

# Università degli Studi di Salerno

Facoltà di Scienze Matematiche, Fisiche e Naturali



Corso di Laurea in Informatica

## ***SVILUPPO DI APPLICAZIONI GOOGLE-BASED CON TECNOLOGIE WMS***

Relatore

Ch.ma Prof.ssa

Monica Sebillo

Candidato

Vittorio Paragallo

Matr.556/001873

Anno Accademico 2007/2008

UNIVERSITÀ DEGLI STUDI DI SALERNO

SUNTO

Sviluppo di applicazioni “Google-based” con tecnologie WMS

a cura di Vittorio Paragallo

556/001873

Relatore:

Prof.sa Monica Sebillo

Dipartimento di Matematica e Informatica

Tesi relativa allo sviluppo di sistemi stand-alone che facciano uso di mappe Google, con uno studio sui prodotti già esistenti, una verifica sulla fattibilità di sviluppo di tali applicazioni e sulla possibilità di integrazione con servizi WMS.

Questo documento è diviso in sei capitoli

Il primo capitolo introduce il contesto in cui si colloca la tesi.

Il secondo capitolo espone, in forma descrittiva, gli obiettivi del progetto e l'analisi di sistemi già esistenti.

Il terzo capitolo mostra parte dello studio effettuato per comprendere il funzionamento e la struttura di Google.

Il quarto capitolo propone una libreria, per lo sviluppo di applicazioni google-based, descrivendone le componenti più importanti.

Il quinto capitolo descrive le funzionalità di un prototipo per dimostrare alcune funzionalità della libreria proposta.

Il sesto capitolo riassume il lavoro che è stato svolto e descrive i risultati ottenuti.

Infine due appendici:

L'appendice A mostra la possibilità di integrazione di dati in Google Earth e fornisce spiegazioni sui sistemi di coordinate adottati.

L'appendice B descrive un paradigma che ha delle relazioni con l'architettura di Google Maps.

## SOMMARIO

CAPITOLO 1 .....	1
<i>introduzione</i> .....	1
1. Rappresentazione di Mappe .....	4
2. Scopo del progetto .....	8
3. Tempificazione delle attività .....	9
4. Struttura del documento.....	10
CAPITOLO 2 .....	11
<i>requisiti</i> .....	11
Introduzione .....	11
1. Obiettivi generali.....	11
2. Sistemi correnti .....	12
2.1 Google Maps .....	12
2.1 Google Earth.....	15
2.1 GeoServer .....	17
3. Sistema proposto .....	18
3.1 Overview.....	19
3.2 Le sorgenti di informazione .....	19
3.3 Requisiti funzionali.....	20
3.4 Requisiti non funzionali .....	20

CAPITOLO 3 .....	23
<i>studio di fattibilità</i> .....	23
Conclusioni .....	29
CAPITOLO 4 .....	30
<i>una libreria per lo sviluppo</i> .....	30
DefaultTileFactory (pacchetto mapviewer) .....	32
ProviderBuilder .....	40
WMSService ( pacchetto org.jdesktop.swingx.mapviewer.wms) .....	40
JXMapView ( pacchetto org.jdesktop.swingx.mapviewer):.....	45
GeoMapView.....	46
OverlayMap.....	48
LegendComponent.....	51
MapLegend .....	52
DefaultLegendPane .....	52
MapObject.....	53
MapPaintableElement.....	54
ScreenShape .....	55
DefaultScreenPolygon, DefaultScreenLine, DefaultScreenPoint .....	56
Conclusioni .....	58
CAPITOLO 5 .....	59
<i>Un prototipo</i> .....	59
Modello dei casi d'uso.....	59
Interfaccia utente.....	63
CAPITOLO 6 .....	64
<i>conclusioni</i> .....	64
APPENDICE A.....	66
<i>approfondimenti su google earth</i> .....	66
APPENDICE B .....	76

<i>il paradigma REST</i> .....	76
REST: METTERE AL CENTRO LE RISORSE.....	77
REST: METODI DI ACCESSO ALLE RISORSE.....	78
<b>INDICE DELLE FIGURE .....</b>	<b>80</b>
<b>BIBLIOGRAFIA.....</b>	<b>81</b>
<b>INDICE.....</b>	<b>84</b>

## RINGRAZIAMENTI

Desidero esprimere un sentito ringraziamento alla professoressa Sebillio per avermi guidato nella stesura di questo documento. Grazie ai Luca Paolino e Davide De Chiara, collaboratori presso l'università degli studi di Salerno, per la disponibilità avuta nei miei confronti.

Un sentito ringraziamento, per l'affetto e l'aiuto che mi hanno saputo dimostrare, va ai miei genitori e a mia sorella, i quali mi hanno sempre sostenuto durante il percorso di studi ed aiutato nei momenti più difficili.

Ringrazio Elizaveta Zuravljova per tutti i momenti in cui, nonostante la distanza che ci separa, è sempre stata al mio fianco, e in particolare per avermi dato la forza di continuare e giungere al termine di questo percorso. In più la ringrazio, assieme Irina Kirchina e Anton Zuravljov, per avermi supportato durante il mio periodo di studio in Finlandia.

Благодарю Елизавету Журавлёву за то, что несмотря на дистанцию, которая нас разделяет, она всегда была рядом и помогла довести эту работу до конца. Также благодарю Киришину Ирину Михайловну и Журавлёва Антона за поддержку во время моей учёбы в Финляндии.<sup>8</sup>

Ringrazio le persone con cui lavoro ogni giorno: Salierno Pasquale, De Gianni Pasquale, Baldassarro Gerardo, Bardaro Guido, Del Vecchio Giuseppe, Lanza Antonio, D'isola Gabriele, Mattia Maria per la loro premurosità nei miei confronti.

---

<sup>8</sup> Scritto in russo per dare la possibilità a tutte le persone ringraziate di poter leggere i ringraziamenti nella propria lingua madre

Desidero infine esprimere la mia gratitudine all'ingegnere Carlo Mugnani e Giovanni Biancardi per la loro disponibilità e cortesia avute nei miei confronti, soprattutto durante gli ultimi giorni di stesura della tesi, per me più difficili.



## DICHIARAZIONE DI NON RESPONSABILITÀ

L'Autore declina ogni responsabilità per qualsiasi utilizzo improprio e/o illegale dei contenuti di questo documento.

Tutte le osservazioni, le considerazioni e il materiale presentato sono forniti a puro scopo didattico e non devono essere intesi in alcun modo come sollecito alla violazione dei termini di licenza emessi da Google.

## *Capitolo 1*

### INTRODUZIONE

La nascita del mappamondo di Google Earth nel giugno del 2005, ha segnato una nuova tendenza seguendo la quale, oggi, milioni di utenti sono in grado di utilizzare, per gli scopi più vari, immagini satellitari di straordinaria definizione, navigare attraverso le mappe di tutto il globo e localizzare su mappa obiettivi di varia natura.

La novità, lanciata in primis da Google e poi seguita da Microsoft e Yahoo, ha tagliato i legami con il passato permettendo una “democratizzazione della geografia”. Internet, infatti, non ammette filtri preventivi su ciò che viene inserito nel circuito globale, ma lascia ai propri utenti la facoltà di scegliere. La possibilità di interagire con la cartina, non limitata solo alla visualizzazione dei suoi contenuti, in teoria non ha limiti.

Quelle applicazioni geografiche, che sembravano utili al più per individuare un luogo o un indirizzo, stanno dando un inaspettato contributo all’evoluzione del rapporto dell’uomo con la rete, facendo diventare il web sempre più vicino ad una rappresentazione digitale della complessità che ci circonda.

Per poter comprender fino in fondo le applicazioni della nuova geografia digitale bisogna dare uno sguardo al passato.

Già dalla nascita dei primi computer si sono diffuse le mappe interattive utilizzate soprattutto in campo militare.

In questo contesto, è dominante la filosofia dei sistemi informativi geografici ( più noti come GIS<sup>1</sup>).

I GIS sono sistemi informatizzati per l'acquisizione, la memorizzazione, il controllo, l'integrazione, l'elaborazione e la rappresentazione dei dati che sono spazialmente riferiti alla superficie terrestre (Arnauld 1993).

Si fa riferimento ad una banca dati che permette l'interazione con molte altre, utilizzando come lingua comune riferimenti geografici assoluti come la latitudine e la longitudine.

I sistemi informativi geografici sono strumenti di analisi e pianificazione molto potenti che si prestano ad innumerevoli usi. Molti di questi in via di sviluppo, altri già facenti parte della vita di tutti i giorni; si pensi alla diffusione capillare di navigatori satellitari per auto che permettono di far interagire l'utente con le mappe corrispondenti al territorio in cui si trova, avere informazioni in tempo reale sulla viabilità, etc.

La produzione di cartografia chiara e precisa richiede una complessa struttura professionale che metta a disposizione fotografie aeree, rilevazioni satellitari e, soprattutto, aggiornamento costante.;

Tutto ciò scoraggia fortemente chi non ha la potenzialità di dar vita ad una rete globale di corrispondenti e specialisti.

---

<sup>1</sup> *Geographical Information System* - Sistema informativo geografico.

Tra i principali fornitori di mappe per uso stradale e/o utilizzabili in rete, si ricordano le società TeleAtlas<sup>2</sup> e NavTech<sup>3</sup> che si sono aggiudicate la gran parte del mercato.

Il servizio Google Maps viene lanciato nel febbraio 2005 e, grazie al ricorso estensivo alla tecnica Ajax<sup>4</sup>, definisce subito un nuovo stato dell'arte nella semplicità d'uso: spostamenti e zoom hanno tempi di attesa bassissimi, la visualizzazione è componibile e si ha la possibilità di sovrapporre a piacere diversi livelli interdipendenti, tra cui immagini satellitari di eccezionale risoluzione (Mda EarthSat e Digital Globe), mappe topografiche (TeleAtlas), informazioni locali (Google Local).

A distanza di soli quattro mesi dal giugno 2005, nasce Google Earth, un'applicazione stand-alone che, una volta caricata ed installata, oltre a consentire una visualizzazione tridimensionale delle mappe, permette ricerche geografiche complesse utilizzando l'enorme database georeferenziato dei motori di Mountain View<sup>5</sup>.

L'unico limite è la fantasia e questa, se consideriamo i risultati ottenuti da questi sistemi fino a oggi, sicuramente non manca.

---

<sup>2</sup> Uno dei più grandi provider di mappe digitali, <http://www.teleatlas.com/>

<sup>3</sup> Un altro fornitore di mappe digitali <http://www.navteq.com/>

<sup>4</sup> Asynchronous JavaScript and XML. È una tecnica di sviluppo web per la creazione di applicazioni interattive.

<sup>5</sup> Il **Googleplex** è il quartiere generale di Google, si trova a Mountain View, nella Contea di Santa Clara, in California, nei pressi di San Jose.

Sulle mappe di Google si possono evidenziare informazioni di qualunque tipo e genere: da dati statistici a informazioni di tipo eterogeneo (testi, immagini e video sono integrati tramite uso di file Xml<sup>6</sup>).

## 1. Rappresentazione di Mappe

Nella creazione di un sistema che utilizza mappe, è fondamentale analizzare la relazione che intercorre tra esse e la loro rappresentazione nello specifico dominio di interesse.

La rappresentazione delle mappe può essere vista in relazione a dei fattori tecnici e dei fattori estetici.

Le mappe, come altre forme di rappresentazione, sono il risultato di un'attività scientifica.

Il processo comprende la formulazione delle ipotesi, la raccolta dei dati, l'analisi, la revisione dei risultati e, di conseguenza, la valutazione della validità o meno dell'ipotesi iniziale. Questo processo è uno strumento fondamentale della scienza conosciuto come “processo ipotetico-deduttivo”.

Creare una mappa significa reperire un insieme di informazioni e prendere decisioni coerenti con l'impostazione “ipotetico-deduttiva”. Queste decisioni vanno dalla scelta delle modalità di rappresentazione al tipo di mappa da utilizzare e al modo di visualizzare simboli, ombre e una miriade di altri elementi. Tale metodo scientifico, però, permette una visualizzazione esatta e coerente delle informazioni, ma non dà alcun aiuto nelle scelte di design della mappa, come ad

---

<sup>6</sup> **eXtensible Markup Language**. È un metalinguaggio creato e gestito dal World Wide Web Consortium (W3C).

esempio la scelta dei colori, delle icone, delle dimensioni e della disposizione degli elementi sul pannello.

Queste ultime sono scelte che attengono all'estetica della cartografia e non sono meno rilevanti di quelle scientifico-procedurali.

Sfortunatamente i limiti tra estetica e scienza non sono ben definiti e ciò implica che scelte fatte in un campo potrebbero risultare incompatibili con l'altro.

Ad esempio un simbolo o componente grafica disegnata con un colore forte (scelta di design), può mettere a fuoco dei punti che, sotto una prospettiva puramente scientifica, dovrebbero essere meno enfatizzati.

Quindi l'obiettivo scientifico di una mappa potrebbe essere compromesso da scelte artistiche inappropriate e analogamente, scelte povere sotto l'aspetto scientifico possono avere delle ripercussioni negative sugli elementi artistici.

La mappa perfetta, partendo da dati e immagini, fonde arte e scienza in uno strumento di comunicazione visuale.

Nel valutare la scelta della mappa, bisogna considerare che con essa si vuole rappresentare qualche aspetto della realtà, così come avviene nel campo dei mass media. Come detto precedentemente le mappe sono una "rappresentazione digitale della complessità che ci circonda" pur nella considerazione che la realtà non può essere rappresentata nella sua interezza a causa della sua stessa complessità.

Tutto ciò che rappresenta la realtà e non è la realtà stessa è un'astrazione. In questo caso un'astrazione è un sottoinsieme limitato della realtà che ci permette di focalizzare il dominio di interesse riducendo la complessità d'analisi.

Per scegliere il livello di astrazione è necessario raggiungere dei compromessi.

Infatti:

- Un maggiore livello di astrazione rappresenta meno informazioni e quindi si allontana maggiormente dalla realtà, ma offre una maggiore semplicità e leggibilità migliorando la comunicazione visuale.
- Un minor livello di astrazione contiene una quantità maggiore di informazioni e quindi più si avvicina alla realtà, ma aumenta la complessità e peggiora la comunicazione visuale.

Questi compromessi devono essere tenuti in considerazione nella creazione e/o visualizzazione di mappe.

Sotto l'aspetto puramente tecnico, è importante dare la definizione di due tipi di mappe elettroniche:

**Mappa raster:** è una riproduzione elettronica di una mappa cartacea. Non dà la possibilità di interagire, è solo un'immagine (vedi figura1: ).

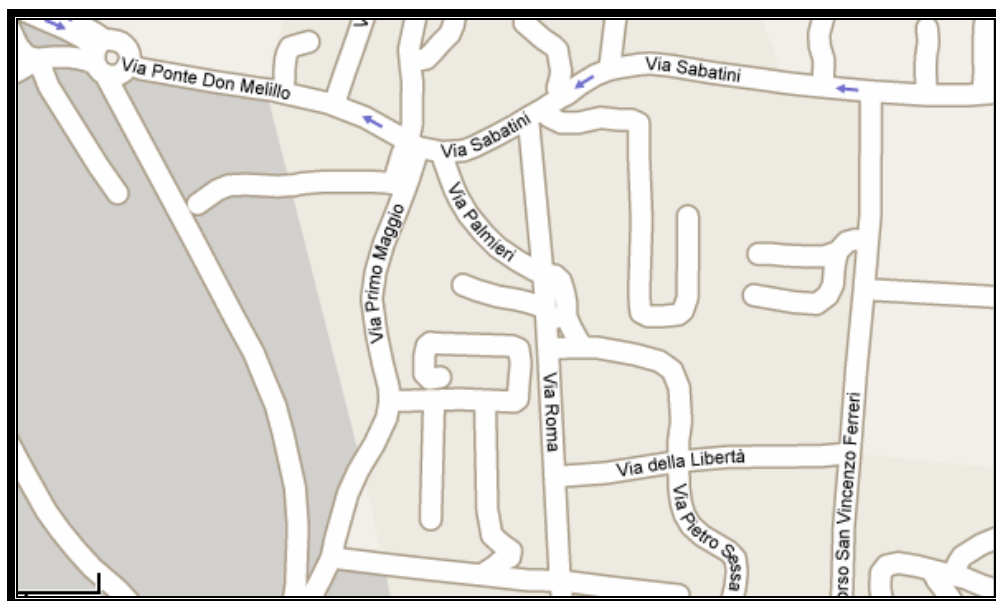


Figura 1 esempio di mappa raster

- **Mappa vettoriale (o navigabile):** è un database strutturato che contiene vari tipi di dati, tra cui la geometria del dato spaziale che si vuole visualizzare, i relativi attributi e operazioni definite sul particolare tipo di dato. Qui di seguito la foto di una mappa visualizzabile sul sito <http://www.swisstrains.ch/> che aggiorna in tempo reale la posizione dei treni delle ferrovie svizzere.



