79*

CAPITOLO 4 - IMPLEMENTAZIONE **81**

4.1 IMPLEMENTAZIONE WEB SERVICE PER GCM 81
4.1.1 ITINERARIO *81*

4.1.2 MULTIMEDIA *83*
4.1.3 NOTIFICA *84*
4.1.4 REGISTRAZIONE *86*
4.2 IMPLEMENTAZIONE CLIENT PER GCM 88
4.2.1 CLASSE MAINACTIVITY *89*
4.2.2 CLASSE GCMBROADCASTRECEIVER *98*
4.2.3 CLASSE GCMSERVICEMESSAGEHANDLER *99*
4.2.4 CLASSE CUSTOMINFOWINDOW *100*
4.2.5 CLASSE VIBRATESERVICE *101*
4.2.6 CLASSE ALERTSERVICE *102*
4.3 IMPLEMENTAZIONE WEB SERVICE PER BOSH 104
4.3.1 LOGIN OPERATION *104*
4.3.2 PUSH OPERATION *106*
4.3.3 POP OPERATION *108*
4.4 IMPLEMENTAZIONE CLIENT PER BOSH 110
4.4.1 CLASSE MAINACTIVITY *111*
4.4.2 CLASSE SERVICEBROADCASTRECEIVER *112*
4.4.3 CLASSE SERVICEMESSAGEHANDLER *113*
4.5 IMPLEMENTAZIONE INTERFACCIA AMMINISTRATIVA 117
4.5.1 INDEX *117*
4.5.2 ADMIN *118*
4.5.3 BROKER *119*
4.5.4 UPLOAD *120*

CAPITOLO 5 - GUIDA ALL'USO 122

5.1 FUNZIONAMENTO DELL'INTERFACCIA UTENTE 122
5.1.1 Funzionamento app *122*
5.1.2 Funzionamento amministratore *128*

CAPITOLO 6 - CONCLUSIONI 131

CONCLUSIONI 131

BIBLIOGRAFIA 133

INDICE DELLE FIGURE

FIGURA 1 - METODO GET 11
FIGURA 2 - METODO POST 11
FIGURA 3 - METODO PUT 12
FIGURA 4 - METODO DELETE 12
FIGURA 5 - SIMPLE ROOT RESOURCE 14
FIGURA 6 – CONTAINER RESOURCE 15
FIGURA 7 – ITEM RESOURCE 16
FIGURA 8 – CLIENT-CONTROLLED CONTAINER RESOURCE 17
FIGURA 9 - CLIENT-CONTROLLED ITEM RESOURCE 18
FIGURA 10 - LAYER ANDROID 21
FIGURA 11 - DIRECTORY SDK MANAGER 27
FIGURA 12 - ANDROID SDK MANAGER 27
FIGURA 13 - JAVA JDK 28
FIGURA 14 - VARIABILI AMBIENTE 29
FIGURA 15 - JAVA BUILD PATH 34
FIGURA 16 - OPENSTREETMAP VIEWER 36
FIGURA 17 - OSMBONUSPACK DEMO 1 37
FIGURA 18 - OSMBONUSPACK DEMO 2 38
FIGURA 19 - MARSHALLING 38
FIGURA 20 - FILE KML 39
FIGURA 21 - UNMARSHALLING 39
FIGURA 22 - FLUSSO SOCKET 43
FIGURA 23 - FLUSSO BOSH 44
FIGURA 24 - FLUSSO WEBSOCKET 46
FIGURA 25 - RICHIESTA WEBSOCKET 46
FIGURA 26 - RISPOSTA WEBSOCKET 47
FIGURA 27 - RICHIESTA RISPOSTA SSE 50
FIGURA 28 - FLUSSO SSE 50
FIGURA 29 - ARCHITETTURA PUBLISH - SUBSCRIBE 51
FIGURA 30 - MESSAGGIO XMPP 53
FIGURA 31 - TIPOLOGIA MESSAGGI 54
FIGURA 32 – FIXED HEADER MQTT 55
FIGURA 33 - ARCHITETTURA MQTT 56
FIGURA 34 - ARCHITETTURA GCM 59
FIGURA 35 - STRUTTURA KML 66
FIGURA 36 - APP INIZIALE 91
FIGURA 37 - INFOWINDOW 93
FIGURA 38 - NOTIFICA SU STATUS BAR 96

FIGURA 39 - RICHIESTA DI AGGIORNAMENTO 96
FIGURA 40 - AVVISO DI PROSSIMITÀ 98
FIGURA 41 - NOTIFICA DI AGGIORNAMENTO 116
FIGURA 42 - LOGIN 117
FIGURA 43 - NOTIFICA GCM 119
FIGURA 44 - NOTIFICA BOSH 120
FIGURA 45 - UPLOAD 121
FIGURA 46 - APP PRINCIPALE 123
FIGURA 47 - APP DETTAGLIO 123
FIGURA 48 - APP MULTIMEDIA 124
FIGURA 49 - APP AVVISO 125
FIGURA 50 - APP SELEZIONE 126
FIGURA 51 - APP ITINERARIO 127
FIGURA 52 - APP PROSSIMITÀ 128
FIGURA 53 - AMMINISTRAZIONE INIZIALE 129
FIGURA 54 - AMMINISTRAZIONE NOTIFICA 130
FIGURA 55 - AMMINISTRAZIONE UPLOAD 130

RINGRAZIAMENTI

Ringrazio tutti coloro che hanno reso possibile questo giorno e soprattutto un ringraziamento speciale alla mia famiglia a cui dedico questa Tesi.

L’interesse per le applicazioni internet ed in particolare per le tecnologie Web, unito alla possibilità di fornire una soluzione pratica ad un problema reale come la comunicazione di informazioni per dispositivi mobili, sono stati i motivi principali che hanno portato alla realizzazione di questo lavoro.

Capitolo 1

INTRODUZIONE

In questa Tesi vengono presentate diverse tecnologie per la gestione di [1] notifiche push per dispositivi mobili: le notifiche push introdotte sono impiegate in ambiente Android e fanno uso di protocolli ed architetture diverse tra loro.
In particolare, sono state realizzate due versioni che forniscono lo stesso identico servizio, ma strutturato e gestito con architetture diverse.
Inoltre vengono impiegati servizi [2] Restful web service su protocollo [3] HTTP come sistemi di riferimento per l'accesso ai contenuti multimediali. Il tutto attraverso un esempio pratico realizzato mediante un'applicazione Android dotata di una mappa del territorio e di diversi itinerari interattivi: le mappe scelte sono open source ed esattamente le [4] OpenStreetMap (OSM), ampiamente utilizzate come valida alternativa alle ben più note mappe di Google.
Le tecnologie innovative che stanno alla base di questo progetto sono le notifiche push basate su [5] GCM (Google Cloud Messaging) e [6] BOSH (Bidirectional-streams Over Synchronous HTTP) quest'ultima più comunemente nota come Long-Polling.
Il GCM è un servizio di notifiche push messo a disposizione da Google che consente di inviare dati da un server alle applicazioni Android. Il servizio permette di archiviare i messaggi e di recapitarli alle applicazioni anche se quest'ultime non sono momentaneamente reperibili.
BOSH rappresenta una valida alternativa al sistema GCM; in quanto fornisce la possibilità di implementare le notifiche push attraverso una tecnica che prevede di rieffettuare immediatamente una richiesta HTTP al verificarsi di ogni notifica.
Per quanto riguarda i Restful web service, nascono per supportare l'integrazione di sistemi eterogenei: i Web Service sono componenti software che possono essere pubblicati, localizzati e utilizzati attraverso il Web. Questa tecnologia dispone di caratteristiche necessarie per offrire l'interoperabilità richiesta per la

progettazione e gestione di sistemi informatici, documenti e protocolli di rete diversi tra loro.

Il modello dei Restful web service è basato sulla definizione di [7] REST (Representational state transfer) che rappresenta un tipo di architettura software basata sulle risorse. Ogni risorsa fornisce un'interfaccia che espone le operazioni utilizzabili. Le richieste ai servizi vengono effettuate attraverso Internet utilizzando il più comune dei protocolli applicativi, l'HTTP.

Questa ricerca si propone di sviluppare e di confrontare diversi approcci alla gestiore delle notifiche remote per dispositivi mobili.

Inoltre, essendo BOSH un sistema basato su web services, permette di superare i limiti dovuti ai firewall che potrebbero bloccare i collegamenti in uscita (da una rete oppure da un computer al quale ci si collega) verso le risorse remote.

I sistemi di notifica sviluppati in questa relazione, mettono a confronto sistemi diversi per le notifiche push:

- Sistema basato su GCM;
- Sistema basato su BOSH.

Pertanto, le parti che giocano un ruolo fondamentale nel caso del GCM sono tre ed esattamente:

- l'applicazione Android, che attraverso le operazioni a disposizione può comunicare coi servizi;
- i servizi che erogano le funzionalità per il reperimento delle risorse;
- il GCM che gestisce l'invio delle notifiche.

Mentre nel caso del sistema basato su BOSH le parti sono due, quali:

- l'applicazione Android, che attraverso le operazioni a disposizione può comunicare coi servizi ;
- i servizi che erogano le funzionalità per il reperimento delle risorse e l'invio di notifiche tramite Long-Polling HTTP;

Il sistema progettato fa uso di un Client (l'applicazione Android) che può ricevere notifiche e richiedere contenuti multimediali per la consultazione di determinate informazioni. Dispone di una mappa cartografica e sistema GPS per la

geolocalizzazione del dispositivo; sulla mappa sono caricati degli itinerari con funzioni integrate che permettono il collegamento ad un insieme di servizi realizzati tramite Restful web service: per svolgere determinate operazioni di supporto come richieste di file multimediali o di file d'itinerario. Tali servizi provvedono ad eseguire operazioni fornendo i contenuti richiesti dal Client.
Le richieste possono essere di due tipologie:

- richiesta di file multimediale;
- richiesta di file d'itinerario.

Nel caso di richiesta di un file multimediale, il Client inoltra ad un servizio la richiesta della risorsa; il servizio dedicato ricevuta la richiesta preleva e restituisce tale risorsa al Client che, una volta ricevuta, l'avvia per consultarne il contenuto.
Nel caso di richiesta di un file di itinerario, il Client deve prima ricevere la notifica di aggiornamento per poi procedere alla richiesta di download del nuovo file di itinerario: in questo caso il Client, ricevuta la notifica, si collega al servizio richiedendogli il nuovo itinerario, ed una volta scaricato lo carica in mappa aggiornando il percorso coi nuovi punti.
Il sistema dispone anche di una applicazione amministrativa da cui è possibile, previa autenticazione, eseguire determinate operazioni tra cui: upload di file multimediali ed invio di notifiche.

Segue una panoramica sui capitoli descritti in questa Tesi.
Nel secondo capitolo vengono introdotti i concetti di base intesi a garantire una comprensione del progetto presentato; successivamente viene trattata la tecnologia che ne ha reso possibile l'implementazione, viene spiegata la teoria dei servizi web, la loro utilità e flessibilità. Inoltre vengono descritti i servizi erogati, i protocolli impiegati, l'uso di librerie [8] OSMDROID ed [9] OSMBONUSPACK ed il loro supporto allo sviluppo di applicazioni per mappe interattive, l'uso delle annotation e delle librerie per la gestione dei WS ed infine la piattaforma Android come soluzione all'implementazione di applicazioni per dispositivi mobili.
Nel terzo capitolo si affronta l'analisi del problema e la sua soluzione; viene introdotta l'architettura di funzionamento dei sistemi mostrando i ruoli e le funzionalità dei Client e dei WS.

Nel quarto capitolo viene descritta l'implementazione di Restful web service e dei client introdotti, dandone una visione tecnica.
Nel quinto capitolo si evidenzia l'uso pratico delle applicazioni sviluppate.
Infine, nell'ultimo capitolo, vengono considerati gli sviluppi futuri del progetto con una trattazione dei possibili miglioramenti ed evoluzioni dei sistemi trattati.

Capitolo 2

2.1 *SERVIZI WEB*

Prima dell'avvento dei Web Service, l'integrazione tra applicazioni scritte in linguaggi di programmazione diversi ed installate su differenti piattaforme operative era molto difficoltosa e gli sviluppatori erano costretti a creare attraverso procedure complesse e con lunghi tempi di sviluppo, un'infrastruttura software in grado di garantire l'interoperabilità tra i due sistemi. In definitiva, un lavoro lungo, costoso e problematico. I Web Service semplificano notevolmente tale compito, modificando lo scenario e rendendo possibile l'interoperabilità attraverso procedure piuttosto semplici o, comunque, notevolmente più agevoli e accessibili di quanto non fosse in passato. Con i Web Service qualsiasi applicazione può essere integrata con un'altra, purché sia Internet-enabled, ovvero sia in grado di comunicare attraverso Internet, dal momento che il nucleo fondamentale dei Web Services è un sistema di messaging in formato [10] XML che utilizza un protocollo di comunicazione Web come HTTP che è un protocollo leggero, standard e di facile implementazione. Le benefiche ripercussioni sono evidenti sia per le aziende, che al loro interno possono così integrare sistemi ed applicativi diversi senza dover sostenere costi proibitivi di sviluppo, sia per l'interoperabilità tra aziende diverse sotto l'aspetto [11] B2B (Business to Business) che risulta così semplificata, sia per rendere possibile ai milioni di utenti del Web la fruizione di servizi altrimenti non disponibili o di difficile realizzazione.

Dal punto di vista organizzativo è plausibile immaginare che ogni azienda possa fornire servizi e al contempo utilizzarne. In particolare, i servizi forniti, che rappresentano le funzionalità che l'azienda intende esportare verso l'esterno, delineano il confine tra ciò che del proprio sistema informativo aziendale è pubblico e ciò che al contrario rimane privato.

D'altro canto, affinché questa interoperabilità sia fattibile (dal punto di vista tecnologico), le aziende devono accordarsi su un linguaggio comune di descrizione dei servizi in modo tale da riconoscere gli elementi che un sistema mette a disposizione. Ciò deve essere accompagnato anche da un meccanismo di ricerca dei servizi esistenti e dalla possibilità di utilizzare il Web come canale di trasmissione. In questo paragrafo ci si concentrerà unicamente sugli aspetti tecnologici soffermandosi, quindi, sui Web Service quale soluzione tecnologica adatta all'interoperabilità dei sistemi. Va altresì sottolineato che il ruolo dei Web Service non si limita solo a un discorso di interoperabilità, ma permette anche di descrivere nuovi servizi realizzati ad hoc, sempre però con l'intento di fornire una soluzione platform-independent. Il primo passo è, quindi, quello di separare nettamente, come del resto avviene anche nei modelli di programmazione basati su componenti, la logica di presentazione da quella applicativa. I Web Service, infatti, si occupano solamente della logica applicativa, e sarà quindi il fruitore del servizio a presentare i dati ottenuti utilizzando il servizio con stili e grafica propri.
Si immagini, ad esempio, che un'azienda voglia, all'interno della propria pagina Web, inserire una piccola casella in cui mostrare il valore della quotazione in tempo reale delle proprie azioni. L'azienda per avere questo tipo di informazione dovrà necessariamente richiedere i dati periodicamente al sistema informativo della Borsa utilizzando un servizio appositamente fornito da quest'ultima; tale servizio richiederà in input il simbolo dell'azione e restituirà in output la quotazione attuale. Il Web master dell'azienda non dovrà fare altro che invocare il servizio e far sì che una porzione della pagina Web dell'azienda sia alimentata dai dati ottenuti in risposta.
Focalizzando l'attenzione sul concetto di servizio è ovvio immaginare, anche alla luce di quanto detto finora, come gli attori in causa siano necessariamente il fornitore e il richiedente. Questo tipo di paradigma è il medesimo che si riscontra nella tipica interazione di tipo client-server.

2.1.1 REST

REST (Representational State Transfer) rappresenta uno stile architetturale impiegato per sistemi distribuiti come il [12] World Wide Web.

I termini Representational State Transfer e REST furono introdotti nel 2000 nella tesi di dottorato di Roy Fielding, uno dei principali autori delle specifiche dell'Hypertext Transfer Protocol (HTTP).
REST si riferisce ad un insieme di principi di architetture di rete, che delineano come le risorse sono definite e indirizzate.
Un concetto importante in REST è l'esistenza di risorse intese come fonti di informazione, a cui si può accedere attraverso un identificatore globale chiamato URI. Per utilizzare le risorse, le componenti di una rete (componenti client e server) comunicano attraverso una interfaccia standard (HTTP) e si scambiano rappresentazioni di queste risorse.
Per comunicare con una risorsa è richiesta la conoscenza di due sole cose: l'identificatore della risorsa e l'azione richiesta.
L'applicazione comunque deve conoscere il formato dell'informazione (rappresentazione) restituita, tipicamente un documento [13] HTML, XML o [14] JSON, ma potrebbe essere anche un'immagine o qualsiasi altro contenuto.
L'approccio architetturale REST è definito dai seguenti sei vincoli applicati:

- Client–server: un insieme di interfacce uniformi separa i client dai server garantendo una maggior separazione ed indipendenza dei ruoli. I client non si devono preoccupare del salvataggio delle informazioni, che rimane all'interno di ogni singolo server, in questo modo la portabilità del codice del client ne trae vantaggio. I server non si devono preoccupare dell'interfaccia grafica o dello stato dell'utente, in questo modo i server sono più semplici e maggiormente scalabili. Server e client possono essere sostituiti e sviluppati indipendentemente fintanto che l'interfaccia non viene modificata.
- Stateless: non viene memorizzata su server alcuna informazione client. Ogni richiesta da ogni client contiene tutte le informazioni necessarie per richiedere il servizio, e lo stato della sessione è contenuto sul client.
- Cacheable: i client possono fare caching delle risposte. Le risposte devono in ogni modo definirsi implicitamente o esplicitamente cacheable o no, in modo da prevenire che i client possano riusare stati vecchi o dati errati. Una gestione ben fatta della cache può ridurre o parzialmente eliminare le comunicazioni client server, migliorando scalabilità e performance.

- Layered system: un client non può dire se è connesso direttamente ad un server di livello più basso od intermedio, i server intermedi possono migliorare la scalabilità del sistema con load-balancing o con cache distribuite. Layer intermedi possono offrire inoltre politiche di sicurezza
- Code on demand: (opzionale) i server possono temporaneamente estendere o personalizzare le funzionalità del client trasferendo del codice eseguibile. Ad esempio questo può includere componenti compilati come Applet Java o linguaggi di scripting client side come JavaScript. "Code on demand" è l'unico vincolo opzionale per la definizione di un'architettura REST.
- Uniform interface: un'interfaccia di comunicazione omogenea tra client e server permette di semplificare e disaccoppiare l'architettura, la quale si può evolvere separatamente.

[15] L'unico vincolo opzionale è il Code on demand, mentre gli altri cinque sono invece obbligatori: se anche uno solo degli altri cinque non è rispettato, allora un servizio non può essere riconosciuto come RESTful. Dall'analisi di questi vincoli si comprende che le architetture RESTful sono fortemente basate sulla più grande infrastruttura informatica creata dall'uomo, il Web, e permettono a qualsiasi sistema ipermediale distribuito di possedere importanti proprietà, quali: semplicità, scalabilità, portabilità, visibilità e tipicamente elevate performance.

Citando Fielding: "La separazione delle responsabilità dettata dalle linee guida REST semplifica l'implementazione dei componenti, riduce la complessità della semantica dei connettori, migliora l'efficacia per la messa a punto delle prestazioni e aumenta la scalabilità dei componenti server-side. I vincoli dei sistemi a strati permettono ai sistemi intermedi (proxy, gateway e firewall) di poter essere introdotti in convenienti punti della rete senza cambiare le interfacce tra i componenti, permettendo così loro di assistere nella gestione della comunicazione e/o di migliorare le prestazioni grazie a cache condivise su larga scala. REST consente l'elaborazione intermedia forzando i messaggi a essere auto-descrittivi: l'interazione tra le richieste è stateless, i metodi standard e i

media type sono usati per indicare la semantica e lo scambio di informazioni, e le risposte esplicitamente indicano come eseguire la cache".

Inoltre, va notato che questi vincoli non indicano il tipo di tecnologia da utilizzare: ci sono però direttive che definiscono come i dati devono essere trasferiti tra i componenti e quali sono i vantaggi nel seguire tali indicazioni.

2.1.2 RESTFUL WEB SERVICE

“ Un servizio RESTful web service è un’interfaccia che mette a disposizione operazioni accessibili attraverso una rete mediante protocollo HTTP”.

I RESTful web service rappresentano la nuova frontiera dei servizi web, permettendo un’alta interoperabilità ed un’applicazione più agevole dei tradizionali servizi web basati sull’architettura [16] SOA (Service oriented architecture).
Alla base dei RESTful web service, troviamo REST (REpresentational State Transfer) che rappresenta un’architettura software per sistemi distribuiti come il World Wide Web.
L’architettura RESTful è di tipo client-server, è senza stato (stateless) ed i dati scambiati sono opportune rappresentazioni di risorse ottenute impiegando formati e codifiche standard del web.
Il concetto centrale per i sistemi RESTful è quello di risorsa: una risorsa è una qualunque entità che possa essere indirizzabile tramite web, come ad esempio una pagina web, un articolo di giornale su un sito di notizie, un’immagine di una web gallery, un tweet ti un determinato utente e tante altre.
La maggior parte delle caratteristiche dello stile architetturale RESTful, combacia con i requisiti della web service architecture: ed è per questa ragione che sono nati i servizi web in stile REST o RESTful web service.
I servizi web RESTful utilizzano il protocollo HTTP per il trasporto delle risorse. Inoltre possono utilizzare una qualsiasi codifica standard, solitamente espressa nei

formati XML o JSON, e dispongono di un insieme prefissato di metodi (comunemente chiamati verbi http) come GET, POST, DELETE e PUT che permettono di sfruttare a pieno la semantica e la ricchezza dei comandi HTTP e le sue funzionalità per la negoziazione dei contenuti.

Le risorse sono organizzate in strutture gerarchiche di tipo directory.

Ogni risorsa dispone di un identificativo univoco all'interno della propria directory.

[17] Le directory e le risorse sono identificate univocamente da un URI, ad esempio la directory http://www.esempio.com/dir1/id indica la risorsa con identificativo id all'interno della struttura dir1.

Quindi si possono distinguere due tipologie di risorse identificate tramite URI:

- Collezione di risorse, se URL termina con "/"
- Risorsa singola, se URL termina con "/id"

I metodi del servizio vengono impiegati attraverso l'invocazione di uno dei verbi HTTP sull'URI della risorsa.

L'interpretazione di questi metodi può essere schematizzati nella seguente logica:

- GET: applicato all'URI di una directory, ritorna un elenco di elementi che vi sono contenuti; mentre se applicato alla URI di una risorsa, ritorna la risorsa stessa (READ).
- PUT: applicato all'URI di una directory, la rimpiazza totalmente con gli elementi inseriti nel payload; mentre se applicato all'URI di una risorsa, la aggiorna coi i dati del payload (UPDATE).
- POST: applicato all'URI di una directory, crea una nuova risorsa al suo interno, le assegna come contenuto il payload del messaggio e ritorna la sua URI (INSERT).
- DELETE: applicato all'URI di una directory, la rimuove assieme a tutto il suo contenuto; mentre se applicato all'URI di una singola risorsa, la rimuove dalla directory (DELETE).

Perché un Web Service sia conforme alle specifiche REST, deve rispettare alcune caratteristiche:

- Architettura basata su client-server.
- Stateless: ogni request/response tra client e server deve rappresentare un'interazione completa; così facendo non è necessario mantenere informazioni aggiuntive sull'utente, portando quindi benefici lato server sulla riduzione di risorse e di complessità.
- Accesso uniforme: ogni risorsa deve avere un unico indirizzo per la sua reperibilità ed ogni risorsa di ogni sistema deve presentare la stessa identica interfaccia, ovvero quella indicata dal protocollo HTTP.

[18] Alcuni esempi pratici sull'uso di chiamate RESTful web service possono essere così riassunte:

- Accesso in lettura ad una risorsa con metodo GET

```
GET /utenti/Rossi HTTP/1.1
```

Figura 1 - Metodo Get

- Creazione di una risorsa con metodo POST

```
POST /utenti HTTP/1.1
Host: server
Content-Type: application/xml
<?xml version = "1.0"?>
<utente>
        <nome>Rossi</nome>
</utente>
```

Figura 2 - Metodo Post

- Modifica di una risorsa con metodo PUT

```
PUT /utenti/Rossi HTTP/1.1
Host: server
Content-Type: application/xml
<?xml version = "1.0"?>
<utente>
        <nome>Bianchi</nome>
</utente>
```

Figura 3 - Metodo Put

- Eliminazione di una risorsa con metodo DELETE

```
DELETE /utenti/Rossi HTTP/1.1
```

Figura 4 - Metodo Delete

Da quello che si evince, RESTful web service permette non solo una implementazione di un'architettura più immediata e scalabile orientata alle risorse, ma anche una sua più facile gestione ed interazione.
La sua forza risiede nell'uso del modello HTTP di cui ne sfrutta le potenzialità, realizzando di fatto un approccio molto più naturale di quanto non facciano i Web Service tradizionali.

2.1.3 AMBIENTE DI SVILUPPO

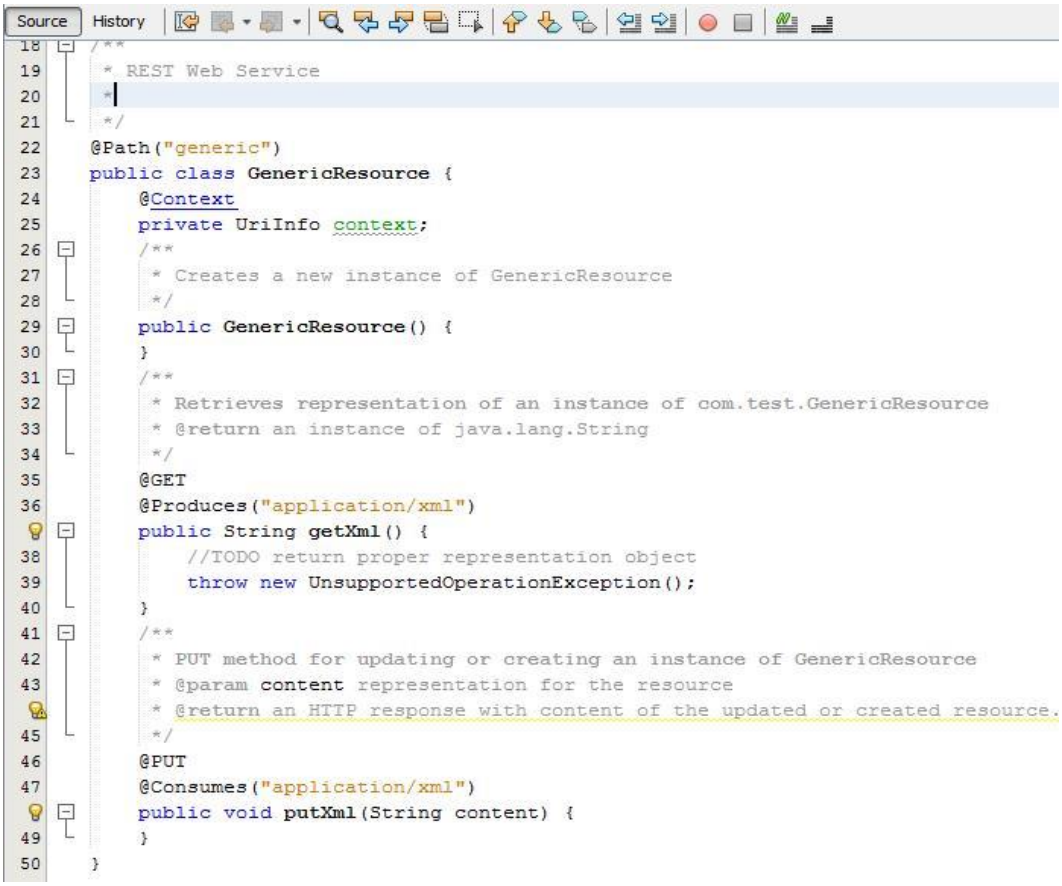
Per lo sviluppo ed il debug del codice è stato scelto come IDE NetBeans.
È stata installata la versione 8.0 per l'implementazione del codice di progetto.
Netbeans per lo sviluppo dei RESTful web service fa uso di una libreria chiamata JAX-RS.

JAX-RS (Java API for RESTful Web Services) è una Java API che fornisce il supporto alla creazione di web service in accordo all'architettura REST.
La libreria usa le annotazioni, introdotte in Java SE 5, per semplificare lo sviluppo ed il deployment dei servizi web.
Dalla versione 1.1 JAX-RS è ufficialmente parte integrante di Java EE 6; col vantaggio di non richiedere alcuna configurazione specifica per il suo utilizzo.
Le annotazioni principali di cui fa uso sono le seguenti:

- @Path: specifica il path relativo di una risorsa: più esattamente associa i metodi di una classe o la classe stessa al path indicato.
- @GET, @PUT, @POST, @DELETE: specificano il tipo di richiesta HTTP per una risorsa.
- @Produces: specifica il MIME Type della risposta
- @Consumes: specifica il MIME type che deve avere la richiesta per essere consumata.
- @PathParam: viene impiegato insieme ai path parametrici ed associa un parametro ad una variabile in ingresso al metodo.

Oltre alla libreria di cui fa uso, NetBeans nella sua configurazione mette a disposizione degli sviluppatori diverse tipologie progettuali, tra cui Java Web per la realizzazione e lo sviluppo di Web Application.
Una volta realizzata l'applicazione web, è possibile creare al suo interno diversi servizi, compresi i RESTful web service.
[19] I RESTful web service presi in considerazione fanno riferimento ai RESTful web service from patterns, che si basano sui seguenti design pattern:

- Simple Root Resource: crea un servizio Restful con una risorsa root con metodi di default GET e PUT; utile per creare servizi semplici.
  Nella sua fase di configurazione è possibile indicare il tipo di path, il nome della classe da utilizzare ed il tipo di MIME da impiegare.