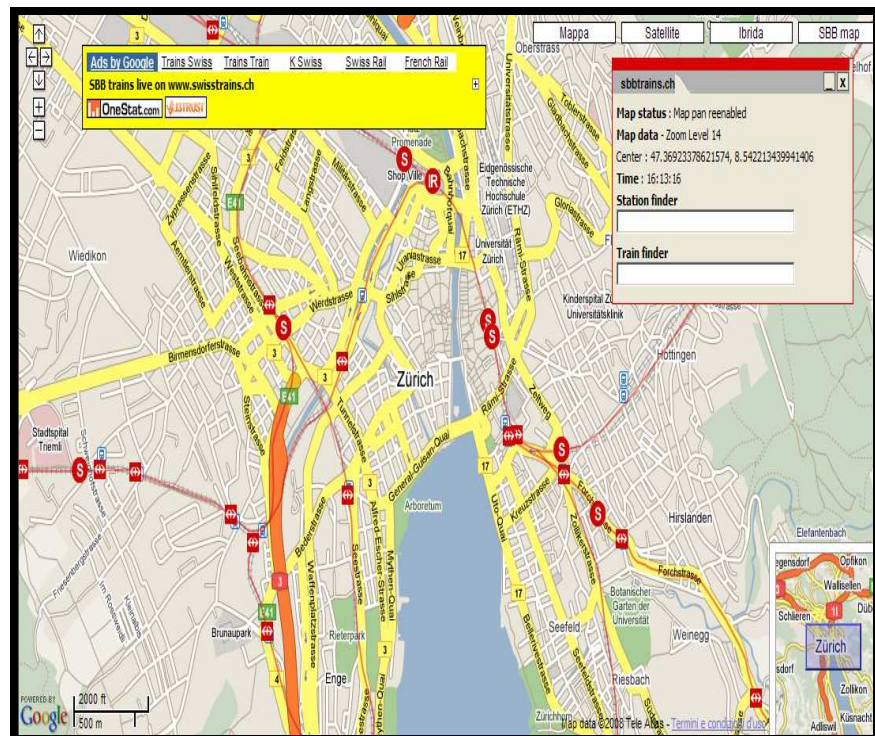


Figura 2 esempio di mappa raster con sovrapposizione di dati vettoriali.

## 2. Scopo del progetto

Il progetto ha lo scopo di verificare la possibilità di creare un'applicazione stand alone che, tramite l'utilizzo di tecnologie WMS<sup>7</sup>, permetta di interagire con il database geografico di mappe Google e, su queste, rappresentare informazioni provenienti da altri database.

In caso di fattibilità sarà descritta una libreria per lo sviluppo di tale applicazione.

<sup>7</sup> Un OGC **Web Map Service (WMS)** produce dinamicamente mappe di dati spazialmente riferiti a partire da informazioni geografiche.

### 3. Tempificazione delle attività

Per la realizzazione della tesi sono state preventivate 150 ore ed effettivamente impiegate 170 ore in 54 giorni lavorativi. Il progetto si è articolato secondo le seguenti fasi:

- Studio dei sistemi informativi territoriali e delle relative potenzialità: è stato studiato il funzionamento dei GIS, i loro campo di applicazione e sono state effettuate delle ricerche sui sistemi preesistenti.
- Studio del funzionamento di Google Earth e Google Maps: studio approfondito sul funzionamento dei due sistemi, le potenzialità e la capacità di integrazione con altri sistemi.
- Studio dei servizi WMS: studio degli standard di trasmissione delle informazioni tramite WMS
- Studio di fattibilità: verifica, in base alle ricerche effettuate della possibilità di procedere con il progetto.
- Progettazione della soluzione: ricerca di una soluzione e progettazione di una libreria per lo sviluppo di applicazioni.

Di seguito si riporta diagramma di Gantt:

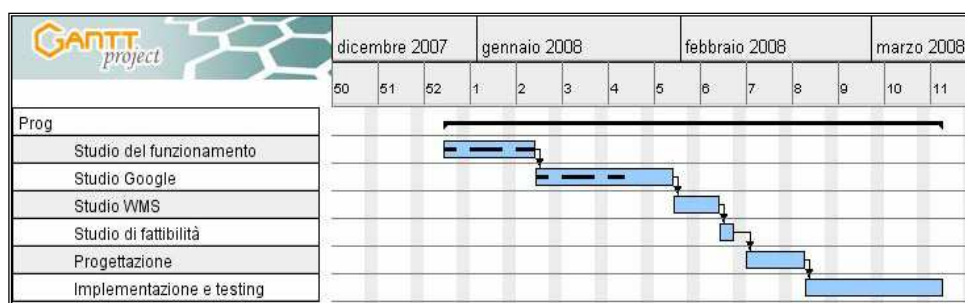


Tabella 1 diagramma di Gantt per le tempistiche del progetto

#### **4. Struttura del documento**

Questo documento è diviso in sei capitoli

Il primo capitolo introduce, in forma descrittiva, lo scopo del progetto e le relative tempistiche. Sono stati individuati i principali protagonisti della cartografia digitale e i relativi strumenti utilizzati (mappe digitali).

Il secondo capitolo espone gli obiettivi generali del progetto e i risultati dello studio sui sistemi già esistenti. In funzione di questi, poi, viene introdotta la libreria proposta.

Il terzo capitolo mostra parte dello studio effettuato per comprendere il funzionamento e la struttura di Google Maps. In base a questo studio viene dimostrata la fattibilità di questo progetto.

Il quarto capitolo descrive la libreria proposta e mette in risalto le componenti più importanti per comprendere le effettive possibilità di utilizzo.

Il quinto capitolo descrive la progettazione di un piccolo sistema implementato per dimostrare alcune funzionalità della libreria proposta.

Il sesto capitolo riassume il lavoro che è stato svolto e descrive i risultati ottenuti.

Infine due appendici:

L'appendice A mostra la possibilità di integrazione di dati in Google Earth e fornisce spiegazioni sui sistemi di coordinate adottati.

L'appendice B descrive un paradigma che ha delle relazioni con l'architettura di Google Maps.

## Capitolo 2

### REQUISITI

#### Introduzione

In base allo studio dei sistemi esistenti, in questo capitolo, si stabiliranno i requisiti che la libreria dovrà soddisfare. Sono stati presi in considerazione tre sistemi: Google Maps, Google Earth e Geoserver. Ognuno di essi presenta, pur con diverse modalità, identiche funzionalità di interazione con mappe digitali. La scelta dei sistemi da analizzare, infatti, è stata presa in base alle modalità con le quali questi offrono i loro servizi. Google Maps fornisce un'API<sup>8</sup> che permette l'utilizzo delle sue funzionalità solo tramite un browser. Google Earth, invece, è un applicativo stand alone<sup>9</sup> che, però, permette di estendere le sue funzionalità in maniera molto limitata. GeoServer<sup>10</sup>, infine, consente di interagire con mappe digitali tramite richieste WMS e, quindi, non presenta alcun problema d'integrazione.

#### 1. Obiettivi generali

Integrare una mappa Google in un'applicazione non web e poter seguire i comandi base. Definire, quindi, quali sono le operazioni base e descriverle sotto forma di requisiti.

---

<sup>8</sup> **Application Programming Interface API** (*Interfaccia di Programmazione di un'Applicazione*), sono ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per un determinato compito.

<sup>9</sup> Caratteristica di un software che può eseguire i compiti per cui è preposto senza che siano richiesti altri componenti.

<sup>10</sup> Server open source (GPL) per la pubblicazione e la modifica di dati geografici ([www.geoserver.org](http://www.geoserver.org)).

## **2. Sistemi correnti**

Nei prossimi paragrafi si mostreranno a livello descrittivo i sistemi esistenti che fanno uso delle tecnologie precedentemente indicate e che rientrano quindi nello studio fatto per comprendere quali sono le effettive possibilità di integrazione. Molta attenzione è stata data al servizio di Google Map e al programma Google Earth.

### **2.1 Google Maps**

Google fornisce un'API per poter interagire con le Google Maps. Per poter valutare le capacità delle API, è stata esaminata l'interfaccia Google Local. Google Local è il nome del sito web fornito da Google che fa uso del Google Maps Programmer Interface (API) per descrivere informazioni, locazioni, e strade all'interno di una mappa. Questa interfaccia è reperibile presso <http://local.google.com/>.

Google Maps usa una combinazione di HTML<sup>11</sup>, JavaScript,<sup>12</sup> mappe, ed elementi interattivi. Questo strumento si chiama Ajax e, attualmente, supporta i browsers Internet Explorer 6.0+, Firefox1.5+, Safari2.0.4+.

---

<sup>11</sup> Hyper Text Mark-Up Language è un linguaggio usato per descrivere i documenti ipertestuali disponibili nel Web.

<sup>12</sup> linguaggio di scripting orientato agli oggetti comunemente usato nei siti web.

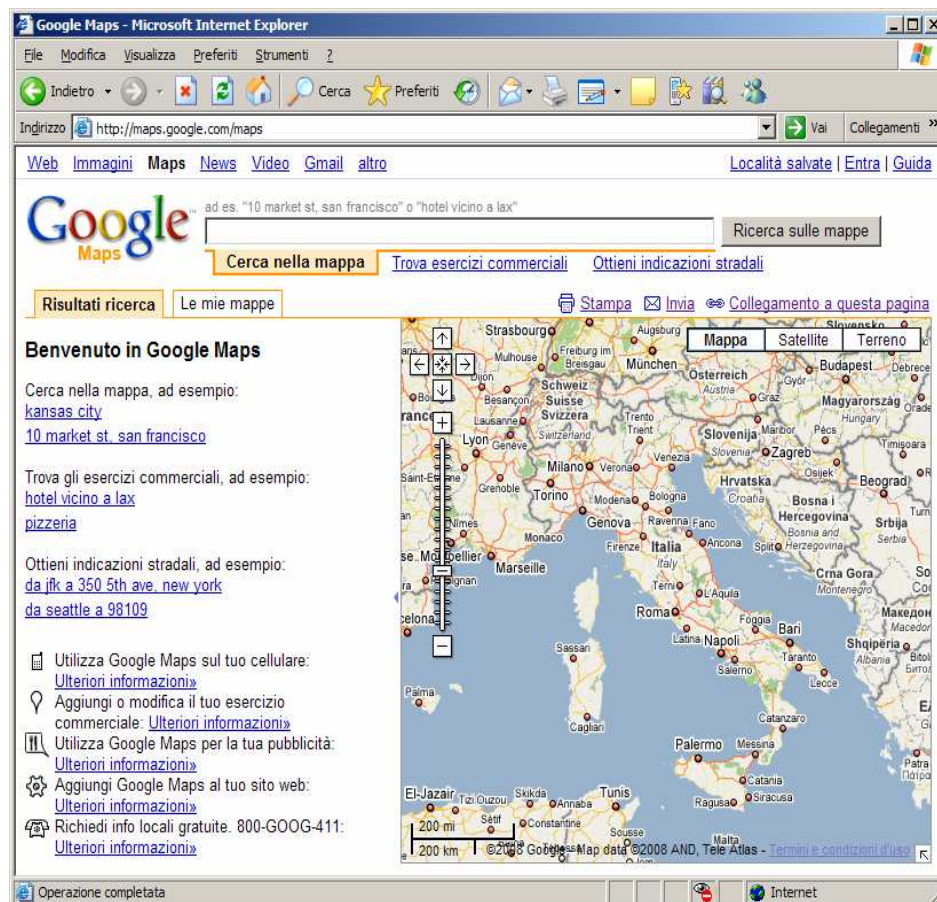


Figura 3 interfaccia Google Maps

Come si vede da questa foto, Il contenuto principale della finestra è il pannello contenente la mappa che mostra la vista corrente. L'utente ha a disposizione dei pulsanti di controllo nell'angolo sinistro e poco sotto uno slider<sup>13</sup> per la regolazione dello zoom. Funzionalità di particolare interesse è quella di poter cambiare il tipo di mappa visualizzata tramite i controlli posti nell'angolo superiore destro. Questa mappa ci permette di definire diversi livelli di zoom, da 10.000 a 100 metri.

Per le aree di minor interesse, in caso di immagine satellitare, lo zoom potrebbe non scendere al di sotto dei 500 metri. Tra le funzionalità offerte dal servizio, si può visualizzare un punto sulla mappa dando in input un indirizzo:

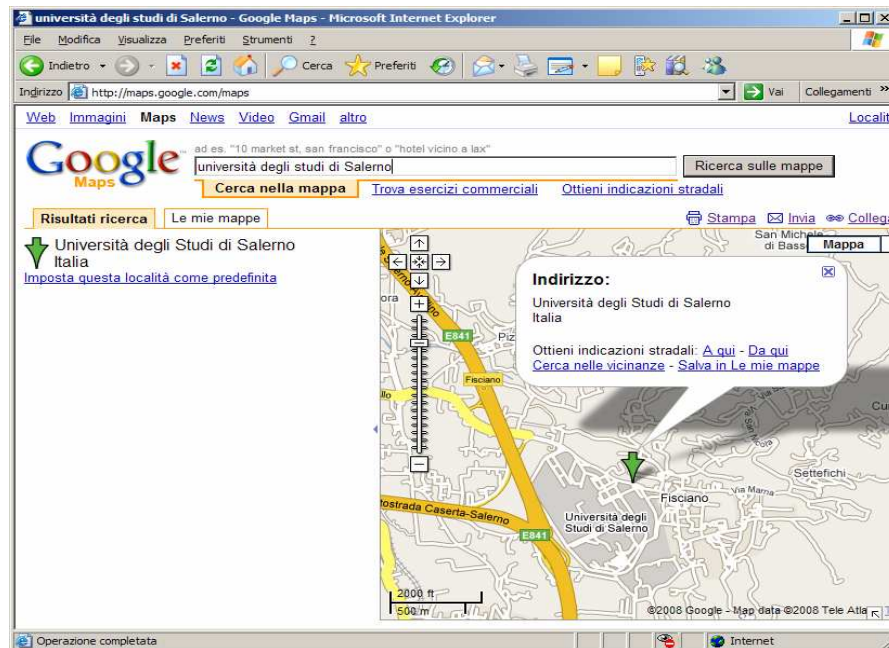


Figura 4 Localizzazione di un punto su una mappa tramite Google Maps.

La mappa si sposterà sul punto interessato ed aggiungerà un marker(in questo caso il disegno della freccia) cliccabile. Cliccando sul punto si riescono ad avere informazioni aggiuntive tramite la visualizzazione di un fumetto in pop-up. Analogamente è possibile sovrapporre linee (ad esempio itinerari) ed altri oggetti poligionali alla mappa.

In questo paragrafo, tramite Google Local, sono state descritte brevemente le operazioni rese disponibili dalla Google Maps API interface: zoom, navigare all'interno della mappa e sovrapporre informazioni.

<sup>13</sup> È un controllo disponibile sull'interfaccia grafica che permette all'utente di specificare valori, di un insieme



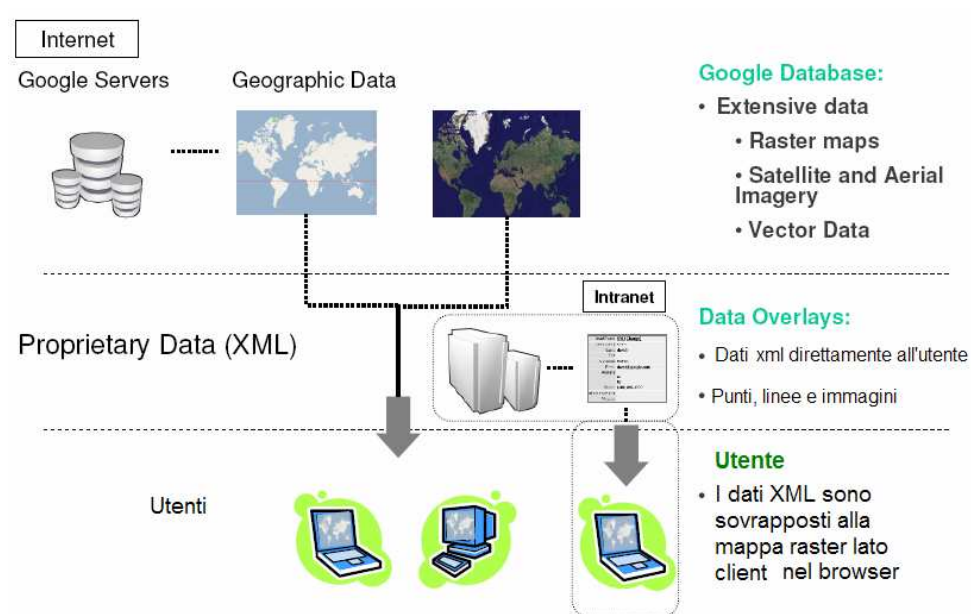


Figura 5: architettura del servizio Google Maps

## 2.1 Google Earth

Google Earth è un'applicazione grafica tridimensionale che permette di visualizzare fotografie aeree e satellitari della Terra con un elevato grado di dettaglio. Nelle principali città del pianeta il programma è in grado di mostrare immagini con una risoluzione inferiore al metro quadrato. Google inizialmente ha acquisito la società Keyhole <sup>14</sup>inc che produceva l'omonimo programma e in seguito ha fuso il programma Keyhole con il servizio Google Maps utilizzando le mappe e le informazioni gestite dal servizio Maps. Il programma oltre a

---

finito lungo uno o due assi.