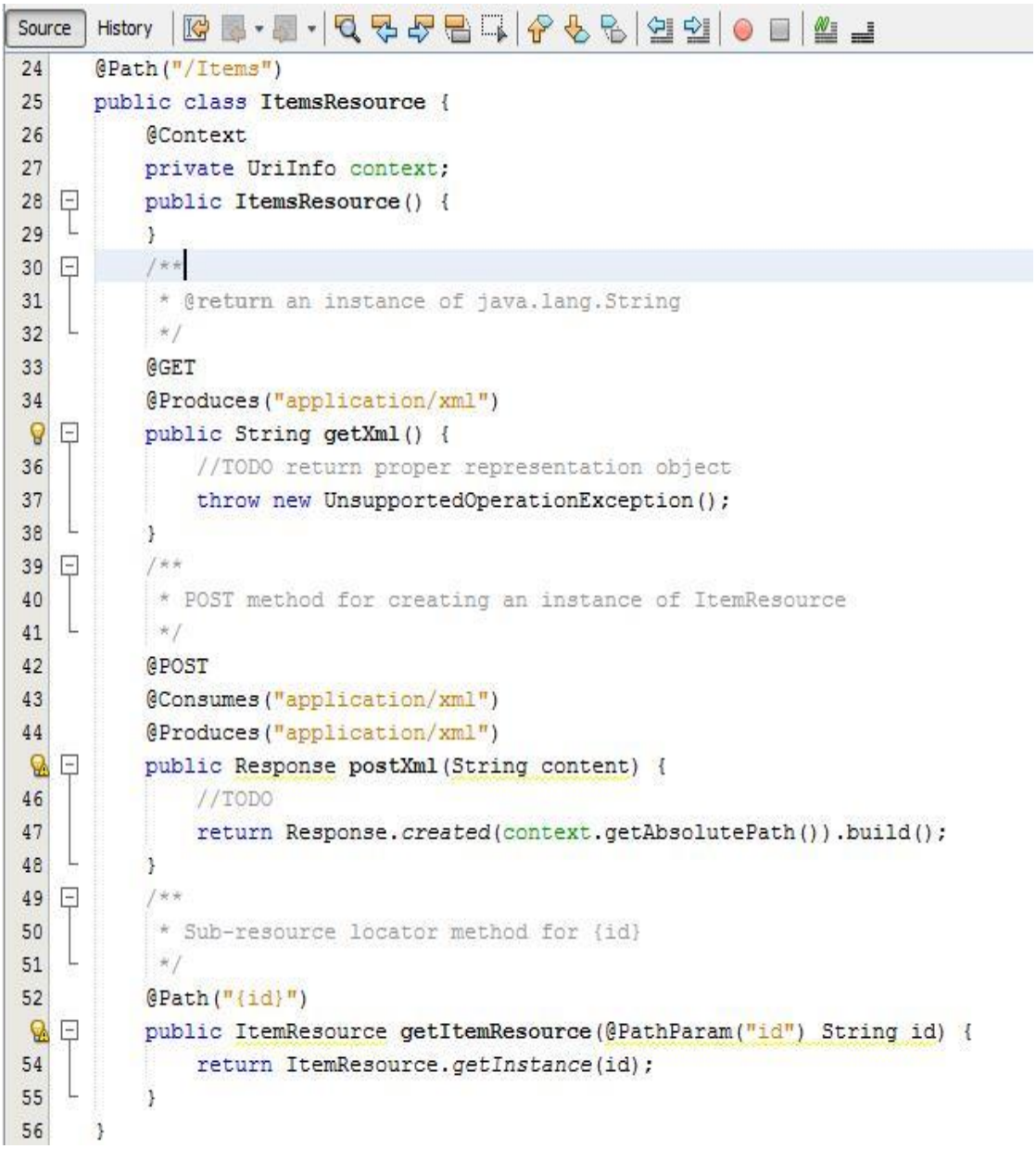
```
Source  History
/**
 * REST Web Service
 *
 */
@Path("generic")
public class GenericResource {
    @Context
    private UriInfo context;
    /**
     * Creates a new instance of GenericResource
     */
    public GenericResource() {
    }
    /**
     * Retrieves representation of an instance of com.test.GenericResource
     * @return an instance of java.lang.String
     */
    @GET
    @Produces("application/xml")
    public String getXml() {
        //TODO return proper representation object
        throw new UnsupportedOperationException();
    }
    /**
     * PUT method for updating or creating an instance of GenericResource
     * @param content representation for the resource
     * @return an HTTP response with content of the updated or created resource.
     */
    @PUT
    @Consumes("application/xml")
    public void putXml(String content) {
    }
}
```

Figura 5 - Simple Root Resource

Da come è possibile notare, lo scheletro creato fornisce due metodi: GET e PUT.
Il metodo GET, ricevuta la richiesta, ritorna un messaggio di tipo MIME application/xml: specificato dall'annotation @Produces.
Il metodo PUT, che può essere utilizzato per aggiornare o creare un'istanza della risorsa, indica che la richiesta deve avere come MIME type application/xml per fare in modo che il servizio possa processarla: specificato dall'annotation @Consumes.

- Container-Item: Crea una coppia di classi, una per le risorse item ed una per la risorsa container. Le risorse degli elementi possono essere create e aggiunte alla risorsa container usanto il metodo POST.
  Va notato che l'URI creato per i nuovi elementi, sono determinati dalla risorsa container.

Classe container:



```
Source  History
24  @Path("/Items")
25  public class ItemsResource {
26      @Context
27      private UriInfo context;
28      public ItemsResource() {
29      }
30      /**
31       * @return an instance of java.lang.String
32       */
33      @GET
34      @Produces("application/xml")
        public String getXml() {
36          //TODO return proper representation object
37          throw new UnsupportedOperationException();
38      }
39      /**
40       * POST method for creating an instance of ItemResource
41       */
42      @POST
43      @Consumes("application/xml")
44      @Produces("application/xml")
        public Response postXml(String content) {
46          //TODO
47          return Response.created(context.getAbsolutePath()).build();
48      }
49      /**
50       * Sub-resource locator method for {id}
51       */
52      @Path("{id}")
        public ItemResource getItemResource(@PathParam("id") String id) {
54          return ItemResource.getInstance(id);
55      }
56  }
```

Figura 6 – Container Resource

Classe per le risorse item:

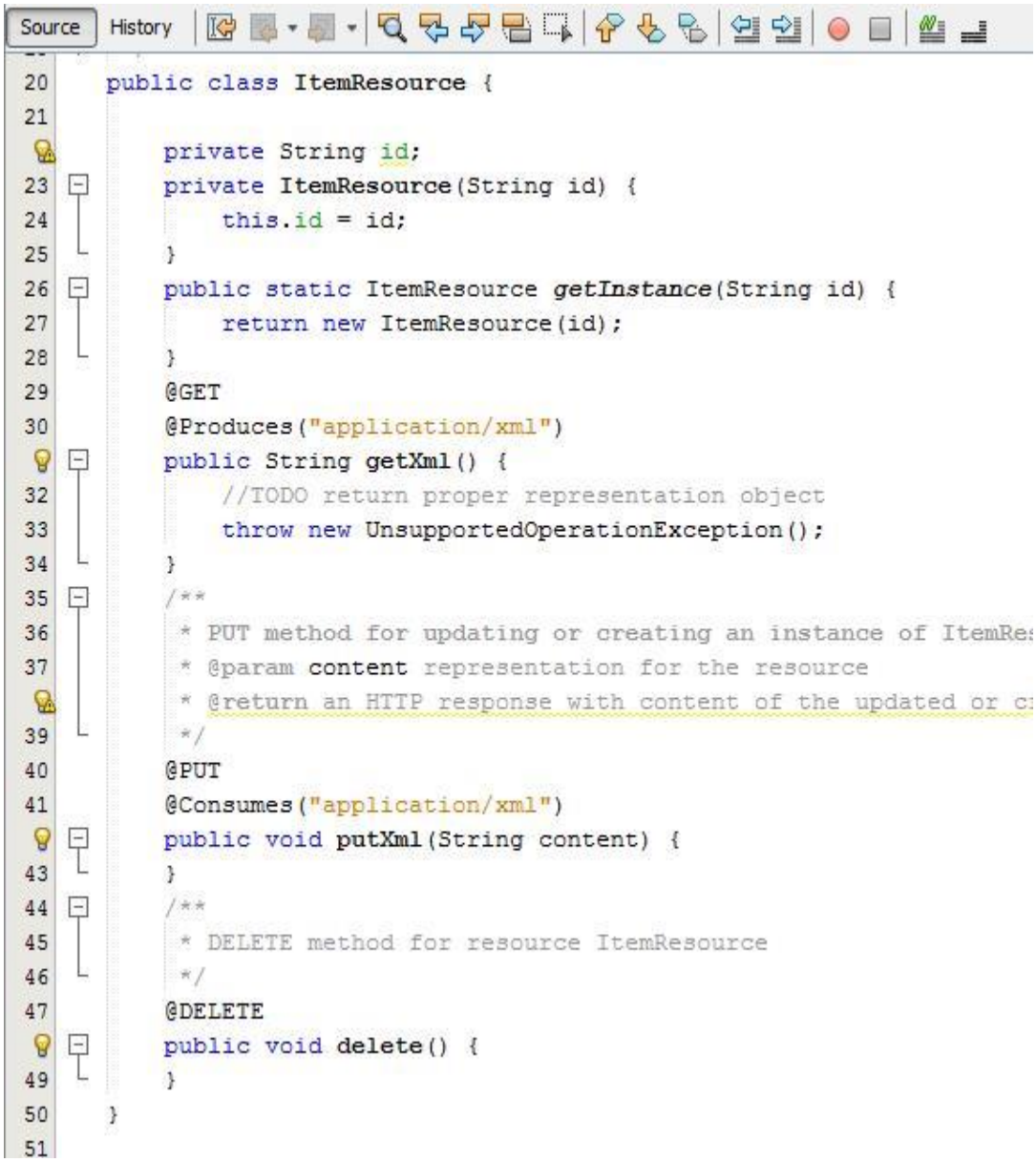


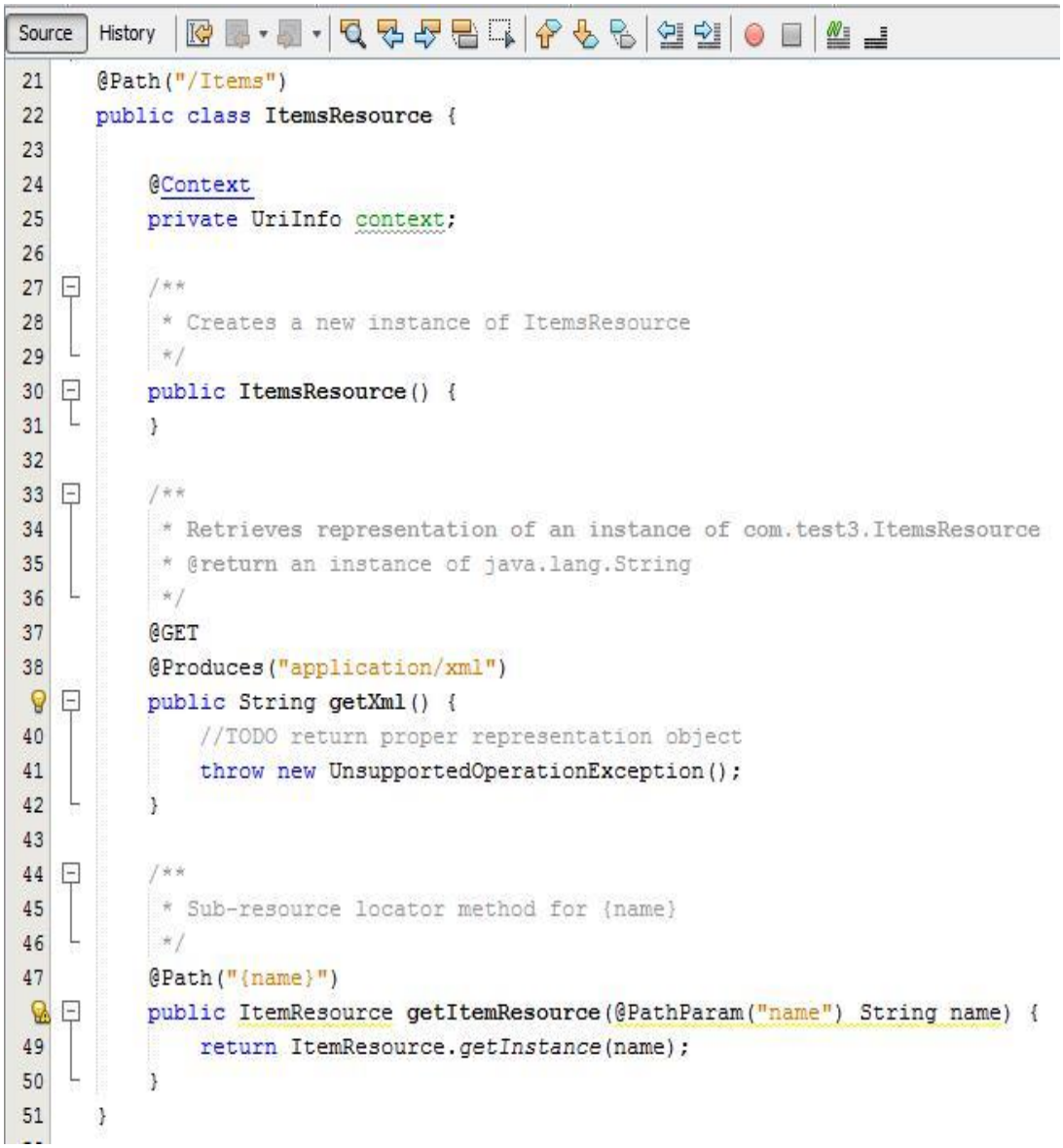
Figura 7 – Item Resource

- Client-Controlled Container-Item: crea una coppia di classi di cui una per le risorse item ed una come risorsa container.

  Questo pattern è simile al modello precedente, ma con la differenza che non c'è il metodo POST nella risorsa container (utilizzato per creare l'elemento risorsa). Invece, le risorse di tipo item sono create usando il metodo PUT direttamente nella classe delle risorse item.

  La ragione di questo è per fare in modo che l'URI della risorsa item possa essere controllata dall'utente e non dalla risorsa container.

Classe container resource:

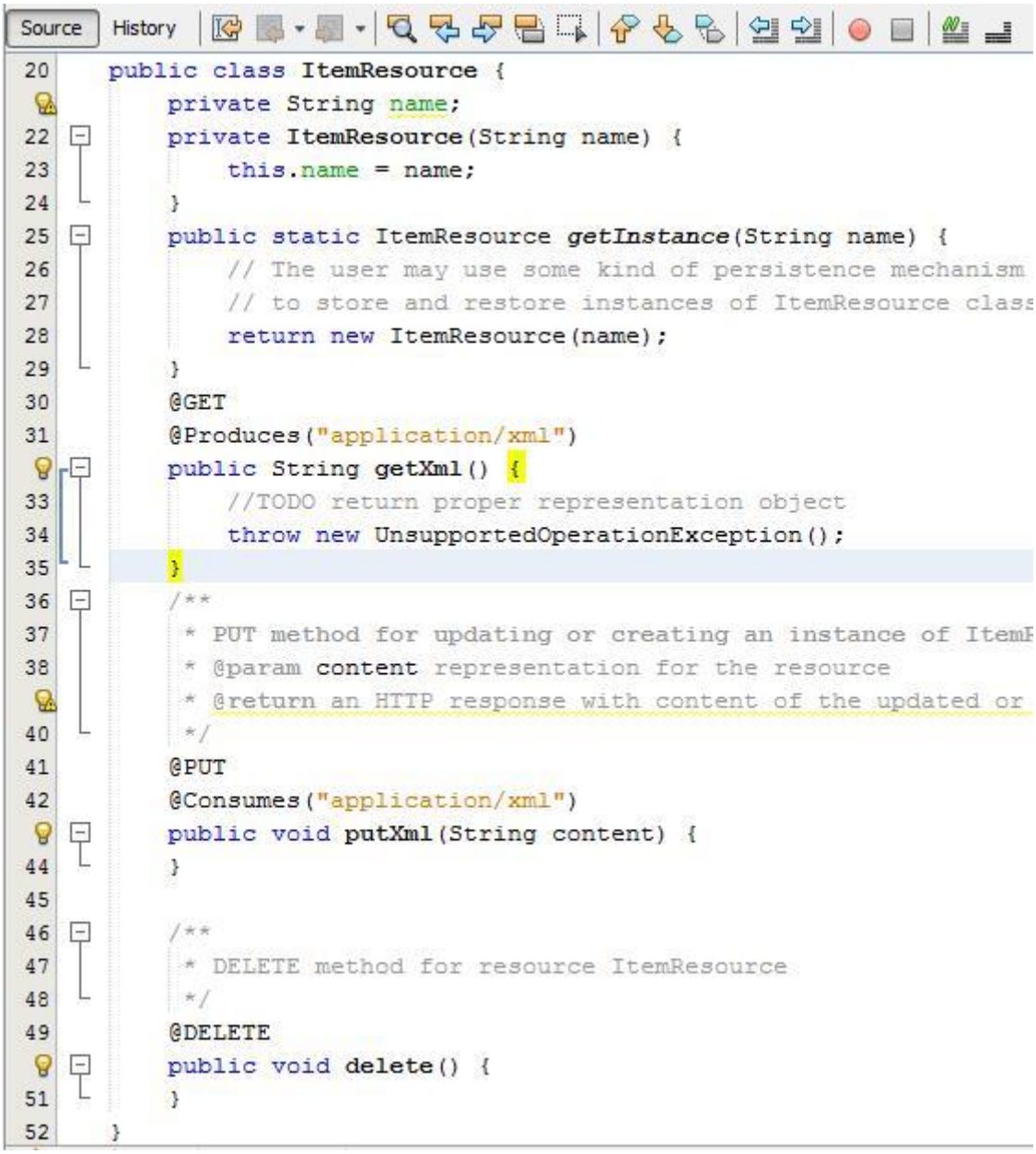


```
21  @Path("/Items")
22  public class ItemsResource {
23
24      @Context
25      private UriInfo context;
26
27      /**
28       * Creates a new instance of ItemsResource
29       */
30      public ItemsResource() {
31      }
32
33      /**
34       * Retrieves representation of an instance of com.test3.ItemsResource
35       * @return an instance of java.lang.String
36       */
37      @GET
38      @Produces("application/xml")
        public String getXml() {
40          //TODO return proper representation object
41          throw new UnsupportedOperationException();
42      }
43
44      /**
45       * Sub-resource locator method for {name}
46       */
47      @Path("{name}")
        public ItemResource getItemResource(@PathParam("name") String name) {
49          return ItemResource.getInstance(name);
50      }
51  }
```

Figura 8 – Client-Controlled Container Resource

In effetti si nota che la risorsa container non ha più il controllo sulla creazione dell'URI; in quanto non dispone più del metodo POST.

Classe item resource:



```
Source  History
20  public class ItemResource {
21      private String name;
22      private ItemResource(String name) {
23          this.name = name;
24      }
25      public static ItemResource getInstance(String name) {
26          // The user may use some kind of persistence mechanism
27          // to store and restore instances of ItemResource class
28          return new ItemResource(name);
29      }
30      @GET
31      @Produces("application/xml")
32      public String getXml() {
33          //TODO return proper representation object
34          throw new UnsupportedOperationException();
35      }
36      /**
37       * PUT method for updating or creating an instance of ItemR
38       * @param content representation for the resource
39       * @return an HTTP response with content of the updated or
40       */
41      @PUT
42      @Consumes("application/xml")
43      public void putXml(String content) {
44      }
45
46      /**
47       * DELETE method for resource ItemResource
48       */
49      @DELETE
50      public void delete() {
51      }
52  }
```

Figura 9 - Client-Controlled Item Resource

Si nota come sia stato inserito il metodo PUT per dare la possibilità all'utente di richiamare il metodo generando l'URI indicato: da notare che nel messaggio si può inserire sia l'url sia l'oggetto da aggiornare o creare, il tutto nel formato di rappresentazione application/xml.

2.2 *ANDROID*

[20] Android è un sistema operativo per dispositivi mobili basato sul kernel Linux ed è sviluppato da Google Inc. a partire dal 2007.

Dispone di un'interfaccia utente basata sulla manipolazione diretta, Android è stato progettato principalmente per i dispositivi mobili touchscreen, quali smartphone e tablet.

Il sistema operativo utilizza input basati sul tocco, come interazione con l'utente, che meglio corrispondono alle azioni del mondo reale: come sfogliare, toccare, e manipolare oggetti sullo schermo direttamente.

Nonostante sia stato progettato principalmente per l'input touchscreen, è stato anche utilizzato in console per videogiochi, fotocamere digitali e altri dispositivi elettronici.

Il codice sorgente di Android è stato rilasciato da Google sotto licenze open source, anche se la maggior parte dei dispositivi Android ultimamente si integrano sempre più con combinazioni di open source e software proprietario.

Android fu inizialmente sviluppato da Android Inc., che venne prima sostenuto finanziariamente da Google e poi acquistato dallo stesso nel 2005.

Android è stato inaugurato nel 2007 con la fondazione della Open Handset Alliance, un consorzio di hardware, software e società di telecomunicazioni dedicato agli standard aperti per dispositivi mobili.

Android è popolare tra le aziende tecnologiche che richiedono un sistema operativo per dispositivi high-tech facile da personalizzare ed a basso costo di realizzazione. La natura aperta di Android ha incoraggiato il suo utilizzo tra comunità di sviluppatori e appassionati per utilizzare il suo codice sorgente come base per la realizzazione di progetti finalizzati all'inserimento di nuove funzionalità o per portare Android su dispositivi che sono stati ufficialmente rilasciati per altri sistemi operativi.

[21] Sul mercato, attualmente, sono disponibili diverse piattaforme di esecuzione oltre ad Android: iOS, Windows Phone, BlackBerry OS, MeeGo, etc....

Molti sviluppatori nonostante la vasta scelta, si orientano sempre più verso l'ambiente Android; questo perché presenta diversi vantaggi, come ad esempio il

linguaggio di programmazione Java che semplifica lo sviluppo di app: il linguaggio Java, rispetto al C ad esempio, gestisce automaticamente il ciclo di vita di un'applicazione ed anche l'allocazione di memoria, rendendo più semplice lo sviluppo.
Android inoltre è completamente open source, quindi è possibile riadattare il codice sorgente personalizzandolo ed ottimizzandolo in base alle proprie esigenze per ottenere una propria versione senza problemi di alcun genere.
Infine un altro fattore importante che va a suo favore è quello di essere il sistema operativo per dispositivi mobili più diffuso al mondo.

2.2.1 ARCHITETTURA ANDROID

Il sistema operativo Android si basa su kernel Linux ed è strutturato a vari livelli o layer, ognuno dei quali fornisce al livello superiore un'astrazione del sistema sottostante.
I layer principali sono quattro, ed esattamente:

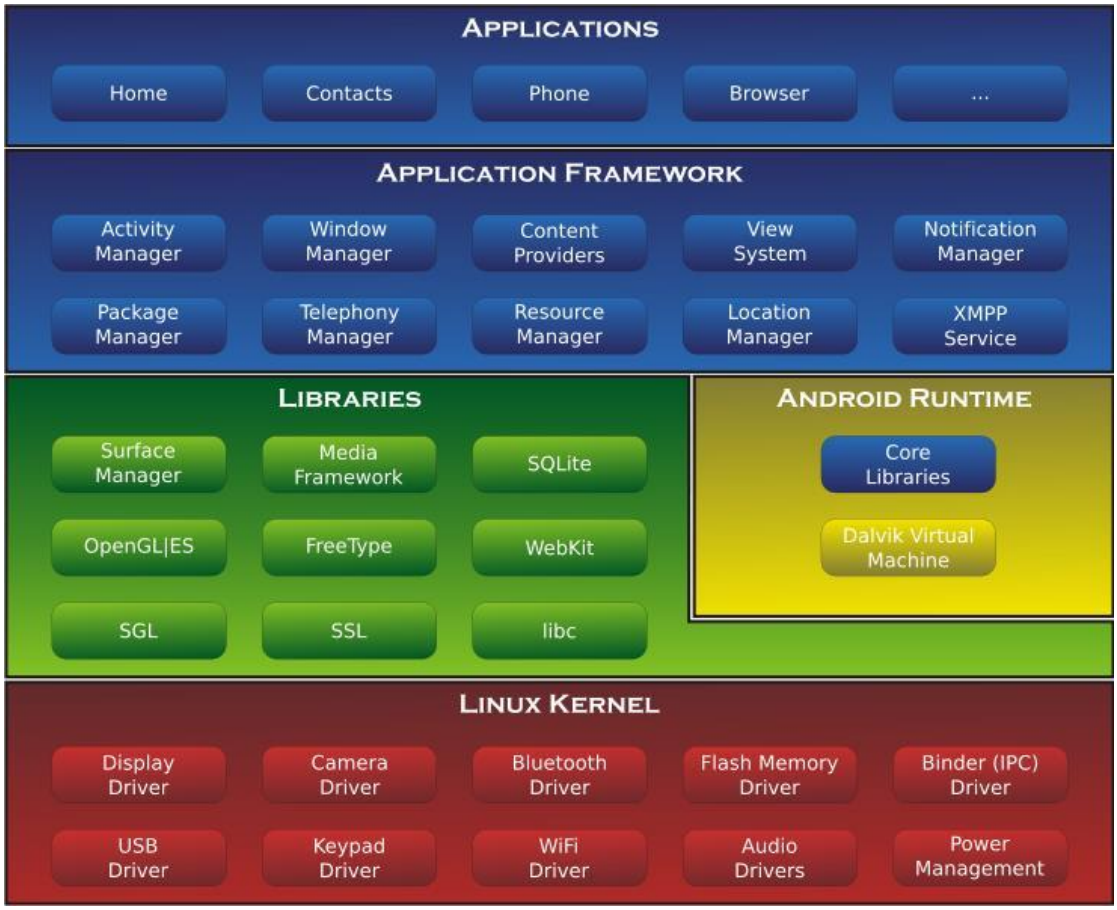


Figura 10 - Layer Android

- Linux Kernel
- Libraries
- Application Framework
- Applications

2.2.1.1 LINUX KERNEL

Android è costituito da un Kernel basato sul kernel Linux 2.6 e 3.x (da Android 4.0 in poi), con middleware, Librerie e API scritte in C (o C++) e software in esecuzione su un framework di applicazioni che include librerie Java compatibili

con librerie basate su Apache Harmony. Android utilizza la Dalvik virtual machine con un compilatore just-in-time per l'esecuzione di Dalvik dex-code (Dalvik Executable), che di solito viene tradotto da codice bytecode Java. La piattaforma hardware principale di Android è l'architettura ARM.
Il Kernel rappresenta il livello di astrazione di tutto l'hardware sottostante, include cioè i driver per la gestione del GPS, WiFi, Bluetooth, fotocamera, audio, USB ecc….

2.2.1.2 LIBRARIES

Il livello superiore riguarda le librerie C/C++ che vengono utilizzate da vari componenti del sistema. Tra queste troviamo:

- Il Media Framework: gestisce i Codec per l'acquisizione e riproduzione dei contenuti audio e video.
- Il Surface Manager: gestisce i componenti di un'interfaccia grafica come ad esempio le View.
- SQLite: il Database Management System (DBMS) utilizzato da Android.
- SGL e OpenGL ES: librerie per la gestione grafica 2D e 3D.
- SSL: libreria per il Secure Socket Layer.
- FreeType: motore di rendering dei font.
- LibC: libreria di sistema per lo standard C.

- Android runtime: ambiente di runtime composto dalla Core Library e la Dalvik Virtual Machine. Insieme rappresentano l'ambiente di sviluppo Android.

2.2.1.3 APPLICATION FRAMEWORK

In questo livello si trovano i gestori e le applicazioni di base del sistema.
Tra i vari gestori presenti troviamo:

- Activity Manager: gestisce tutto il ciclo di vita delle Activity. Le Activity sonoentità associate ad una schermata dell'applicazione. Il compito dell'Activity Manager è quindi quello di gestire le varie Activity sul display del terminale e di organizzarle in uno stack in base all'ordine di visualizzazione sullo schermo.

- Content Providers: gestiscono la condivisione di informazioni tra i vari processi attivi.

- Window Manager: gestisce le finestre relative alle applicazioni.

- Il Telephony Manager: gestisce le funzioni base del telefono quali chiamate ed SMS.

- Resource Manager: gestisce tutte le informazioni relative ad una applicazione (file di configurazione,file di definizione dei layout, immagini utilizzate, ecc).

- Package Manager: gestisce i processi di installazione e rimozione delle applicazioni dal sistema.

- Location Manager: mette a disposizione dello sviluppatore una serie di API

che si occupano della localizzazione (tramite GPS, rete cellulare o WiFi).

- System View: gestisce l'insieme degli elementi grafici utilizzati nella costruzione dell'interfaccia verso l'utente (bottoni, griglie, text boxes, ecc...).

- Notification Manager: gestisce le informazioni di notifica tra il dispositivo e l'utente.

2.2.1.4 APPLICATIONS

All'ultimo livello troviamo le applicazioni utente. Le funzionalità base del sistema come per esempio il telefono, il calendario, la rubrica, sono applicazioni scritte in Java; così come le applicazioni scritte da terze parti: Android le gestisce tutte allo stesso modo, cioè garantendo loro gli stessi diritti e privilegi.

2.2.2 APP ANDROID

Un'applicazione, chiamata anche app, indica un programma ideato appositamente per dispositivi mobili quali smartphone o tablet e che permette agli utenti di usufruire di funzioni sempre più ricche e personalizzate per esperienze che vanno dai giochi, alla musica, ai social network, alla gestione di eventi e documenti.
Rispetto ad un programma standard pensato per i tradizionali computer, le applicazioni Android sono più leggere e pensate per sfruttare al meglio gli ambienti limitati in cui si trovano: in termini di risorse di memoria e di elaborazione. Questo perché la capacità di calcolo e di capienza di un dispositivo mobile è al quanto limitata rispetto ai computer desktop o portatili.

Un'applicazione Android per poter funzionare deve avere tre componenti fondamentali, quali:

- Un file AndroidManifest.xml in cui viene descritto il comportamento e la configurazione dell'applicazione, nonché le risorse richieste per il suo funzionamento.
- La directory src contenente il codice sorgente.
- La cartella res contenente le risorse necessarie all'applicazione: come ad esempio la sottodirectory layout e drawable.

Gli elementi che compongono una applicazione Android sono diversi e rappresentano l'ossatura di qualsiasi progetto Android.
Tra questi troviamo:

- Activity: è una finestra che contiene l'interfaccia dell'applicazione con lo scopo di gestire l'interazione tra l'utente e l'applicazione stessa. Le applicazioni possono definire un qualsiasi numero di Activity per poter trattare diversi parti del software: ognuna svolge una particolare funzione e deve essere in grado di salvare il proprio stato in modo da poterlo ristabilire successivamente come parte del ciclo di vita dell'applicazione.

- Service: i servizi sono processi che per loro natura svolgono delle operazioni autonome e che vengono richiamati dalle attività al bisogno. Il sistema operativo fornisce alle applicazioni vari servizi già pronti all'uso, per ottenere l'accesso all'hardware o a risorse esterne.

- Content Provider: sono dei contenitori di dati generati dalle applicazioni che ne forniscono una condivisione; i dati possono essere contenuti nel file system, in un database SQLite, sul web o in una qualunque locazione di dati.

- Broadcast Receiver: permettono alle apps di ricevere segnali rivolti a tutte le apps in esecuzione, per la condivisione di dati o di segnali di servizio (come ad esempio quello di batteria scarica).

2.2.3 AMBIENTE DI SVILUPPO

Android permette di sviluppare con diversi sistemi operativi come Windows, Linux oppure Mac.
Per lo sviluppo vengono fornite diverse soluzioni ed inoltre non è richiesta la dotazione di un dispositivo Android per poter sviluppare: in quanto l'ambiente di sviluppo fornisce anche un emulatore con cui testare le app realizzate.
Tra le principali soluzioni offerte per lo sviluppo troviamo:

1) Possibilità di scaricare ed utilizzare il pacchetto [22] Eclipse ADT (Android Development Tools): è la soluzione più immediata e completa, poiché viene fornito tutto l'occorrente in un unico folder (Soluzione adottata in questa Tesi).

2) Possibilità di utilizzare l'ambiente Android Studio: ambiente di sviluppo ufficiale di Android, ma ancora in versione beta nel momento in cui si sta redigendo la presente.

3) Possibilità di integrare l'ambiente di sviluppo Android in Eclipse tramite plugin: in caso si disponesse già di Eclipse; soluzione che vedremo nel dettaglio.

Mentre le prime due sono immediate, l'ultima soluzione è quella che richiede più attenzione ed è anche quella che meglio si presta alla comprensione di tutti i componenti richiesti per lo sviluppo nel Mondo Android.
Per cui a titolo di esempio viene mostrato l'uso di questa configurazione e di quali siano le sue parti principali.
Il primo step consiste nell'installare l'SDK, sigla che indica Software Development Kit, utilizzato come kit di sviluppo software.
Il kit va scaricato dal sito di Android http://developer.android.com stando attenti a scegliere la versione del sistema operativo che si dispone. Una volta scaricato ed estratto, va selezionato al suo interno il file SDK Manager come evidenziato in figura

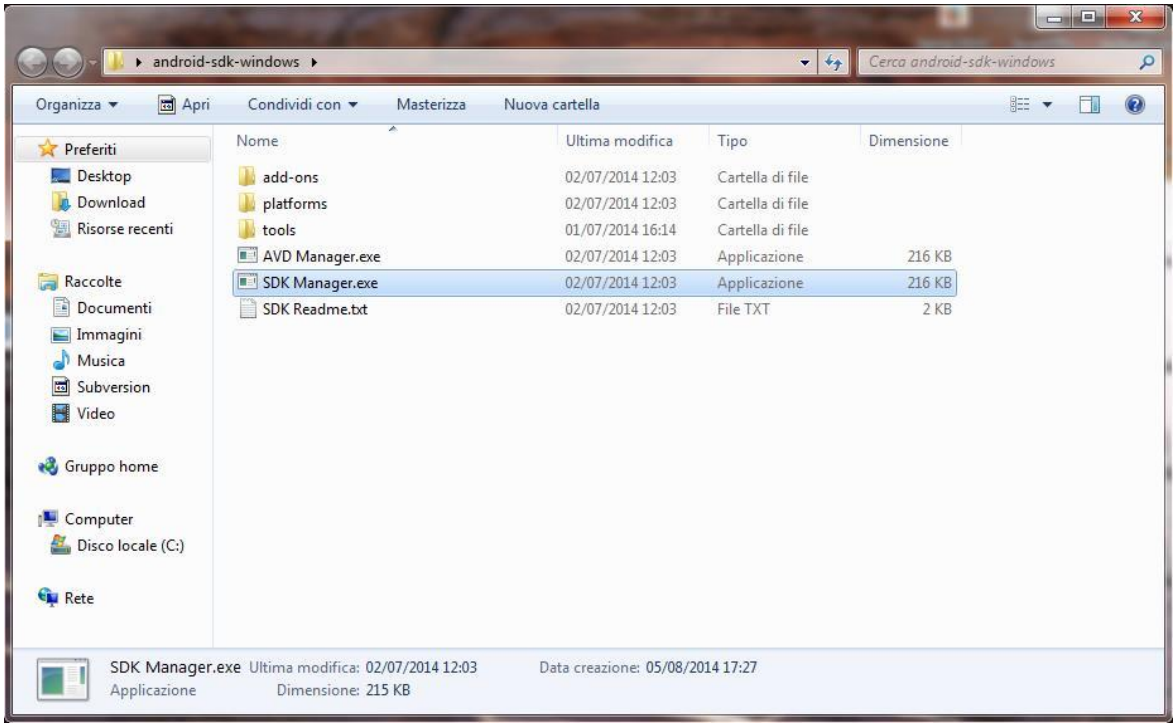


Figura 11 - Directory SDK Manager

avviando l'SDK Manager si possono così installare le versioni d'interesse.

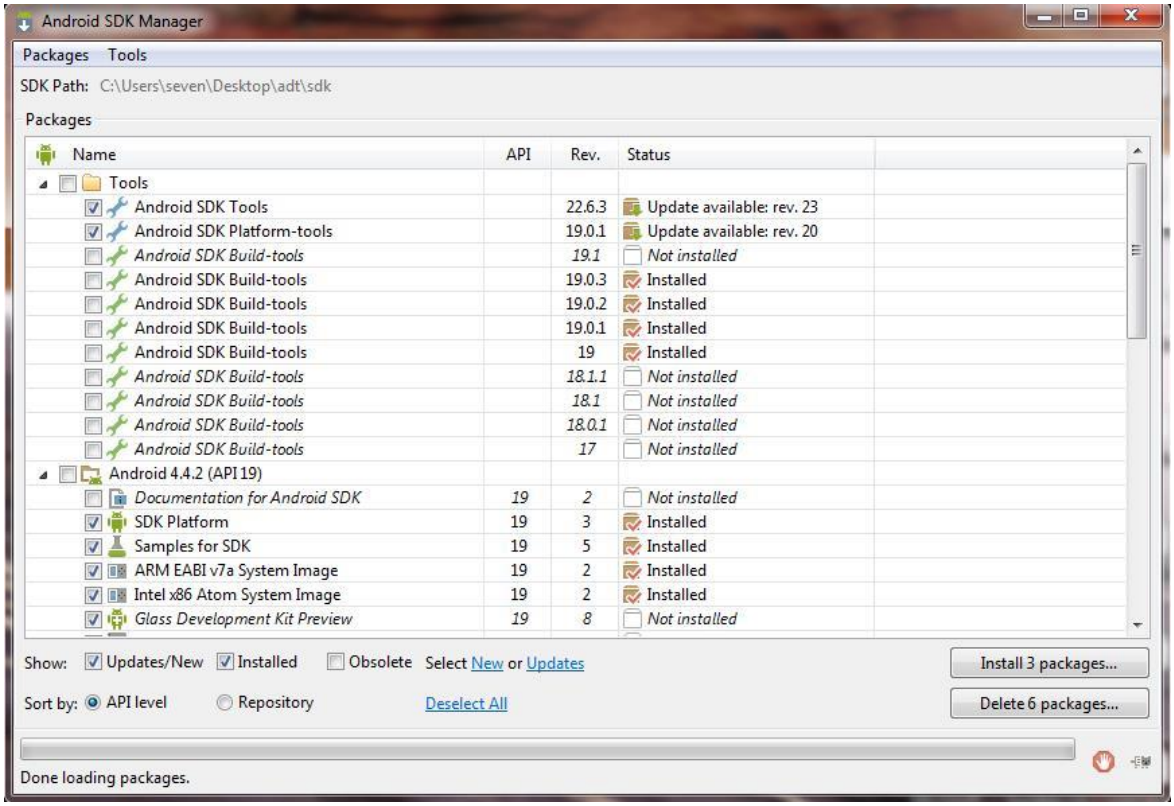


Figura 12 - Android SDK Manager

Una volta che si sono scelte le versioni SDK che si vogliono installare, basta selezionare il pulsante Install.

La seconda fase riguarda l'installazione di un altro componente necessario all'ambiente di sviluppo: il JDK (Java Development Kit).

Per ottenere il JDK è necessario scaricarlo dal sito http://www.oracle.com ed una volta ultimato il download installarlo nel proprio sistema.

Java SE Development Kit 8u11

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux x86	133.58 MB	jdk-8u11-linux-i586.rpm
Linux x86	152.55 MB	jdk-8u11-linux-i586.tar.gz
Linux x64	133.89 MB	jdk-8u11-linux-x64.rpm
Linux x64	151.65 MB	jdk-8u11-linux-x64.tar.gz
Mac OS X x64	207.82 MB	jdk-8u11-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	135.66 MB	jdk-8u11-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	96.14 MB	jdk-8u11-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	135.7 MB	jdk-8u11-solaris-x64.tar.Z
Solaris x64	93.18 MB	jdk-8u11-solaris-x64.tar.gz
Windows x86	151.81 MB	jdk-8u11-windows-i586.exe
Windows x64	155.29 MB	jdk-8u11-windows-x64.exe

Figura 13 - Java JDK

Per avviare correttamente gli eseguibili, bisogna includere la cartella bin del JDK nel PATH di Windows.

Il PATH di Windows include tutti i percorsi che possono essere invocati dalla shell.

Il compilatore javac.exe ed il runtime java.exe risiedono all'interno della cartella <JAVA_HOME>\bin bisogna quindi aggiungerla alla variabile PATH.

In pratica:

1) Fare click sul pulsante START e selezionare Pannello di Controllo.

2) Selezionare Sistema e Sicurezza.

3) Fare click su Sistema.

4) Scegliere sulla sinistra la voce Impostazioni di sistema Avanzate e poi Variabili d'Ambiente.

5) Dalla nuova schermata bisogna selezionare la variabile path, fare click sul pulsante modifica e incollare il percorso del JDK/bin.

Nel caso di Windows 7 o Windows 8 il percorso potrebbe esse simile a questo: C:\Program Files\Java\jdk1.7.0_65\bin\ come illustrato nell'immagine seguente

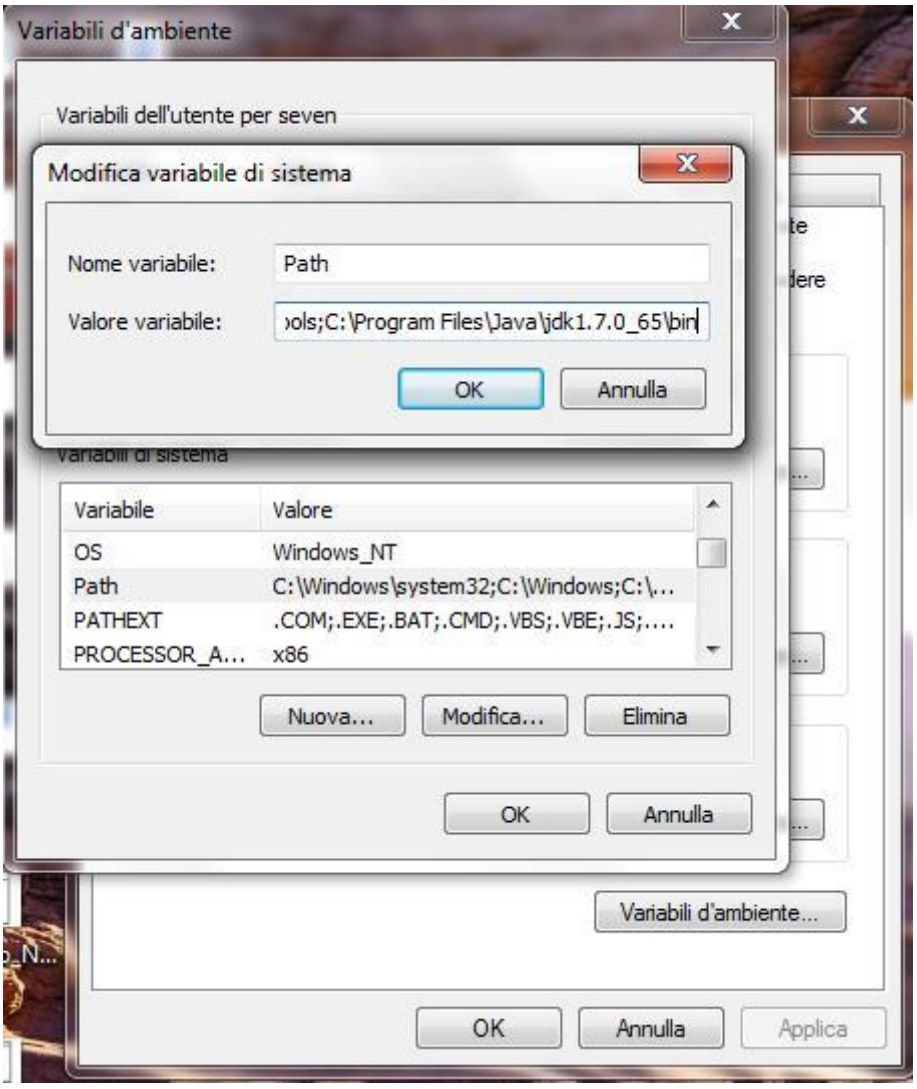


Figura 14 - Variabili Ambiente

2.3 *MAPPE*

Una mappa è una rappresentazione semplificata dello spazio che evidenzia relazioni tra componenti, oggetti o regioni, di quello spazio.
Una mappa è comunemente una rappresentazione bidimensionale, geometricamente accurata, di uno spazio tridimensionale, come ad esempio una carta geografica. Più in generale, le mappe possono essere usate per rappresentare qualsiasi proprietà locale del mondo o parte di esso, o qualsiasi altro spazio, anche mentale o concettuale.
Un ruolo importante riguardo le mappe è dato dalla Cartografia che rappresenta l'insieme di conoscenze tecniche e artistiche finalizzate alla rappresentazione simbolica di informazioni geografiche in relazione al luogo per il quale si realizzano.
Le mappe, inoltre, per poter essere utilizzate come punti di riferimento sul territorio, necessitano di un processo di georeferenziazione che permette loro di associare le coordinate ai punti che le compongono: in modo che ad ogni punto sulla mappa corrispondano coordinate specifiche.
La georeferenziazione permette quindi di utilizzare le mappe come guide o stradari di riferimento, che con l'ausilio di sistemi GPS diventano dei veri e propri navigatori.
Tra le mappe maggiormente utilizzate in diversi ambiti, compreso il mobile, troviamo le Google Maps e le OpenStreetMap.
In seguito viene fornita una trattazione specifica di entrambe le mappe.

2.3.1 GOOGLE MAPS

[23] Google Maps (nome precedente Google Local) è un servizio accessibile dal relativo sito web che consente la ricerca e la visualizzazione di mappe geografiche di buona parte della Terra.

Oltre a questo è possibile ricercare servizi in particolari luoghi, tra cui ristoranti, monumenti, negozi, inoltre si può trovare un possibile percorso stradale tra due punti e visualizzare foto satellitari di molte zone con diversi gradi di dettaglio (per le zone che sono state coperte dal servizio si riescono a distinguere in molti casi le case, i giardini, le strade e così via). Le foto sono statiche (non in tempo reale), buona parte delle quali sono riferite alla fine degli anni novanta. Oltre a queste funzioni, Google Maps offre anche una ricerca di attività commerciali sulle stesse mappe.
Le mappe di Google forniscono quindi un valido strumento per la georeferenziazione di attività, punti di interesse e tanto altro.
Per i dispositivi mobili che supportano la piattaforma Java (o anche quelli basati sui sistemi operativi Android, iOS, Windows Mobile, Palm OS o Symbian OS) e che dispongono di una connessione a Internet esiste dal 2006 un'apposita versione Google Maps, detta Google Maps Mobile, che permette di accedere alle mappe e di usufruire del servizio di navigazione gps come un vero e proprio Navigatore satellitare da tali dispositivi.
Google Maps è basato su una variante della Proiezione di Mercatore. Se la Terra fosse perfettamente sferica la proiezione sarebbe la stessa di quella di Mercatore. Google Maps usa le formule della Proiezione sferica di mercatore, ma le coordinate di google maps sono del GPS basate sui dati del sistema geodetico mondiale WGS 84.

2.3.2 OPENSTREETMAP

OpenStreetMap (OSM) è un progetto collaborativo finalizzato a creare mappe a contenuto libero del mondo. Il progetto punta ad una raccolta mondiale di dati geografici, con scopo principale la creazione di mappe e cartografie.
La caratteristica fondamentale dei dati geografici presenti in OSM è che possiedono una licenza libera. È cioè possibile utilizzarli liberamente per qualsiasi scopo con il solo vincolo di citare la fonte e usare la stessa licenza per eventuali lavori derivati dai dati di OSM. Tutti possono contribuire arricchendo o correggendo i dati. Il progetto è simile a Google Map Maker con la differenza che

ogni contributo fornito volontariamente dagli utenti resta di proprietà dell'utente stesso con una licenza Creative Commons.

Le mappe sono create usando come riferimento i dati registrati da dispositivi GPS portatili, fotografie aeree ed altre fonti libere. Sia le immagini renderizzate che i dati vettoriali, oltre che lo stesso database di geodati sono rilasciati sotto licenza Open Database License.

OpenStreetMap è stato ispirato da siti come Wikipedia: la pagina in cui la mappa è consultabile espone in evidenza un'etichetta "Modifica" per procedere con la modifica dei dati ed il progetto è accompagnato da un archivio storico delle modifiche (cronologia e log). Gli utenti registrati possono caricare nei database del progetto tracce GPS e modificare i dati vettoriali usando gli editor forniti.

Di base, i dati per la realizzazione delle mappe sono ricavati da schizzi realizzati da volontari che intraprendono sistematici rilievi sul territorio muniti di unità GPS portatili capaci di registrare il percorso, insieme a smartphone, computer portatili o registratori vocali; le informazioni raccolte vengono introdotte via computer nel database, in formato vettoriale.

Molti collaboratori usano programmi come GPSBabel per convertire i dati GPS dal formato grezzo (NMEA) o da formati proprietari al formato GPX (un'applicazione dell'XML). I dati, come latitudine/longitudine, sono raccolti nel formato WGS84 e sono normalmente visualizzati sulla proiezione di Mercatore.

I rilievi sul territorio vengono effettuati da volontari (autodefinitisi "mappatori", dall'inglese mappers) a piedi, in bici, auto, treno o qualsiasi altro mezzo di trasporto (benché la bicicletta rimanga il mezzo di trasporto preferito dai mapper nelle aree urbane), usando un'unità GPS, unitamente in alcuni casi a computer portatili, registratori vocali digitali e fotocamere digitali per la raccolta di dati (in alcuni casi si è rivelato utile anche la raccolta di informazioni mediante domande ai passanti). Alcuni contributori concentrano la propria attività nella propria città di residenza, raccogliendo dati in maniera sistematica. Inoltre, periodicamente vengono organizzati dai volontari dei mapping party, campagne di mappatura concentrate in un luogo preciso a cui i volontari partecipano divisi in squadre; queste attività possono durare anche più di un giorno nel caso di "mappature intensive".

2.4 *LIBRERIE*

Le librerie presentate di seguito permettono una facile integrazione delle mappe sui dispositivi mobili, offrendo oltre a questo personalizzazioni dei punti di interesse così come funzionalità in grado di arricchire e rendere più interattive le mappe stesse.
Le librerie forniscono un set di operazioni per intervenire ed interagire sulle mappe; è possibile ad esempio aggiungere degli oggetti su determinate zone del territorio ed associare a questi oggetti delle informazioni di attinenza, quali: il dettaglio del punto di riferimento, file multimediali che descrivono l'oggetto, descrizioni particolari o di orientamento e tanto altro a cui la fantasia può contribuire.
Oltre alle librerie per le mappe, un'altra library utilizzata in questo progetto è JavaAPIforKML che si occupa di fornire operazioni di xml-binding tramite funzionalità marshalling ed unmarhsalling.

2.4.1 OSMDROID

Osmdroid è una libreria che fornisce la possibilità di integrare una mappa attraverso una classe base chiamata MapView.
Questa libreria include anche un sistema tile provider con cui è possibile selezionare il tipo di provider a cui collegarsi per la visualizzazione delle mappe di riferimento, ed inoltre fornisce supporto overlay che permette di creare sovrapposizioni per tracciare icone, rilevamento di posizione, e disegno di oggetti e forme.
[24] La sua integrazione all'interno di un progetto può avvenire aggiungendo la libreria jar nel build path dell'ambiente di sviluppo

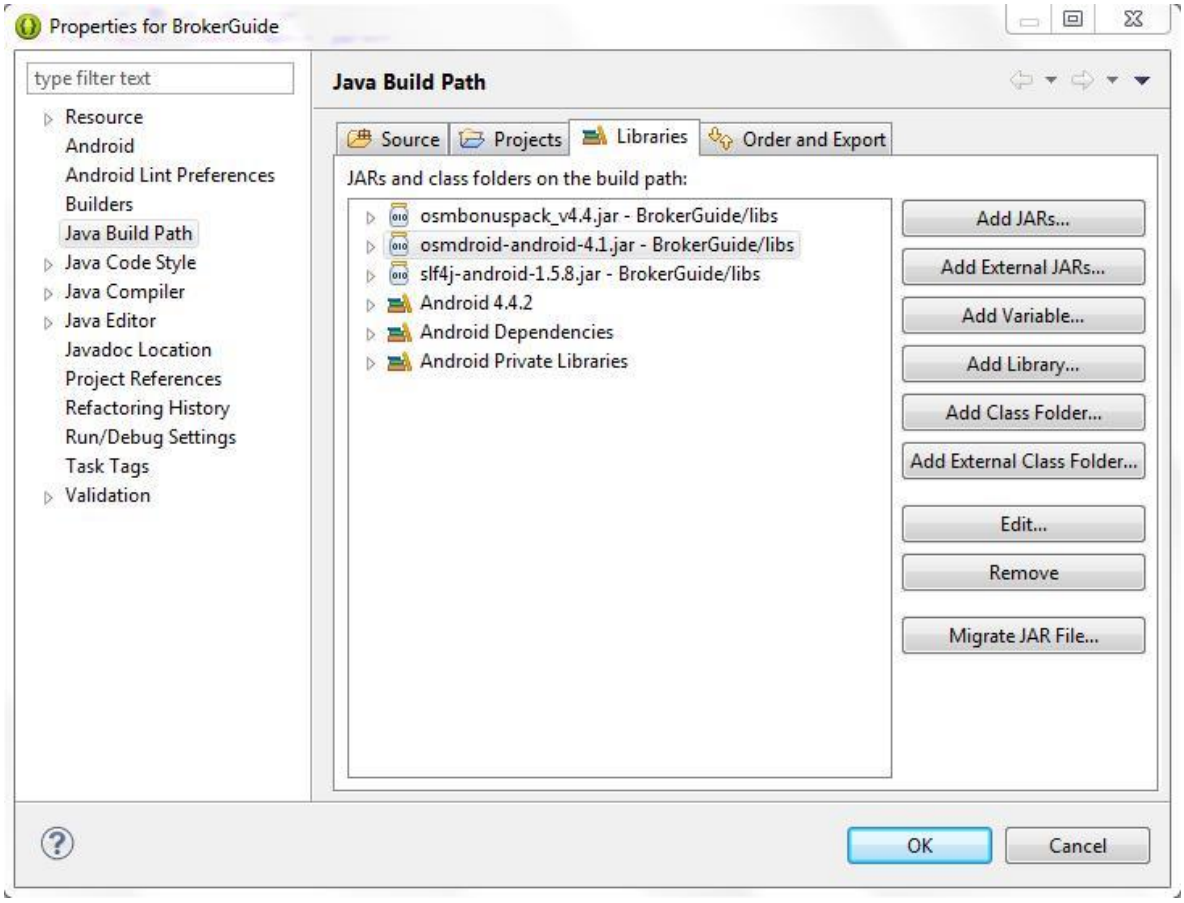


Figura 15 - Java Build Path

Oltre a questo vanno aggiunti al file AndroidManifest.xml i seguenti permessi:

```
<uses-sdk android:targetSdkVersion="16" android:minSdkVersion="7" />

<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission    android:name="android.permission.ACCESS_WIFI_STATE"
/>
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

A questo punto creando un'activity è possibile all'interno del metodo onCreate aggiungere la mappa

```
MapView map = (MapView) findViewById(R.id.map);
map.setTileSource(TileSourceFactory.MAPNIK);
map.setBuiltInZoomControls(true);
map.setMultiTouchControls(true);
IMapController mapController = map.getController();
mapController.setZoom(18);
```

Infine nel file contenente la struttura dell'interfaccia, solitamente activity_main.xml, va aggiunto l'oggetto MapView

```
<org.osmdroid.views.MapView
        android:id="@+id/map"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
/>
```

Con questi semplici passaggi è possibile configurare una mappa consultabile in tutta comodità dal proprio dispositivo mobile. Inoltre può essere sfruttata come base di partenza per uno sviluppo più avanzato come vedremo nei capitoli implementativi.
Segue una illustrazione sulla grafica di Osmdroid.

Figura 16 - OpenStreetMap Viewer

2.4.2 OSMBONUSPACK

OSMBonusPack è una libreria nata per estendere le funzionalità di Osmdroid.
Ricordiamo che Osmdroid è una libreria per interagire con i dati di OpenStreetMap all'interno di un'applicazione Android, ed offre oggetti per la creazione e manipolazione di mappe come MapView, MapController, Overlay e tanti altri.
La librea Osmbonuspack va invece ad aggiungersi alle funzionalità di base di Osmdroid, estendendone le funzionalità.
Questa libreria integrata ad Osmdroid, aggiunge diverse classi interessanti, tra cui:

- Marker: permette di creare degli oggetti marcatori;
- Routes e Directions: per percorsi e orientazioni;

- Points of Interests: per i punti di interesse;
- Marker Clustering: per il raggruppamento di marcatori;
- Polyline: per la realizzazione di polilinee e poligoni;
- Supporto per file KML e contenuto GeoJSON;
- Geocoding e Reverse Geocoding;
- Integrazione di un Cache Management Tool per le mappe off-line

La sua integrazione ed utilizzo all'interno di un progetto, seguono le indicazioni viste per la libreria Osmdroid: va agginto il file jar nel build path del progetto come visto in precedenza.
Seguono alcune illustrazioni sugli oggetti grafici offerti dalla libreria.

Figura 17 - OSMBonusPack Demo 1

Figura 18 - OSMBonusPack Demo 2

2.4.3 JAVAAPIFORKML

[25] L'obiettivo principale delle API Java per KML (JAK) è quello di fornire un'implementazione che permetta di generare automaticamente oggetti kml da file strutturati dallo standard kml.
Si tratta di una API orientata agli oggetti che semplifica e permette l'uso conveniente e facile di [26] KML in ambienti Java.
Con questa libreria è possibile, ad esempio, creare un file kml semplicemente con poche righe di codice

```
final Kml kml = new Kml();
kml.createAndSetPlacemark()
   .withName("London, UK").withOpen(Boolean.TRUE)
   .createAndSetPoint().addToCoordinates(-0.126236, 51.500152);
kml.marshal(new File("HelloKml.kml"));
```

Figura 19 - Marshalling

In questo codice viene creato un oggetto kml e si definiscono successivamente una etichetta Londo e le coordinate associate.
Infine si salva il tutto in un file chiamato “HelloKml.kml” tramite la funzione marshalling.
Un'altra funzionalità è quella del processo inverso. Ad esempio, se abbiamo un file kml già creato

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:atom="http://www.w3.org
    <Placemark>
        <name>London, UK</name>
        <open>true</open>
        <Point>
            <altitudeMode>clampToGround</altitudeMode>
            <coordinates>-0.126236,51.500152</coordinates>
        </Point>
    </Placemark>
</kml>
```

Figura 20 - File kml

Lo possiamo caricare in memoria costruendo l’oggetto corrispondente tramite il seguente codice

```
final Kml kml = Kml.unmarshal(new File("HelloKml.kml"));
final Placemark placemark = (Placemark) kml.getFeature();
Point point = (Point) placemark.getGeometry();
List<Coordinate> coordinates = point.getCoordinates();
for (Coordinate coordinate : coordinates) {
    System.out.println(coordinate.getLatitude());
    System.out.println(coordinate.getLongitude());
    System.out.println(coordinate.getAltitude());
}
```

Figura 21 - Unmarshalling

Il caricamento in memoria, in quest’ultimo caso, si chiama unmarshalling e permette di trasformare la struttura ed il contenuto del file kml nel suo oggetto equivalente.

2.5 *NOTIFICHE*

[27] Una notifica push è un breve messaggio asincrono inviato da un server ad una specifica applicazione installata nel dispositivo finale. Questi messaggi sono ampiamente usati dalle case produttrici per informare l'utente che è disponibile una nuova versione del sistema operativo o che è disponibile un aggiornamento di un'applicazione. Sviluppatori di terze parti potrebbero utilizzare queste notifiche per informare l'utente che è disponibile un aggiornamento nei contenuti dell'applicazione o un evento relativo all'applicazione. Per esempio, le push notification possono essere utilizzate in un applicazione per informare l'utente dell'andamento delle proprie azioni nel mercato azionario o il cambiamento del risultato in un match sportivo.

Le notifiche push rappresentato un aspetto innovativo e rivoluzionario per le applicazioni che sono in grado di supportarle; e sono ampiamente utilizzate in tutti quei contesti in cui sia richiesto l'invio o la comunicazione di determinate informazioni agli utenti di riferimento.

Solitamente, nei contesti di Information Technology, ci si riferisce alle notifiche push come a richieste che vengono inizializzate direttamente dai server o dai publisher: diversamente da come avviene nei contesti più comuni in cui sono i client ad inizializzare la connessione.

Le notifiche push, oltre agli esempi già menzionati, sono impiegate per diversi scopi tra cui il publish-subscribe che prevede di fornire aggiornamenti tramite la sottoscrizione degli eventuali client ai topic d'interesse: per fare in modo che si possano ricevere le sole comunicazioni associate all'argomento prescelto. Il synchronous conferencing, utilizzato per le comunicazioni tra più utenti: come ad esempio i canali online; od ancora l'Istant Messaging impiegato per le ormai conosciutissime chat o programmi di messaggistica. Generalmente vengono utilizzate per fare in modo che l'utente sia informato in caso di attività importanti.

Le tecniche utilizzate per realizzare le notifiche push spaziano dal LONG POLLING (BOSH), all'SSE (Server Sent Event), ai WEBSOCKET.

Il concetto che raccoglie queste tecnologie può essere riassunto in Comet, che rappresenta una famiglia di tecniche impiegate per questo obiettivo.

Entrando in una trattazione più dettagliata, nell'architettura pull method tradizionale, il controllo è gestito dal client: la comunicazione viene inizializzata su richiesta di quest'ultimo e termina quando riceve la risposta.
Con il progredire della tecnologia, e delle aspettative degli utenti, il modello pull ha iniziato a non essere più sufficiente nel soddisfare i bisogni dell'utenza.
Pensiamo a quelle applicazioni web che offrono dei meccanismi di comunicazione in tempo reale tra gli utenti, o più in generale a tutti i mezzi di interazione disponibili: per far fronte a queste esigenze, hanno infatti bisogno di un'architettura client-server con Server Push.
[28] Un server di tipo push, offre la possibilità di poter inviare dati al client appena sono disponibili, senza dover attendere che vengano richiesti.
Un attuale ingegnere Google, Alex Russel, nell'articolo "Comet: Low Latency Data for the Browser" ha coniato il termine Comet per descrivere l'architettura di un'applicazione che utilizza "long-held HTTP requests" (ovvero richieste HTTP mantenute incomplete per lungo tempo) per permettere ad un web server di poter inviare dati ad un client in maniera asincrona. Questo tipo di comunicazione permette quindi una comunicazione orientata agli eventi, che possono essere trasmessi al client, nel momento stesso in cui sono generati lato server. La comunicazione Comet-oriented viene implementata utilizzando tecniche di complessità e natura diverse, alcune nate come "hack", sfruttando cioè caratteristiche particolari di strumenti già esistenti.

2.5.1 TECNOLOGIE

Le tecnologie prese in esame fanno riferimento ai modelli impiegati per le notifiche push.
Sebbene l'obiettivo comune tra queste tecnologie sia l'invio dei messaggi ai propri destinatari, ci sono differenze sostanziali che ne determinano una netta distinzione.
Quello che contraddistingue tali tecnologie, sono la tecnica impiegata e l'architettura utilizzata nei diversi approcci.

Tra i principali modelli troviamo:

- Socket
- Bosh
- WebSocket
- Sse

2.5.1.1 SOCKET

[29] In informatica ci si riferisce ai socket come ad un'astrazione software realizzata per poter utilizzare interfacce standard per la trasmissione e ricezione di dati attraverso una rete, oppure come sistema per la comunicazione tra processi (Inter-Process Communication).
Il socket può essere visto come il punto in cui un processo accede al canale di comunicazione tramite una porta, ottenendo una comunicazione tra processi che lavorano su macchine fisicamente separate.
Dal punto di vista di un programmatore, un socket rappresenta un oggetto grazie al quale si può leggere e scrivere dati che devono essere trasmessi o ricevuti.
I tipi fondamentali di socket sono due:

- i socket su protocollo IP, usati per le comunicazioni tramite protocollo di trasporto TCP o UDP;
- gli Unix domain socket (noti anche come socket locali), usati nei sistemi operativi POSIX per comunicazioni tra processi residenti sullo stesso computer.

I socket su protocollo IP permettono comunicazioni tra processi remoti grazie ai quali è possibile instaurare canali di trasmissione bidirezionali.
Più precisamente, un socket rappresenta un’area di memoria che permette attraverso interfacce specifiche di potervi accedere in lettura e scrittura; questo rende possibile da un lato ricevere le informazioni che arrivano dai processi

remoti e dall'altro inviare le informazioni a tali processi: in realtà un socket corrisponde ad una porta tra il processo di un'applicazione ed il protocollo di trasporto TCP, dove quest'ultimo ha il ruolo di inoltrare il messaggio ai livelli sottostanti per farlo recapitare al processo destinatario identificato dalla coppia IP:PORTA.

La flessibilità dei socket li rende particolarmente elastici sia sul tipo di connessione che si vuole instaurare, sia sul tipo di stile di comunicazione: è possibile ad esempio effettuare comunicazioni affidabili oppure non affidabili, scegliere il tipo di modalità d'invio come flusso di byte o di caratteri e tante altre caratteristiche che ne fanno un vero e proprio modello.

L'architettura di una comunicazione socket tra client e server è la seguente:

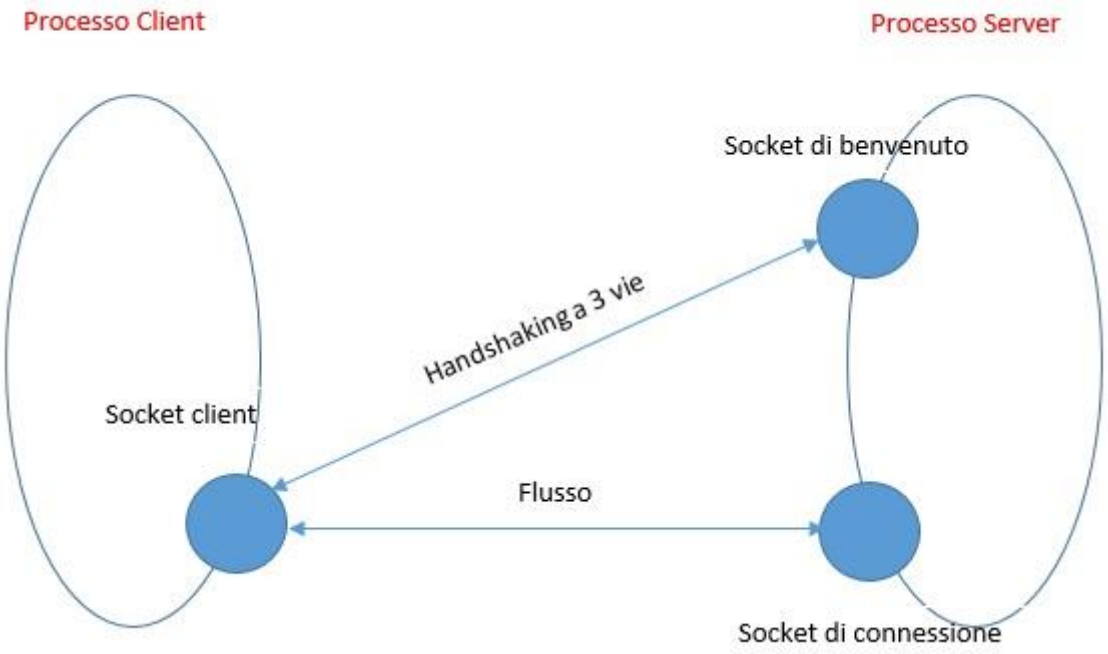


Figura 22 - Flusso Socket

2.5.1.2 BOSH

Bosh è l'acronimo di Bidirectional streams Over Synchronous HTTP, ed è un protocollo di trasporto che permette di instaurare una connessione bidirezionale

tra due entità, come ad esempio un client ed un server, usando chiamate request/response HTTP sincrone: senza effettuare polling o richieste asincrone.