<sup>14</sup> Una compagnia che sviluppava software ed era specializzata nelle applicazioni di visualizzazione di dati geospaziali. Fu comprata da Google nel 2004.



consentire la visualizzazione delle informazioni, consente anche al singolo utente di inserire delle informazioni aggiuntive condivisibili con altri utenti.

Google Earth può essere utilizzato fornendo coordinate geografiche, indirizzi o semplicemente navigando sulla carta con il mouse. La maggior parte delle grandi città sono disponibili in alta risoluzione in modo da potere evidenziare dettagli quali edifici, strade e automobili. Il livello di dettaglio dipende dall'importanza del luogo; infatti la maggior parte della crosta terrestre è coperta con una risoluzione di 15 metri.

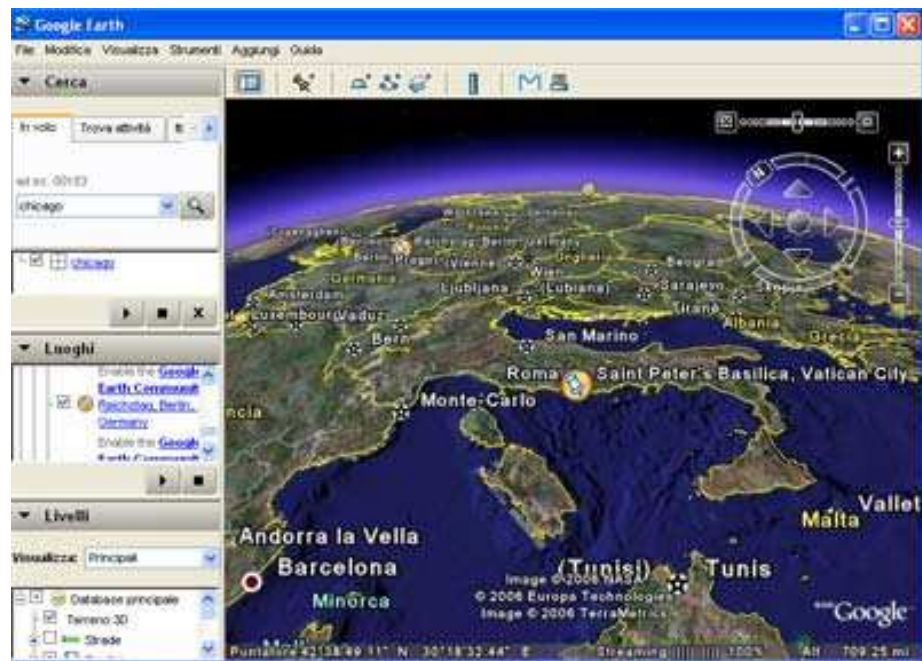


Figura 6 ambiente di Google Earth

Google Earth da la possibilità di essere esteso tramite il caricamento di file KML<sup>15</sup> o KMZ<sup>16</sup>. Questi file permettono di aggiungere overlay, anche personalizzati sulle mappe. Questo rende il sistema molto flessibile in termini di manipolazione dei dati. Purtroppo non ha la possibilità di estendere le funzionalità del suo ambiente di base. Non è possibile, cioè, utilizzarlo come un framework<sup>17</sup> da integrare in altre applicazioni.

## 2.1 GeoServer

GeoServer è un server che permette di pubblicare e modificare dati georeferenziati utilizzando gli open standards (OGC).

Geoserver consente la pubblicazione dei dati in molti formati: mappe/immagini (tramite Web Map Server), dati (Web Feature Service) e permette agli utenti di aggiornare, inserire e cancellare dati. Allo stesso modo GeoServer ha la capacità di ricavare informazioni da fonti eterogenee (vedere Figura 7: formati di input/output Geoserver).

---

<sup>15</sup> Il **KML** (**Keyhole Markup Language**) è un linguaggio basato su XML creato per gestire dati geospaziali in tre dimensioni nei programmi Google Earth, Google Maps e Google Mobile.

<sup>16</sup> Versione compressa (ZIP) di un file KML, ma con estensione .kmz

<sup>17</sup> Il **framework** è una struttura di supporto su cui un software può essere organizzato e progettato.

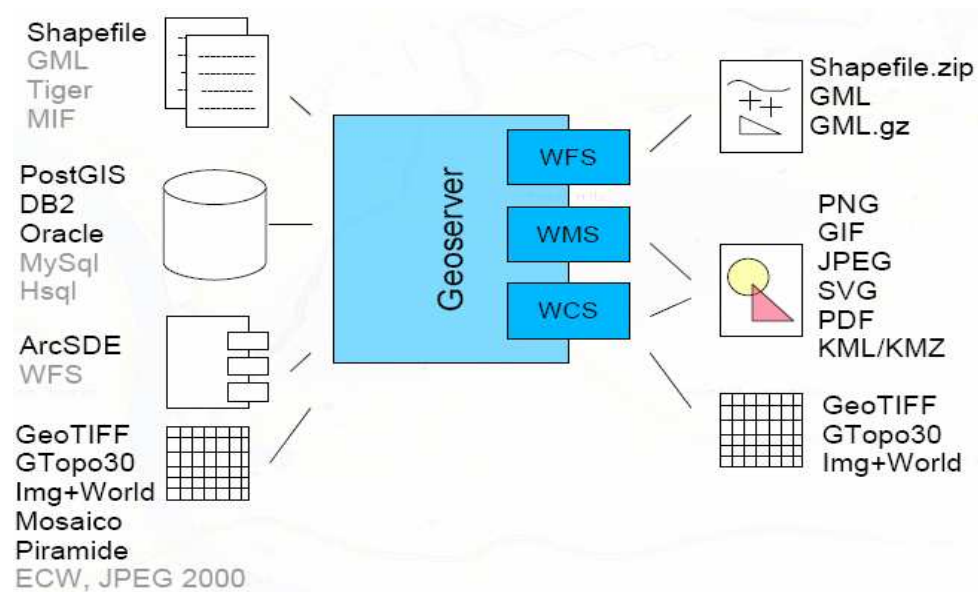


Figura 7: formati di input/output Geoserver

### 3. Sistema proposto

In base all'analisi fatta nei paragrafi precedenti sulle componenti esistenti, si può constatare che, pur esistendo degli standard OGC<sup>18</sup>, manca una convergenza comune sulle modalità di accesso al servizio.

Le Google Maps API si presentano molto flessibili e di facile integrazione in altri sistemi, ma sono web-based.

<sup>18</sup> **Open Geospatial Consortium (OGC**, in precedenza OpenGIS Consortium) è un'organizzazione internazionale no-profit, basata sul consenso volontario, che si occupa di definire specifiche tecniche per i servizi geospaziali e di localizzazione (location based). Le specifiche definite da OGC sono pubbliche (PAS) e disponibili gratuitamente. Le più importanti sono: WMS, WFS, WCS, GML, CAT, CAT, SFS, SQL.

Il software Google Earth è più performante ed estende le funzionalità di Google Maps ampliandone le possibilità di utilizzo, ma non è di facile integrazione in altri sistemi.

Le funzionalità di Geoserver sono molto interessanti e ci permettono molta flessibilità perché non sono vincolate ad un particolare sistema di interazione.

Si propone una libreria java per la creazione di applicazioni, non web-based, capaci di interagire, tramite richieste WMS, con mappe digitali provenienti da Google e da altre fonti. In questa libreria, quindi, convoglieranno le principali caratteristiche dei sistemi visti in precedenza: avere la possibilità di integrare mappe google (Google Maps), essere stand-alone (Google Earth), poter caricare altre informazioni attraverso richieste WMS (Geoserver).

### **3.1 Overview**

Il pacchetto permetterà ai programmatori di integrare nelle proprie applicazioni, mappe provenienti da servizi WMS tra cui Google. Questa libreria implementerà anche dei metodi per le operazioni base da effettuare su una mappa. Qui vengono individuati i requisiti della libreria e vengono presentate le diverse metodologie utilizzabili per soddisfare tali requisiti. Per dimostrare le funzionalità verrà scritta una classe Demo. Al momento, non sono noti framework per lo sviluppo di tali applicazioni.

### **3.2 Le sorgenti di informazione**

Per comprendere a fondo le funzionalità da sviluppare, si è fatto riferimento alla documentazione presente su <http://maps.google.com/support/?hl=it> riguardante le API Google Maps, e il Sistema Google Earth ed al “Web Map Service Implementation Specification” versione 1.1.1.

### 3.3 Requisiti funzionali

La libreria dovrà implementare dei metodi per inserire le mappe in un pannello visuale o in una classe da esso derivata. Sulla mappa, mostrata sul pannello, dovrà essere possibile effettuare le seguenti operazioni:

- Operazioni GIS di base (Zoom, pan);
- Aggiunta di punti e relativi metodi di renderizzazione;
- Disegno di geometrie con una specifica renderizzazione;
- Disegno di layers;
- Gestione della sovrapposizione e calcolo delle coordinate;
- Ricezione degli eventi mouse sulla mappa;
- Cambiare a run-time il provider della mappa e, quindi, il tipo di mappa visualizzato;
- Interazione con una legenda associata ai layers caricati sulla mappa;

### 3.4 Requisiti non funzionali

Sono molteplici i requisiti non funzionali di un pacchetto software per la visualizzazione e interazione con mappe digitali e alcuni di questi, influenzano fortemente diversi aspetti dello sviluppo. Tra questi vi sono:

- Velocità;
- Semplicità di utilizzo;

- Affidabilità;
- Scalabilità;
- Modularità;

e sono motivati dal largo uso che la comunità di programmatori java fa di librerie esterne.

Un pacchetto software per l'interazione con le mappe Google, deve essere affidabile ad evitare che l'improvvisa mancata visualizzazione di una mappa possa causare la perdita di informazioni non memorizzate su file. Simili errori possono essere causati dall'eterogeneità delle mappe offerte dai servizi WMS.

È importante che la libreria offra al programmatore delle funzioni ad alto livello, consentendo di sfruttare le potenzialità del WMS e ignorando tutti i dettagli del suo funzionamento.

Tutte le classi implementate dovranno essere facilmente estendibili, si dovrà mantenere un livello di astrazione tale da permettere al programmatore la personalizzazione e l'adattamento degli oggetti. Tali caratteristiche consentiranno di avere una libreria duttile e facilitare, di conseguenza, l'interoperabilità con altri sistemi/librerie.

Per aumentare la portabilità della libreria il codice sarà scritto in Java<sup>19</sup>.

I metodi forniti dalla libreria per il disegno di aree e punti, dovranno avere complessità lineare, logaritmica, costante o ,comunque, un tempo di risposta minore di 4 secondi.

---

<sup>19</sup> Il linguaggio **Java** è un linguaggio di programmazione orientato agli oggetti, derivato dal C++ (e quindi indirettamente dal C) e creato da James Gosling e altri ingegneri di Sun Microsystems.

La mappa dovrà essere predisposta alla visualizzazione di informazioni provenienti da applicazioni Google come file KML.

## STUDIO DI FATTIBILITÀ

Per verificare la possibilità di creare un sistema che faccia uso di mappe Google, è essenziale comprendere il meccanismo attraverso il quale:

- Il server Google risponde alle query dei clients (Google Earth, Google Local);
- Le mappe sono disposte in relazione alle loro specifiche geografiche.
- Il client comunica con il server (protocollo<sup>20</sup>).

Partendo dall'interfaccia Google Local sono stati fatti diversi esperimenti tesi a scoprire la semantica del codice ottenuto dal server in risposta a delle query che gli venivano sottoposte. Per fasi successive, poi, è stato analizzato come varia il file html ottenuto dal server al variare delle richieste che gli venivano inoltrate. Le prove sono state ripetute per ogni livello di zoom sul tipo di mappa consultata (stradale, satellitare, orografica). Per i primi test sono state eseguite, in rapida successione, delle operazioni di spostamento e zoom della mappa. Ad ognuna di queste operazioni sul client sono state caricate le porzioni di mappa mancante (Figura 9) tramite l'accostamento di quadrati aventi le stesse dimensioni. Questo lascia supporre che la mappa del globo sia divisa in una matrice di quadrati. Ad avvalorare questa ipotesi, la documentazione di Google Earth (vedere appendice A) indica un metodo

---

<sup>20</sup> Un **protocollo** è un insieme di regole fisse e comuni su come deve essere organizzato lo scambio di informazioni tra due entità.



simile per il prodotto stand-alone di Google. Continuando a zoomare, si può constatare che la zona viene nuovamente divisa in quadrati di uguale dimensione. Da quest'ultimo esperimento si è ipotizzato che, per ogni livello di zoom, vi sia uno strato (Figura 8) suddiviso in un determinato numero di celle. Misurato, sul monitor con risoluzione 1152X864, il lato di un generico quadrato caricato, questo ha ricoperto sullo schermo una linea verticale di circa 9 centimetri. Considerato l'altezza del monitor di circa 30 centimetri si è potuta impostare la proporzione (si tratta di valori approssimati):

$$(\approx 9):(\approx 30) = x : 864 \quad X \approx 256$$

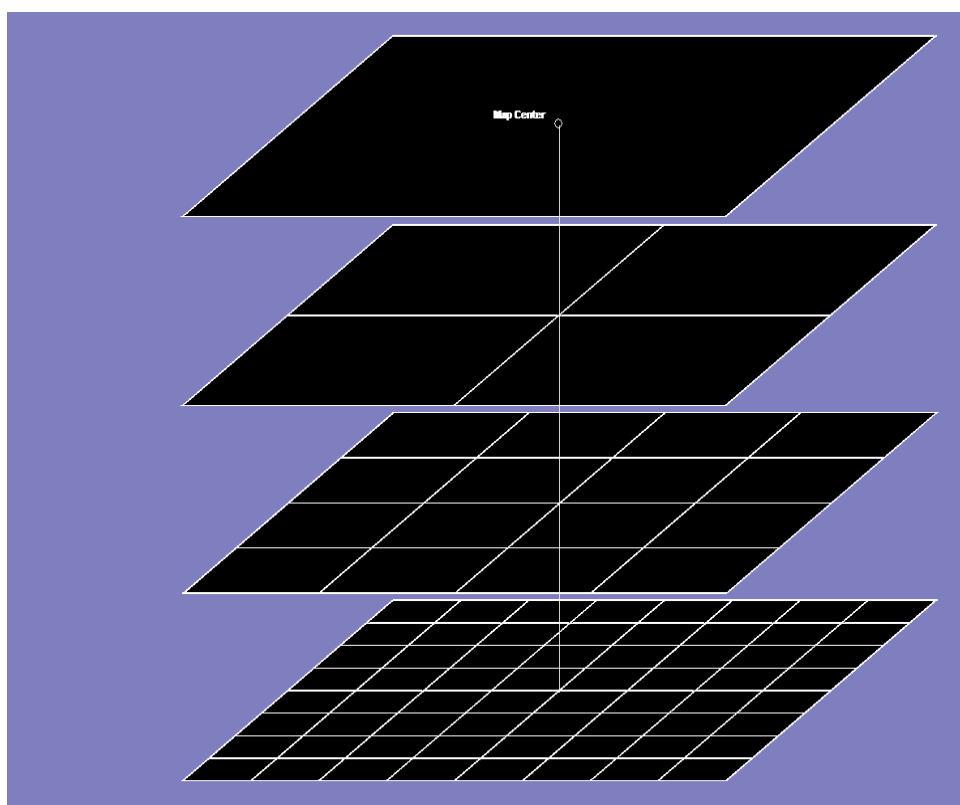


Figura 8 I diversi layers di una stessa mappa per la gestione dello zoom.