Il funzionamento adottato dal modello BOSH per le notifiche push è il seguente: il client avvia una richiesta HTTP al server con cui vuole comunicare. Il server, ricevuta la richiesta, non restituisce immediatamente la risposta al client se non ha informazioni da fornirgli, ma blocca l'invio della risposta fino a quando non ha dati da inviargli. Dopo aver ricevuto una risposta, il client effettua immediatamente un'altra richiesta HTTP, in modo che il server possa sempre inviargli i dati nell'immediata disponibilità. Se durante l'attesa di una risposta il cliente deve inviare i dati al server, si apre una seconda connessione HTTP.

Nel modello BOSH ci possono essere al massimo due connessioni HTTP attive nello stesso momento: una su cui il server può inviare dati al client come risposta ed una su cui il cliente può inviare dati al server.

Tale modello è comunemente noto come Long Polling.

L'architettura è rappresentata dal seguente diagramma di sequenza:

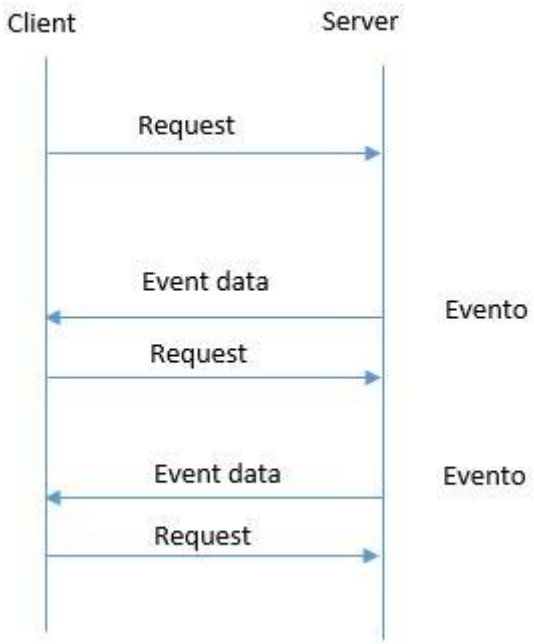


Figura 23 - Flusso Bosh

2.5.1.3 WEBSOCKET

[30] Il WebSocket è una tecnologia che fornisce canali di comunicazione full duplex su una singola connessione TCP.
E' stato standardizzato dal W3C come parte integrande dell'HTML5.
Il protocollo WebSocket è basato su TCP. Il suo unico rapporto col protocollo applicativo HTTP è che la sua fase iniziale di negoziazione di connessione (handshake), richiede una request upgrade al server: grazie alla quale viene poi instaurata una connessione TCP.
Tale modello rende possibile una maggior interazione tra un browser ed un server, facilitando il contenuto dal vivo e la creazione di giochi real-time.
Questo metodo fornisce al server la possibilità di inviare il contenuto al browser senza essere sollecitato dal client, e consentendo al contempo di ricevere messaggi: mantenendo la connessione aperta.
Inoltre, le comunicazioni sono effettuate sulla porta 80 che è a garanzia per tutti quegli ambienti che bloccano le connessioni utilizzando firewall o proxy. Il protocollo WebSocket è attualmente supportato da diversi browser, tra cui Google Chrome, Internet Explorer, Firefox, Safari ed Opera.
Il WebSocket, per la sua gestione, richiede applicazioni specifiche server side che possano implementarlo.
Tra queste implementazioni, le principali individuate sono:

- Apache Tomcat dalla versione 7.0.27
- Jetty Web Server
- Node.js
- Nginx
- Netty
- Implementazioni create ad hoc tramite PHP, od altri linguaggi lato server, che implementano il protocollo WebSocket e librerie lato client o browser con interpreti che li supportano.

La sua architettura può essere espressa dal seguente schema:

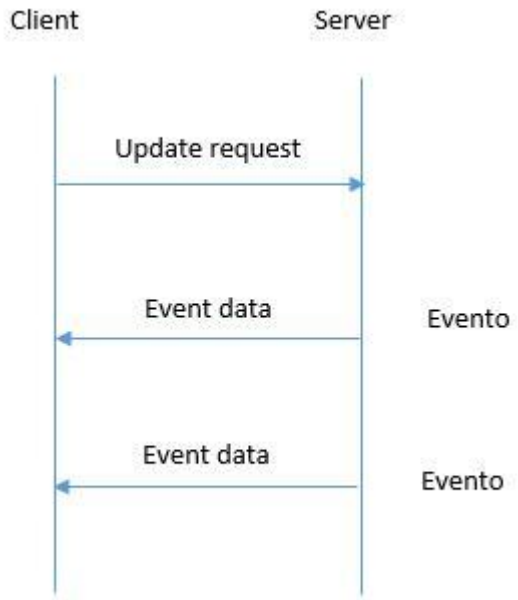


Figura 24 - Flusso Websocket

[31] Entrando più nel dettaglio, quello che avviene in una sessione di connessione è espresso da due step, quali: una prima fase chiamata handshake, ed una seconda fase per il trasferimento.
L'handshake del client è una normale richiesta http così composta:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Figura 25 - Richiesta Websocket

- l'URI della richiesta è l'endpoint della connessione WebSocket
- Connection:Upgrade richiede di modificare l'attuale connessione
- Upgrade:websocket indica il nuovo protocollo
- Origin: indica al server da quale client proviene la richiesta ed è obbligatorio se il client è un browser

- Sec-WebSocket-Version è la versione richiesta dal protocollo; la prima versione definitiva è la numero 13
- Sec-WebSocket-Key che è un valore pseudocasuale di 16 byte codificato in base64

Il client nella richiesta può specificare altre opzioni,

- Sec-WebSocket-Protocol: elenco di sub-protocolli di livello applicativo che il client vuole usare
- Sec-WebSocket-Extensions: insieme di estensioni di WebSocket supportate dal client
- Cookie
- Autenticazione

Mentre la struttura della risposta si presenta come segue:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

Figura 26 - Risposta Websocket

Per essere certi che l'handshake sia completato, lo status code dev'essere 101.
Inoltre, gli header field Upgrade e Connection devono essere presenti; mentre Sec-WebSocket-Accept serve a confermare l'accettazione della richiesta.
In seguito, nella fase di trasferimento dati, il WebSocket crea un socket TCP nell'infrastruttura applicativa del WWW grazie a cui può aprire un canale tra il client ed il server e poter comunicare.

2.5.1.4 SSE

[32] Il Server sent events (SSE) è una tecnologia grazia a cui un browser può ricevere automaticamente gli aggiornamenti da un server attraverso una

connessione HTTP. Il Server-Sent Events è stato standardizzato come parte del protocollo [33] HTML5 dal Consorzio [34] W3C.

Di fatto, il Server-sent event è uno standard che descrive come i server possono avviare la trasmissione dei dati verso i client una volta che è stata stabilita una connessione iniziale.

Sono comunemente utilizzati per inviare messaggi di aggiornamento o flussi di dati continui da un server ad un client: attraverso comunicazioni unidirezionali; questo significa che è concesso solo ed esclusivamente al server l'inoltro dei messaggi.

I messaggi comunicati dal modello Server-sent event avvengono in tempo reale: questo approccio è simile ai WebSocket, in quanto anche quest'ultimi garantiscono comunicazioni real time, ma con la differenza che la comunicazione non è solamente one-way.

Ad ogni modo, il Server-sent event è il modello che più si abbina al publish-subscribe.

Entrando più nel dettaglio, questa tecnica utilizza l'oggetto Javascript EventSource che mette a disposizione un insieme di API che forniscono la possibilità ad un server di inviare eventi ad un client.

Attraverso questo oggetto, il client può richiedere al server un particolare URL allo scopo di ricevere gli eventi associati.

Per ogni istanza di EventSource si deve specificare l'url della risorsa web verso la quale effettuare la richiesta (per stabilire il canale di comunicazione): la connessione aperta verso l'indirizzo specificato attraversa 3 stati:

- CONNECTING (valore numerico 0), stato nel quale si trova la connessione al momento della creazione dell'oggetto: ovvero, la connessione non è ancora stata stabilita.

- OPEN (valore numerico 1), la connessione è stata stabilita e il client è in attesa di un evento.

- CLOSED (valore numerico 2), la connessioneè chiusa e il browser non tenta di riconnettersi; la connessione può assumere questo stato sia a causa di un errore fatale, sia conseguentemente all'invocazione del metodo close.

Quando lo stato assunto della connessione è OPEN, il client può ricevere un evento; per ciascuno tipo di evento, viene associato un metodo handler che definiscono il comportamento da tenere quando l'evento corrispondente viene generato:

- open, tipo di evento generato all'apertura della connessione; il gestore dell'oggetto Javascript associato a tale evento è onOpen.
- message, tipo di evento generato all'arrivo di un messaggio dal server; il gestore dell'oggetto Javascript associato a tale evento è onMessage.
- error, tipo di evento generato per comunicare il verificarsi di un errore; il gestore dell'oggetto Javascript associato a tale eventoè onError.

Al momento della ricezione di un evento, viene invocato il metodo Javascript onMessage(event); a quel punto i dati inviati sono disponibili nel campo data dell'oggetto event. Questa tecnica prevede inoltre che il contenuto di ogni evento inviato dal server sia della forma:

- data: [stringa del messaggio]

La tecnica Server-Sent Event al momento è supportata nativamente solo da recenti versioni dei browser Chrome e Safari: essa infatti è una tecnica nata solo recentemente, e specifica per la gestione di eventi in implementazioni server push.
Un vantaggio dell'SSE è che lavora su protocollo HTTP.
Inoltre, questa tecnica, presenta diverse caratteristiche che altre tecniche non hanno: come la riconnessione automatica in caso di disconnessione, l'event ids (possibilità di registrarsi ad una pagina specifica), e la possibilità di inviare eventi arbitrari.
Le comunicazioni richiedono nella request l'Accept header settato a test/event-stream e nella response il Content-type header settato a test/event-stream.

```
=> Request
GET /stream HTTP/1.1 (1)
Host: example.com
Accept: text/event-stream

<= Response
HTTP/1.1 200 OK (2)
Connection: keep-alive
Content-Type: text/event-stream
Transfer-Encoding: chunked
```

Figura 27 - Richiesta Risposta Sse

Uno schema riassuntivo del suo funzionamento può esser dato dalla seguente illustrazione.

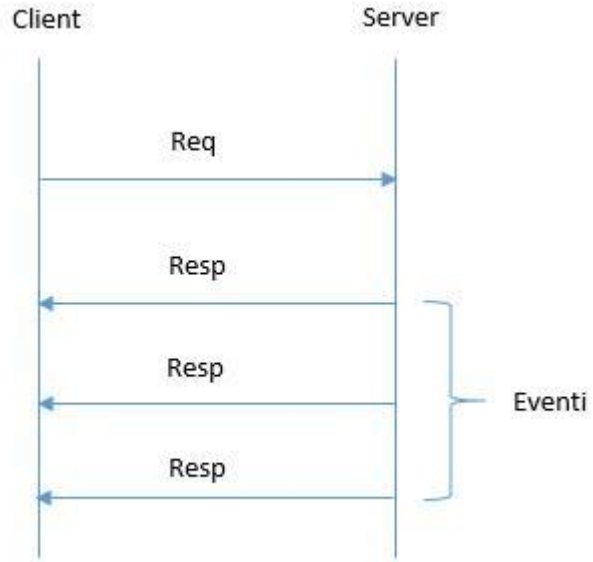


Figura 28 - Flusso Sse

2.5.2 PUBLISH-SUBSCRIBE

[35] Il publish/subscribe (o publisher/subscriber) è un modello di messaggistica impiegato per la comunicazione asincrona fra diversi processi, oggetti o altri agenti.

In questo modello, i mittenti ed i destinatari comunicano attraverso un tramite, detto broker. Il mittente di un messaggio (publisher) non deve essere a

conoscenza dell'identità dei destinatari (subscriber); deve invece pubblicare il proprio messaggio al broker. I destinatari si rivolgono a loro volta al broker abbonandosi (subscribe) alla ricezione dei messaggi relativi al topic di interesse prescelto dagli stessi. Il broker quindi inoltra ogni messaggio inviato da un publisher a tutti i subscriber interessati a quel messaggio.

In genere, il meccanismo di sottoscrizione consente ai subscriber di specificare a quali messaggi sono interessati e ricevere così le sole notifiche relative agli argomenti prescelti.

Questo schema implica che ai publisher non sia noto quanti e quali sono i subscriber e viceversa. Questo può contribuire alla scalabilità del sistema.

E' quindi un sistema in cui non c'è un'interazione diretta tra le parti interagenti, ma la comunicazione richiede l'ausilio di un intermediario: identificato dal broker, che riveste un ruolo cruciale garantendo e dispensando il publisher sia sulla conoscenza degli abbonati, sia sull'invio dei messaggi; poiché è il broker a dover gestire le sottoscrizioni e le comunicazioni inoltrategli dal publisher.

Segue una sua possibile architettura:

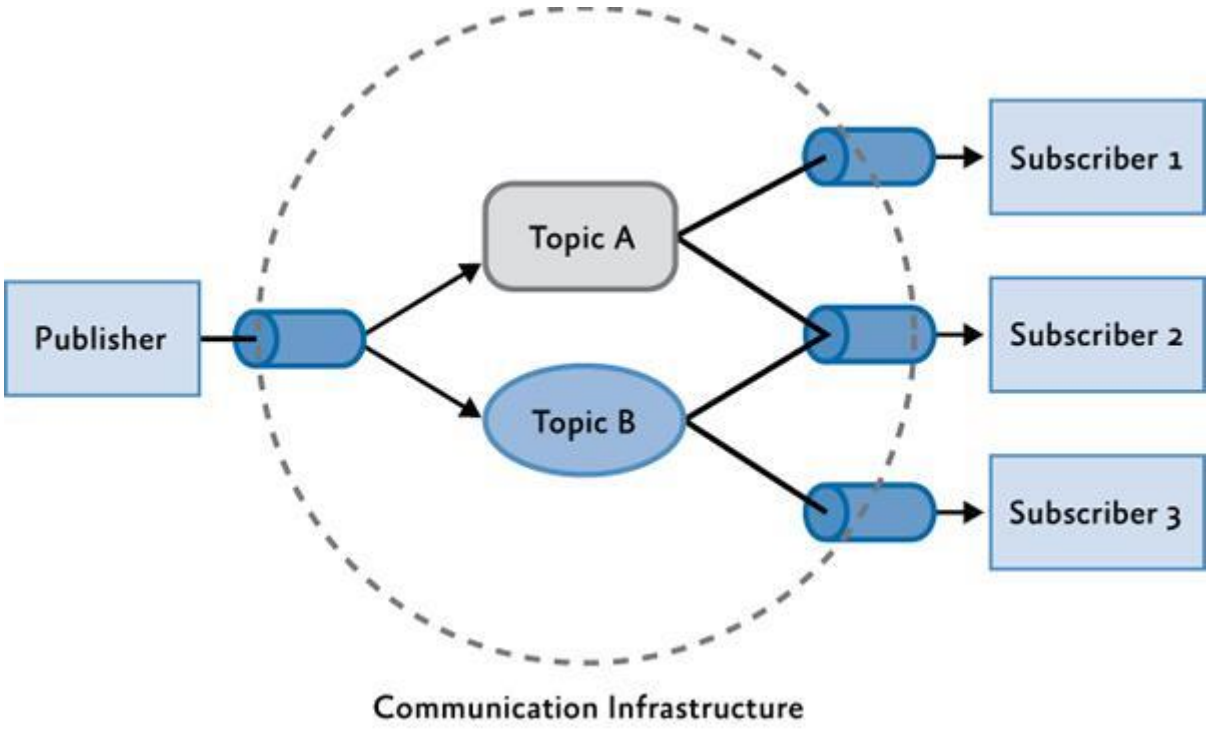


Figura 29 - Architettura Publish - Subscribe

2.5.3 XMPP

[36] L'XMPP (Extensible Messaging and Presence Protocol) è un protocollo di comunicazione basato su XML (Extensible Markup Language).
Il protocollo è stato sviluppato per la realizzazione di messaggi in tempo reale e messaggistica istantanea.
Progettato per essere estensibile, il protocollo è stato utilizzato per diversi scopi, tra cui il publish-subscribe, sistemi di segnalamento per il VoIP, video, trasferimento di file, giochi, Internet of Things (IoT), ed applicazioni come la smart grid e servizi di social networking.
A differenza di molti protocolli di instant messaging, l'XMPP è definito uno standard aperto: le implementazioni possono essere sviluppate utilizzando qualsiasi licenza software; anche se molte implementazioni di server, client, e librerie sono distribuite come software open-source, esistono anche numerose implementazioni software freeware e commerciali.
Il protocollo di trasporto originale e nativo per l'XMPP è il TCP (Transmission Control Protocol), che utilizza flussi XML aperti su connessioni TCP.
L'architettura XMPP è basata sul modello client-server.
[37] In XMPP ogni entità in rete è indirizzabile in maniera univoca attraverso un indirizzo chiamato JID che può essere di due tipi:

- BARE: node@domain, identifica solamente il nodo;
- FULL: node@domain/resource, identifica un'entità all'interno di un nodo

La struttura del messaggio è composta da un contenitore che dispone di un tag di apertura ed uno di chiusura <stream></stream>.

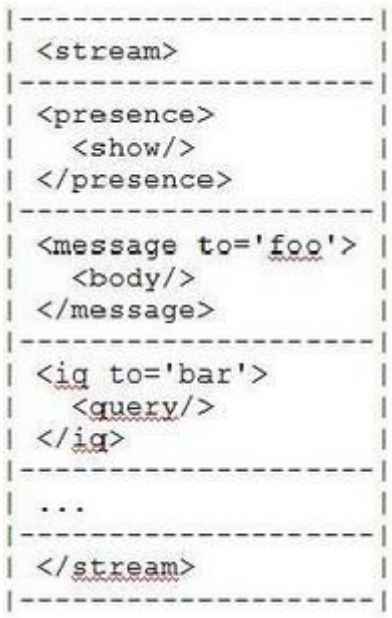


Figura 30 - Messaggio XMPP

All'interno di questo messaggio è possibile trasmettere diversi elementi XML, tra i quali:

- XML Stanza
- Elementi di negoziazione TLS e/o SASL

L'XML Stanza è un blocco di informazioni strutturate che viene spedito da un'entità all'altra all'interno dell'XML Stream.
L'XML Stanza è identificata da un tag di apertura ed uno di chiusura.
Ci sono diversi tipi di XML Stanza che si differenziabo a seconda dei messaggi che si vogliono inviare; di default il core XMPP prevede:

- <message/>: meccanismo push tramite il quale una entità invia informazioni a un'altra.
- <presence/>: un messaggio publish-subscribe per trasmettere la propria disonibilità in rete (se ad esempio online od off line).
- <iq/>: per info o query, è un messaggio request/response simile ad http che permette a due entità di scambiarsi informazioni. La comunicazione viene riconosciuta grazie l'identificatore iq che identifica i messaggi.

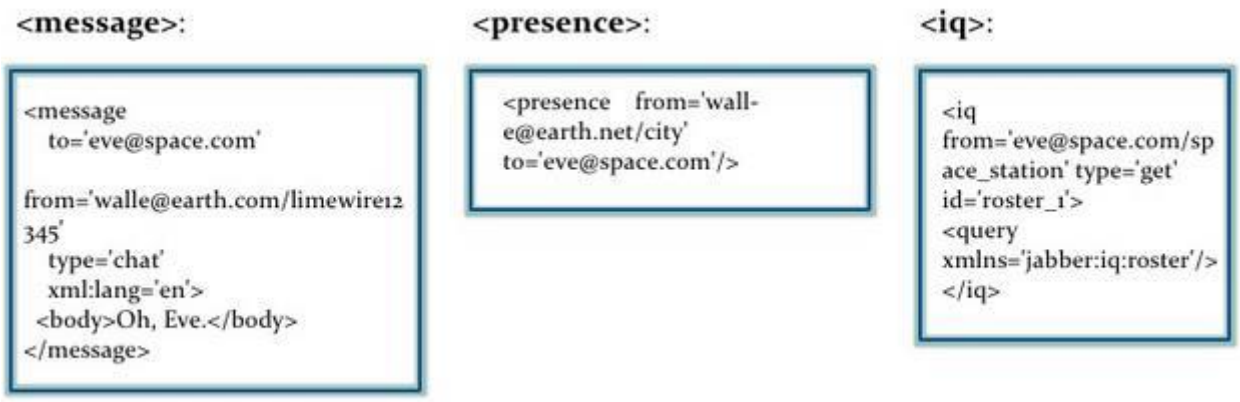


Figura 31 - Tipologia Messaggi

2.5.4 MQTT

[38] MQTT è un protocollo di messaggistica leggero ampiamente utilizzato per il publish-subscribe. È progettato per connessioni remote in cui è richiesta un ridotta quantità di dati in trasmissione: per problemi di banda o ti traffico limitato.
E' stato progettato per essere aperto, semplice, leggero e facile da implementare.
Utile in tutti quei contesti in cui la rete ha una bassa capacità di banda od una scarsa affidabilità; oppure per tutti quei dispositivi o sistemi con capacità elaborative o di memorizzazione limitate.
Anche l'MQTT, come tanti altri protocolli, si appoggia al protocollo TCP.
[39] Tra le caratteristiche principali del protocollo, troviamo:

- il pattern publish-subscribe per la notifica di messaggi one-to-many;
- trasporto di messaggi con basso contenuto di payload;
- l'uso del protocollo TCP/IP come base di connettività di rete;
- tre tipologie di qualità del servizio (QoS) per la consegna dei messaggi:
    - al più una volta: il messaggio può anche non arrivare.
      Ad esempio, un sensore ambientale che invia dati con una certa regolarità; se ad un certo punto un messaggio di rilevamento non arriva, non ha importanza se il messaggio successivo arriva subito dopo andando a rimpiazzare il valore precedente;

  - almeno una volta, in cui l'arrivo dei messaggi è assicurato, ma potrebbero esserci duplicati;
  - esattamente una volta, dove il messaggio viene assicurato per arrivare esattamente una ed una sola volta. Questo livello potrebbe essere utilizzato, ad esempio, con i sistemi di fatturazione dove però la duplicazione o perdita di tali dati potrebbe portare ad applicazioni di oneri errati;
- un basso overhead a causa delle ridotte dimensioni per ridurre al minimo il traffico di rete;
- un meccanismo di notifica alle parti interessate di una disconnessione anomala di un client, utilzzando la funzione Last Will and Testament

Il formato del messaggio è composto da tre parti principali:

- Fixed header: contiene informazioni riguardo il tipo di messaggio, il livello di qualità del servizio, alcuni flag e la lunghezza del messaggio

bit	7	6	5	4	3	2	1	0
byte 1	Message Type				DUP flag	QoS level		RETAIN
byte 2	Remaining Length							

Figura 32 – Fixed Header MQTT

- Variable header: alcuni tipi di messaggi MQTT possono contenere un componente di intestazione variabile che può risiedere tra il Fixed header ed il payload del messaggio.

- Payload: contiene l'informazione utile del messaggio.

Segue un'esposizione di una sua possibile architettura.

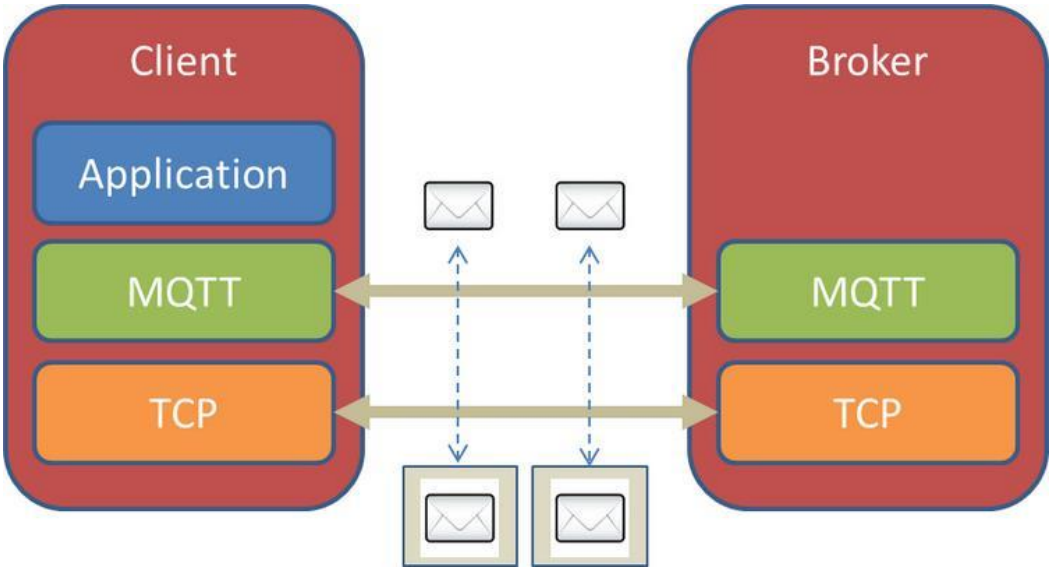


Figura 33 - Architettura MQTT

2.5.5 GCM

Google Cloud Messaging (GCM) è un servizio gratuito messo a disposizione da Google che aiuta gli sviluppatori a realizzare applicazioni per l'invio di dati dai server ai dispositivi Android.

I messaggi comunicati dal server ai dispositivi, potrebbero riguardare avvisi di aggiornamento disponibili per il download (ad esempio, un messaggio multimediale) così come informazioni disponibili alla consultazione (come ad esempio, una notifica di una nuova e-mail) oppure messaggi con contenuti informativi diretti con un payload massimo di 4Kb.

Inoltre, il servizio GCM gestisce ogni aspetto riguardante l'accodamento dei messaggi e la loro consegna alle applicazioni Android; permette attraverso configurazioni specifiche di poter gestire l'accodamento dei messaggi con flessibilità: è possibile ad esempio poter scegliere il periodo massimo entro cui notificare il messaggio in caso di mancata consegna; così come collassare le notifiche accorpandole in un'unica notifica in modo di sovrascrivere i messaggi vecchi coi nuovi a condizione che portino la stessa intestazione.

Infine, la consegna del messaggio dal GCM ai dispositivi è diretta e prescinde dallo stato in cui si trova il dispositivo (background, foreground o shutdown).

2.5.5.1 CARATTERISTICHE

- Permette l'invio di messaggi ai dispositivi Android da server di terze parti.
- Non fornisce garanzie sull'ordine o la consegna dei messaggi.
- L'applicazione Android sul dispositivo può anche non essere in esecuzione per ricevere i messaggi; sarà il sistema a svegliarla attraverso un componente chiamato BroadcastReceiver che permette di ricevere avvisi direttamente dal sistema.
- Non controlla i messaggi ma si limita ad inoltrarli verso i dispositivi.
- E' richiesto Android 2.3 o superiore e Google Play Store installato o un simulatore che supporti la versione 2.2 di Android con le Google APIs installate.
- Per le versioni di Android superiori alla 4.04 non è richiesto un Account Google per il suo utilizzo (cosa necessaria per le versioni precedenti), ma è sufficiente che sia installato il solo play store.

2.5.5.2 ARCHITETTURA

In questo paragrafo vengono descritti gli elementi coinvolti nel processo di inoltro di un messaggio.
Questi elementi si dividono in due categorie:

Componenti: entità fisiche che svolgono ruoli specifici nel modello GCM.

- Mobile Device: dispositivo mobile con applicazione Android configurata per comunicare col GCM.

- Application Server: un'applicazione server che invia i dati ai dispositivi tramite GCM.
- GCM Server: server di Google che inoltre i messaggi ricevuti dall'Application Server verso i dispositivi.

Credenziali: identificativi e token coinvolti nelle diverse fasi per garantire che tutte le parti siano state autenticate e che il messaggio è nella posizione corretta.

- Sender ID: identificativo alfanumerico rilasciato dalla APIs console di Google (chiamato Project ID) che abilita il servizio GCM per una determinata applicazione.
- Application ID: un identificativo univoco dell'applicazione, viene creato in base al nome dell'applicazione stessa dal sistema operativo in modo di garantire che i messaggi siano inoltrati all'applicazione corretta.
- Registration ID: un identificativo rilasciato dal server GCM all'applicazione android che permette di ricevere i messaggi. L'applicazione una volta che ha ricevuto l'identificativo dal GCM, lo invia al Server Provider che lo utilizza per identificare l'app e per spedirgli i messaggi tramite il GCM.
- Google User Account: perché il GCM lavori, il dispositivo mobile deve includere almeno un account Google se utilizza una versione di Android inferiore alla 4.04.
- Sender Auth Token: un'API key che viene salvata sul server provider che permette di avere accesso al servizio di Google. L'API key è inclusa nell'intestazione delle richieste POST che vengono inviate al GCM.

2.5.5.3 FLUSSO CICLO DI VITA

Segue una rappresentazione grafica del flusso sul ciclo di vita del servizio GCM.

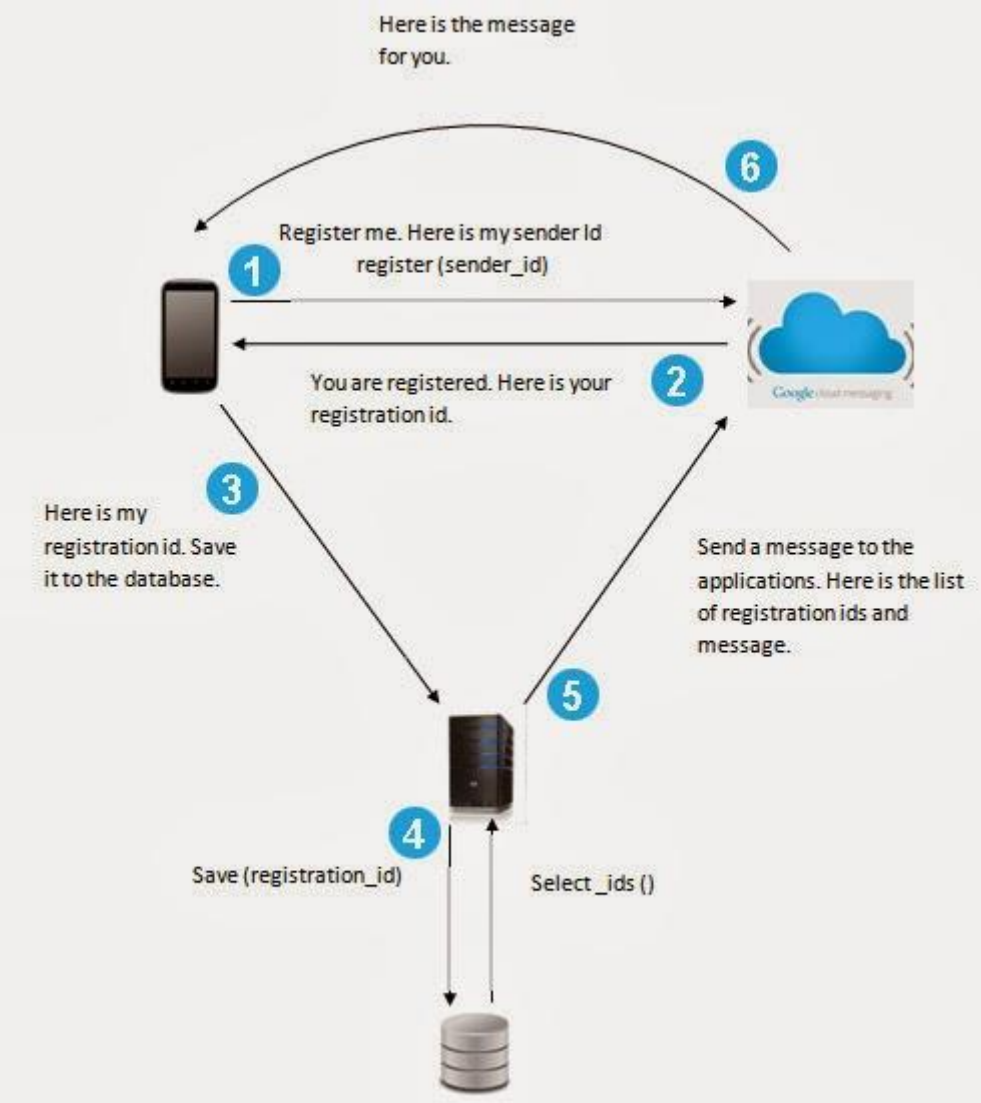


Figura 34 - Architettura GCM

1) L'applicazione Android richiede al GCM di registrarsi e gli invia il Sender ID ottenendo di conseguenza la Registration ID.
2) Il GCM riceve dall'applicazione il Sender ID e restituisce la Registration ID.
3) L'applicazione invia la Registration ID all'Application Server.
4) L'Application Server, memorizza la Registration ID nel database.
5) Quando è richiesto l'invio di un messaggio, l'Application Server preleva dal database la Registration ID e l'invia al server GCM insieme al messaggio.
6) Il GCM Server invia il messaggio all'applicatione Android.