Lo zoom è inverso. La vista della mappa del pianeta completo (quindi senza zoom) ha parametro  $\text{zoom}=17$ , mentre  $\text{zoom} = 0$  indica il massimo zoom disponibile. Nel primo caso la matrice è formata da una sola cella contenente l'immagine (Tile) indicizzata da  $x=0$  e  $y=0$ . Ogni volta che si aumenta lo zoom, si scende di un livello(vedere Figura 8) e la matrice passerà da  $1 \times 1$  a  $2 \times 2$ ,  $4 \times 4$  e così via. Considerando un generico livello di zoom  $\lambda$ , quindi, la matrice sarà  $2^{17-\lambda} \times 2^{17-\lambda}$ .



Figura 9 Caricamento di mappe da Google Maps

È questa una deduzione confermata dallo studio dell'url. Si è localizzato sulla mappa l'indirizzo "Sede centrale Google, 1600 Amphitheatre Parkway, Mountain View, CA 94043"(vedere Figura 10)."

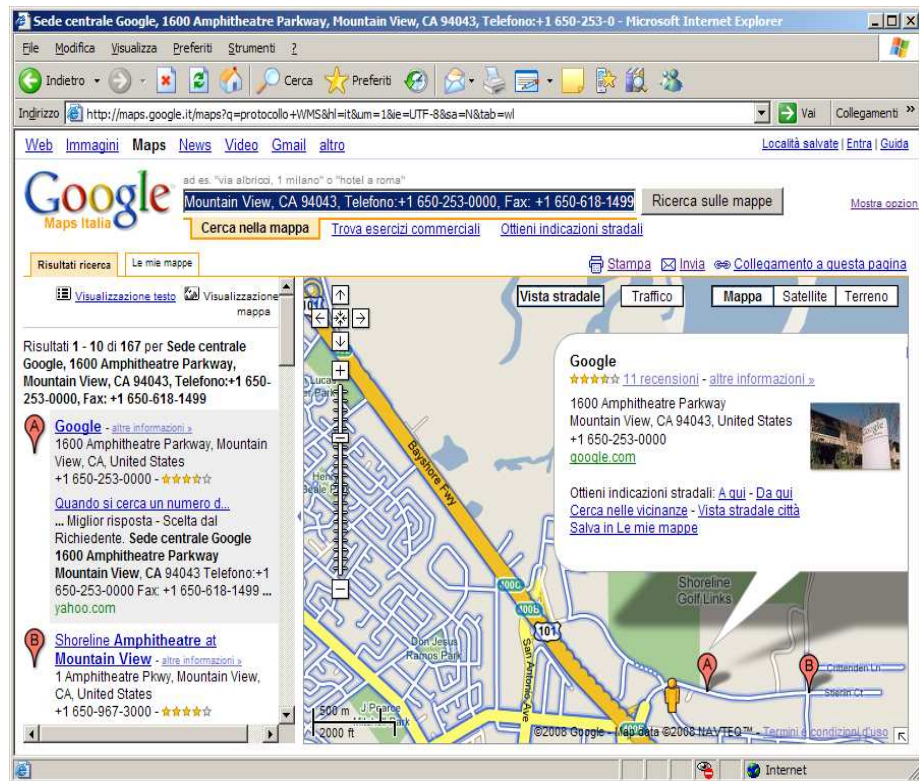


Figura 10 Localizzazione di un punto su Google Maps

e si è analizzato il relativo file html. Si riporta parte del file. Considerare il codice (sottolineato):

```

.....

'Visualizzazione ingrandita della mappa',11014: 'Oggi',11151: 'Non utilizzare termini
inappropriati quali: %1$s',11929: 'Nickname non disponibile',10953: 'Se continui, le
modifiche non salvate andranno perdute.',11250: '\x3ca
href\x3d\x22%1$s\x22\x3eAccedi\x3c/a\x3e per visualizzare i segnalibri
esistenti',11245:'Trascina per modificare percorso',0:
");GLoadApi(["http://mt0.google.com/mt?n\x3d404\x26v\x3dw2.69\x26h\x3dit\x
26", "http://mt1.google.com/mt?n\x3d404\x26v\x3dw2.69\x26h\x3dit\x26", "http://mt2.

```

```

google.com/mt?n\x3d404\x26v\x3dw2.69\x26h\x3dit\x26","http://mt3.google.com/mt
?n\x3d404\x26v\x3dw2.69\x26h\x3dit\x26"],
["http://kh0.google.it/kh?n\x3d404\x26v\x3d25\x26h\x3dit\x26","http://kh1.googl
e.it/kh?n\x3d404\x26v\x3d25\x26h\x3dit\x26","http://kh2.google.it/kh?n\x3d404\x26
v\x3d25\x26h\x3dit\x26","http://kh3.google.it/kh?n\x3d404\x26v\x3d25\x26h\x3dit\x
26"],
["http://mt0.google.com/mt?n\x3d404\x26v\x3dw2t.69\x26h\x3dit\x26","http://mt
1.google.com/mt?n\x3d404\x26v\x3dw2t.69\x26h\x3dit\x26","http://mt2.google.com/
mt?n\x3d404\x26v\x3dw2t.69\x26h\x3dit\x26","http://mt3.google.com/mt?n\x3d404\
x26v\x3dw2t.69\x26h\x3dit\x26"],"","",false,"G",{,"http://mt0.google.com/mt?n\
x3d404\x26v\x3dw2p.64\x26h\x3dit\x26","http://mt1.google.com/mt?n\x3d404\x26
v\x3dw2p.64\x26h\x3dit\x26","http://mt2.google.com/mt?n\x3d404\x26v\x3dw2p.64\
x26h\x3dit\x26","http://mt3.google.com/mt?n\x3d404\x26v\x3dw2p.64\x26h\x3dit\x
26"]);if (window.GJsLoaderInit)
{GJsLoaderInit("http://maps.google.com/intl/it_it/mapfiles/102/maps2"
+ "/main.js");}function GUnload() {if (window.GUnloadApi) {GUnloadApi();}}var
_mIsRtl = false;var _mF = [
,false,true,true,100,4096,"bounds_cipppt.txt","cities_cipppt.txt","local/add/.....
...flagStreetView",true,true,400,true,true,,true,,true,"/maps/c/ui/HovercardLaunche
r/dommanifest.js.....

```

Facendo un confronto con le stringhe presenti nella pagina riusciamo a dedurre che la successione di caratteri “\x3d” equivale al segno “=” e la successione “\x26” equivale al segno “&”. Continuando con esperimenti di questo tipo, si è ipotizzato che Google utilizzi quattro server per fornire le mappe: mt0, mt1, mt2, mt3 mentre per le mappe satellitari sono kh0, kh1, kh2, kh3. Questa divisione, si suppone, sia un metodo per bilanciare uniformemente le richieste dei client. Analizzando gli URL delle immagini che vengono caricate si arriva all’indirizzo:

“http://mt{*k*}.google.com/mt?n=404&v={*j*}&x={*xtile*}&y={*ytile*}&zoom={*z*} ”

dove  $k=\{ 0, 1, 2, 3, \}$ ,  $j=\{ w2.69, w2t.69, w2p.64 \}$ <sup>22</sup>, xtile e ytile indicano le coordinate matriciali della cella contenente l'immagine da caricare,  $z=\{0-17\}$ . Questo è l'URL per le mappe non satellitari. Tramite il parametro "v" si indica il tipo di mappa (stradale, orografica, ibrida) ad esempio al momento della stesura di questo testo, tramite il parametro w.69 si richiede la mappa stradale e tramite il w2t.69 la mappa orografica. La richiesta delle mappe satellitari, invece, utilizza un altro tipo di codifica. L'URL ricavato è:

"http://kh{k}.google.com/mt?n=404&v=25&t={una sequenza di caratteri}&"

dove  $k=\{ 0, 1, 2, 3 \}$ , il parametro "v" ha la stessa funzione di quello visto nell'url precedente<sup>23</sup>, e il parametro "t" è una stringa che inizia con la lettera "t" e continua con caratteri definiti dall'insieme { "q", "r", "s", "t" }. La stringa definita per il parametro t indicizza la mappa da visualizzare per un determinato livello di zoom. Il primo livello, cioè quello composto da una matrice 1x1, è indicato dalla stringa "t". Aggiungendo un carattere si fa uno zoom e si indicizzerà l'immagine da caricare nel seguente modo (tenere presente la Figura 8 e considerare che ogni immagine ad ogni livello di zoom viene divisa in 4 parti uguali) :

- Aggiungendo il carattere "q" si indicherà il quadrante superiore sinistro;
- Aggiungendo il carattere "r" si indicherà il quadrante superiore destro;
- Aggiungendo il carattere "s" si indicherà il quadrante inferiore destro;
- Aggiungendo il carattere "t" si indicherà il quadrante inferiore sinistro;

---

<sup>22</sup> questo numero può variare frequentemente, durante la stesura di questo documento è cambiato due volte

<sup>23</sup> Anche questo numero, in analogia a quello per le mappe stradali, è soggetto a variazioni.

## Conclusioni

Il servizio web offerto da Google è molto semplice. Il sistema multiserver, che utilizza per la distribuzione delle mappe, da l'idea di un'architettura in stile "rest."<sup>24</sup> Il server map di google alleggerisce il carico computazionale dei client dividendo il mondo in una matrice quadrata, è possibile così, richiedere una porzione di mappa specificando la posizione x/y dell'immagine all'interno della matrice. Lato client, a questo punto, resta solo l'onere di calcolare le porzioni di mappa da caricare, in base ad una specifica posizione geografica e alla quantità di mappa che necessita di essere visualizzata. Sfortunatamente Google non offre un servizio wms e, l'utilizzo delle mappe senza i metodi forniti dalle Google Map API, non rientra nei termini di licenza. In rete vi sono diversi documenti che spiegano come è possibile integrare le mappe Google in un servizio wms lato server. Ad oggi, tuttavia, non sono reperibili applicazioni stand-alone che integrano mappe Google.

---

<sup>24</sup> Representational State Transfer. Vedere appendice.

## *Capitolo 4*

### UNA LIBRERIA PER LO SVILUPPO

Per creare una libreria facile da utilizzare e soprattutto da estendere, è stato fondamentale scegliere un linguaggio di programmazione di tipo object-oriented<sup>25</sup>. Tra i vari linguaggi disponibili è stato scelto Java. Questo, infatti, è caratterizzato da un'ottima flessibilità ed è molto diffuso tra i programmatori. Questo rende più facile lo sviluppo e l'integrazione di componenti scritte da terzi.

Per la creazione della libreria è stata estesa una libreria SwingX creata dalla Swing Labs<sup>26</sup> che facilita la gestione delle richieste http e la visualizzazione delle mappe tramite pannelli. In particolare, sono state sfruttate le enormi potenzialità del pacchetto "org.jdesktop.swingx.mapviewer" (vedere Figura 11).

Qui di seguito si riportano le principali componenti analizzate delle quali, alcune, fanno parte del pacchetto mapviewer.

---

<sup>25</sup> La **programmazione orientata agli oggetti (OOP, Object Oriented Programming)** è un paradigma di programmazione, che prevede di raggruppare in un'unica entità (la classe) sia le strutture dati che le procedure che operano su di esse, creando per l'appunto un "oggetto" software dotato di proprietà (dati) e metodi (procedure) che operano sui dati dell'oggetto stesso.

<sup>26</sup> Swing Labs è un laboratorio open source per la ricerca di nuovi modi per rendere le applicazioni swing facili da scrivere, con migliori performance e un migliore aspetto grafico.

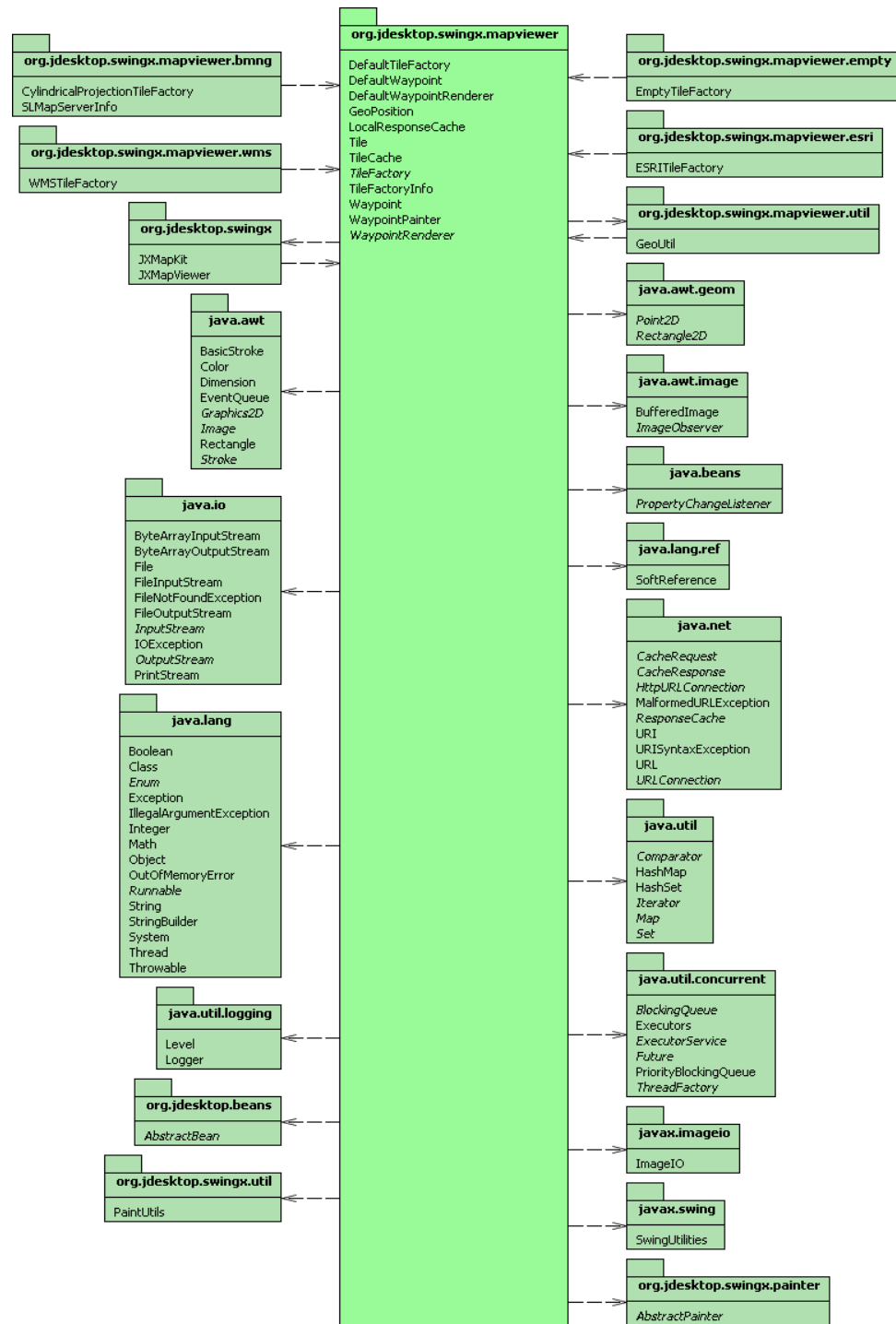


Figura 11 Il pacchetto mapviewer



## DefaultTileFactory (pacchetto mapviewer)

Questo è uno degli oggetti più importanti per i nostri scopi. Infatti DefaultTileFactory è la classe che si occupa del caricamento dei raster tramite http. Il suo ruolo non si limita a scaricare l'immagine in memoria, ma fornisce funzioni per convertire le coordinate geografiche in pixel e viceversa, utilizza tecniche di threading per migliorare l'affidabilità. Gli URL delle immagini da caricare vengono prelevati da una coda a priorità e vengono effettuati tre tentativi di caricamento dell'immagine, se questi falliscono si procede a prelevare dalla coda a priorità altri "Tiles".