2.5.5.4 INVIO NOTIFICA

Per fare in modo che l'Application Server possa inviare correttamente una notifica al dispositivo, è importante che si sia definita (tramite la console di Google) l'API key che permetta di comunicare con il GCM Server.
Inoltre l'applicazione Android dev'essere stata registrata: ottenendo prima la Registration ID dal GCM Server ed inviando poi tale ID all'Application Server.
Seguono gli eventi che si verificano all'attivarsi di un invio da parte dell'Application Server:

1. L'Application Server invia un messaggio al GCM Server.
2. Il GCM Server salva la notifica se il dispositivo non è momentaneamente raggiungibile.
3. Quando il dispositivo ritorna raggiungibile, il GCM Server gli recapita la notifica.
4. Il sistema operativo Android recapita la notifica ricevuta dal server verso l'applicazione corretta identificata dal Registration ID. Questo passaggio avviene attraverso un intent broadcast che attiva l'applicazione opportuna. Da notare che l'Applicazione Android non necessita di essere in esecuzione per ricevere la notifica.
5. L'applicazione Android processa il messaggio ricevuto.

2.5.5.5 RICEZIONE NOTIFICA

Seguono gli eventi che si verificano quanto l'applicazione Android riceve una notifica.

1. Il sistema riceve il messaggio ed estrae i dati a coppie di chiave-valore dal payload del messaggio, se presenti.
2. Il sistema passa i dati grezzi all'applicazione tramite l'Intent com.google.android.c2dm.intent.RECEIVE

va notato che non avviene nessun controllo da parte del sistema sui dati passati.

3. L'Applicazione Android estrae i dati grezzi dall'Intent receiver ed elabora i dati.

2.5.5.6 COMUNICAZIONE GCM

La parte che segue riguarda i requisiti richiesti per comunicare con il GCM Server, come si svolge la comunicazione, il formato dei messaggi ed i parametri richiesti.

Requisiti

1) E' necessario attivare il servizio Google Cloud Messaging nella Google APIs console raggiungibile all'indirizzo https://code.google.com/apis/console/ e salvare l'API Key da inserire nella richiesta http diretta al GCM Server.
2) L'Application Server dev'essere in grado di instaurare una connessione HTTPS verso il GCM Server.
3) L'Application Server deve poter registrare permanentemente sia la Registration ID sia l'API Key.

L'Application Server per comunicare la notifica al GCM Server, invia un messaggio tramite il metodo POST del protocollo http alla risorsa seguente:

https://android.googleapis.com/gcm/send

Formato dei messaggi

Il messaggio di richiesta è formato da due parti: l'intestazione ed il corpo del messaggio; che sono rispettivamente l'HTTP header e l'HTTP body.

Il GCM Server per il corpo del messaggio accetta solamente due formati: JSON e PLAIN TEXT. Il formato JSON rispetto al PLAIN TEXT è più strutturale e personalizzabile.

L'intestazione deve contenere l'API key ed il formato scelto.

Esempio di intestazione:

```
Authorization: key=API KEY
Content-Type:    application/json (per dizionario JSON)
 application/x-www-form-urlencoded;charset=UTF-8 (per testo semplice)
```

Il corpo del messaggio deve contenere sia le Registration IDs a cui inviare il messaggio, sia il dizionario JSON contenente l'informazione da recapitare.

Inoltre, si possono specificare altri parametri (opzionali) esposti nelle figure seguenti.

Esempio di corpo del messaggio:

```
{ "collapse_key": "score_update",
 "time_to_live": 108,
 "delay_while_idle": true,
 "data": {
   "score": "4x8",
   "time": "15:16.2342"
 },
 "registration_ids":["4", "8", "15", "16", "23", "42"]
}
```

Seguono i parametri utilizzati per la comunicazione col GCM

Parametro	Descrizione
registration_ids	Rappresenta una vettore contenente l'elenco delle registration id associate

	ai dispositivi e registrate nel GCM. Deve contenere almeno uno e al massimo 1000 registration id.
collapse_key	Rappresenta una stringa arbitraria che identifica i messaggi che possono essere accorpati tra loro quando il dispositivo non è raggiungibile: in modo che venga recapitato solo l'ultimo messaggio.
data	Un dizionario JSON i cui campi rappresentano coppie chiave – valore. Il limite massimo del messaggio è di 4kb. I valori possono essere di qualsiasi tipologia, ma si consiglia l'uso di stringhe, poiché il GCM li converte in questo formato.
delay_while_idle	Indica che il messaggio non deve essere inviato al dispositivo finché questo non è attivo.
time_to_live	Specifica per quanto tempo il messaggio deve essere conservato nel GCM se il dispositivo non è in linea: di default è di 4 settimane.

Formato risposta

Se il GCM riceve ed elabora con successo una richiesta, restituisce il codice 200 del protocollo HTTP; mentre quando la richiesta viene respinta, la risposta contiene un codice relativo al tipo di causa che ha portato al fallimento della comunicazione (un codice come ad esempio 400, 401 o 503).
Esempio di risposta GCM:

```
{ "multicast_id": 216,
 "success": 3,
 "failure": 3,
 "canonical_ids": 1,
 "results": [
   { "message_id": "1:0408" },
   { "error": "Unavailable" },
   { "error": "InvalidRegistration" },
   { "message_id": "1:1516" },
   { "message_id": "1:2342", "registration_id": "32" },
   { "error": "NotRegistered"}
 ]
}
```

I possibili parametri restituiti nel messaggio dal GCM sono i seguenti

Parametro	Descrizione
multicast_id	Id univoco che identifica il messaggio multicast
success	Numero di messaggi elaborati senza errori
failure	Numero di messaggi che non possono essere elaborati
canonical_ids	Rappresenta l'id dell'ultima registrazione valida in caso di conflitti.
results	Matrice di oggetti che rappresenta lo stato dei messaggi elaborati, può contenere: - message_id: stringa che rappresenta il messaggio quando è stato elaborato

	correttamente. - registration_id: se presente significa che il server GCM ha elaborato il messaggio, ma ha un altro canonical_id registrato per tale dispositivo, il mittente deve sostituire gli ID per le richieste future; altrimenti le richieste successive potrebbero essere respinte. - error: stringa contenente l'errore che si è verificato nell'elaborazione del messaggio.

2.5.6 FILE KML

Il file kml (Keyhole Markup Language) è un linguaggio basato su XML creato per la gestione dei dati geospaziali.

KML è diventato uno standard aperto internazionale riconosciuto dall'Open Geospatial Consortium.

Il file Kml specifica un set di elementi (segnalibri geografici, immagini, poligoni, modelli 3D, descrizioni ed etichette testuali) da visualizzare su mappe come Google Earth, Maps, OpenStreetMap e tante altre che riconoscano il formato trattato.

Ogni locazione deve avere obbligatoriamente una longitudine ed una latitudine, oltre ad altri dati opzionali che possono rendere la visualizzazione più dettagliata e specifica: come l'inclinazione, inquadratura e quota del punto di vista che nel loro insieme forniscono una vista.

Segue un esempio di struttura kml.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.0">
<Placemark>
  <description>New York City</description>
  <name>New York City</name>
  <Point>
    <coordinates>40.714172,-74.006393,0</coordinates>
  </Point>
</Placemark>
</kml>
```

Figura 35 - Struttura Kml

[40] Dalla struttura del file si può notare che la radice è sempre kml, ma il suo contenuto può contenere più Placemark ed avere associato, per ciascuno di questi, un certo tipo di geometria: Point, LineString, LinearRing, Polygon, MultiGeometry o Model.

In questo caso si può notare come venga incluso il tag Point con i riferimenti obbligatori quali latitudine e longitudine richiesti dalle specifiche.

Capitolo 3

3.1 *ANALISI DEL PROBLEMA*

Il problema in questione riguarda la gestione di notifiche push da parte di sistemi centralizzati per l'invio di informazioni ai dispositivi mobili.
Le soluzioni individuate e messe a confronto in questa Tesi sono due: la prima attraverso l'uso di un servizio offerto da Google chiamato GCM, la seconda come implementazione di un sistema che impiega la tecnica BOSH (Long-Polling).
I servizi di notifiche push come Google Cloud Messaging mettono a disposizione sistemi in grado di comunicare coi propri dispositivi mobili.
Tuttavia la loro gestione è comunque limitata sia sul piano quantitativo, per esempio la dimensione massima dei messaggi od il numero massimo di dispositivi a cui poter inviare i messaggi, sia sugli interventi e personalizzazioni del servizio stesso: il GCM è visto come una scatola nera e non è possibile parametrizzarlo andando ad aggiungere strutture dati diverse da quelle proposte dal protocollo del servizio, per cui non è possibile ampliarlo od aggiungere e sviluppare altre funzionalità oltre a quelle offerte dal servizio.
Mentre per quanto riguarda il modello Bosh, così progettato, permette di avere diversi vantaggi rispetto ai sistemi GCM o similari.
Uno dei punti a favore di BOSH è la semplificazione dell'architettura incentrata non più su tre attori, come nel caso del GCM, ma su due entità che permettono di ridurre notevolmente le fasi di intermediazione: ne deriva una miglior gestione e manutenzione.
Inoltre, il sistema BOSH non si presenta come scatola nera; per cui è possibile intervenire aggiungendo od ampliando le funzionalità di base offerte dal modello iniziale. Oltre a questo, va precisato che essendo di natura aperta e trasparente, favorisce anche quei prerequisiti richiesti nel trattamento di informazioni private o sensibili in termini di sicurezza: in quanto uno dei principi della sicurezza

informatica asserisce che non si può considerare sicuro un sistema che non si conosce.

Infine, va considerato anche un aumento prestazionale sull'abbattimento dei tempi di latenza dovuti ad un'architettura più diretta e quindi più veloce rispetto ai sistemi GCM.

Senza considerare la parte relativa alla configurazione del sistema: il GCM è molto più complesso e vincolante del sistema presentato; non è possibile ad esempio inviare messaggi con carichi utili superiori ai 4kb, così come non è possibile inoltrare un messaggio a più di 1000 destinatari.

Queste limitazioni vengono superate dal modello presentato.

Un altro aspetto rilevante riguarda la consegna dei messaggi: l'ordine d'invio dei messaggi grazie al sistema presentato viene rispettato, o per lo meno l'invio del messaggio avviene in una sequenza d'ordine prestabilita (utile in quei sistemi in cui l'ordine ha una certa rilevanza); mentre nei sistemi GCM lo stesso messaggio potrebbe essere ripartito tra più server che potrebbero inviarlo in sequenze completamente casuali.

La soluzione alternativa al GCM proposta da questa Tesi fa uso di un sistema implementato tramite tecnologia BOSH che sfrutta un RESTful web service come soluzione gestionale delle notifiche push, comunicante con un'applicazione client installata sul dispositivo mobile, avvalendosi del protocollo HTTP.

Si può pensare a questa soluzione come un sistema in grado di gestire le notifiche indirettamente, il cui intermediario (frapposto tra il client ed il mittente dei messaggi) è un servizio realizzato con la tecnologia RESTful Web Service.

La scelta di gestire le notifiche tramite RESTful web service, consente anche di risolvere problematiche relative a possibili restrizioni dovute ai firewall. Supponiamo che ci sia un palmare collegato ad un computer di una rete Intranet, e che questa rete abbia un firewall che blocchi la maggior parte delle porte verso Internet. Con una comunicazione che non usi l'http, c'è il rischio che l'informazione possa essere bloccata dai firewall; mentre col sistema proposto sarebbe possibile eludere il controllo del firewall, in quanto: la tecnologia presentata per la gestione delle notifiche fa uso del protocollo HTTP che si appoggia alla porta 80 (porta solitamente aperta per la navigazione su web).

Il sistema proposto permette quindi di bypassare eventuali limitazioni, sia restrittive, come nel caso di un firewall, sia di sviluppo e personalizzazioni (cosa

impossibile in sistemi chiusi), sia di invio per quantità di contenuto che di destinatari senza limitazioni.

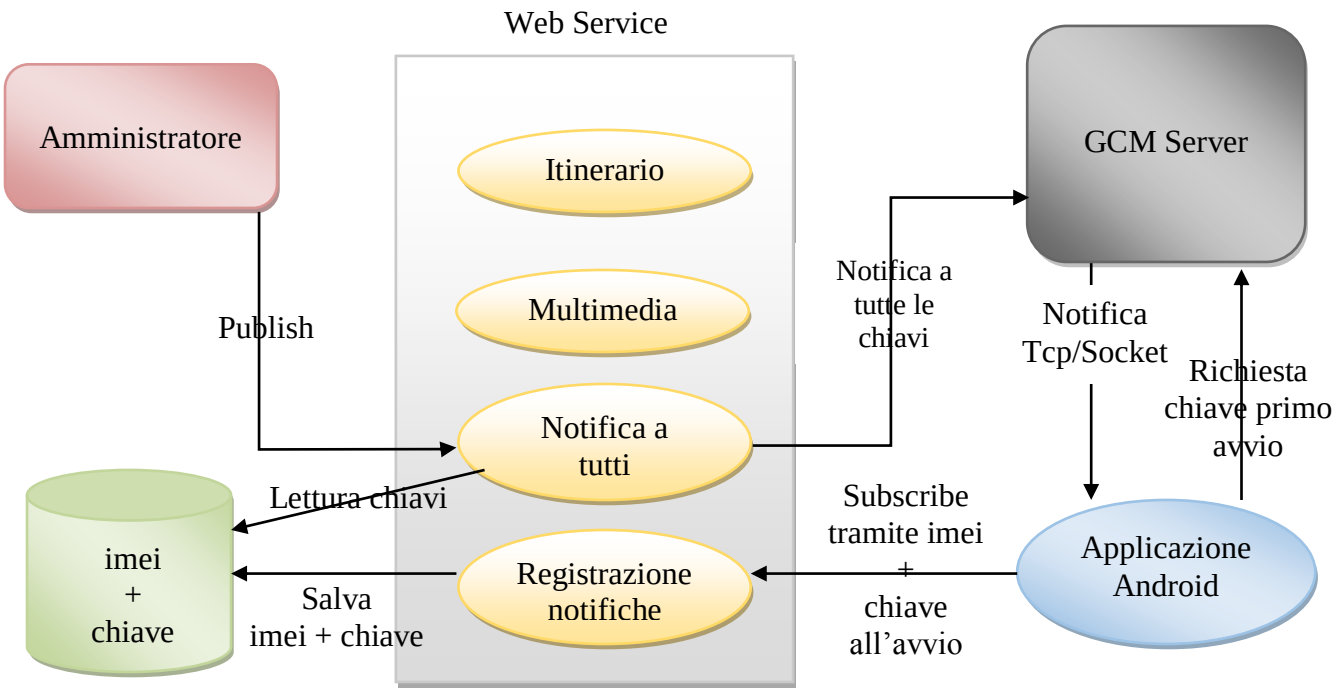
Per rendere pratica l'attuazione di notifica, è stata implementata un'applicazione Android che permette di ricevere notifiche gestendo contenuti multimediali e percorsi di itinerario.

3.2 *ARCHITETTURA*

In questo paragrafo vengono descritte le due architetture impiegate in questa tesi, il sistema GCM ed il sistema BOSH, che sono state utilizzate per realizzare un'applicazione Android specifica per l'utilizzo delle notifiche.

3.2.1 APPLICAZIONE ANDROID CON GCM

L'architettura del sistema è rappresentata nella figura seguente:



La struttura architetturale si compone di un client, di un Web Service e di un servizio.

Il client è una applicazione installata sul dispositivo mobile che gestisce, in base a diverse funzionalità, la ricezione dei messaggi ricevuti dal GCM Server. Il client può solo ricevere notifiche e comunica col Web Service attraverso una libreria installata sul proprio sistema chiamata Google Play Services che permette di aprire una connessione TCP/SOCKET col GCM Server. La connessione è instaurata dal client ogni volta che il segnale radio è attivo. L'applicazione Android non richiede di essere in esecuzione, poiché il sistema di ricezione è a carico dell'ambiente di esecuzione che grazie alla libreria Google Play avvia un intent receive passando il messaggio da notificare esattamente all'app destinataria.

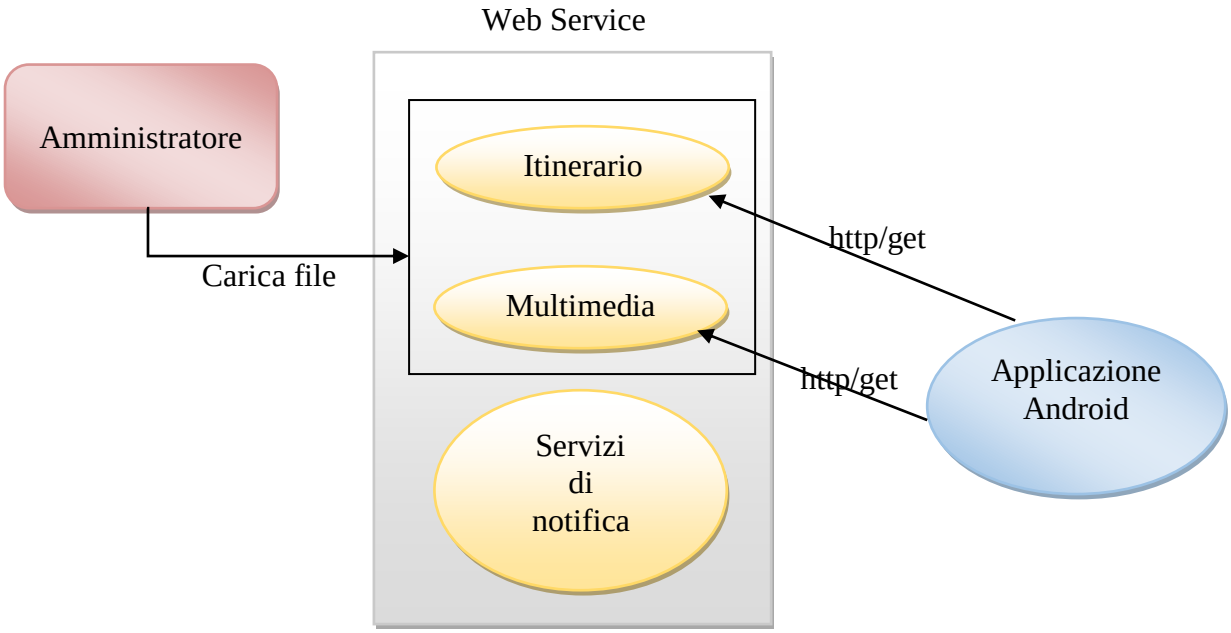
Il GCM Server si occupa di inoltrare le notifiche alle applicazioni Android indicate tra i destinatari di notifica e lavora come intermediario per fare in modo che i messaggi che gli vengono consegnati dal web service li recapiti ai dispositivi a cui sono destinati.

Il Web Service, di architettura RESTful, è in ascolto di possibili richieste da parte di un amministratore o di una entità abilitata all'invio di notifiche.

Le notifiche che vengono inoltrate, si appoggiano al protocollo HTTP ed usano un formato di rappresentazione chiamato JSON che permette una maggior strutturazione del dato.

Oltre alle notifiche, il web server gestisce la fornitura di dati multimediali richiesti direttamente dall'applicazione Android.

Segue l'architettura di funzionamento per l'interazione coi servizi Itinerario e Multimedia.



Nello schema appena esposto si possono notare le due funzioni messe a disposizione dell'Amministratore per il caricamento dei file di itinerario e dei file multimediali.

Inoltre sono mostrate le due chiamate che permettono all'applicazione Android di ricevere i file richiesti.

Nel caso del servizio Itinerario, quando l'applicazione Android riceve la notifica dal servizio GCM, ha la possibilità di richiedere direttamente il file di itinerario indicato nella notifica: se l'utente accetta di scaricarlo, si attiva un canale di comunicazione request-response su protocollo HTTP con metodo GET che permette all'applicazione di ricevere il file d'itinerario e caricarlo sulla propria mappa.

Nel caso del servizio Multimedia, l'applicazione Android ha la possibilità (selezionando un punto di interesse) di poter accedere ai suoi contenuti

multimediali tramite richieste dirette effettuate sempre su protocollo HTTP con metodo GET al fine di poter visionare i contenuti associati.
Inoltre, un aspetto importante per l'accesso ai contenuti del Web Service è dato dalla sessione, che si occupa di fornire l'accesso all'applicazione Android in modo che quest'ultima possa consultare le risorse d'interesse: file di itinerario e file multimediali.
La sessione viene creata in fase di inizializzazione e viene assegnata all'applicazione Android solo se riconosciuta dal Web Service: per fare in modo che il servizio riconosca l'applicazione Android, quest'ultima passa al servizio web un idlogin con una sequenza di cui solo le due parti sono a conoscenza; e se riconosciuta viene rilasciata una sessione che permette poi all'app di avere accesso ai contenuti multimediali.

3.2.1.1 FUNZIONALITÀ DEL WEB SERVICE

Le operazioni erogate dal Web Service (che nell'architettura GCM svolge il ruolo di Application Server) riguardano l'invio di notifiche push (Notifica), la registrazione delle applicazioni Android (Registrazione), l'invio di itinerari su connessione diretta ed esplicita dei client (Itinerario) e l'invio di informazioni multimediali (Multimedia) sempre su richiesta esplicita e diretta dei client.
Mentre le funzioni di Registrazione e di Notifica coinvolgono il GCM Server, le operazioni di Itinerario e Multimedia sono gestite direttamente tra il client ed il web server e non richiedono alcuna interazione da parte del GCM.
Ad esempio, un utente potrebbe richiedere un contenuto multimediale di un determinato punto di interesse, in questo caso si attiverebbe una connessione diretta tra il client (l'applicazione android) ed il web server per la visualizzazione del contenuto richiesto.
Così come la notifica che se accettata dal client, si attiverebbe una connessione diretta al web server per scaricare ed aggiornare la propria mappa sulla base del nuovo itinerario.

- La funzione Notifica riceve un messaggio composto di tre parti:

- Titolo: rappresenta il titolo della notifica.
- Messaggio: contine il contenuto descrittivodel messaggio.
- Filekml: indica il nome del file di itinerario disponibile

Crea, in seguito, un messaggio POST includendo queste tre parti in un pacchetto composto di diverse informazioni tra cui l'API key del servizio GCM da richiamare, le Registration ID delle applicazioni Android a cui inviare il messaggio ed il campo data contenente il messaggio composto dalle tre parti presentate: il payload vero e proprio del messaggio.

- La funzione Registrazione riceve un messaggio composto di due parametri, ed esattamente:
    - key: parametro contenente il registration id dell'applicazione android
    - imei: parametro contenente un identificativo del dispositivo

  e li registra permanentemente in un file xml

- La funzione Itinerario riceve una richiesta direttamente dall'applicazione android, composta di una parametro corrispondente al nome del file di itinerario che l'applicazione android richiede: attraverso una richiesta http/get. Restituisce il file indicato nel parametro della request.

- La funzione Multimedia riceve una richiesta composta di un parametro indicante il nome del file multimediale: attraverso una richiesta http/get. Ritorna il file indicato nel parametro della richiesta.

3.2.1.2 FUNZIONALITÀ DEL CLIENT

Le principali operazioni svolte dal client possono essere suddivise in due parti:

- computazione funzionale locale: gestisce il caricamento di un itinerario sulla mappa, permette di interagire coi punti di interesse associati, monitora e controlla la posizione GPS del dispositivo permettendo, attraverso un monitoraggio costante, l'avviso di prossimità della posizione del dispositivo rispetto i punti d'itinerario.
  Permette inoltre di consultare la mappa cartografica come riferimento di base dell'applicazione android e di poter interagire con questa in termini di zoom ed operazioni di pan.

- Computazione funzionale local-network (riguarda l'interazione con la rete): permette l'utilizzo di funzioni per le richieste di file multimediali e di file d'itinerario a seguito di notifiche d'aggiornamento.
  Nel primo caso, dopo aver selezionato un punto di interesse, si apre una finestra che riporta informazioni sul punto selezionato ed alcune icone da cui è possibile scaricare e visionare i contenuti associati.
  Selezionando, ad esempio, un'icona l'applicazione android si collega al web service attraverso una connessione HTTP richiedendo il file associato; a questo punto il file viene scaricato e riprodotto localmente.
  Nel secondo caso, la funzione di richiesta avviene a seguito di una notifica che permette all'utente di scegliere se scaricare il nuovo itinerario o rifiutare la richiesta. In caso di accettazione, l'applicazione android apre una connessione col web service, sempre in HTTP, andando a scaricare il file di itinerario per caricarlo poi su mappa.

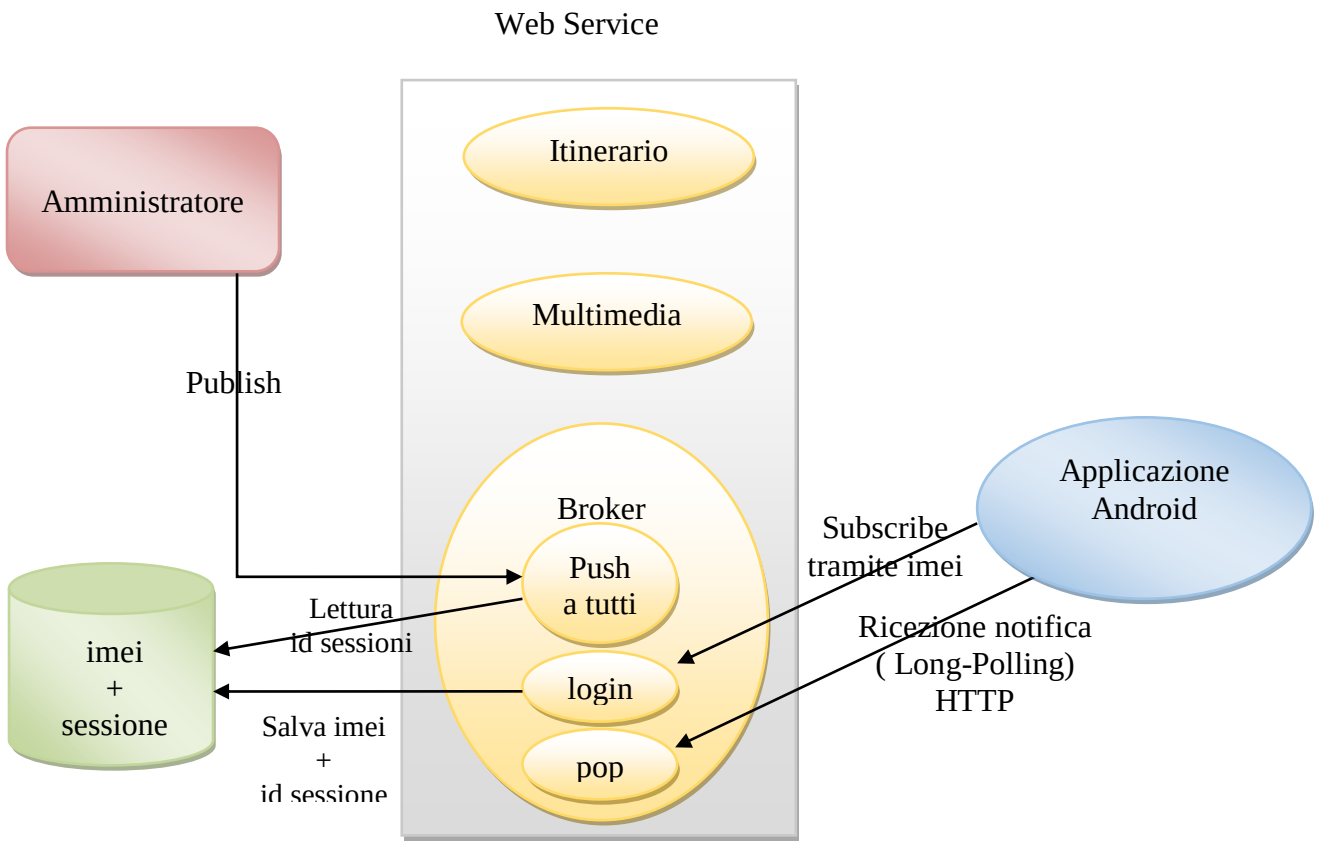
3.2.1.3 FUNZIONALITÀ DEL GCM

Come visto in precedenza per il funzionamento del GCM Server, le operazioni svolte da questa entità riguardano la fase di registrazione richiesta per le applicazioni android e l'invio di notifiche ai dispositivi associati.

Avviene una prima interazione tra l'applicazione android ed il GCM per la fase appena menzionata (di registrazione) ed in seguito interazioni che coinvolgono solamente il web service (per l'invio di notifiche).

3.2.2 APPLICAZIONE ANDROID CON BOSH

L'architettura del sistema è rappresentata nello schema seguente.



La struttura architetturale si compone di un client e di un Web Service.
Il client è una applicazione installata sul dispositivo mobile che gestisce, in base a diverse funzionalità, la ricezione dei messaggi ricevuti dal Web Service. Il client può solo ricevere notifiche e comunica col Web Service attraverso connessioni HTTP request/response.
La connessione è instaurata dal client ogni volta che il segnale radio è attivo. L'applicazione Android non richiede di essere in esecuzione, poiché è attivo un servizio in background che si occupa del collegamento col Web Service e grazie al quale è possibile ricevere le notifiche e passare il messaggio da notificare esattamente all'app destinataria.
Il Web Server si occupa di inoltrare le notifiche alle applicazioni Android e di gestire la fornitura di dati multimediali richiesti direttamente da queste.
Il Web Server è in ascolto di possibili richieste da parte di un amministratore o di un'entità abilitata all'invio di notifiche.
Le notifiche vengono passate al Broker che le inoltra ai dispositivi connessi.
Il Web Service gestisce attraverso il Broker le connessioni ai dispositivi grazie ad una tecnica chiamata Long Polling, che permette di mantenere attiva una connessione bloccando la request della richiesta finché non sono disponibili notifiche da inoltrare.
Il Broker gestisce anche una coda di messaggi che permette di notificare un messaggio anche quando il dispositivo non è raggiungibile.
La comunicazione è di tipo client – server e la rappresentazione del dato scambiato tra le due parti viene gestita attraverso messaggi formattati in XML.
Per quanto riguarda l'uso della sessione l'applicazione Android, collegandosi al Web Service con l'operation login, passa un parametro per farsi identificare (l'imei); il WS crea una associazione data dalla coppia (imei,sessione) e gli restituisce un messaggio per indicare all'app che è loggata. A questo punto l'app esegue una nuova request con operation pop che viene bloccata dal service fino a che non ci sono informazioni da notificare; quando arriva la notifica o se scade il timeout termina la connessione e quindi la request viene rilasciata. In seguito l'app si ricollega rieffettuando una nuova richiesta con operation pop e così via finché non termina la sessione per cause di caduta di connessione o mancanza di

segnale. Il timeout settato per il blocco della request viene fatto terminare poco prima del tempo di scadenza della sessione, in modo di mantenere quest'ultima sempre attiva.
Se avvenisse il contrario, avremmo che terminerebbe prima la sessione e nella fase successiva alla scadenza del timeout (per il rilascio della request), l'app non sarebbe più autenticata.
Mentre se la sessione dovesse scadere, causa disconnessione o mancata copertura, alla prima connessione disponibile gli verebbe restituito un messaggio d'errore obbligando l'app a rieffettuare la fase di login.
Riguardo il tempo effettivo per la scadenza di sessione, è possibile scegliere i tempi in base alle proprie esigenze: in questo caso si è optato per un tempo di scadenza pari a 30 minuti, corrispondente al valore impiegato da Tomcat come session timeout di default. Mentre per il tempo di timeout della request si è scelto un valore di poco inferiore: 25 minuti.

3.2.2.1 FUNZIONALITÀ DEL WEB SERVICE

Le funzionalità Itinerario e Multimedia sono identiche a quanto esposto nei paragrafi precedenti, mentre la parte sostanziale è fornita dal Broker che si occupa di gestire l'intera fase di comunicazione di notifiche attraverso connessioni persistenti.
Il Broker è composto di tre funzionalità principali:

- login: è l'operazione invocata dal client quando richiede di stabilire una connessione col servizio Broker.
  Il Web Server, in questa fase, associa alla richiesta di login una sessione che viene abbinata all'imei del dispositivo che ha effettuato la richiesta.
- pop: questa operazione viene invocata dal client subito dopo aver effettuato il login, e si occupa di instaurare una connessione di tipo Long Polling col client. In questa fase, se il client si era precedentemente disconnesso perdendo dei messaggi che nel frattempo erano stati inviati dal Broker ed il Broker non

trovando il dispositivo ha provveduto a memorizzarli nella coda associata, il client riceve dal Broker tutti i messaggi che nel frattempo gli sono stati accodati e solo quando la coda è vuota si attiva una nuova connessione persistente in modalità long polling.

- push: è l'operazione che permette ad un amministratore od entità di inviare le notifiche ai dispositivi associati.

  Il funzionamento è il seguente: l'amministratore invia attraverso un'interfaccia apposita un messaggio POST il cui payload contiene il titolo, il messaggio ed il nome del file di itinerario da recapitare ai dispositivi.

  Il Broker alla ricezione del messaggio, provvede ad inoltrarlo a tutti i dispositivi connessi ed eventualmente accodarlo per tutti quei dispositivi al momento irraggiungibili.

3.2.2.2 FUNZIONALITÀ DEL CLIENT

Le funzionalità del client sono identiche alle funzionalità esposte nel sistema GCM, salvo per la tecnica di connessione utilizzata.

In questo caso il client instaura una connessione diretta col web service attraverso una tecnica chiamata Long Polling. Questo gli permette di ricevere istantaneamente dal web service gli aggiornamenti (itinerari) in tempo reale e senza ritardi di alcun genere.

Inoltre il modello BOSH così progettato, ridefinisce lo scenario comunicativo in soli due attori: il web service ed il clinet (l'applicazione Android installata sul dispositivo); garantendo così una comunicazione più istantanea e diretta.

Sebbene il modello bosh permetta la bidirezionalità del flusso dati, in una comunicazione tra client e server, il modello così gestito è mono-direzionale e prevede lo scambio di informazioni dal service al client.

La tecnica utilizzata prevede chiamate sincrone, cioè bloccanti: lato server la request del client viene bloccata e rilasciata solamente in presenza di informazioni da restituire.

3.2.3 INTERFACCIA AMMINISTRATIVA

L'interfaccia amministrativa è stata realizzata per fornire uno strumento di controllo sui servizi.
Questa interfaccia permette, ad esempio, ad un amministratore di poter comporre un messaggio ed inviarlo ai dispositivi associati.
Inoltre permette di poter caricare su server file multimediali o di itinerario e garantire così la loro reperibilità ai dispositivi mobili.
Segue un elenco delle operazioni messe a disposizione dall'interfaccia esposta:

- Notifica GCM: è la form che permette di inoltrare la notifica tramite CGM.
  La form dispone di tre campi che riguardano il titolo del messaggio, il corpo del messaggio che rappresenta una descrizione dello stesso, ed il nome del file kml.
- Notifica Broker: rappresenta la form impiegata per inviare le notifiche tramite Broker. Anche questa form contiene i tre campi visti nel punto precedente, in quanto si occupa sempre dell'invio di notifica ai dispositivi associati, ma non passa l'informazione al GCM; l'inoltra invece direttamente ai dispositivi.
  Per cui, a livello di interfaccia grafica, valgono gli stessi campi descritti sopra: quello che cambia è l'architettura comunicativa utilizzata.
- Carica File: permette di caricare un file sul server. Questa funzione fornisce un pulsante di selezione grazie al quale è possibile selezionare un file.
  Vanno distinte due tipologie diverse di file, che sono: file con estensione kml, e file multimediali. Mentre per i file kml, una volta caricati, avviene un controllo sul tipo di estensione; questo non vale se viene caricato un file multimediale, poiché essendo troppo vasta la variabilità e la natura dei diversi formati, si è pensato di considerare come file multimediale tutto ciò che non è file kml. Va ricordato che i file kml, in questo caso, sono file di itinerario e rappresentano i soli contenuti informativi notificati ai dispositivi mobili.

Capitolo 4

Nel capitolo seguente si espongono le due implementazioni trattate nelle architetture precedenti.
La prima parte riguarda lo sviluppo del Web Service e del Client Android relativi alla tecnologia GCM.
La seconda parte riguarda lo sviluppo del Web Service e del Client Android relativi alla tecnologia BOSH (Long-Polling).

4.1 *IMPLEMENTAZIONE WEB SERVICE PER GCM*

Il Web Service è stato implementato in Java utilizzando l'ambiente di sviluppo NetBeans 8.0 (per l'editing, testing e debugging) e la libreria JAX-RS integrata nell'ambiente di sviluppo (per le operazioni di gestione dei servizi web tramite annotation). Una volta ultimato è stato esposto tramite il container Tomcat 8.0.3.
L'implementazione del Web Service è basata sulle seguenti classi principali che espongono operazioni di tipo richiesta/risposta:

- Itinerario
- Multimedia
- Notifica
- Registrazione

4.1.1 ITINERARIO

La funzione Itinerario si occupa di recuperare gli itinerari memorizzati sul server. Quando il client invia una richiesta di ricezione di un itinerario al WS, quest'ultimo esegue l'ordine prelevando l'informazione d'interesse. Quando il WS ha ottenuto il file, procede per restituirlo al client che ne ha fatto richiesta.
Si riporta di seguito il codice della funzione menzionata:

```
@GET
@Path("/kml/{file}")
@Produces({MediaType.APPLICATION_XML})
public Kml getItinerarioKml(@PathParam("file") String file){

    HttpSession session = request.getSession(false);
    if(session == null) {
        return null;
    }

    String url = servletContext.getRealPath("/WEB-INF/kml");

    if (System.getProperty("os.name").startsWith("Windows")) {
        url = url + "\\" + file + ".kml";
    } else {
        url = url + "/" + file + ".kml";
    }

    final Kml kml = Kml.unmarshal(new File(url));


    return kml;
}
```

La funzione restituisce un oggetto di tipo kml contenente informazioni formattate in xml.
L'annotation @GET indica che la richiesta dev'essere di tipo GET; mentre @Path esplicita il percorso che deve seguire la richiesta per indicare il file d'interesse: il percorso oltre all'indirizzo base, deve indicare la risorsa che si vuole ottenere, e questa risorsa è espressa da "{file}" che viene passato al metodo come parametro per la ricerca ed individuazione del file.
L'annotation @Produces({MediaType.APPLICATION_XML}), indica che l'operazione, in risposta alla richiesta, produce contenuto XML.
Una volta ottenuto il parametro, il metodo ricerca all'interno della directory /WEB-INF/kml il file; ed appena individuato lo carica in memoria creando un oggetto di tipo kml attraverso un processo chiamato unmarshal.

L'oggetto Kml è un oggetto che viene implementato grazie alla libreria JavaAPIforKml-2.2.0.jar che si occupa di creare l'istanza dal file registrato sul server. L'istanza viene creata grazie all'uso di una libreria integrata chiamata JAXB per l'xml binding che permette di effettuare mashalling ed unmarshalling di oggetti Java: in altre parole, è possibile serializzare oggetti Java in XML e di effettuare l'operazione inversa; quindi dalla rappresentazione XML riottenere l'oggetto Java.
Questo oggetto viene poi restituito al client formattato in xml.
Va notato che sia la libreria JavaAPIforKml, sia JAX-RS impiegano JAXB per le operazioni di serializzazione; questo significa che nel metodo appena presentato avvengono due operazioni: la prima di tipo umarshalling effettuata dalla libreria JavaAPIforKml che carica il file kml in un oggetto in memoria; la seconda elaborazione avviene invece ad opera della libreria JAX-RS che serializza (in maniera del tutto automatica sempre attraverso JAXB) con un marshalling l'oggetto Kml da restituire al client. Quest'ultima operazione avviene proprio grazie all'annotation @Produces({MediaType.APPLICATION_XML}).

4.1.2 MULTIMEDIA

La funzione Multimedia si occupa di restituire un file multimediale al client che ne effettua richiesta. Quando il client invia un messaggio, passandolo al WS, inserisce nella request il nome del file che vuole gli venga restituito. Il Web Service esegue l'ordine andando a costruire il messaggio di response creando uno stream che invia al client.
Si riporta di seguito il codice della funzione menzionata:

```
//Calcolo il tipo di mime
String mime = Files.probeContentType(path);

//set response headers
//response.setContentType("audio/mpeg");
response.setContentType(mime);

response.addHeader("Content-Disposition", "attachment; filename=" + fileName);

response.setContentLength((int) file.length());

FileInputStream input = new FileInputStream(file);
buf = new BufferedInputStream(input);
int readBytes = 0;
//read from the file; write to the ServletOutputStream
while ((readBytes = buf.read()) != -1) {
    stream.write(readBytes);
}
```

Il metodo esposto riceve come input un parametro che indica il nome del file, in seguito viene creata la response andando a settare il tipo di mime rilevato in base al file da restituire; viene aggiunta l'intestazione col nome del file e settata la dimensione richiesta per la composizione del messaggio di ritorno.
Infine il file viene serializzato e scritto nell'oggetto output stream venendo così trasmesso al client.

4.1.3 NOTIFICA

La funzione Notifica, si occupa di inoltrare un messaggio a determinati destinatari. Quando, ad esempio, un amministratore di rete invia una richiesta di notifica tramite la sua interfaccia amministrativa, questa notifica viene passata ad un servizio (una servlet che corrisponde nell'architettura GCM al server terzo) che compone ed invia il messaggio da inoltrare al GCM che a sua volta l'inoltrerà ai dispositivi indicati.
Questo messaggio viene composto nel seguente modo:

```
String titolo = request.getParameter("titolo");
String messaggio = request.getParameter("messaggio");
String filekml = request.getParameter("filekml");
Content c = new Content();
//c.addRegId("APA91bGjlyKqs6ikuFw4O-aer9meDdqsYdWs3JJmv-JBJYmjYmpVetc

for(int i = 0; i < listaID.length; i++){
    c.addRegId(listaID[i]);
}

c.createData(titolo, messaggio, filekml);
c.setCollapseKey("collassa pacchetti");
System.out.println( "Sending POST to GCM" );
String apiKey = "AIzaSyANUFqolqrT2QfOEnAAwX4KJE7c_0hIOHQ";
POST2GCM.post(apiKey, c);

response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
out.println("{\"result\":\"Notifica inoltrata con successo\"}");
```

Il metodo esposto riceve in input tre parametri che sono

- titolo: corrisponde al titolo della notifica;
- messaggio: corrisponde all'informazione associata al messaggio;
- filekml: rappresenta il nome del file da scaricare per aggiornare il proprio itinerario.

Ottenuti questi tre parametri si procede alla creazione del messaggio da inoltrare al GCM. In seguito viene definita e aggiunta l'API key richiesta per contattare il servizio GCM e viene creata una connessione http con metodo post contenente il messaggio formattato in json; ed infine viene inviato il tutto al GCM.

```
// 1. URL
URL url = new URL("https://android.googleapis.com/gcm/send");
// 2. Open connection
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
// 3. Specify POST method
conn.setRequestMethod("POST");
// 4. Set the headers
conn.setRequestProperty("Content-Type", "application/json");
conn.setRequestProperty("Authorization", "key=" + apiKey);
conn.setDoOutput(true);
// 5. Add JSON data into POST request body
// 5.1 Use Jackson object mapper to convert Content object into JSON
ObjectMapper mapper = new ObjectMapper();
// 5.2 Get connection output stream
DataOutputStream wr = new DataOutputStream(conn.getOutputStream());
// 5.3 Copy Content "JSON" into
mapper.writeValue(wr, content);
//mapper.configure(SerializationFeature.INDENT_OUTPUT, true);
//Stampa la stringa json passata al server GCM
System.out.println(mapper.writeValueAsString(content));
// 5.4 Send the request
wr.flush();
```

4.1.4 REGISTRAZIONE

La funzione Registrazione si occupa di registrare i dispositivi nel server terzo: chiamato nell'architettura GCM Application Server.
Questa funzione riceve in ingresso due parametri:

```
public Registrazione() {
}

@POST
@Produces(MediaType.TEXT_HTML)
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
public String postCreaElementInXml(@FormParam("key") String key, @FormParam("imei") String imei)
```

- key: corrisponde alla registration key fornita in fase di registrazione dal GCM all'applicazione Android installata sul dispositivo mobile;

- imei: riguarda il codice che identifica in maniera univoca il dispositivo. A seconda dei casi può essere sia il codice imei, se è un cellulare, oppure il MAC address della scheda di rete del dispositivo.

Ottenuti i due parametri, il metodo si accerta che il dispositivo da cui proviene la richiesta non sia già stato registrato, in caso affermativo la chiave corrispondente viene sostituita con l'ultima chiave passata: come mostrato in figura.

```
for(int s = 0; s < listOfElements.getLength() ; s++){

    //Esegue il casting per trasformare listOfElements di tipo Node in element di tipo Element
    Element element = (Element) listOfElements.item(s);

    String verificaAttributo = element.getAttribute("imei");

    if(imei.equals(verificaAttributo) ){
        //il dispositivo presenta gia' una registrazione: sostituisco la chiave con quella passata
        element.setTextContent(key);
        TransformerFactory transformerFactory = TransformerFactory.newInstance();
        Transformer serializer = transformerFactory.newTransformer();
        serializer.transform(new DOMSource(document), new StreamResult(new FileOutputStream(url)));
        return "<html><head></head><body>Codice esistente.</body></html>";
    }
```

Mentre se non esiste viene inserito un nuovo noto contenente la nuova chiave e l'imei con attributo del nodo

```
//Registro l'id nel file xml

Element radice = (Element) root;

Element nodoid = document.createElement("Id");

nodoid.setAttribute("imei", imei);

nodoid.appendChild(document.createTextNode(key));

radice.appendChild(nodoid);

TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer serializer = transformerFactory.newTransformer();

serializer.transform(new DOMSource(document), new StreamResult(new FileOutputStream(url)));


return "<html><head></head><body>Registrazione creata con successo.</body></html>";
```

Segue una rappresentazione del file xml contenente i dati registrati

```
<root>
<Id imei="x5xxxxxxxx0xxxx">APA91bGjlyYdWs3JJmv...</Id>
<Id imei="2xx7xxxxxx5xxxx">QE12mPJCi4qw_w4A_Tp...</Id>
</root>
```

Da come si evince nell'immagine soprastante , per ogni dispositivo che richiede la registrazione al server terzo, si assegna un id che lo identifica grazie al codice imei ed inoltre gli viene aggiunta la chiave come nodo testuale utile per gli invii delle notifiche.

4.2 *IMPLEMENTAZIONE CLIENT PER GCM*

Il Client è stato implementato in Java utilizzando l'ambiente di sviluppo Eclipse ADT con integrato l'SDK Android: Eclipse ADT Bundle.
La configurazione del progetto prevede che il dispositivo abbia installato una versione di Android con almeno l'SDK API 14: corrispondente ad Android 4.0.
Le classi più importanti ''ai fini del progetto'' sono quelle per la conduzione degli eventi e la gestione di connessione col GCM e l'Application Server.
Per la connessione tra il client ed il GCM viene impiegata una libreria chiamata google-play-services_lib inclusa nel progetto dell'applicazione android.
Questa libreria permette di interfacciarsi al GCM fornendo funzioni specifiche e garantendo la comunicazione tra il client ed il server.
Il progetto è composto dalle seguenti classi principali:

- MainActivity
- GcmBroadcastReceiver
- GcmMessageHandler
- CustomInfoWindow
- VibrateService
- AlertService

4.2.1 CLASSE MAINACTIVITY

La classe MainActivity rappresenta il fulcro del progetto, la parte più rappresentativa e centrale dell'applicazione. E' composta di diverse funzioni, tra quelle principali abbiamo: onCreate, loadMap, getRegId, DownloadTask, myLocationListener quest'ultima definita dall'interfaccia LocationListener. La funzione onCreate permette di impostare la configurazione di avvio dell'applicazione istanziando ad esempio gli oggetti base dell'app come la mappa e l'itinerario. Il metodo loadMap permette di caricare su una mappa i punti di un itinerario, il metodo getRegId fornisce la registrazione dell'applicazione android al GCM Server. La classe DownloadTask si occupa di effettuare il download di un file d'itinerario; mentre l'oggetto myLocationListener fornisce il monitoraggio della posizione GPS, associando a queste posizioni l'azione relativa. Di seguito viene illustrato il codice della funzione onCreate:

```
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

//Rilevo il codice imei
TelephonyManager mngr =
(TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
imei = mngr.getDeviceId();
if(imei == null){
WifiManager manager = (WifiManager)
getSystemService(Context.WIFI_SERVICE);
WifiInfo info = manager.getConnectionInfo();
imei = info.getMacAddress();
}

//Mappa con markatori ed infoWindow

map = (MapView) findViewById(R.id.map);
map.setTileSource(TileSourceFactory.MAPNIK);
map.setBuiltInZoomControls(true);
map.setMultiTouchControls(true);

mapController = map.getController();
mapController.setZoom(18);

if (savedInstanceState != null) {
CenterPosition center =
savedInstanceState.getParcelable("CENTER_POSITION");
Toast.makeText(getBaseContext(),
" Lan: " + center.getCenter().getLatitude() +
" Lng: " + center.getCenter().getLongitude(),
```

```
Toast.LENGTH_SHORT).show();

mapController.setCenter(center.getCenter());
mapController.setZoom(center.getZoom());
}

loadDefaultFile();

// Crea un'istaza progress bar impiegata per l'aggiornamento
dell'itinerario
mProgressDialog = new ProgressDialog(this);
mProgressDialog.setMessage("A message");
mProgressDialog.setIndeterminate(true);
mProgressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
mProgressDialog.setCancelable(true);

createSession();

verifyRegistration();

//GPS
locationManager =
(LocationManager)getSystemService(Context.LOCATION_SERVICE);

Location lastLocation =
locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
if(lastLocation != null){
updateLoc(lastLocation);
}

//Avvia il servizio gps per monitorare l'avviso di prossimita'
intent = new Intent(getApplicationContext(), AlertService.class);
startService(intent);

}
```

Come si può notare, la prima istruzione va a rilevare il codice imei del dispositivo, poi crea l'oggetto map che rappresenta la mappa di base ed abilita oltre ad altri settaggi lo zoom ed il multi-touch; in seguito carica l'itinerario di default tramite il metodo loadDefaultFile che corrisponde all'itinerario che viene fornito con la prima installazione dell'applicazione android. Segue il metodo createSession che crea una sessione con l'Application Server per fare in modo che l'app possa accedere ai file multimediali del server terzo (l'Application Server), ed inoltre viene invocato il metodo verifyRegistration che permette di verificare e registrare l'app al servizio GCM.

Infine viene creato l'oggetto lastLocation per mostrare in mappa l'ultima posizione nota del dispositivo, e viene successivamente lanciato il service AlertService che è il servizio dedicato al monitoraggio del gps.

Segue un'immagine dell'app appena avviata.

Figura 36 - App Iniziale

Il codice del metodo loadMap viene presentato di seguito:

```
//Carica i punti dell'itinerario sulla mappa
public void loadMap(){
ArrayList<Point> punti = Collection.getInstance().getData();

map.getOverlays().clear();

//Ridisegno la posizione del gps: altrimenti verrebbe persa
if(gpsLocation != null){
position = new Marker(map);
position.setPosition(new GeoPoint(gpsLocation.getLatitude(),
gpsLocation.getLongitude()));
position.setAnchor(Marker.ANCHOR_CENTER, Marker.ANCHOR_BOTTOM);
position.setIcon(getResources().getDrawable(R.drawable.gps_icon_red));
position.setTitle("me");
map.getOverlays().add(position);
map.invalidate();
}
```

```
PathOverlay myOverlay= new PathOverlay(Color.rgb(249, 112, 4), this);
myOverlay.getPaint().setStyle(Style.STROKE);

for(int i = 0; i < punti.size(); i++){
Point punto = (Point) punti.get(i);
Marker temp = new MarkerWithLabel(map,
punto.getLinks().get("ID").toString());
temp.setPosition(punto.getPoint());
temp.setAnchor(Marker.ANCHOR_CENTER, Marker.ANCHOR_BOTTOM);
temp.setIcon(getResources().getDrawable(R.drawable.blue_poi));
temp.setTitle(punto.getName() + " ");
temp.setSubDescription((String) punto.getLinks().get("DESCRIZIONE"));
map.getOverlays().add(temp);
map.invalidate();

myOverlay.addPoint(punto.getPoint());

//Effettuo il focus su mappa del punto con ID = 1: lo start
dell'itinerario
if("1".equals(punto.getLinks().get("ID")))
mapController.setCenter(punto.getPoint());

//Personalizzo l'infoWindow
LinkMultimediali links = new LinkMultimediali();
links.setAudio((String) punto.getLinks().get("AUDIO"));
links.setVideo((String) punto.getLinks().get("VIDEO"));
links.setImmagine((String) punto.getLinks().get("IMMAGINE"));
links.setTesto((String) punto.getLinks().get("DOCUMENTO"));
temp.setRelatedObject(links);
temp.setInfoWindow(new CustomInfoWindow(map, this));

Collection.getInstance().getData().get(i).setMarker(temp);

}
map.getOverlays().add(myOverlay);
map.invalidate();
}
```

Dal codice si può vedere che viene creata un'istanza di un oggetto che rappresenta una collezione di punti d'itinerario. In seguito viene ridefinita la posizione GPS del dispositivo, disegnando la posizione su mappa, e successivamente vengono creati dei marcatori a cui si associano i punti dell'istanza caricata: attraverso l'oggetto MarkerWithLabel. Infine viene creata una infoWindow personalizzata per la descrizione del punto d'interesse.

A questo punto con la selezione su mappa del marcatore, dovrebbe apparire una finestra di dettaglio con dei link collegati ai file multimediali associati a quel punto e scaricabili dal server terzo (l'Application Server).

Segue una grafica del funzionamento.

Figura 37 - InfoWindow

La funzione getRegId, come accennato in precedenza, si occupa di registrare l'applicazione android al servizio GCM.

Questa funzionalità è rappresentata dal codice seguente.

```
public void getRegId() {
    new AsyncTask<Void, Void, String>() {
        @Override
        protected String doInBackground(Void... params) {
            String msg = "";
            try {
                if (gcm == null) {
                    gcm = GoogleCloudMessaging.getInstance(getApplicationContext());
                }
                regid = gcm.register(SENDER_ID);
                msg = "Device registered, registration ID=" + regid;
                Log.i("GCM", msg);
                // Persist the regID - no need to register again.
                storeRegistrationId(getApplicationContext(), regid);
                //Invio regid al server per fare in modo che possa notificare all'app gli eventi
                sendToServer();
            } catch (IOException ex) {
                msg = "Error :" + ex.getMessage();

            }
            return msg;
        }

        @Override
        protected void onPostExecute(String msg) {
            Log.i("GCM", msg);
        }
    }.execute(null, null, null);
}
```

Il metodo esegue in background la fase di registrazione invocando l'oggetto GoogleCloudMessaging.getInstance fornito dalla classe com.google.android.gms.gcm.GoogleCloudMessaging integrata nella già menzionata libreria google-play-services_lib.

Questa invocazione permette di ottenere un'istanza dell'oggetto GoogleCloudMessaging da cui poter invocare il metodo register (passandogli il sender_id/project id) ed ottenere il registration id fornito dal GCM.

Ottenuto l'id, l'applicazione android, lo passa all'Application Server col metodo sendToServer: in modo che quest'ultimo possa registrarlo per gli invii delle notifiche.

La classe DownloadTask permette di gestire il download dei file di itinerario notificati dall'amministratore di sistema ai client associati.

Il cuore della classe è mostrato dal codice seguente.

```
URL url = new URL(sUrl[0]);

MainActivity.httpClient.setBase(url);
input = (InputStream) MainActivity.httpClient.getKml("itinerario/kml/", filekml);


// expect HTTP 200 OK, so we don't mistakenly save error report
// instead of the file
if (MainActivity.httpClient.getConnection().getResponseCode() != HttpURLConnectior
    return "Server returned HTTP " + MainActivity.httpClient.getConnection().getRe
}

// this will be useful to display download percentage
// might be -1: server did not report the length
int fileLength = MainActivity.httpClient.getConnection().getContentLength();

// download the file
input = MainActivity.httpClient.getConnection().getInputStream();
output = new FileOutputStream("/sdcard/itinerario.kml");

byte data[] = new byte[4096];
long total = 0;
int count;
while ((count = input.read(data)) != -1) {
    // allow canceling with back button
    if (isCancelled()) {
        input.close();
        return null;
    }
    total += count;
    // publishing the progress....
    if (fileLength > 0) // only if total length is known
        publishProgress((int) (total * 100 / fileLength));
    output.write(data, 0, count);
}
```

Il download viene attivato solo su accettazione dell'utente che ricevuta la notifica deve accettarla per poter scaricare l'itinerario notificato

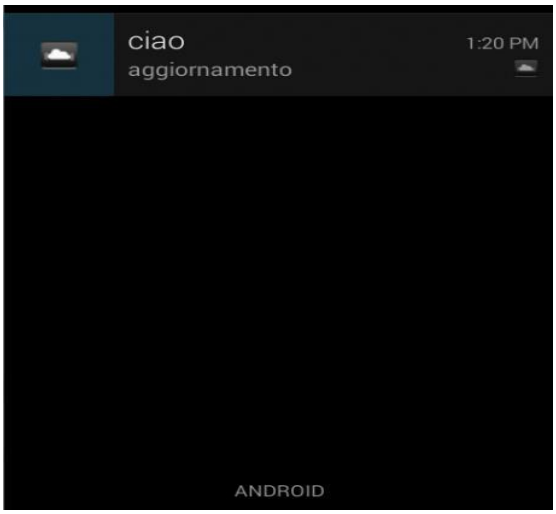


Figura 38 - Notifica su status bar

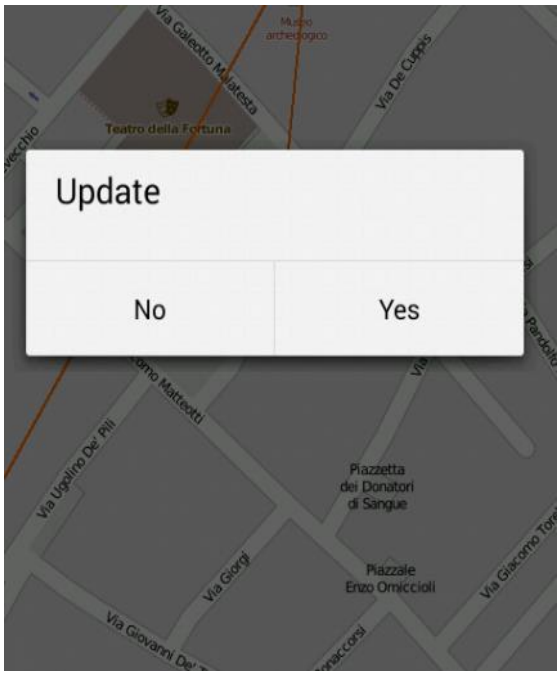


Figura 39 - Richiesta di aggiornamento

Selezionando Yes, si attiva il download del file ed il suo successivo caricamento: vengono caricati i punti dell'itinerario scaricato.

L'interfaccia myLocationListener permette di monitorare la posizione del dispositivo sulla mappa per l'avviso di prossimità.

Quando la posizione del dispositivo si avvicina entro un certo raggio da uno dei punti di interesse dell'itinerario caricato, si attiva una vibrazione con notifica per avvertire l'utente che si trova in una posizione prossima ad un Point of Interest (POI). Segue il codice descritto.

```
private LocationListener myLocationListener = new LocationListener() {

    public void onStatusChanged(String provider, int status, Bundle extras) {
    }

    public void onProviderEnabled(String provider) {
    }

    public void onProviderDisabled(String provider) {
    }

    public void onLocationChanged(Location location) {
        updateLoc(location);
    }
};
```

L'oggetto dispone di diversi metodi, tra cui onLocationChange che viene invocato ad ogni cambio posizione.
Questo metodo invoca a sua volta un altro metodo chiamato updateLoc che esegue un controllo sui punti prossimi alla posizione ricevuta.

```
float dist = Integer.MAX_VALUE;
Location dest = new Location("");
Collection punti = Collection.getInstance();

//Verifica la prossimita' della posizione attuale con le diverse destinazioni (punti di interesse)
for(int i = 0; i < punti.getData().size(); i++){

    GeoPoint tempPoint = punti.getData().get(i).getPoint();
    dest.setProvider(punti.getData().get(i).getName());
    dest.setLatitude(tempPoint.getLatitude());
    dest.setLongitude(tempPoint.getLongitude());
    dist = loc.distanceTo(dest);

    if (dist < 20 && !search(visitato, dest)) {
        visitato.add(dest);
        //Attivo geofence
        if(cerchio != null) map.getOverlays().remove(cerchio);
        cerchio = new CircleOverlay(this, dest.getLatitude(), dest.getLongitude(), 20);
        map.getOverlays().add(cerchio);
        break;

    } else if(dist > 30 && search(visitato, dest)) {
        //Rimuovo geofence
        if(cerchio != null) map.getOverlays().remove(cerchio);
        remove(visitato, dest);
    }

}
```

Come si può notare dal codice, se la posizione ricade entro un certo raggio da uno dei punti disponibili, allora viene disegnato un cerchio trasparente centrale al punto oltre ad una notifica di avviso.

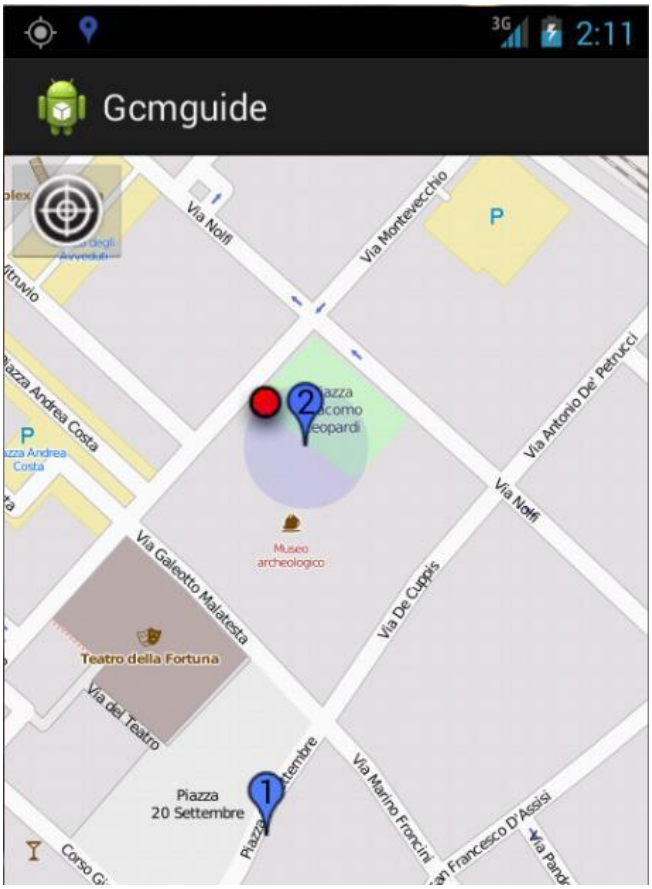


Figura 40 - Avviso di prossimità

4.2.2 CLASSE GCMBROADCASTRECEIVER

La classe GcmBroadcastReceiver gestisce la ricezione delle notifiche catturando l'avviso e delegando la gestione del messaggio alla classe GcmMessageHandler.
La classe si compone del seguente codice:

```
 9 public class GcmBroadcastReceiver extends WakefulBroadcastReceiver {
10
11     @Override
12     public void onReceive(Context context, Intent intent) {
13
14         // Explicitly specify that GcmMessageHandler will handle the intent.
15         ComponentName comp = new ComponentName(context.getPackageName(), GcmMessageHandler.class.getName());
16
17         // Start the service, keeping the device awake while it is launching.
18         startWakefulService(context, (intent.setComponent(comp)));
19         setResultCode(Activity.RESULT_OK);
20     }
21 }
```

Questa classe estende WakefulBroadcastReceiver che permette di mantenere attivo il servizio finché non termina la procedura: siccome il sistema android non garantisce che un processo una volta avviato rimanga attivo o persista per tutta la durata richiesta, è necessario fare in modo che esegua fino in fondo tutte le operazioni; cosa resa possibile dalla classe esposta.
Il GcmBroadcastReceiver viene attivato dalla ricezione di un messaggio, ed al suo risveglio avvia la classe GcmMessageHandler passandogli la notifica ricevuta.

4.2.3 CLASSE GCMSERVICEMESSAGEHANDLER

La classe GcmMessageHanler viene invocata dal GcmBroadcastReceiver, e si occupa di gestire il messaggio di notifica ricevuto dal GCM Server.
I metodi principali sono:

- onHandleIntent: si occupa di ricevere il messaggio ed avviare la funzione showNotification.
  Il codice della funzione è riportato di seguito:

```
protected void onHandleIntent(Intent intent) {
    Bundle extras = intent.getExtras();

    GoogleCloudMessaging gcm = GoogleCloudMessaging.getInstance(this);
    // The getMessageType() intent parameter must be the intent you received
    // in your BroadcastReceiver.
    String messageType = gcm.getMessageType(intent);

    titolo = extras.getString("title");
    messaggio = extras.getString("message");
    MainActivity.filekml = extras.getString("filekml");
    showToast();
    Log.i("GCM", "Received : (" + messageType + ")  " + extras.getString("title"));

    //Attivo notifica
    if(titolo != null && messaggio != null && !"".equals(titolo.trim()) && !"".equal
        showNotification(getApplicationContext());


    GcmBroadcastReceiver.completeWakefulIntent(intent);
}
```

  Come si può notare, vengono prelevati i parametri del messaggio: titolo, messaggio e filekml (contenente il nome del file di itinerario da scaricare).

Infine viene avviato il metodo showNotification.

- showNotification: avvia la notifica di ricezione di un messaggio sulla barra di stato dell'applicazione.