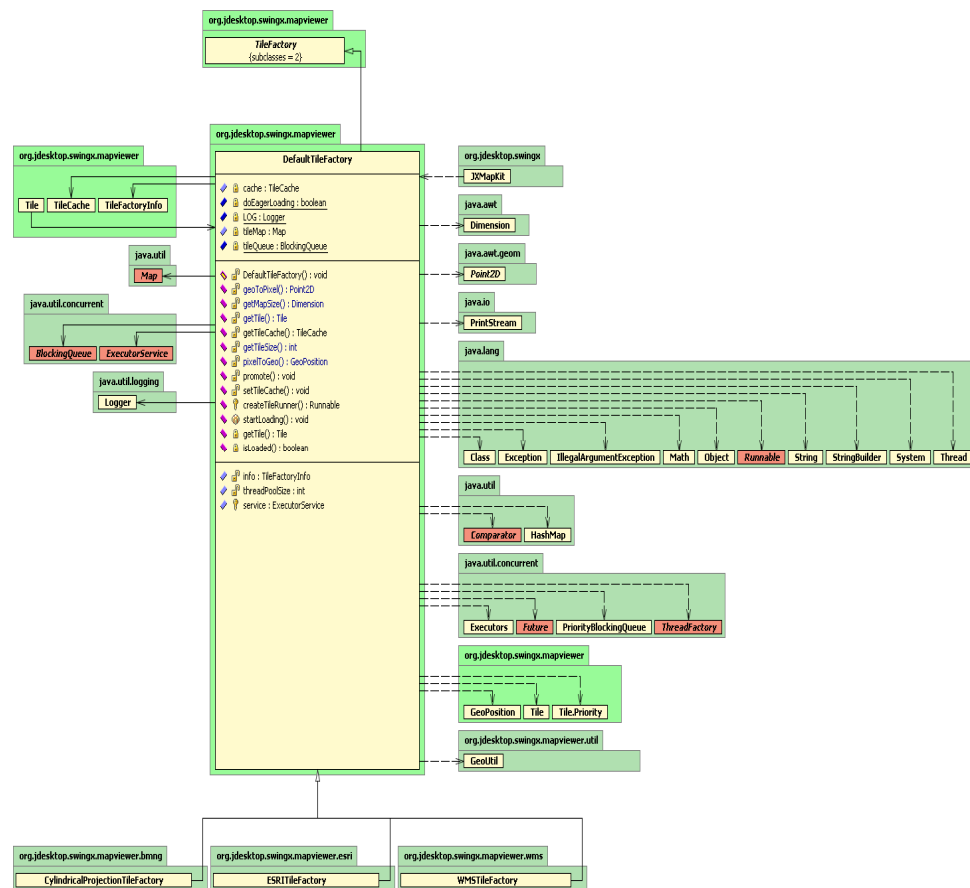


Figura 12 DefaultTileFactory

Le informazioni utilizzate da DefaultTile Factory sull URL da chiamare sono memorizzate in un oggetto di tipo TileFactoryInfo:

```
package org.jdesktop.swingx.mapviewer;

@author joshy

public class TileFactoryInfo
{
.....
.....
    public TileFactoryInfo(int minimumZoomLevel, int maximumZoomLevel,
int totalMapZoom,
        int tileSize, boolean xr2l, boolean yt2b,
        String baseURL, String xparam, String yparam, String
zparam) {
        this.minimumZoomLevel = minimumZoomLevel;
        this.maximumZoomLevel = maximumZoomLevel;
        this.totalMapZoom = totalMapZoom;
        this.baseURL = baseURL;
        this.xparam = xparam;
        this.yparam = yparam;
        this.zparam = zparam;
        this.setXr2l(xr2l);
        this.setYt2b(yt2b);

        this.tileSize = tileSize;

        // init the num tiles wide
        int tilesize = this.getTileSize(0);

        longitudeDegreeWidthInPixels = new double[totalMapZoom+1];
        longitudeRadianWidthInPixels = new double[totalMapZoom+1];
        mapCenterInPixelsAtZoom = new Point2D.Double[totalMapZoom+1];
        mapWidthInTilesAtZoom = new int[totalMapZoom+1];

        // for each zoom level
        for (int z = totalMapZoom; z >= 0; --z) {
            longitudeDegreeWidthInPixels[z] = (double)tilesize / 360;
            longitudeRadianWidthInPixels[z] = (double)tilesize /
(2.0*Math.PI);
            int t2 = tileSize / 2;
            mapCenterInPixelsAtZoom[z] = new Point2D.Double(t2, t2);
            mapWidthInTilesAtZoom[z] = tileSize / this.getTileSize(0);
            tileSize *= 2;
        }

    }

.....
.....
.....
```

```

public String getTileUrl(int x, int y, int zoom) {

    String ypart = "&" + yparam + "=" + y;

    if(!yt2b) {
        int tilemax = getMapWidthInTilesAtZoom(zoom);
        ypart = "&" + yparam + "=" + (tilemax/2-y-1);
    }
    String url = baseUrl +
        "&" + xparam + "=" + x +
        ypart +
        "&" + zparam + "=" + zoom;
    return url;
}

```

Questo oggetto, tramite il costruttore, memorizza le informazioni necessarie per completare l'URL e inizializza dei vettori:

```

longitudeDegreeWidthInPixels = new double[totalMapZoom+1];

```

ad esempio conterrà il numero di pixel per grado di longitudine ad un dato livello di Zoom.

Un grado di latitudine corrisponde sulla superficie terrestre a circa 110 km, come si può calcolare dalla proporzione  $1^\circ:360^\circ = d:2\pi R$  ove d è l'equivalente in km di un grado e R è il raggio terrestre. Analogamente nel ciclo “for” possiamo leggere le istruzioni:

```

longitudeDegreeWidthInPixels[z] = (double)tilesize / 360;
longitudeRadianWidthInPixels[z] = (double)tilesize / (2.0*Math.PI);

```

che permettono di definire a quanti pixel corrisponde un grado/radiante.

```

int t2 = tilesize / 2;
mapCenterInPixelsAtZoom[z] = new Point2D.Double(t2, t2);
mapWidthInTilesAtZoom[z] = tilesize / this.getTileSize(0);
tilesize *= 2;

```

La seconda istruzione memorizza il centro della mappa e le dimensioni a diversi livelli di zoom. L'ultima istruzione assicura che ad un maggiore livello di zoom vi sia un maggior numero di pixel per grado.

Il metodo `getTileUrl` restituisce URL dal quale viene caricata la mappa, sovrascrivendo questo metodo si può adattare l'oggetto a qualunque indirizzo web o locale.

```
package org.jdesktop.swingx.mapviewer;

@author josh

public class DefaultTileFactory extends TileFactory
{
    public DefaultTileFactory(TileFactoryInfo info)
    { this.info = info; }

    private static final boolean doEagerLoading = true;

    private TileFactoryInfo info;
    private int threadPoolSize = 4;
    private ExecutorService service;

    private Map<String, Tile> tileMap = new HashMap<String, Tile>();

    private TileCache cache = new TileCache();

    public Tile getTile(int x, int y, int zoom) {
        return getTile(x, y, zoom, true);
    }

    private Tile getTile(int tpx, int tpy, int zoom, boolean eagerLoad) {
        int tileX = tpx;
        int numTilesWide = (int)getMapSize(zoom).getWidth();
        if (tileX < 0) {
            tileX = numTilesWide - (Math.abs(tileX) % numTilesWide);
        }

        tileX = tileX % numTilesWide;
        int tileY = tpy;
        String url = getInfo().getTileUrl(tileX, tileY, zoom);
        Tile.Priority pri = Tile.Priority.High;
        if (!eagerLoad) {
            pri = Tile.Priority.Low;
        }
        Tile tile = null;
        if (!tileMap.containsKey(url)) {
            if (!GeoUtil.isValidTile(tileX, tileY, zoom, getInfo())) {
                tile = new Tile(tileX, tileY, zoom);
            } else {
                tile = new Tile(tileX, tileY, zoom, url, pri, this);
                startLoading(tile);
            }
            tileMap.put(url, tile);
        } else {

```

```

        tile = tileMap.get(url);
        if (tile.getPriority() == Tile.Priority.Low && eagerLoad
&& !tile.isLoaded()) {
            promote(tile);
        }
    }
}

.....
.....
    return tile;
}

/*
private void eagerlyLoad(int x, int y, int zoom) {
    TilePoint tl = new TilePoint(x,y);
    if(!isLoaded(tl, zoom)) {
        getTile(tl, zoom, false);
    }
}
*/

protected synchronized ExecutorService getService() {
    if(service == null) {
        //System.out.println("creating an executor service with a
threadpool of size " + threadPoolSize);
        service = Executors.newFixedThreadPool(threadPoolSize, new
ThreadFactory() {
            private int count = 0;

            public Thread newThread(Runnable r) {
                Thread t = new Thread(r, "tile-pool-" + count++);
                t.setPriority(Thread.MIN_PRIORITY);
                t.setDaemon(true);
                return t;
            }
        });
    }
    return service;
}

synchronized void startLoading(Tile tile) {
    if(tile.isLoading) {
        System.out.println("already loading. bailing");
        return;
    }
    tile.isLoading = true;
    try {
        tileQueue.put(tile);
        getService().submit(createTileRunner(tile));
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
}

```

```

protected Runnable createTileRunner(Tile tile) {
    return new TileRunner();
}

private class TileRunner implements Runnable {

    protected URI getURI(Tile tile) throws URISyntaxException {
        if(tile.getURL() == null) {
            return null;
        }
        return new URI(tile.getURL());
    }

    public void run() {

        final Tile tile = tileQueue.remove();

        int trys = 3;
        while (!tile.isLoaded() && trys > 0) {
            try {
                BufferedImage img = null;
                URI uri = getURI(tile);
                img = cache.get(uri);
                if(img == null) {
                    byte[] bimg = cacheInputStream(uri.toURL());
                    img = PaintUtils.loadCompatibleImage(new
ByteArrayInputStream(bimg));
                    cache.put(uri,bimg,img);
                    img = cache.get(uri);
                }
                if(img == null) {
                    System.out.println("error loading: " + uri);
                    LOG.log(Level.INFO, "Failed to load: " + uri);
                    trys--;
                } else {
                    final BufferedImage i = img;
                    SwingUtilities.invokeLaterAndWait(new Runnable() {
                        public void run() {
                            tile.image = new
SoftReference<BufferedImage>(i);
                            tile.setLoaded(true);
                        }
                    });
                }
            } catch (OutOfMemoryError memErr) {
                cache.needMoreMemory();
            } catch (Throwable e) {
                LOG.log(Level.SEVERE,
                    "Failed to load a tile at url: " +
tile.getURL() + ", retrying", e);
                //temp
                System.err.println("Failed to load a tile at url:
" + tile.getURL());
                e.printStackTrace();
            }
        }
    }
}

```

```

        ///temp
        Object oldError = tile.error;
        tile.error = e;
        tile.firePropertyChangeOnEDT("loadingError",
oldError, tile.error);
        if (trys == 0) {
tile.firePropertyChangeOnEDT("unrecoverableError", null, tile.error);
        } else {
            trys--;
        }
    }
    tile.isLoading = false;
}

private byte[] cacheInputStream(URL url) throws IOException {
    InputStream ins = url.openStream();
    ByteArrayOutputStream bout = new ByteArrayOutputStream();
    byte[] buf = new byte[256];
    while(true) {
        int n = ins.read(buf);
        if(n == -1) break;
        bout.write(buf,0,n);
    }
    return bout.toByteArray();
}
}
}

```

In queste righe di codice sono stati mostrati alcuni dei metodi più importanti della classe originale:

- `getTile`: prima richiama l'URL dall'oggetto `TileFactoryInfo`, poi verifica se la mappa richiesta è già presente nel `Map` in cui viene mantenuta una cache delle mappe già caricate. Se la mappa è presente, viene richiamata tramite `Map.get()` altrimenti, viene verificata la correttezza dell'URL e creata una nuova mappa inserendola nell'oggetto `Map`. Nel caso in cui la mappa sia presente nella collezione, ma ancora non sia stata caricata e il suo thread ha una priorità bassa, il metodo incrementa la priorità del thread. Il caricamento dell'oggetto `Tile` è richiesto tramite la chiamata del

metodo `startLoading`. È da evidenziare il modo in cui il metodo calcola il valore `x` della `Tile` da caricare, tramite l'operazione:

```
if (tileX < 0)
{
    tileX = numTilesWide - (Math.abs(tileX) % numTilesWide);
}

tileX = tileX % numTilesWide;
```

Se verrà richiesta una `Tile` esterna alla matrice (vedere Figura 16 per comprendere i dettagli della `Tile`) l'operazione di modulo indicizzerà la `Tile` da caricare come se la matrice fosse collegata agli estremi e, quindi, simulando la circolarità del pianeta.

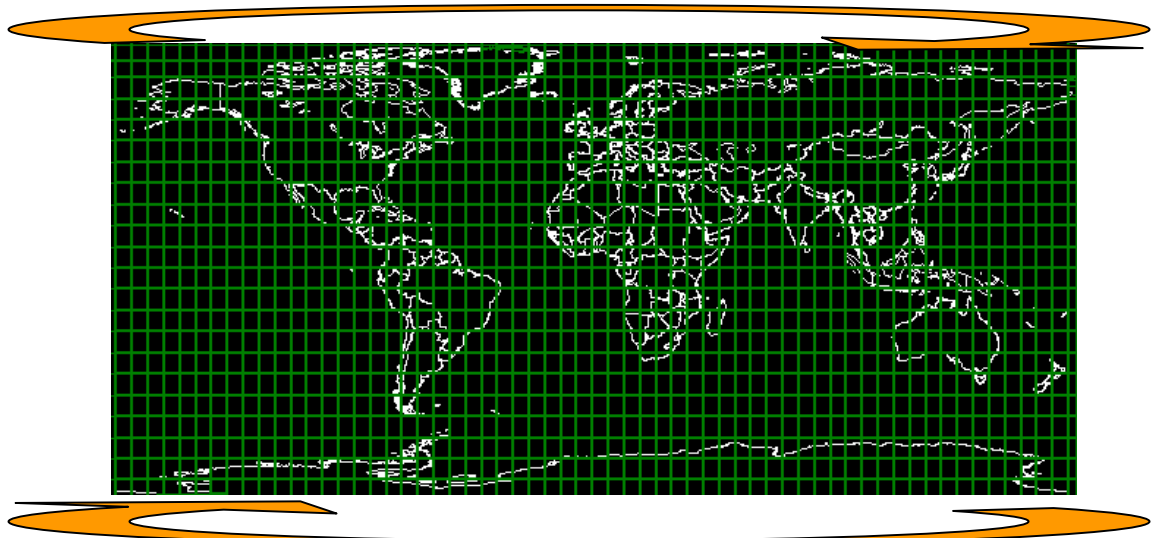


Figura 13 simulazione della geometria circolare del pianeta.

- `startLoading`: aggiunge l'oggetto `Tile` alla relativa coda, crea e sottomette all'`executorservice` una nuova istanza della inner class `Tile Runner` che viene eseguita come nuovo thread.



- **TileRunner**: è l'oggetto che contiene il codice per caricare la mappa. Preleva l'oggetto **Tile** dalla coda e verifica se è presente in cache. Se non è presente l'immagine viene caricata da rete.

## ProviderBuilder

è una classe molto semplice che permette di ottenere dei **TileFactory** preimpostati per diversi provider tra cui, Blue Marble, Openstreet Maps, Google Maps.

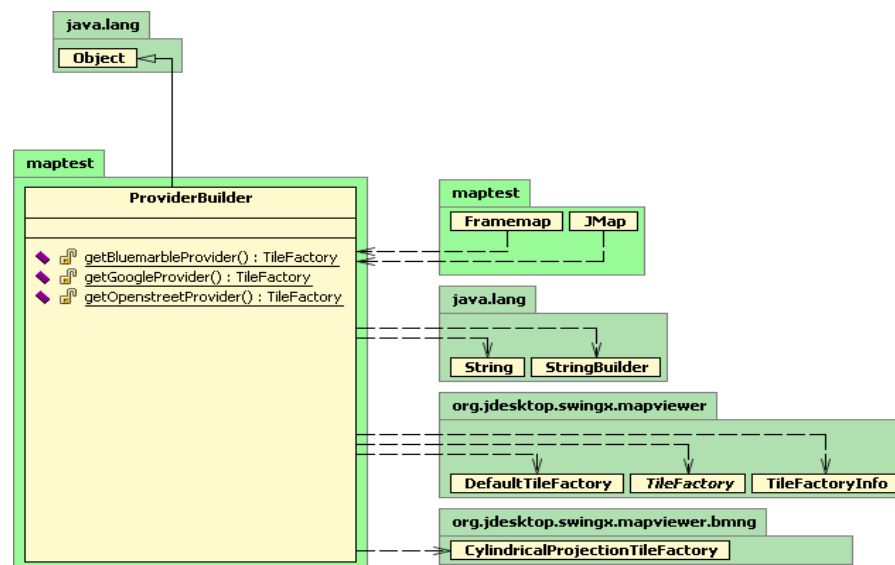


Figura 14 ProviderBuilder

## WMSService (pacchetto `org.jdesktop.swingx.mapviewer.wms`)

Questa classe rappresenta un servizio WMS. Per poterla utilizzare è necessario passarne un'istanza al costruttore della classe **WMSTileFactory**. Il metodo **toWMSURL()** può essere sovrascritto per adattare l'URL a seconda del server che si vuole interrogare.

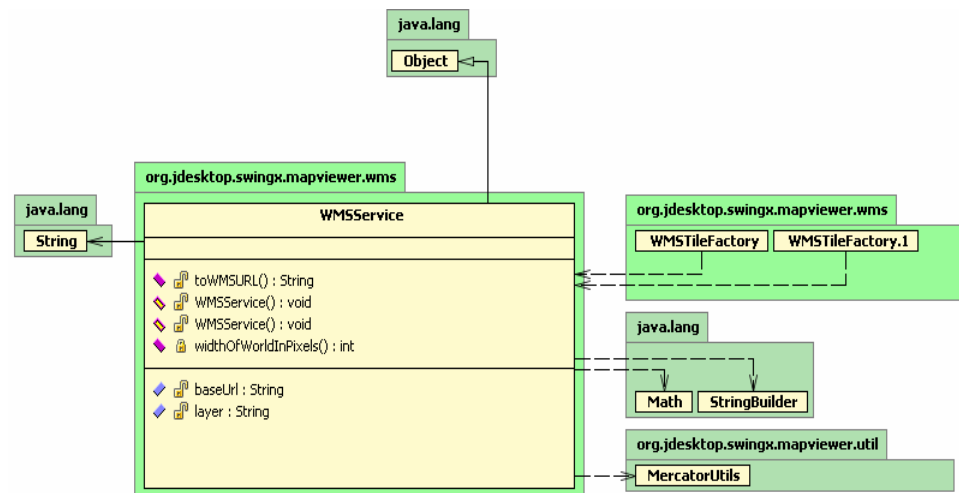


Figura 15 WMSService

```

package org.jdesktop.swingx.mapviewer.wms;

import org.jdesktop.swingx.mapviewer.*;
import org.jdesktop.swingx.mapviewer.util.MercatorUtils;

@author josh

public class WMSService
{
    private String baseUrl;
    private String layer;

    public WMSService()
    {
        setLayer("BMNG");
        setBaseUrl("http://wms.jpl.nasa.gov/wms.cgi?");
    }
    public WMSService(String baseUrl, String layer) {
        this.baseUrl = baseUrl;
        this.layer = layer;
    }

    public String toWMSURL(int x, int y, int zoom, int tileSize)
    {
        String format = "image/jpeg";
    }
}
  
```

```

String layers = "BMNG";
String styles = "";
String srs = "EPSG:4326";
int ts = tileSize;
int circumference = widthOfWorldInPixels(zoom, tileSize);
double radius = circumference / (2* Math.PI);
double ulx = MercatorUtils.xToLong(x*ts, radius);
double uly = MercatorUtils.yToLat(y*ts, radius);
double lrx = MercatorUtils.xToLong((x+1)*ts, radius);
double lry = MercatorUtils.yToLat((y+1)*ts, radius);
String bbox = ulx + "," + uly + "," + lrx + "," + lry;
String url = getBaseUrl() +
    "version=1.1.1&request="+
    "GetMap&Layers="+layer+
    "&format="+format+
    "&BBOX="+bbox+
    "&width="+ts+"&height="+ts+
    "&SRS="+srs+
    "&Styles="+styles+
    //"&transparent=TRUE"+
    "";
return url;
}

```

Come si può constatare dal codice, viene inizializzato di default il server della Nasa <http://wms.jpl.nasa.gov/wms.cgi>.

È possibile comunque utilizzare qualunque servizio impostando un diverso URL e diversi parametri.

Le operazioni definite dal servizio WMS sono:

GetCapabilities: restituisce informazioni sul servizio offerto dal server, sul tipo di dati che può inviare e sui parametri che può accettare dal client;

GetMap: restituisce una mappa i cui parametri geografici e dimensionali sono ben definiti;

GetFeatureInfo: restituisce informazioni riguardo particolari elementi mostrati sulla mappa. La disponibilità di questa operazione sul server non è assicurata.

E i parametri più comuni:

Version:(opzionale) questo parametro specifica il numero di versione del protocollo. Il numero di versione è formato da tre numeri interi positivi, separati da punto, nella forma “x,Y,Z”. Questi numeri sono soggetti a cambiamento dopo ogni revisione del “Web Map Service Implementation Specification” da parte dell’Open Gis Consortium. I clients devono usare lo stesso numero di versione che il server ha dichiarato di supportare attraverso le informazioni restituite con `getCapabilities()`. Un server può anche supportare diverse versioni, queste possono essere scoperte dal client attraverso delle regole di negoziazione.

Service: indica quale tipo di servizio viene invocato, tra quelli disponibili per una determinata istanza.

Request: indica quale operazione di servizio si sta invocando, `GetCapabilities`, `GetMap` o `GetFeatureInfo`.

Format: indica il tipo di formato di risposta ad una operazione. I formati attraverso i quale il servizio web può rispondere, devono essere definiti nel suo `GetCapabilities XML`.

Exception: indica il formato attraverso il quale il richiedente vuole essere notificato le eccezioni.

SRS (Spatial Reference System): indica il sistema di coordinate di riferimento. È possibile fare uso di due namespace, EPSG e AUTO. Il primo fa uso delle tabelle dell’ “European Petroleum Survey Group”. È possibile anche utilizzare il valore NONE in caso di informazioni geografiche il cui riferimento spaziale non è definito.

BBOX (Bounding box): è un insieme di quattro numeri decimali o interi. Questi rappresentano il minimo X, il minimo y, il massimo x, il massimo y.

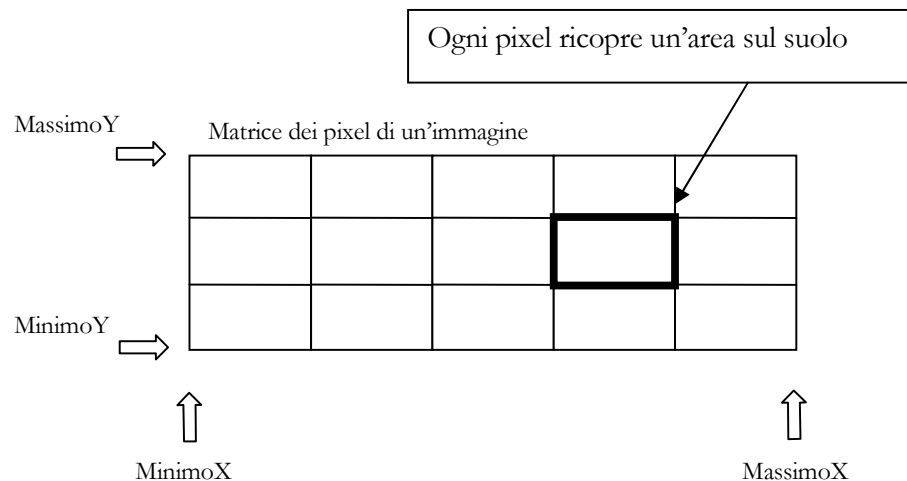


Figura 16 BBox

Layers: indica il tipo di layer che deve essere restituito con la mappa.

Styles: questo parametro indica la lista di stili con cui ogni layer deve essere renderizzato.

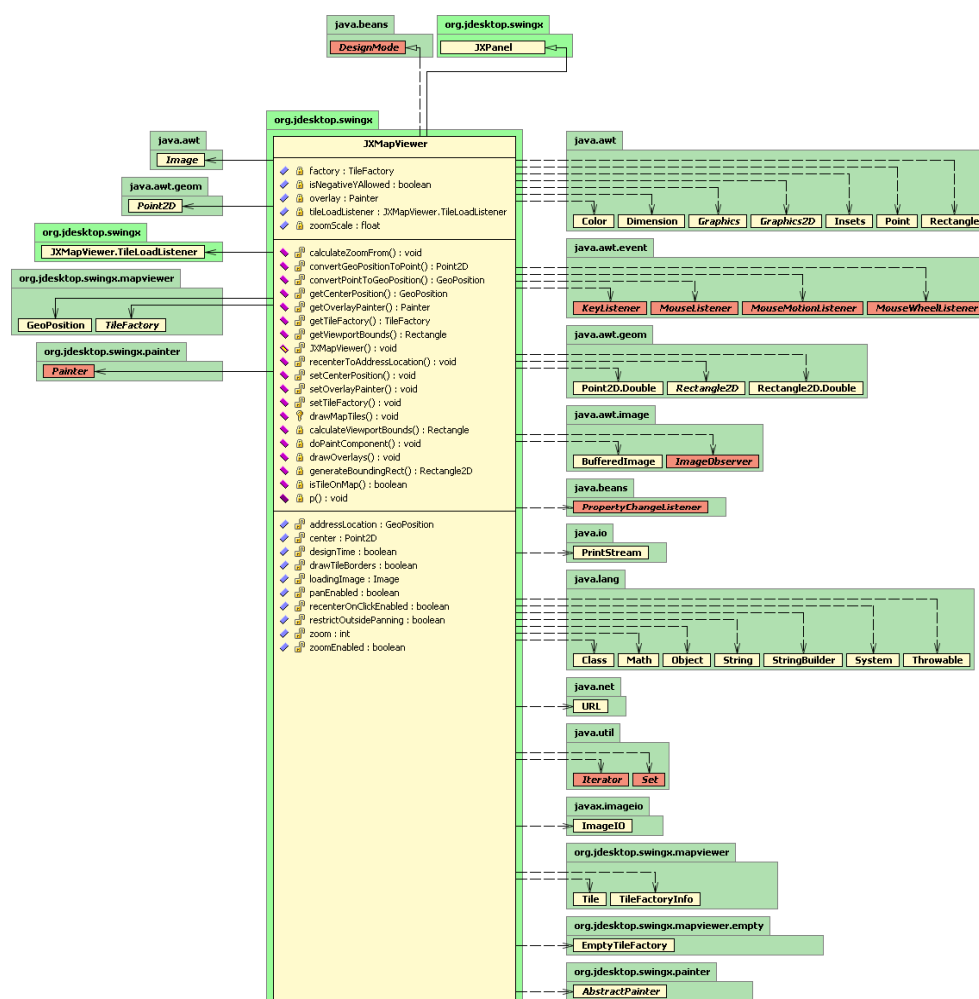
Width, Height: Indicano la dimensione, espressa in pixel, dell'immagine contenente la mappa da produrre.

Transparent: questo parametro può avere valore "false" o "true". Specifica se lo sfondo della mappa debba essere trasparente o meno. Questo permette di sovrapporre mappe provenienti da diverse request GetMap().

BGColor: indica il colore dello sfondo espresso in esadecimale nella forma RGB.

## JXMapView (pacchetto `org.jdesktop.swingx.mapviewer`):

Questo è un utilissimo oggetto fornito con il pacchetto *swing-x*. È predisposto per la visualizzazione di una mappa digitale ed è indipendente dal tipo di mappa e dal tipo di codifica necessario. Una volta impostato il `TileFactory`, permette di visualizzare la mappa e di interagire attraverso operazioni di pan e zoom. Gli evento mouse quindi sono riconosciuti.

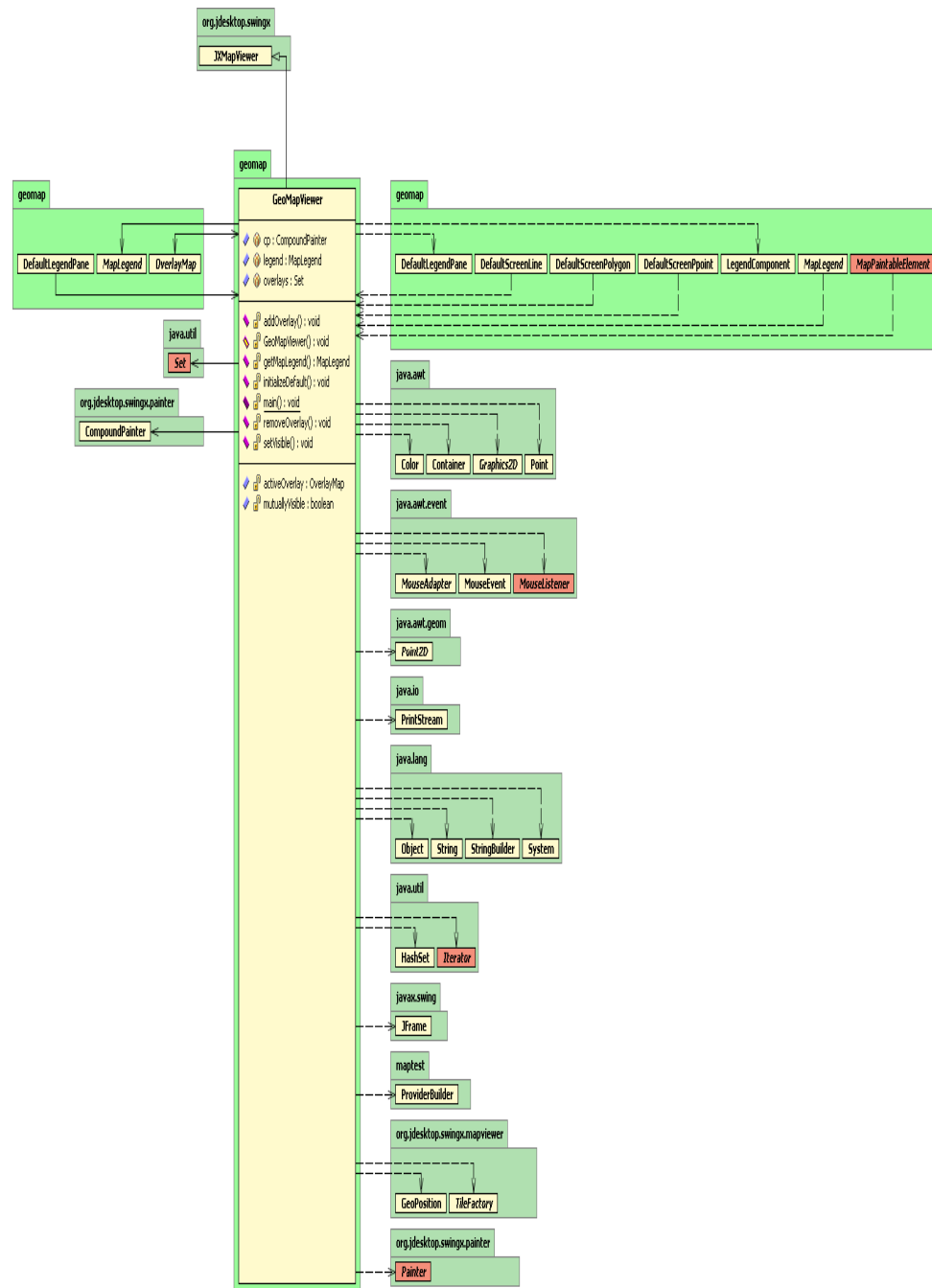


## GeoMapView

Questa classe è un'estensione di JXMapView. La classe vista in precedenza non è di facile utilizzo e per la visualizzazione di grandi overlays complica la stesura del codice. Il programmatore, infatti, è tenuto a fare attenzione alla differenza tra le coordinate dell'immagine sullo schermo in pixel e le coordinate geografiche. La gestione degli eventi diventa un altro ostacolo se si vogliono associare eventi diversi a diversi tipi di overlay. È necessario quindi, avere un migliore livello d'astrazione e progettare le figure geometriche che vengono disegnate come degli oggetti di tipo mappa. Applicando un modello di tipo MVC<sup>27</sup>, si è reso opportuno creare un meccanismo di delega affinché l'oggetto GeoMapView funga da "Viewer" nei confronti degli oggetti MapObject, che diventeranno il relativo "Model". Il ruolo del controller sarà preso da un oggetto di tipo OverlayMap (più avanti descritto). GeoMapView conserva tutti gli OverlayMap in un oggetto di tipo CompoundPainter; un overlay rappresenta un painter. Ad ogni operazione di disegno il compoundpainter viene chiamato a disegnare tutti gli overlay. In più vi è la possibilità di definire: il layer attivo (cioè quello a cui verrà delegata la ricezione di eventi), i layer visibili e la possibilità di ottenere una legenda creata automaticamente in base ai layers disponibili. L'utilizzo di questa classe è semplice e non richiede al programmatore la conoscenza del funzionamento a basso livello. Con una sola istruzione, si può aggiungere un'intero overlay e la relativa componente alla legenda. Per meglio comprendere il significato di questo paragrafo, vedere la parte di testo relativa al map overlay. Infine questa classe contiene un'istanza di MapLegend, che indica un componente visuale rappresentante una legenda degli overlay caricati.