```
private void showNotification(Context con) {

    //Context context = getApplicationContext();
    Intent intent = new Intent(con, MainActivity.class);
    intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_SINGLE_TOP);
    intent.putExtra("DIALOG", true);
    PendingIntent pending = PendingIntent.getActivity(con, 0, intent, 0);

    Bitmap aBitmap = BitmapFactory.decodeResource(getResources(), R.drawable.ic_stat_gcm);

    NotificationManager ns = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

    Notification notify = new Notification.Builder(con)
    .setContentTitle(titolo)
    .setContentText(messaggio)
    .setSmallIcon(R.drawable.ic_stat_gcm)
    .setLargeIcon(aBitmap)
    .setAutoCancel(true)
    .setDefaults(Notification.DEFAULT_SOUND)
    .setContentIntent(pending)
    .build();

    ns.notify(++index,notify);
```

4.2.4 CLASSE CUSTOMINFOWINDOW

Questa classe viene impiegata come riferimento per la descrizione e gli indirizzi multimediali associati ai punti di interesse (POI).
L'indirizzo delle risorse fa riferimento all'URL del WS a cui collegarsi per prelevare tali contenuti.
Viene esposto un frammento del codice della classe analizzata:

```
public CustomInfoWindow(MapView mapView, MainActivity m) {
        super(R.layout.bonuspack_bubble, mapView);

        this.main = m;

        Button btn = (Button)(mView.findViewById(R.id.bubble_moreinfo));


        btn.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                    Toast.makeText(view.getContext(), "Button clicked", Toast.LENGTH_LONG).show();

                    if (mSelectedPoi.getVideo() != null && !"".equals(mSelectedPoi.getVideo())){
                        //Verifico se il link e' del server oppure esterno
                        if(!mSelectedPoi.getVideo().toLowerCase().contains("http")){
                            //Il link e' del server: avvio il download del file e la sua apertura successiva tramit
                            downloadFileMultimediale(mSelectedPoi.getVideo());
                        }else{
                            //Il link e' esterno, avvio direttamente l'action view senza effettuare il download
                            Intent myIntent = new Intent(Intent.ACTION_VIEW, Uri.parse(mSelectedPoi.getVideo()));
                            view.getContext().startActivity(myIntent);
                        }
                    }


            }
        });
```

Con questo sistema alla selezione di uno dei pulsanti associati al punto, si crea un collegamento all'Application Server andando a scaricare il file relativo.

4.2.5 CLASSE VIBRATESERVICE

La classe VibrateService viene invocata dalla classe AlertService e si occupa di attivare la modalità di vibrazione del telefono per fare in modo che l'utente possa percepirne il movimento e quindi notare la sua posizione all'interno di un raggio di prossimità.
Il codice è molto semplice ed è rappresentato in figura

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    // TODO Auto-generated method stub
    //return super.onStartCommand(intent, flags, startId);
    Thread thread = new Thread() {
        @Override
        public void run() {
            super.run();

            Vibrator v = (Vibrator)getSystemService(Context.VIBRATOR_SERVICE);

            // pass the number of millseconds fro which you want to vibrate the
            // have passed 1000 so phone will vibrate for 1 seconds.

            v.vibrate(1000);

            mHandler.post(new Runnable(){
                public void run(){
                    Toast.makeText(getApplicationContext(), "Phone is Vibrating"
                }
            });


        }
    };

    //thread.setPriority(Process.THREAD_PRIORITY_BACKGROUND);
    thread.start();

    //Se il sistema termina il servizio, non deve ricrearlo
    return START_NOT_STICKY;
}
```

La parte principale è determinata dalla chiamata del servizio VIBRATOR_SERVICE a cui viene passato un parametro temporale che indica per quanto tempo il dispositivo deve vibrare: in questo caso si parla di mille millisecondi corrispondenti ad un secondo.

4.2.6 CLASSE ALERTSERVICE

La classe AlertService viene invocata dalla classe MainActivity per monitorare l'avviso di prossimità sulla posizione GPS del dispositivo anche quando la MainActivity non è attiva.

La parte più rilevante è quella determinata dal seguente metodo

```
public void onLocationChanged(Location loc) {

    float dist = Integer.MAX_VALUE;
    Location dest = new Location("");
    Collection punti = Collection.getInstance();

    //Verifica la prossimita' della posizione attuale con le diverse destinazioni (punt
    for(int i = 0; i < punti.getData().size(); i++){

        GeoPoint tempPoint = punti.getData().get(i).getPoint();
        dest.setProvider(punti.getData().get(i).getName());
        dest.setLatitude(tempPoint.getLatitude());
        dest.setLongitude(tempPoint.getLongitude());
        dist = loc.distanceTo(dest);

        if (dist < 20 && !search(visitato, dest)) {
            Log.v("asd", "yes: " + dist);

            visitato.add(dest);

            //Attivo vibrazione
            Intent intent = new Intent(getApplicationContext(), VibrateService.class);
            intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            startService(intent);

            //Attivo notifica
            showNotificationLocation(i, dest, getApplicationContext());
            break;

        } else if(dist > 30 && search(visitato, dest)) {
            Log.v("asd", "no1: " + dist);
            remove(visitato, dest);
        }else {
            Log.v("asd", "no2: " + dist);

        }

    }
}
```

Il metodo si attiva ad ogni cambio posizione GPS e rileva in base ai punti di interesse la prossimità del dispositivo sulla base delle sue coordinate.

A differenza del metodo onLocationChange del MainActivity (che è stato integrato per il solo scopo di disegnare sia il punto del dispositivo sulla mappa sia il geofance), questo metodo si attiva sempre sul cambio posizione GPS, ma non interagisce con la mappa: la sua funzione principale è quella di notificare sullo status bar un avviso di prossimità ed una vibrazione che allerti l'utente.

4.3 *IMPLEMENTAZIONE WEB SERVICE PER BOSH*

L'implementazione del servizio web per BOSH è stata realizzata con gli stessi strumenti impiegati per il GCM: utilizzando sempre lo stesso ambiente di sviluppo, quale NetBeans 8.0.
Le funzioni del servizio che rimangono inalterate sono: Itinerario e Multimedia.
Queste operazioni sono identiche a quando descritto nell'implementazione del Web Service per GCM.
Quello che cambia radicalmente, oltre all'architettura, è l'operazione di notifica.
In questa implementazione è stata introdotta una classe chiamata Broker che integra la funzionalità di notifica.
Questa classe permette di gestire, oltre alle notifiche, i collegamenti diretti ai dispositivi: permette di instaurare connessioni sincrone con i diversi client, mantenendo attivi i canali di comunicazione finché non si dispone di carichi utili da notificargli.
Il Broker è composto di tre operation principali:

- Login
- Push
- Pop

4.3.1 LOGIN OPERATION

Il metodo login rappresenta la fase in cui il broker accetta le connessioni dei client creando una sessione con cui riconoscerli.
Segue il codice dell'operation login.

```
HttpSession session = request.getSession();
Element root = data.getDocumentElement();
String operation = root.getTagName();
String user;
Document answer = null;
OutputStream os;

switch (operation) {
    case "login":
        user = ((Text) root.getChildNodes().item(0)).getData();
        System.out.println("login received: " + user);
        synchronized (this) {
            if(sessioni.get(user) != null &&
               !sessioni.get(user).getId().equals(session.getId())){
                sessioni.get(user).invalidate();
                ob.wait();
            }
            sessioni.put(user, session);
            contexts.put(user, new LinkedList<Document>());
        }

        session.setAttribute("user", user);
        session.setMaxInactiveInterval(10);
        answer = mngXML.newDocument();
        answer.appendChild(answer.createElement("logged"));
        os = response.getOutputStream();
        mngXML.transform(os, answer);
        os.close();

        break;
```

In un primo momento viene creata la sessione, se non esiste, ed estratto dal nodo radice del messaggio della request il tipo di operation da eseguire.

Se il tipo di operation è login, il controllo entra nello scope corrispondente; a questo punto viene prelevato l'identificativo del dispositivo (che può essere imei o MAC address della scehda di rete utilizzata per il collegamento) e memorizzato nella variabile user. In seguito si controlla la sessione, se esiste già una sessione associata a quello user, ma diversa dalla sessione appena creata per lo stesso user (sovrapposizione di sessioni aperte per lo stesso user), allora la sessione precedente viene rimossa e si mantiene l'ultima.

Successivamente vengono settati dei parametri di sessione come lo "user" e l'intervallo massimo di inattività concesso, oltre il quale scatta la scadenza della sessione. Infine viene restituita una risposta al client e chiusa la connessione.

Alla scadenza della sessione, si attiva un listener che intercetta l'invalidate: il codice che esegue questa operazione è il seguente.

```
@Override
public void sessionDestroyed(HttpSessionEvent event) {
    String utente = (String) event.getSession().getAttribute("user");
    contexts.remove(utente);
    sessioni.remove(utente);
    System.out.println("SESSIONE RIMOSSA: "+ utente + " - " + event.get
    ob.notify();
}
```

Questo metodo distrugge la sessione andando a rimuovere dalla memoria anche la relativa coda di messaggi associati all'utente.

4.3.2 PUSH OPERATION

Questa fase prevede l'invio di notifica ai dispositivi associati ed è invocata dall'amministratore di sistema attraverso l'interfaccia amministrativa.
Segue l'esposizione del codice push

```
case "push":
    System.out.println("push received");

    /*
     * Invio il messaggio a tutti i client connessi.
     * Per i clienti loggati ma non ancora connessi,
     * inserisco il messaggio in una coda per recapitarglielo nella request successiva
     */
    for (String destUser : contexts.keySet()) {
        Object value = contexts.get(destUser);
        if (value instanceof AsyncContext) {
            try{
                //String test = (String) ((HttpServletRequest) ((AsyncContext) value).getR
                OutputStream aos = ((AsyncContext) value).getResponse().getOutputStream();
                mngXML.transform(aos, data);
                aos.close();
                ((AsyncContext) value).complete();
                //Resetto la coda
                contexts.put(destUser, new LinkedList<Document>());
            }catch(Exception ab){//Eccezione per la chiusura della connessione del client
                contexts.remove(destUser);
                continue;
            }
        } else {
            /*
            Il messaggio viene accodato solo se la sessione non è ancora scaduta, perche'
            e' scaduta allora l'utente con la sua coda viene rimosso dalla struttura quind
            e' impossibile accodargli un messaggio.
            */
            ((LinkedList<Document>) value).addLast(data);
        }
    }
```

Da come si può notare, viene iterata la lista degli utenti gestiti dal broker e per ogni utente viene verificato il tipo di oggetto associato: l'utente in base alla struttura contexts

```
HashMap<String, Object> contexts = new HashMap<String, Object>();
```

può avere associato uno dei due oggetti presentati di seguito: un oggetto di tipo LinkedList oppure un oggetto di tipo AsyncContext.

L'oggetto LinkedList è impiegato per la creazione di una coda di messaggi che viene associata all'utente che non ha una connessione attiva; in modo di recapitargli i messaggi accodati nella connessione successiva.

L'oggetto AsynContext rappresenta invece la connessione associata ad un particolare utente, e se esiste (durante questo ciclo nell'operation push) permette al broker di inoltrargli la notifica.

Per l'invio della notifica viene creato un oggetto OutputStream e successivamente tramite il metodo mngXML.transform il messaggio viene serializzato e trasmesso come flusso al client.

4.3.3 POP OPERATION

In questa operation avviene il collegamento vero e proprio, in cui il broker blocca la request del client e la rilascia solo se dispone di informazioni da notificargli.
Segue il codice dell'operation pop.

```
case "pop":
    user = (String) session.getAttribute("user");
    System.out.println("pop received from: " + user);
    boolean async;
    synchronized (this) {
        LinkedList<Document> list = (LinkedList<Document>) contexts.get(user);
        if (async = list.isEmpty()) {
            request.setAttribute("org.apache.catalina.ASYNC_SUPPORTED", true);
            AsyncContext asyncContext = request.startAsync();
            asyncContext.setTimeout(5000);  //Timeout richiesto per bloccare la risposta
            asyncContext.addListener(new AsyncAdapter() {
                @Override
                public void onTimeout(AsyncEvent e) {
                    String user = "";
                    try {
                        AsyncContext asyncContext = e.getAsyncContext();
                        user = (String) ((HttpServletRequest) asyncContext.getRequest()).get
                        System.out.println("timeout event launched for: "+ user);
                        ManageXML mngXML = new ManageXML();
                        Document answer = mngXML.newDocument();
                        answer.appendChild(answer.createElement("timeout"));
                        boolean confirm;
                        synchronized(Broker.this) {
                        if (confirm = (contexts.get(user) instanceof AsyncContext))
                            contexts.put(user,new LinkedList<Document>());
                        }
                        if (confirm) {
                        OutputStream tos = asyncContext.getResponse().getOutputStream();
                        mngXML.transform(tos, answer);
                        tos.close();
                        asyncContext.complete();
                        }
                    } catch (Exception ex) {
```

```
                    System.out.println(ex);
                    contexts.remove(user);
                }
            }
        });
        contexts.put(user, asyncContext);
    } else {
        answer = list.removeFirst();
    }
}
if (!async) {
    os = response.getOutputStream();
    mngXML.transform(os, answer);
    os.close();
}
break;
```

In questo codice si nota che inizialmente si estrae lo user tramite l'oggetto session e successivamente si verifica se la sua coda dispone di messaggi che potrebbero essergli stati lasciati dal broker durante la sua assenza.
Questo controllo avviene tramite il seguente codice

```
LinkedList<Document> list = (LinkedList<Document>) contexts.get(user);
if (async = list.isEmpty()) {
```

Se la coda non è vuota, allora si estrae un elemento e lo si restituisce al client che si ricollega subito dopo rieseguendo una nuova request e ripetendo lo stesso processo finché la coda non si svuota: il meccanismo di estrazione di un elemento dalla coda e della sua restituzione è dato dalla porzione di codice che segue.

```
    } else {
        answer = list.removeFirst();
    }
}
if (!async) {
    os = response.getOutputStream();
    mngXML.transform(os, answer);
    os.close();
}
break;
```

Una volta svuoltata la coda, il controllo entra nel cuore dell'operation bloccando la richiesta tramite la chiamata

```
AsyncContext asyncContext = request.startAsync();
```

setta il timeout di blocco della request ed aggiunge un listener che si attiva quanto si raggiunge il timeout.
In questa fase il collegamento tra broker e client è aperto e se arriva una notifica tramite metodo push, questa viene comunicata istantaneamente ai client connessi, od eventualmente accodata per gli utenti momentaneamente disconnessi.
Quando si attiva il timeout, si crea un messaggio di risposta al client e si chiude la connessione invalidandola.
Dopodiché il broker è di nuovo in ascolto per eventuali richieste di connessione.

4.4 *IMPLEMENTAZIONE CLIENT PER BOSH*

Il Client è stato implementato sempre in Java utilizzando lo stesso ambiente di sviluppo impiegato per il sistema GCM: Eclipse ADT con integrato l'SDK Android (chiamato anche Eclipse ADT Bundle).
A differenza della configurazione prevista per il client GCM, non è richiesta la libreria google-play-services_lib né i metodi e le funzionalità impiegate dal sistema GCM.
In questo modello la fase di registrazione prevista per il GCM non è più richiesta.
Inoltre l'architettura è notevolmente semplificata e ricade nel paradigma client – server: l'aspetto sostanziale è dato dal fatto che la connessione al server è diretta e non passa come nel modello precedente tramite un terzo elemento (il GCM).
La novità di questo approccio, rispetto il precedente, è la completa revisione e riprogettazione del sistema di connessione previsto per l'interazione tra l'applicazione android ed il servizio.
Il progetto è composto dalle seguenti classi principali:

- MainActivity
- ServerBroadcastReceiver

- ServerMessageHandler
- CustomInfoWindow
- VibrateService
- AlertService

Le ultime tre classi sono identiche a quelle presentate nel progetto GCM, per cui per una loro trattazione si rimanda al contenuto descritto in quei paragrai.

4.4.1 CLASSE MAINACTIVITY

La classe MainActivity, similmente alla classe MainActivity del sistema GCM, è il cuore operativo dell'applicazione android.
La parte più rappresentativa e centrale dell'applicazione è identica ai metodi esposti dalla MainActivity del GCM, ma con la differenza che in questo modello è stata esclusa la fase di registrazione che era invece richiesta dal GCM; quindi presenta le stesse identiche funzionalità salvo per i metodi previsti per l'interazione e registrazione col GCM.
Le sue funzioni centrali sono sempre onCreate, loadMap, DownloadTask, myLocationListener quest'ultima definita dall'interfaccia LocationListener.
E' stata esclusa dall'elenco la parte relativa alla registraizone, data dal metodo: getRegId.
Come descritto in precedenza, la funzione onCreate permette di impostare la configurazione di avvio dell'applicazione istanziando ad esempio gli oggetti base dell'app come la mappa e l'itinerario. Il metodo loadMap permette di caricare su una mappa i punti di un itinerario, la classe DownloadTask si occupa di effettuare il download di un file d'itinerario; mentre l'oggetto myLocationListener fornisce il monitoraggio della posizione GPS, associando a queste posizioni l'azione relativa. Per una trattazione più approfondita si rimanda al codice descritto nell'implementazione client per gcm.

4.4.2 CLASSE SERVICEBROADCASTRECEIVER

La classe ServerBroadcastReceiver è una delle novità introdotte da questo modello, che rispetto al sistema precedente permette di rilevare diversi eventi ed azionare l'avvio del servizio che si occupa del collegamento col Broker.
La classe estende l'oggetto BroadcastReceiver utilizzato per catturare eventi che si verificano nel sistema Android.
Questo oggetto è un componente in grado di registrare eventi e di notificarli all'applicazione: eventi che vanno dalla ricezione di un messaggio, alla batteria in esaurimento, al display spento e tanti altri che se gestiti possono essere monitorati per azionare delle azioni in reazione a quegli eventi.
Segue il codice implementato del BroadcastReceiver.

```
public class ServerBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {

        if (intent.getAction().equals(ConnectivityManager.CONNECTIVITY_ACTION)) {
            ConnectivityManager connMgr = (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);
            NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();

            if(networkInfo != null && networkInfo.isConnected()){
                Intent brokerserver = new Intent(context, ServerMessageHandler.class);
                context.startService(brokerserver);
                Toast.makeText(context, "Connected", Toast.LENGTH_LONG).show();
            }else{
                Intent brokerserver = new Intent(context, ServerMessageHandler.class);
                context.stopService(brokerserver);
                Toast.makeText(context, "Not Connected", Toast.LENGTH_LONG).show();
            }
        }else if (intent.getAction().equals(Intent.ACTION_USER_PRESENT)) {
            Intent brokerserver = new Intent(context, ServerMessageHandler.class);
            context.startService(brokerserver);
            Toast.makeText(context, "USER_PRESENT", Toast.LENGTH_LONG).show();
        }


    }
```

Il metodo onReceive si attiva ogni volta che si verifica un evento di tipo CONNECTIVITY_ACTION oppure ACTION_USER_PRESENT.
Questi eventi registrati opportunamente nel file AndroidManifest.xml permettono al metodo onReceive di avviarsi ad ogni azione corrispondente.
Il codice imputato al risveglio del BroadcastReceiver è il seguente

```
<receiver android:name="com.example.brokerguide.ServerBroadcastReceiver">
    <intent-filter>
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
        <action android:name="android.intent.action.USER_PRESENT" />
    </intent-filter>
</receiver>
```

L'evento di tipo CONNECTIVITT_ACTION si verifica ad ogni cambio di connessione: si può verificare ad esempio se si perde la connessione, oppure se avviene un cambio di connessione da rete GSM a wifi o viceversa; in sostanza si verifica quando cambia lo stato di connessione.
Mentre l'evento ACTION_USER_PRESENT si verifica quanto l'utente sblocca il display del proprio dispositivo.
Questi eventi, una volta attivati, avviano un servizio chiamato ServerMessageHandler che si occupa di instaurare la connessione col server.

4.4.3 CLASSE SERVICEMESSAGEHANDLER

La classe ServerMessageHandler è la classe più rilevante del progetto, si occupa di gestire la connessione col server in maniera del tutto autonoma indipendentemente dal ciclo di vita della MainActivity.
ServerMessageHandler estende la classe service e ne ridefinisce i metodi principali.
Questo processo, una volta avviato, instaura una connessione con server per ricevere le notifiche d'interesse.
Le parti principali della classe sono:

- onStartCommand: è il metodo invocato in fase di avvio dal servizio.
- Viewer: è la classe che si occupa di gestire la connessione col server
- showNotification: permette di gestire la notifica del messaggio inviando un avviso sulla barra di stato del dispositivo.

Segue il codice di onStartCommand

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {

    if(viewer == null || !viewer.isAlive()){
        viewer = new Viewer();
        viewer.start();
    }

    return START_STICKY;
}
```

Il metodo viene invocato ogni volta che si avvia il servizio: sia per la prima invocazione che per le successive.

All'interno del suo scope, viene avviato il thread che si occupa della connessione col server, ovvero l'oggetto viewer.

Prima di procedere al suo avvio, il metodo esegue un controllo di verifica in caso fosse già attivo.

Segue il codice dell'oggetto viewer.

```
class Viewer extends Thread {

    public void run() {
        try {
            hc.setBase(new URL(BASE));
            mngXML = new ManageXML();
            Document data = mngXML.newDocument();
            Element root = data.createElement("login");
            root.appendChild(data.createTextNode(MainActivity.imei));
            data.appendChild(root);

            answer = hc.execute("Broker", data);

            while (true) {
                data = mngXML.newDocument();
                root = data.createElement("pop");
                data.appendChild(root);
                // Riceve le risposte di tipo push
                Document answerView = hc.execute("Broker", data);

                if (answerView.getDocumentElement().getTagName().equals("push")) {
                    titolo = answerView.getDocumentElement().getChildNodes().item(0).getTextContent().toString();
                    System.out.println(titolo);
                    messaggio = answerView.getDocumentElement().getChildNodes().item(1).getTextContent().toString();
                    System.out.println(messaggio);
                    MainActivity.filekml = answerView.getDocumentElement().getChildNodes().item(2).getTextContent().toSt
                    System.out.println(MainActivity.filekml);

                    showToast();

                    Log.i("Server", "Received : (" + titolo + ")  ");

                    // Attivo notifica
                    if (titolo != null && messaggio != null && !"".equals(titolo.trim()) && !"".equals(messaggio.trim())
                        showNotification(getApplicationContext());

                }

            }
```

Questa classe viene eseguita come thread per rendere più indipendente il processo.

Nella fase inziale il thread compone un messaggio con il contenuto corrispondente all'operation che vuole far eseguire al server.

In questo caso l'operation è "login" che permette al server di associare a quel dispositivo una sessione per identificarlo.

Ottenuta la risposta dal server, si entra in un ciclo while in cui si richiede costantemente l'attivazione di una connessione tramite l'operation "pop" che permette al server di bloccare la request della richiesta client e di restituire contenuti solo quando disponibili.

Per cui il client una volta attivata la connessione col server (attraverso l'operation "pop") si blocca in attesa di risposta. Il server tiene bloccata la request del client rilasciandola solo quando ha informazioni da inviargli.

Una volta rilasciata la connessione, il client riceve la notifica e la processa ed in seguito si ricollega col server: questo meccanismo di comunicazione è chiamato Long Polling.

Il metodo invocato dal viewer, ottenuta la notifica dal server, è showNotification; che permette di avvisare l'utente inoltrando la notifica sulla barra di stato.

```
private void showNotification(Context con) {

    // Context context = getApplicationContext();
    Intent intent = new Intent(con, MainActivity.class);
    intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_SINGLE_TOP);
    intent.putExtra("DIALOG", true);
    PendingIntent pending = PendingIntent.getActivity(con, 0, intent, 0);

    Bitmap aBitmap = BitmapFactory.decodeResource(getResources(), R.drawable.ic_stat_gcm);

    NotificationManager ns = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

    Notification notify = new Notification.Builder(con)
            .setContentTitle(titolo).setContentText(messaggio)
            .setSmallIcon(R.drawable.ic_stat_gcm).setLargeIcon(aBitmap)
            .setAutoCancel(true).setDefaults(Notification.DEFAULT_SOUND)
            .setContentIntent(pending).build();

    ns.notify(++index, notify);

}
```

Come mostra il codice, quando viene invocato il metodo showNotification si attiva la MainActivity ed appare sulla barra di stato la notifica ricevuta.

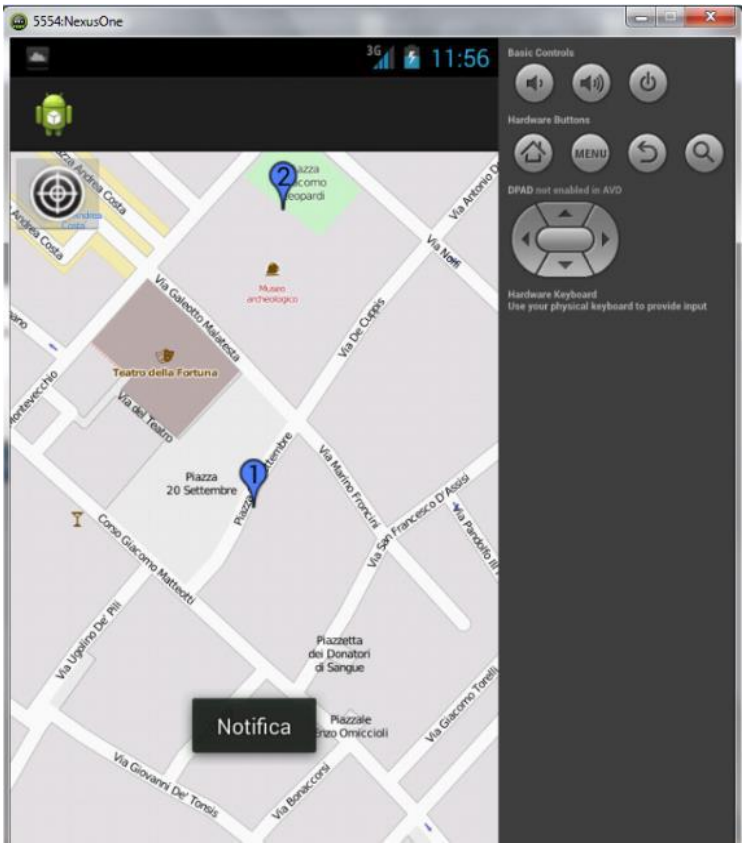


Figura 41 - Notifica di aggiornamento

4.5 *IMPLEMENTAZIONE INTERFACCIA AMMINISTRATIVA*

L'implementazione dell'interfaccia amministrativa, permette di azionare le notifiche verso i dispositivi e di caricare i file per gli itinerari ed i file multimediali. L'applicazione web è stata realizzata con http, javascript per la parte client side, e jsp per la parte server side. Inoltre sono state utilizzate le librerie jquery per la parte javascript e per le funzionalità ajax e bootstrap per l'interfaccia grafica.
Le parti che compongono l'applicazione web sono le seguenti:

- index
- admin
- broker
- upload

4.5.1 INDEX

Rappresenta il punto di accesso all'applicazione e la sua struttura è composta di un template iniziale (index.html) che fornisce le modalità di autenticazione.

Figura 42 - Login

Ottenuto l'accesso all'area amministrativa, si accede nel cuore operativo dell'applicazione.

Segue il codice che si occupa della fase di login

```
<script type="text/javascript">
    function loginHash() {
        var password = $.sha1($('#inputPassword').val());
        var username = $('#inputName').val();

        document.frm.inputPassword.value = password;
        document.frm.submit();

        return false;
    }
</script>

</head>
<body>
<div class="bs-example">
    <form name="frm" style="width:400px; margin: 0 auto;" action="login.jsp" method="POST"

        <div class="form-group">
            <label for="inputName">Name</label>
            <input class="form-control" type="text" id="inputName" name="username" placehol
        </div>
```

La fase di autenticazione richiede l'invio dei parametri nome utente e password al server, più precisamente alla pagina login.jsp che si occupa di controllare la validità delle credenziali. Se l'utente viene riconosciuto, accede alle funzionalità amministrative.

4.5.2 ADMIN

La jsp admin rappresenta l'interfaccia che permette di inviare notifiche ai dispositivi tramite il sistema GCM, più esattamente invia la richiesta di notifica passandola al GCM Service.

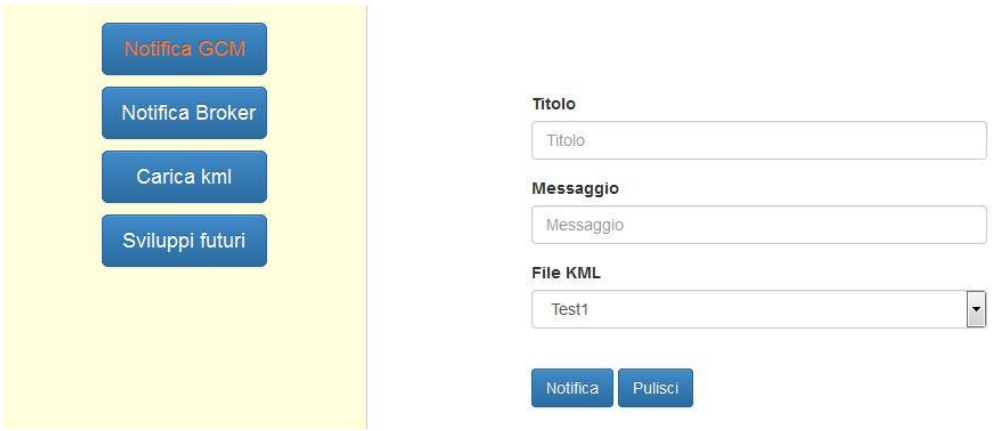


Figura 43 - Notifica GCM

Il codice che si occupa di questo è appunto admin.jsp

```
function inviaNotifica() {
    var titolo = $('#titolo').val();
    var messaggio = $('#messaggio').val();
    var filekml = $('#filekml').val();