---

<sup>27</sup> Model View Controller,





## OverlayMap

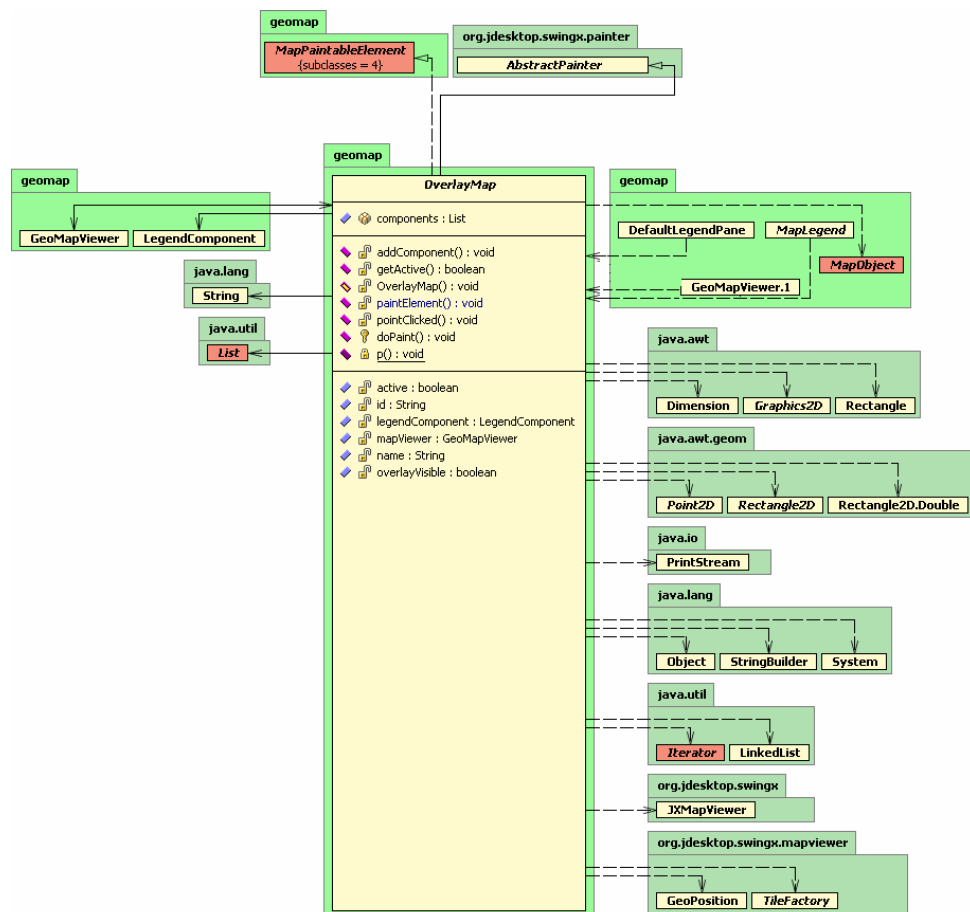
È la classe che rappresenta un insieme di oggetti mappa che compongono un overlay. Per creare un'overlay sul `GeoMapView` è necessario passargli un'istanza di una classe che estende `OverlayMap`. Questa classe è definita astratta perché, non conoscendo a priori le operazioni da eseguire in caso di eventi, non implementa il metodo `pointClicked()`. Qui di seguito un maggiore chiarimento sui principali metodi.

- `addComponent`: prende in input un parametro di tipo `MapObject`, che rappresenta un oggetto componente l'overlay e lo memorizza in una lista contenente tutti i componenti.
- `getActive`: ritorna un valore booleano se la mappa è attiva. Se una mappa è attiva può rispondere agli eventi.
- `isOverlayVisible/setOverlayVisible`: sono dei metodi per definire se la mappa è visibile o meno.
- `doPaint`: è un metodo ereditato dalla classe `AbstractPainter`. Questo è richiamato dal `GeoMapView` ogni volta che è necessario disegnare l'overlay. Questo metodo è molto importante perché se il parametro `overlayVisible` è impostato a `true`, prepara il contesto grafico in relazione alla porzione di mappa visualizzata. In più, istanzia una lista di oggetti *Rectangle2D* che definisco le porzioni visibili di mappa sullo schermo. Tutto ciò permette al programmatore di ignorare i dettagli di visualizzazione sullo schermo della mappa. Il calcolo delle zone visibili è necessario per non eseguire codice per disegnare oggetti che non sono nel campo visivo della mappa sullo schermo. Dopo aver preparato il contesto grafico, questo metodo, richiama `paintElement()`;

- `paintElement`: è un metodo che fa parte dell'interfaccia `"MapPaintableElement"`. Questo richiama il metodo `paintElement` di tutti i componenti di tipo `ObjectMap` memorizzati nella lista di `OverlayMap` e che rappresentano un'istanza di `"MapPaintableElement"`. Questo permette al programmatore di ignorare i dettagli grafici di ogni elemento. In teoria affida il compito ad ogni oggetto di tipo `MapPaintableElement` di disegnare se stesso.
- `pointClicked`: è un metodo chiamato da `GeoMapView`, su un `OverlayMap` attivo, ogni volta che intercetta un'azione di click del mouse. I parametri di input sono due:
  - un oggetto di tipo `Point` che indica le coordinate del punto cliccato espresse in pixel;
  - un oggetto di tipo `GeoPosition` che indica le coordinate geografiche per punto cliccato.

Il programmatore dovrà sovrascrivere questo metodo per impostare particolari operazioni a seguito di eventi.

- `getLegendComponent/setLegendComponent`: fa riferimento ad un oggetto di tipo `LegendComponent` utilizzato da `GeoMapServer` per creare una legenda degli overlay. Nel componente restituito vi è installato un ascoltatore di eventi che in base agli eventi ricevuti, permette di impostare i parametri `"overlayVisible"` e `"active"`.

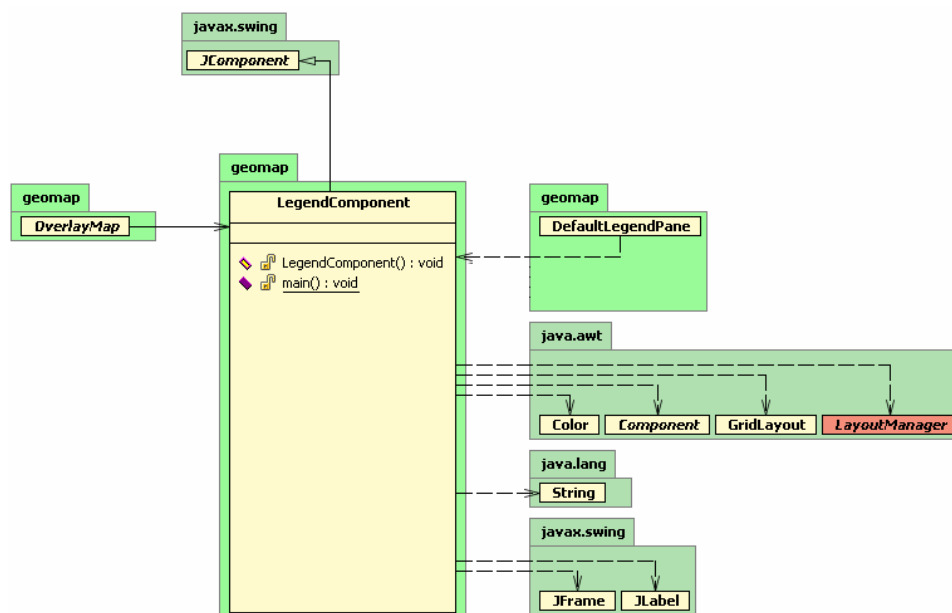


## LegendComponent

Crea una componente legenda da poter aggiungere ad un pannello(vedere esempio di pannello con tre LegendComponent) . È sufficiente specificare nel costruttore, un oggetto di tipo Color e una stringa (anche in formato html).

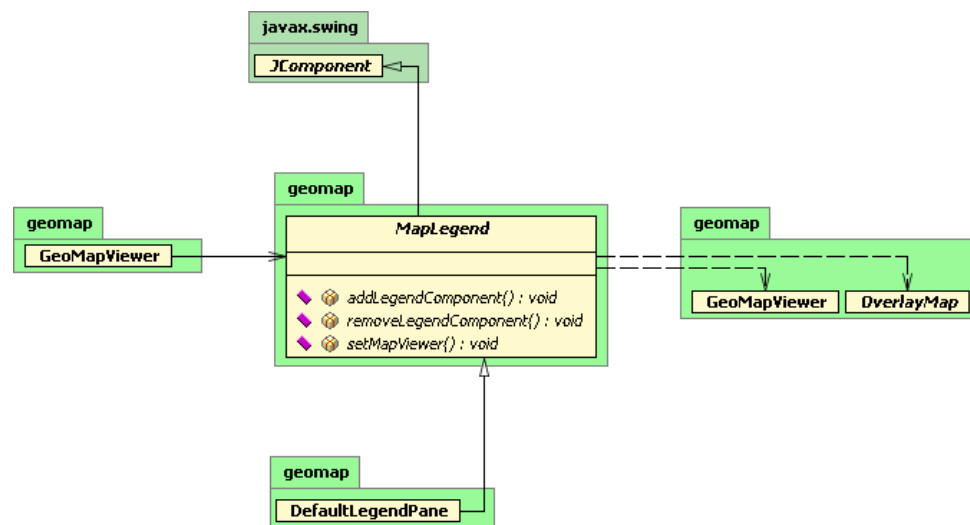


Figura 17: esempio di pannello con tre LegendComponent



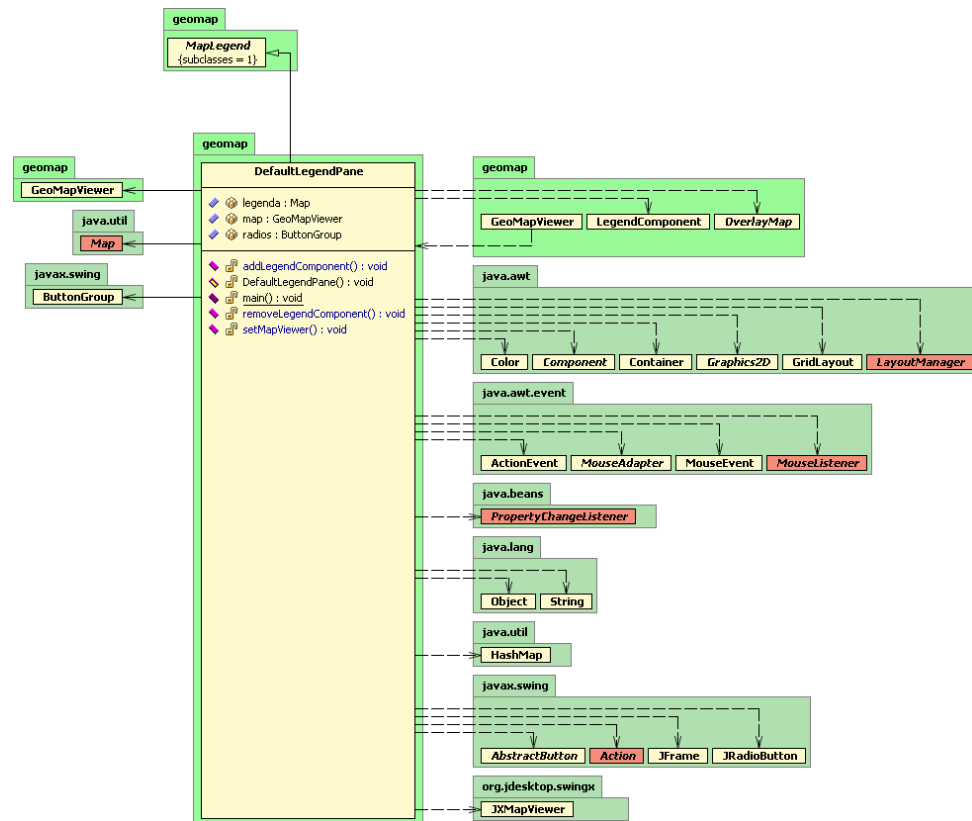
## MapLegend

È una classe astratta utilizzata come interfaccia. Tale scelta è dovuta all'alta probabilità che future versioni di questa libreria implementeranno alcuni metodi. Per il momento, questa classe è implementata da DefaultLegendPane.



## DefaultLegendPane

Questa classe estende `MapLegend`. È istanziata da `GeoMapView` che, all'aggiunta di ogni overlay, richiama il metodo `addLegendComponent`. Questo metodo accetta come parametro di input un oggetto di tipo `OverlayMap`, dal quale ricava il `LegendComponent`, vi aggiunge un campo radio e due ascoltatori di eventi. Il primo per intercettare le azioni sul bottone radio e settare il relativo overlay enabled. Il secondo per intercettare l'azione del mouse sull'icona corrispondente all'overlay per renderlo visibile o meno.

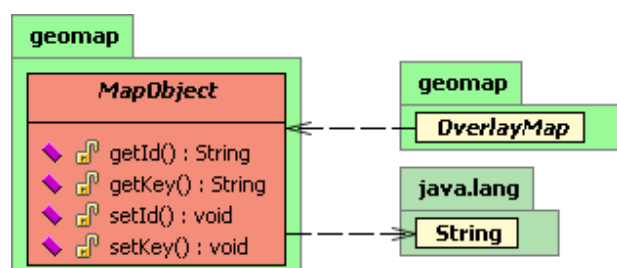


## MapObject

Nel concepire la struttura di questo oggetto, si è fatto riferimento al diagramma degli oggetti KML. MapObject è una semplice interfaccia per la gestione di due parametri, ma concettualmente permette immaginare gli overlay come collezioni di oggetti. I metodi dichiarati agiscono su due parametri, “id” e “key” che le classi, che implementano questa interfaccia, possono definire o meno (impostare a “null”). Il parametro id consente di referenziare oggetti condivisi o anche di utilizzare tecniche di update<sup>28</sup>. Il parametro key invece, pur non avendo ancora un impiego reale, è stato progettato per consentire, in versioni successive della

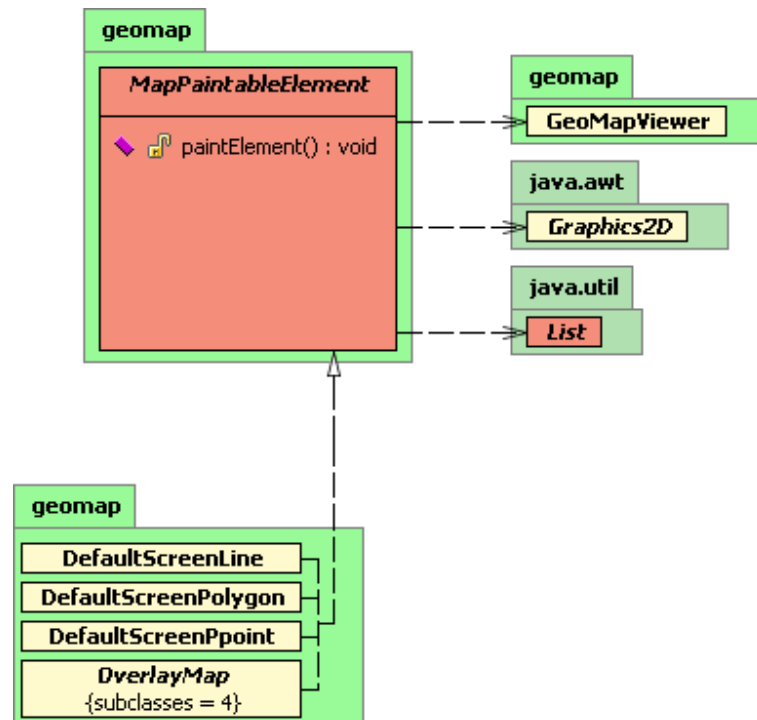
<sup>28</sup> Consultare la “KML 2.1 Reference”

libreria, l'utilizzo di tecniche di caricamento dinamico. Queste si saranno indispensabili, nella gestione di grandi quantità di informazioni, per minimizzare la complessità computazionale e temporale. Ad esempio, nel campo key, si potrebbe voler scrivere solo un riferimento (posizione in un file, query SQL, etc..) ad un oggetto per poi caricare tutte le informazioni in un secondo momento e solo in caso di effettivo bisogno.



## MapPaintableElement

Questa interfaccia ha premesso la delega delle operazioni di disegno dal **GeoMapView** fino all'oggetto che deve essere disegnato. Questa catena di deleghe si basa sul concetto secondo il quale, nessuno conosce l'oggetto meglio dell'oggetto stesso. Quindi un oggetto che implementa questa interfaccia è capace di disegnarsi. Tutti gli oggetti che devono essere disegnati devono implementare questa interfaccia. Quando sarà necessaria un'operazione di disegno, sarà chiamato il metodo `paintElement` con tre parametri in input: un'istanza di **GeoMapView**, un contesto grafico già predisposto per il disegno e una lista di rettangoli che compongono la zona visibile della mappa.