    $.ajax({
        url: "../Notifica",
        type: 'POST',
        data: {titolo: titolo, messaggio: messaggio, filekml: filekml},
        contentType: "application/x-www-form-urlencoded; charset=utf-8",
        dataType: "json",
        success: function(data) {
            var obj = eval(data);
            alert(obj.result);
        }
    });
    return false;
}
```

La funzione invia un messaggio http di tipo post al servizio Notifica che provvede a spedirlo al GCM, ed attende una risposta con contenuto formattato in json (dataType:"json").

4.5.3 BROKER

La jsp broker rappresenta l'interfaccia che permette di inviare notifiche ai dispositivi tramite connessione diretta con tecnologia BOSH, più esattamente

invia la richiesta di notifica senza passarla al GCM Service come invece avviene nel sistema precedente.

La sua interfaccia è mostrata dal focus della voce di menù indicato in figura.

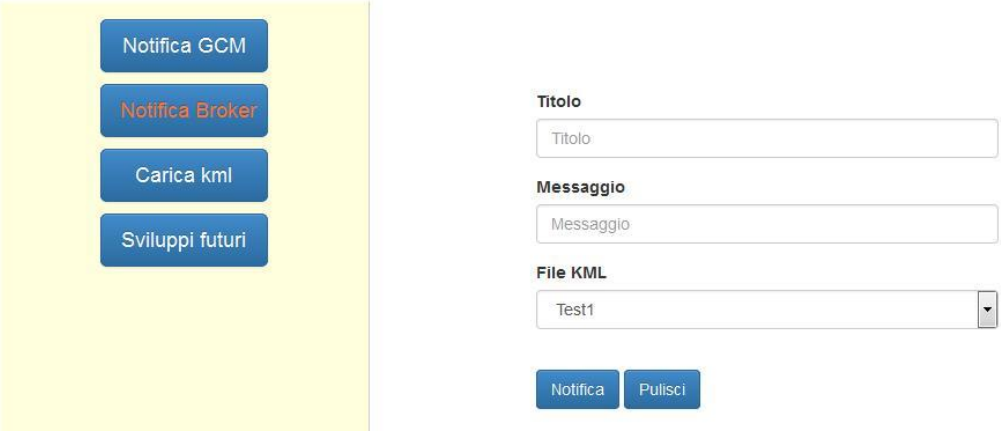


Figura 44 - Notifica Bosh

Segue il codice che gestisce l'interfaccia

```
function sendMessageToBroker() {
    var titolo = $('#titolo').val();
    var messaggio = $('#messaggio').val();
    var filekml = $('#filekml').val();
    var messaggio = "<push><titolo>"+titolo+"</titolo><messaggio>"+messaggio+"</messaggio><filekml>"+filekml+"</filekml></push>";

    $.ajax({
        url: "../Broker",
        type: 'POST',
        processData: false,
        //data: {titolo: titolo, messaggio: messaggio, filekml: filekml},
        //data: "<push><titolo>ciao</titolo></push>",
        data: messaggio,
        //contentType: "application/x-www-form-urlencoded; charset=utf-8",
        dataType: "xml",
        success: function(data) {

            alert(data.documentElement.childNodes[0].nodeValue);

        }
    });
    return false;
}
```

Il codice invia un messaggio al Broker aspettando come valore di ritorno un oggetto formattato in xml: per la risposta sull'esito del processo.

A questo punto è il Broker che si prende carico di inoltrare il messaggio ai dispositivi.

4.5.4 UPLOAD

La jsp upload fornisce un'interfaccia per il caricamento dei file sul server.
Questa interfaccia mette a disposizione un pulsante da cui è possibile selezionare i file e caricarli così in maniera agevole.
L'interfaccia si presenta nel seguente modo.

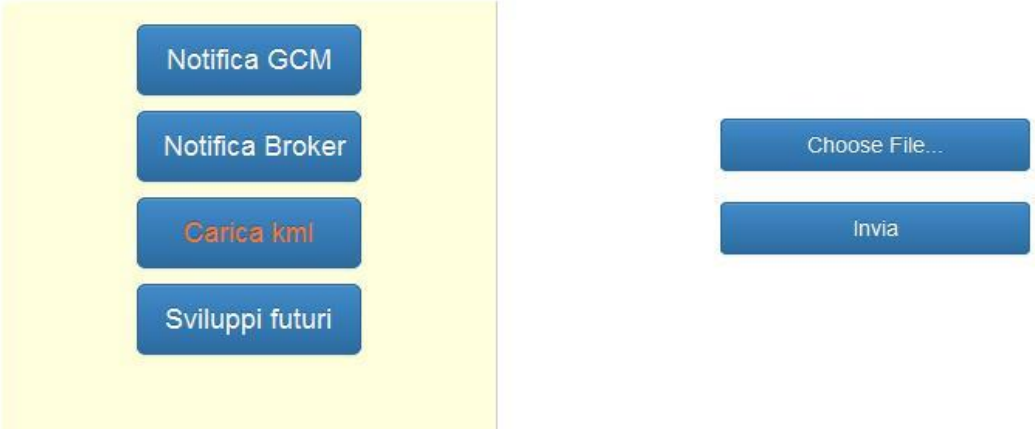


Figura 45 - Upload

Ed il codice che gestisce la sua implementazione è il seguente.

```
//Verifico se si tratta di un file kml oppure di altro: se e' kml va mes
if(saveFile.toLowerCase().contains("kml")){
    url = context.getRealPath("/WEB-INF/kml");
    saveFile = url + "/" + saveFile;
    File ff = new File(saveFile);
    FileOutputStream fileOut = new FileOutputStream(ff);
    //fileOut.write(dataBytes, startPos, (endPos - startPos));
    fileOut.write(dataBytes, startPos, (boundaryLocation - startPos));
    fileOut.flush();
    fileOut.close();
}else{
    url = context.getRealPath("/WEB-INF/filemultimediali");
    saveFile = url + "/" + saveFile;
    File ff = new File(saveFile);
    FileOutputStream fileOut = new FileOutputStream(ff);
    //fileOut.write(dataBytes, startPos, (endPos - startPos));
    fileOut.write(dataBytes, startPos, (dataBytes.length - startPos));
    fileOut.flush();
    fileOut.close();
}
```

Questa è la parte principale di upload.jsp che una volta ottenuto il file, lo salva nella directory opportuna: se è un file kml lo deposita in /WEB-INF/kml, altrimenti in /WEB-INF/filemultimediali.

Capitolo 5

5.1 *Funzionamento dell'interfaccia utente*

In questo paragrafo si evidenzia l'uso pratico delle applicazioni realizzate, mostrando le diverse interazioni che accompagnano gli scenari messi a disposizione dal sistema.
La guida introduttiva si articola in due parti: la parte riguardante il funzionamento dell'applicazione Android, e la parte riguardante il funzionamento dell'amministratore.
A titolo dimostrativo, verrà preso in esame uno dei due sistemi realizzati: sia il sistema GCM, sia il sistema Bosh sono identici riguardo le interazioni utente.

5.1.1 Funzionamento app

All'avvio dell'applicazione l'app si presenta nel seguente modo

Figura 46 - App principale

Come mostrato in figura, viene caricato sulla mappa un itinerario di default fornito con l'installer dell'applicazione Android: rappresentato dai punti blu numerati.

Questi punti di itinerario sono interattivi e permettono all'utente di selezionarli per visionare dettagli e contenuti multimediali associati.

Ad esempio, selezionando un punto di interesse, come mostrato nella figura seguente, si può riscontrare da subito un dettaglio del punto; per poi procedere alla visualizzazione dei suoi contenuti selezionando i pulsanti associati.

Figura 47 - App dettaglio

Segue uno screenshot del download di un video associato ad uno dei pulsanti sopra esposti.

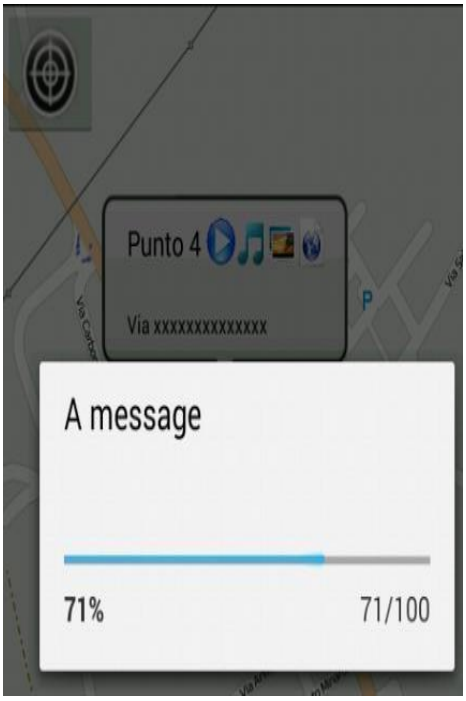


Figura 48 - App multimedia

Alla fine del download parte in automatico la visualizzazione del video: eventualmente potrebbe richiedere la selezione di uno dei programmi installati sul dispositivo tra i più idonei a gestire quel tipo di contenuto.

Un altro aspetto interessante è quello fornito dall'interazione di notifica.

Quando arriva una notifica, il dispositivo mostra sulla barra di notifiche un'icona, come mostra l'immagine seguente.

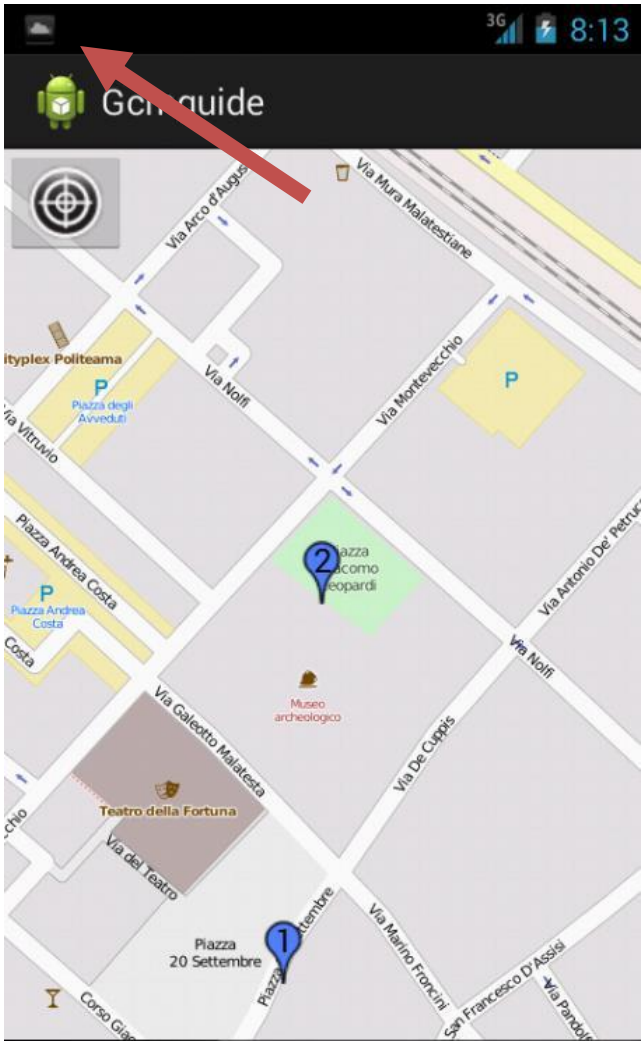


Figura 49 - App avviso

Selezionando la notifica appena arrivata, si presenta una finestra di dialogo a cui l'utente può rispondere rifiutando l'aggiornamento di itinerario oppure accettandolo.

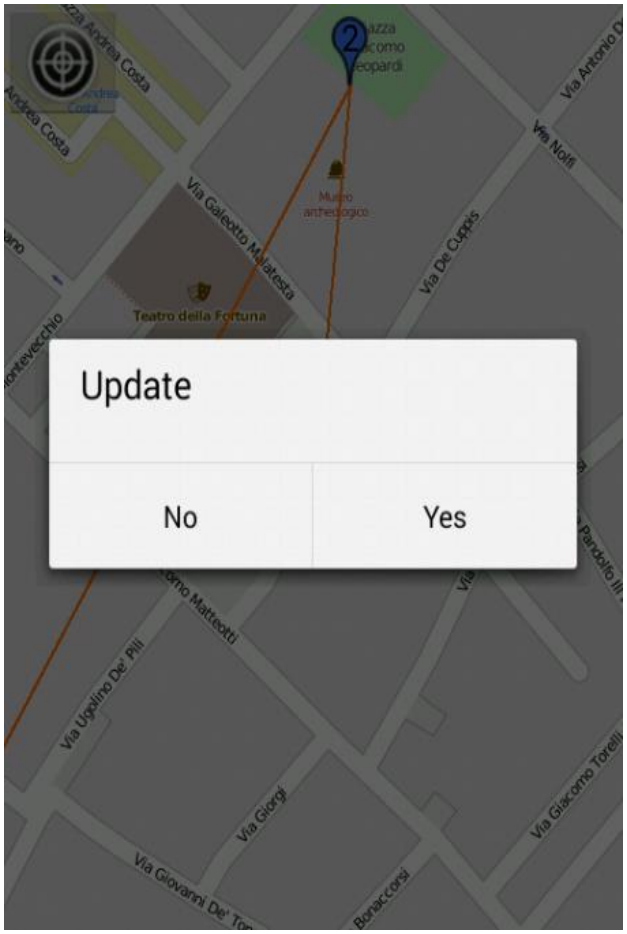


Figura 50 - App selezione

Se l'utente rifiuta, ritorna direttamente alla mappa; mentre se accetta si attiva il download del nuovo itinerario: che verrà caricato subito dopo il completamento del download.

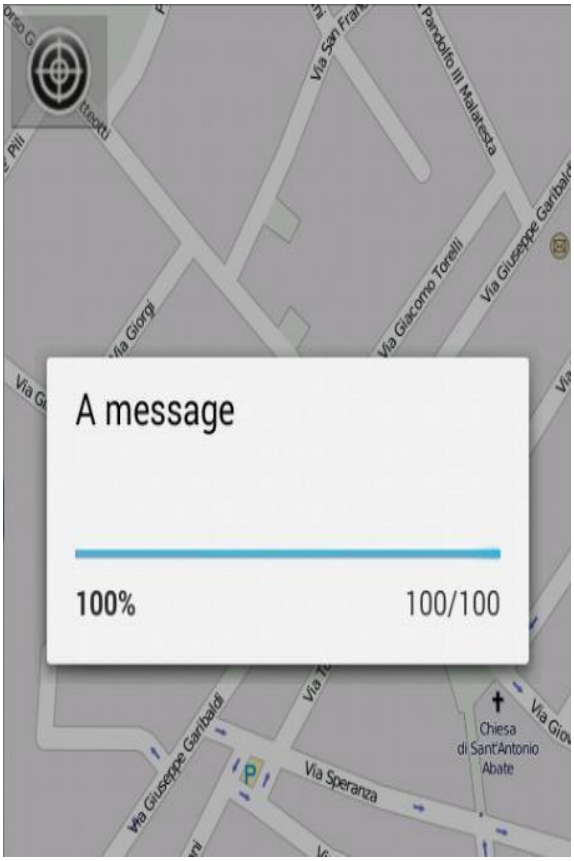


Figura 51 - App itinerario

A questo punto viene caricato il nuovo itinerario e l'utente può riprendere a visionare il nuovo percorso.

Un altro aspetto rilevante, è l'avviso di prossimità: quando l'utente si trova in vicinanza ad un punto di interesse, gli viene comunicato attraverso una vibrazione ed una notifica.

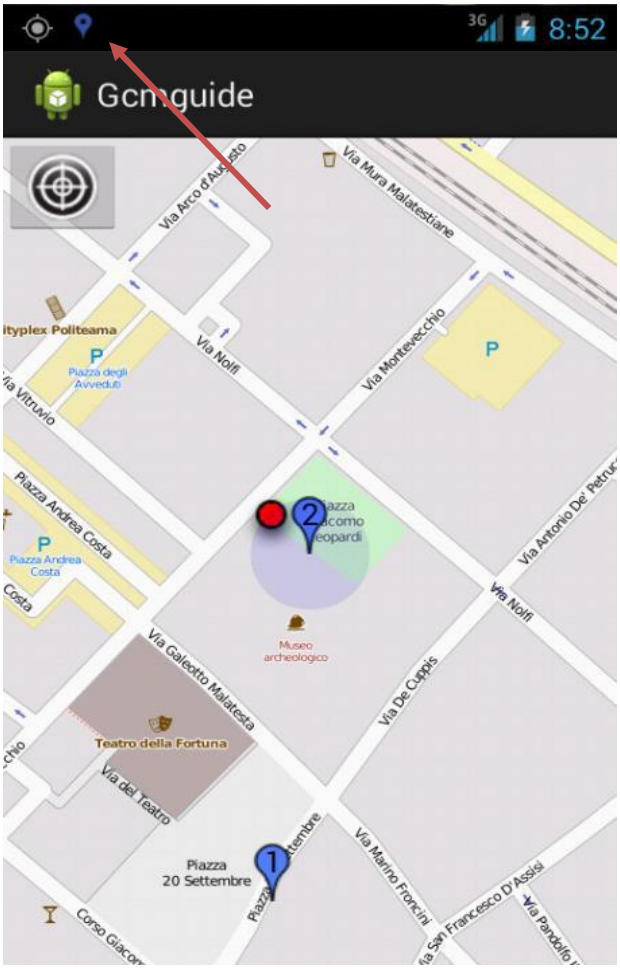


Figura 52 - App prossimità

Selezionando il messaggio, si viene proiettati esattamente sul punto a cui fa riferimento la notifica.

5.1.2 Funzionamento amministratore

La parte amministrativa permette ad un amministratore di poter eseguire diverse operazioni, quali: l'invio di una notifica ed il caricamento di file di itinerario e multimediali.

Accedendo all'interfaccia amministrativa, ci troviamo di fronte diverse opzioni

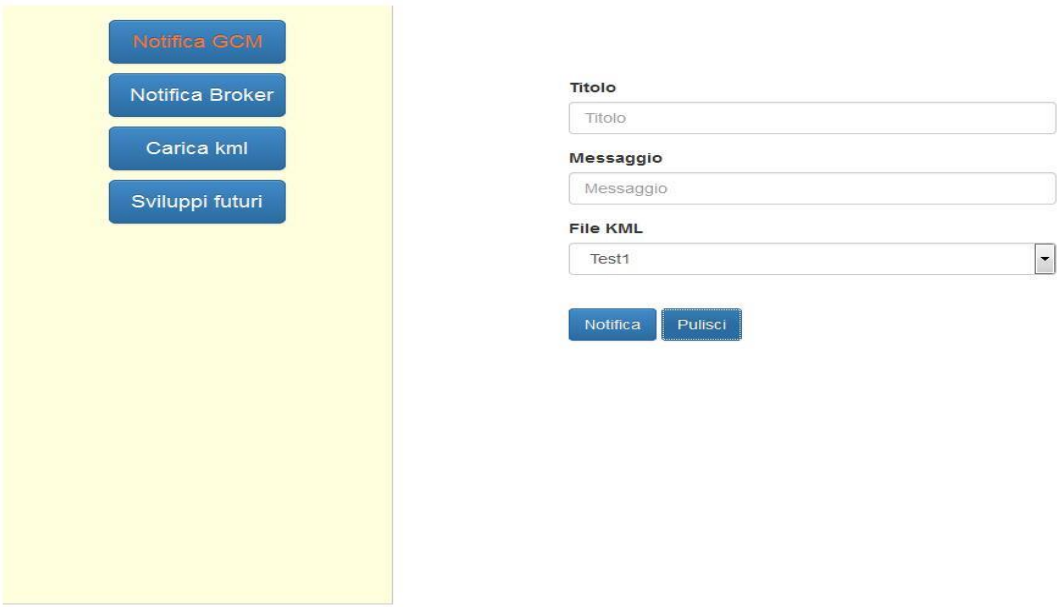


Figura 53 - Amministrazione iniziale

Le due voci principali riguardano le notifiche dei messaggi e permettono (sia per la Notifica GCM, sia per la Notifica Broker) di inviare una notifica ai dispositivi registrati.

Il funzionamento è al quanto semplice e consiste nel compiare la form laterale indicando esattamente il titolo, il corpo del messaggio ed il nome del file di itinerario che si vuole far scaricare ai dispositivi.

Una volta completati i campi, si può procedere all'invio di notifica selezionando il pulsante "Notifica".

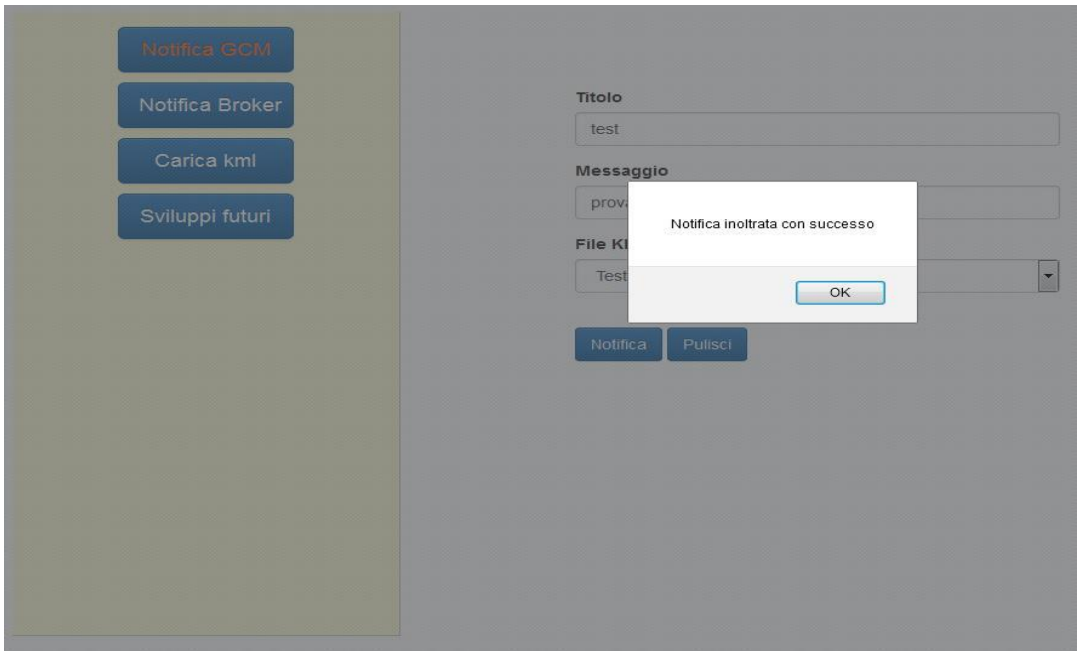


Figura 54 - Amministrazione notifica

Per quanto riguarda la parte di gestione dei file di itinerario e multimediale, accedendo all'area "Carica kml" si entra in un pannello da cui è possibile selezionare e caricare i file d'interesse.

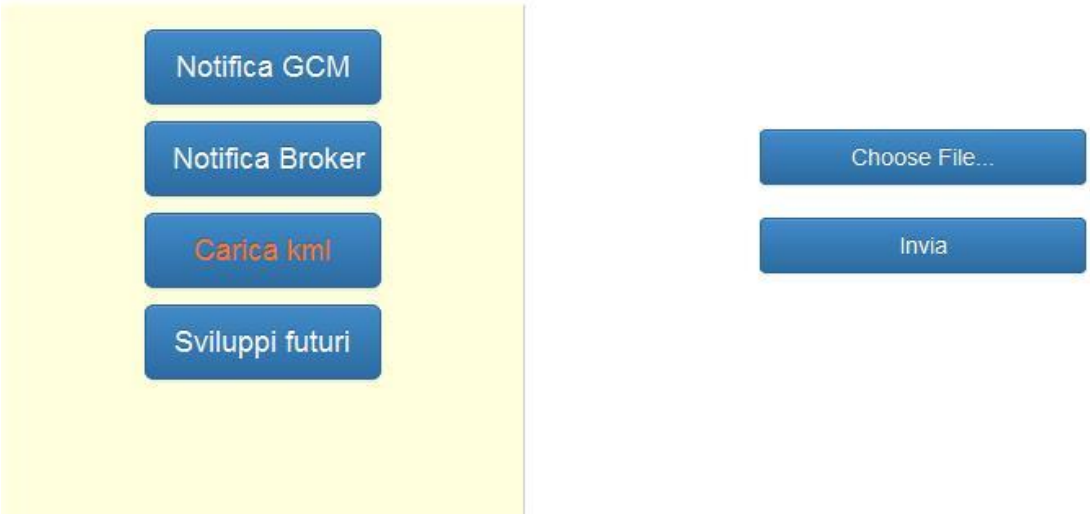


Figura 55 - Amministrazione upload

Selezionando un file, a seconda che si tratti di un file di itinerario (con estensione kml) o di un file multimediale (qualsiasi altro file che non abbia estensione kml), verrà caricato dal sistema nella directory opportuna.

Nel caso del file di itinerario, il sistema oltre a caricarlo nella directory di riferimento, provvede anche ad aggiungerlo al menù "File KML" visto in precedenza.

Capitolo 6

CONCLUSIONI

I servizi di notifiche push come Google Cloud Messaging mettono a disposizione sistemi in grado di comunicare coi propri dispositivi mobili.
Tuttavia la loro gestione è comunque limitata sia sul piano quantitativo, per esempio la dimensione massima dei messaggi od il numero massimo di dispositivi a cui poter inviare i messaggi, sia sugli interventi e personalizzazioni del servizio stesso: il GCM è visto come una scatola nera e non è possibile parametrizzarlo andando ad aggiungere strutture dati diverse da quelle proposte dal protocollo del servizio, per cui non è possibile ampliarlo od aggiungere e sviluppare altre funzionalità oltre a quelle offerte dal servizio.
Mentre per quanto riguarda il modello Bosh, così progettato, permette di avere diversi vantaggi rispetto ai sistemi GCM o similari.
Uno dei punti a favore di BOSH è la semplificazione dell'architettura incentrata non più su tre attori, come nel caso del GCM, ma su due entità che permettono di ridurre notevolmente le fasi di intermediazione: ne deriva una miglior gestione e manutenzione. Inoltre, il sistema BOSH non si presenta come scatola nera; per cui è possibile intervenire aggiungendo od ampliando le funzionalità di base offerte dal modello iniziale. Oltre a questo, va precisato che essendo di natura aperta e trasparente favorisce anche quei prerequisiti richiesti nel trattamento di informazioni private o sensibili in termini di sicurezza: in quanto uno dei principi della sicurezza informatica asserisce che non si può considerare sicuro un sistema che non si conosce.
Infine, va considerato anche un aumento prestazionale sull'abbattimento dei tempi di latenza dovuti ad un'architettura più diretta e quindi più veloce rispetto ai sistemi GCM.

Senza considerare la parte relativa alla configurazione del sistema: il GCM è molto più complesso e vincolante del sistema presentato; non è possibile ad esempio inviare messaggi con carichi utili superiori ai 4kb, così come non è possibile inoltrare un messaggio a più di 1000 destinatari in un'unica notifica.

Queste limitazioni vengono superate dal modello presentato.

Per quanto riguarda gli sviluppi futuri, le soluzioni fornite sebbene riguardino un'applicazione specifica per la gestione delle notifiche per dispositivi mobili, possono essere generalizzate all'uso di altri servizi, ad esempio: servizi di news, di chat, di forum, ecc. Un possibile sviluppo di questo genere potrebbe prevedere, sempre a carico del WS, la fornitura dei servizi menzionati; si potrebbe gestire ad esempio un servizio di news estendendo le funzionalità attraverso l'integrazione del protocollo publish-subscribe.

Un altro sviluppo potrebbe riguardare la portabilità del progetto: estendendo tramite implementazioni specifiche l'applicazione ad altri ambienti di esecuzione come ad esempio iOS di Apple oppure Windows phone.

Per quanto riguarda la gestione dei dati gestita dal WS (visto che non memorizza permanentemente l'informazione, ma la processa in memoria temporanea), si potrebbe ottimizzarla facendo in modo che i messaggi vengano registrati permanentemente fornendo così una certa persistenza dei dati.

Questo sistema garantirebbe al client di recuperare i messaggi anche in caso di guasti del Server di notifica o disconnessione.

Per quanto riguarda i file multimediali, un ulteriore sviluppo potrebbe riguardare la trasmissione dei file in streaming: attualmente è previsto il download completo dei contenuti multimediali per poterli visionare; per cui se l'utente non riesce a scaricare completamente un video, non può visionarlo in alcun modo. Mentre in streaming potrebbe visionarlo anche parzialmente, oltre a vederlo istantaneamente non appena aziona la richiesta.

Infine un'altra considerazione va rivolta alle notifiche che nel sistema presentato vengono impiegate solo per comunicare i file di itinerario: si potrebbe impiegare lo stesso meccanismo per notificare non solo contenuti diversi, ma anche azioni che potrebbero permettere di pilotare od attivare processi di elaborazione nell'applicazione Android: ad esempio, aggiungere in calendario una nota di un appuntamento inoltrata dal datore di lavoro ai propri dipendenti.

BIBLIOGRAFIA

[1] Wikipedia, http://en.wikipedia.org/wiki/Push_technology

[2] Leonard Richardson and Sam Ruby, RESTful Web Services

[3] Wikipedia, http://it.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[4] OpenStreetMap, http://www.openstreetmap.org/about

[5] Google Cloud Messaging for Android, https://developer.android.com/google/gcm/index.html

[6] Wikipedia, http://en.wikipedia.org/wiki/BOSH

[7] Wikipedia, http://en.wikipedia.org/wiki/Representational_state_transfer

[8] Guida Osmdroid, https://code.google.com/p/osmdroid/

[9] Guida Osmbonuspack, https://code.google.com/p/osmbonuspack/

[10] Wikipedia, http://it.wikipedia.org/wiki/XML

[11] Wikipedia, http://it.wikipedia.org/wiki/Business-to-business

[12] Wikipedia, http://it.wikipedia.org/wiki/World_Wide_Web

[13] Wikipedia, http://it.wikipedia.org/wiki/HTML

[14] Wikipedia, http://it.wikipedia.org/wiki/JSON

[15] Articolo su REST, http://www2.mokabyte.it/cms/article.run?permalink=mb168_restmaturitymodel-1

[16] Wikipedia, http://it.wikipedia.org/wiki/Service-oriented_architecture

[17] G. Della Penna, Web Service Architecture, www.di.univaq.it/gdellape

[18] Articolo su RESTful web service, https://www-10.lotus.com/ldd/lqwiki.nsf/dx/restfulServices-Basics.pdf/$file/restfulServices-Basics.pdf

[19] Documentazione Oracle sui RESTful web service per Netbeans, http://www.oracle.com/technetwork/documentation/index.html

[20] Wikipedia, http://it.wikipedia.org/wiki/Android

[21] Tesi F. Forcolin, Sviluppo di App per sistema operativo Android

[22] Eclipse ADT, http://developer.android.com/sdk/index.html

[23] Wikipedia, http://it.wikipedia.org/wiki/Google_Maps

[24] Guida Osmdroid, https://github.com/osmdroid/osmdroid

[25] Guida JAK, http://labs.micromata.de/projects/jak.html

[26] Wikipedia, http://it.wikipedia.org/wiki/Keyhole_Markup_Language

[27] Tesi F. Nuzzo, Analisi e progettazione di architetture private cloud per servizi di push notification verso dispositivi mobile

[28] Tesi M. Ambrosin, Realizzazione componente push in ambiente Google Web Toolkit

[29] Wikipedia, http://it.wikipedia.org/wiki/Socket_(reti)

[30] Wikipedia, http://it.wikipedia.org/wiki/WebSocket

[31] Lucidi F. Scioscia, Linguaggi e tecnologie web

[32] Wikipedia, http://en.wikipedia.org/wiki/Server-sent_events

[33] Wikipedia, http://it.wikipedia.org/wiki/HTML5

[34] Wikipedia, http://it.wikipedia.org/wiki/World_Wide_Web_Consortium

[35] Wikipedia, http://it.wikipedia.org/wiki/Publish/subscribe

[36] Wikipedia, http://it.wikipedia.org/wiki/Extensible_Messaging_and_Presence_Protocol

[37] Seminario R. Bertolini e F. Felice, Seminario di Protocolli di reti: XMPP http://www.slideshare.net/fabio87/xmpp-1486152

[38] Wikipedia, http://it.wikipedia.org/wiki/MQTT

[39] Specifiche MQTT, http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html

[40] Riferimenti KML, https://developers.google.com/kml/documentation/kmlreference