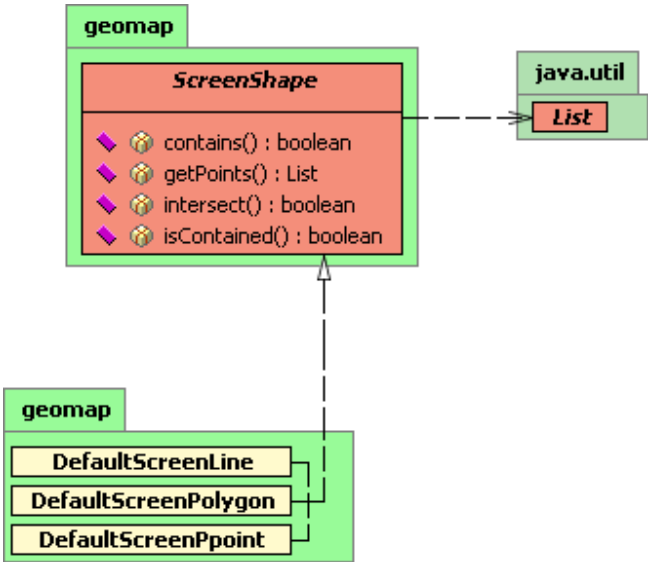
## ScreenShape

Questa interfaccia descrive una generica figura geometrica che può essere disegnata sullo schermo. Dichiara quattro metodi:

- **contains**: dato in input un oggetto di tipo **ScreenShape**, restituisce **true** se questo è contenuto nell'oggetto di cui è stato chiamato il metodo;
- **getPoints**: restituisce una lista di oggetti **Point** che rappresentano i punti in cui è definita la figura geometrica. Nel caso di figure geometriche bidimensionali o composite, la lista indicherà solo i contorni;
- **intersects**: restituisce **true** se vi è intersezione con l'oggetto **ScreenShape** dato in input.



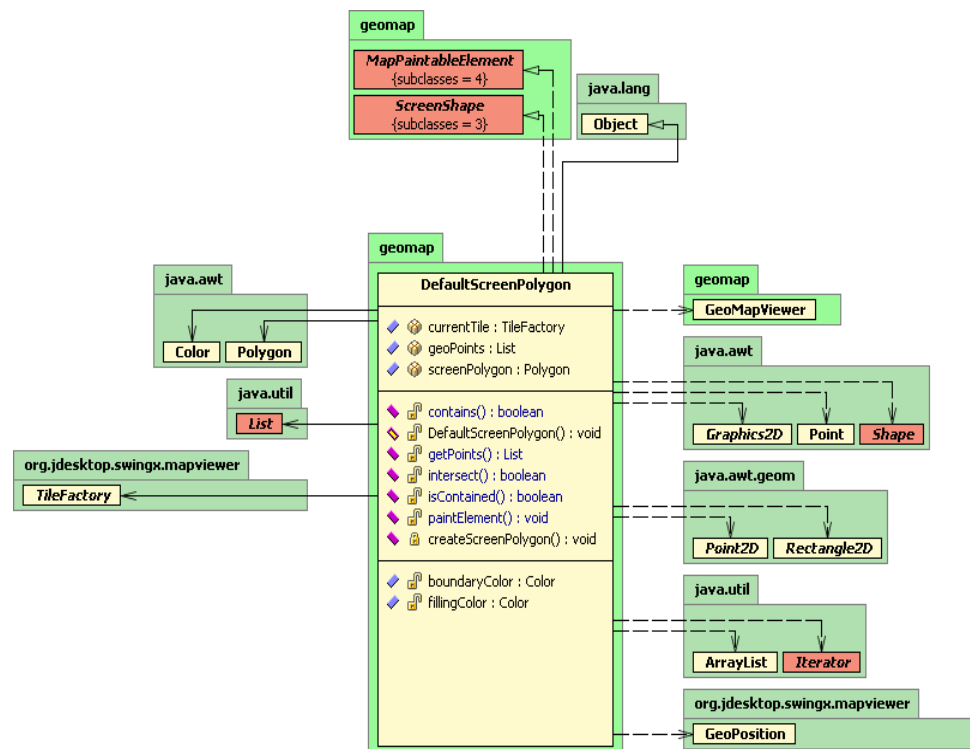
- `isContained`: restituisce `true` se tutti i punti della figura sono contenuti nella lista di `Rectangle` data in input. Questo metodo è utilizzato per disegnare solo le figure che appartengono alla porzione di mappa visualizzata sullo schermo. In un caso particolare questo metodo risulta inefficace. Ad esempio, nel caso in cui l'area visuale dello schermo sia inscritta nella figura geometrica, il metodo ritorna un valore `false` in quanto tutti i punti della figura sono esterni all'area visualizzata nello schermo.



## DefaultScreenPolygon, DefaultScreenLine, DefaultScreenPoint

Si è data un'implementazione di default dell'interfaccia `ScreenShape` per le principali figure geometriche: linee, punti, poligoni. Per brevità si presenterà solo la classe `DefaultScreenPolygon`.

DefaultScreenPolygon conserva informazioni importanti per la sua visualizzazione come colore di riempimento e colore dei contorni. Per implementare i metodi dell'interfaccia ScreenShape, sfrutta le capacità dell'oggetto "Polygon" fornito dalle librerie di base di Java. Questo oggetto è istanziato durante la prima operazione di disegno. Infatti, al momento della creazione di DefaultScreenPolygon, può non essere noto l'oggetto TileFactory utilizzato da GeoMapView. Ciò implica che non è possibile richiamare i metodi dell'interfaccia ScreenShape prima che questa figura sia stata disegnata almeno una volta. Un vincolo così forte, purtroppo, non giova alla semplicità d'utilizzo di questi oggetti e richiede molta attenzione da parte del programmatore.



## **Conclusioni**

Sono stati descritti i componenti principali della libreria sviluppata. Questi sono stati progettati in considerazione dei requisiti funzionali e dei requisiti non funzionali descritti nel secondo capitolo. Nonostante ciò, alcuni ancora presentano dei problemi di tipo logico o sono carenti sotto l'aspetto della complessità computazionale.

La libreria presentata, lungi dall'essere esaustiva, vuole rappresentare un approccio allo sviluppo modulare di applicazioni, presentando delle linee guida attraverso le interfacce da essa definite.

## *Capitolo 5*

### UN PROTOTIPO

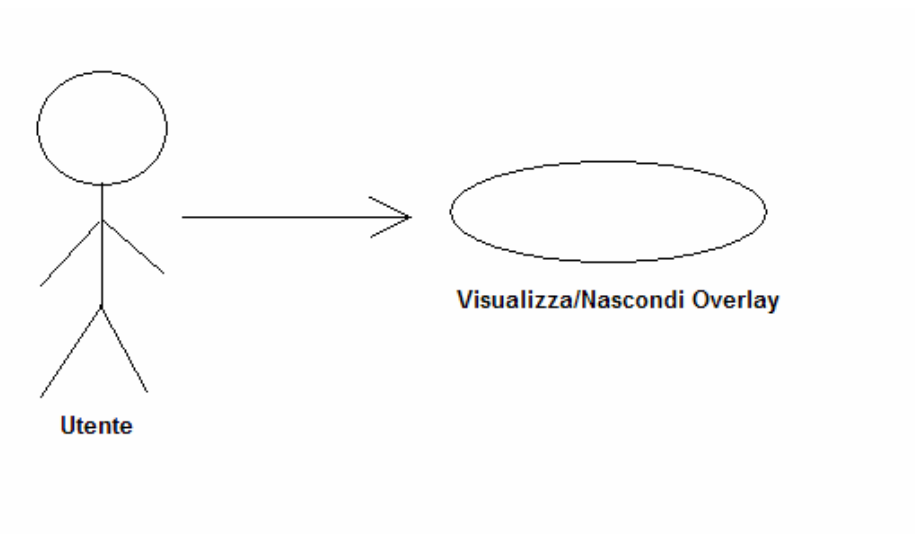
Alla fine della fase di implementazione del pacchetto, verrà proposta una classe demo che testerà alcune delle funzionalità sviluppate in considerazione dei requisiti funzionali. La classe demo sarà un'applicazione java che farà uso della libreria progettata nel capitolo precedente. Tale applicazione sarà composta da una mappa contenente un pannello, il quale consentirà, attraverso dei pulsanti, di aggiungere o rimuovere overlays.

#### **Modello dei casi d'uso**

Qui di seguito verranno mostrati dei casi d'uso per la classe demo. Questi casi d'uso, per brevità, sono applicati allo scenario in cui gli unici attori del sistema sono un generico utente ed il sistema stesso.

- Caso d'uso codice X1: con questo caso d'uso si mostra il flusso di eventi mediante il quale un generico utente comanda al sistema di visualizzare o nascondere degli overlay, attraverso azioni su una leggenda.

CODICE: X1
------------



**NOME USE CASE:** Visualizza/NascondiOverlay

---

**PARTECIPANTI:** Inizializzato da Utente

---

**FLUSSO DI EVENTI:**

1. L'utente clicca su uno dei bottoni della legenda;
2. Se il layer associato al bottone della mappa è già visualizzato, il sistema lo nasconde, altrimenti lo visualizza.

---

**CONDIZIONI DI INGRESSO:**

- Il client ha instaurato una connessione con il server per le mappe di fondo di Google;
- Ha caricato alcuni layer;

- La legenda mostra l'elenco dei layer caricati e/o visualizzati;
- La mappa è visualizzata sullo schermo.

---

**CONDIZIONI DI USCITA:** L'utente ha visualizzato/nascosto gli overlay.

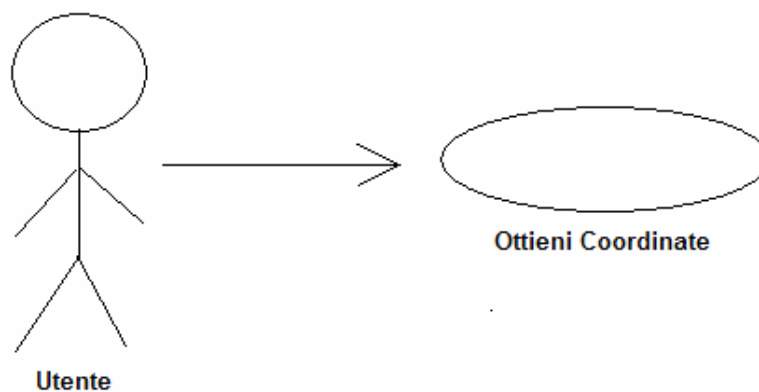
---

ECCEZIONI: La connessione al provider delle mappe digitali è fallita.

---

Caso d'uso codice X2: con questo caso d'uso si mostra il flusso di eventi con cui un generico utente riesce ad avere le coordinate geografiche di un punto cliccando sulla mappa.

CODICE: X2
------------



**NOME USE CASE:** OttieniCoordinate

---

**PARTECIPANTI:** Inizializzato da Utente

---

**FLUSSO DI EVENTI:**

1. L'utente clicca in un punto qualsiasi della mappa;
2. Il sistema mostrerà le coordinate geografiche del punto cliccato;

---

**CONDIZIONI DI INGRESSO:**

- Il client ha instaurato una connessione con il server per le mappe di fondo di Google;
- La mappa è visualizzata sullo schermo.

---

**CONDIZIONI DI USCITA:** Le coordinate del punto cliccato sono state stampate a video.

---

**ECCEZIONI:** La connessione al provider delle mappe digitali è fallita.

---

## Interfaccia utente

Qui di seguito si riporta uno screen mock-up dell'aspetto che avrà il programma Demo:

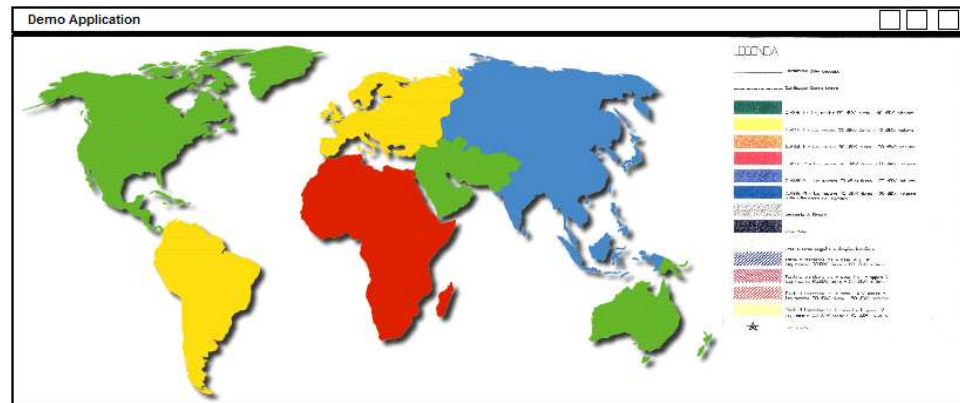


Figura 18 Screen Mock-up

Lo screen mock-up in Figura 18 denota l'aspetto visuale che avrà il prototipo. L'area visualizzata dal pannello sarà ricoperta per la maggior parte, dall'immagine della mappa. In un'altra area di dimensioni minori, sarà visualizzata una legenda. Nella barra superiore vi saranno tre pulsanti per ridurre ad icona, ingrandire a pieno schermo, e chiudere l'applicazione.



## *Capitolo 6*

### CONCLUSIONI

Nel corso del lavoro è stato affrontato il problema dell'integrazione di mappe Google in applicazioni stand alone. Ad oggi, infatti, non sono noti sistemi o tecnologie capaci di implementare tali funzionalità. Probabili cause della mancanza di tali applicativi, i termini del contratto di licenza delle Google Maps API's, i quali prevedono l'utilizzo del servizio con il ricorso esclusivo alle API fornite dalla società.

Momento di rilevante importanza è stato quello dello studio di fattibilità. Per poter utilizzare le informazioni geografiche di Google, infatti, è stato indispensabile "scoprire" un metodo per interagire con il Google Map Server. Un primo tentativo di effettuare richieste WMS è fallito in quanto il server, non offre alcun supporto per gli standard OGC. Successivamente un'attenta analisi del codice che Google carica lato Client, quando si utilizza l'interfaccia Google Map, ha permesso di scoprire il tipo di codifica con il quale il client richiede una determinata porzione di mappa e il formato delle stringhe di richiesta URL. L'esito positivo di test basati su gli url scoperti ha determinato la fattibilità del progetto.

La scelta del linguaggio Java per la creazione di una libreria, ha consentito l'utilizzo di alcuni oggetti già esistenti.

La libreria proposta, infatti, è un'estensione del pacchetto *org.jdesktop.swingx.mapviewer* della libreria *swing-x* prodotta dalla *SwingLabs* ed è concepita per nascondere al programmatore i dettagli di connessione e di codifica delle mappe. Si presenta come uno strumento duttile, facile da utilizzare ed estendibile

Si giunge alla conclusione che si possono sviluppare applicazioni che integrano mappe Google e, quindi, applicazioni “Google-based”, ma non è possibile utilizzarle senza un'autorizzazione di Google.

## *A p p e n d i c e   A*

### APPROFONDIMENTI SU GOOGLE EARTH

I seguenti paragrafi sono tratti integralmente dalla guida utente di Google Earth disponibile all'indirizzo:

[http://earth.google.com/userguide/v4/ug\\_importdata.html](http://earth.google.com/userguide/v4/ug_importdata.html).

Si ritiene che tali approfondimenti siano importanti per:

- Ipotizzare nuovi casi di applicazione della libreria presentata nella tesi;
- Comprendere le differenze tra i vari sistemi di coordinate e verificare come questi siano influenzati dal tipo di proiezione utilizzata;
- Ideare nuove estensioni della libreria presentata nella tesi;

#### **Importazione di dati in Google Earth**

Puoi utilizzare la funzione di importazione dati di Google Earth per importare i tuoi dati geografici personalizzati in Google Earth e visualizzarli come faresti con qualsiasi [livello](#) nella finestra Livelli. La funzione di importazione ti permette di importare sostanzialmente due tipi di dati:

- Dati vettoriali: i dati vettore sono costituiti da punti, linee, percorsi e poligoni. In genere, i dati dei punti vengono utilizzati per descrivere

luoghi specifici, ad esempio il centro di una città. Linee e percorsi possono essere utilizzati per strade o confini, mentre i poligoni possono essere utilizzati per delineare appezzamenti di terreno o altre aree, ad esempio un lago. Puoi importare in Google Earth i dati vettore messi a disposizione da altri fornitori di dati, come ad esempio ESRI (Environmental Systems Research Institute). Inoltre, puoi utilizzare file di testo generici per l'importazione dei dati di punti da te definiti. Una volta importati i dati vettore in Google Earth, puoi modificarne l'aspetto o il contenuto procedendo come per la modifica di segnaposto e cartelle. Inoltre, puoi utilizzare Modelli di stile per una formattazione visivamente più immediata dei dati. I [dati KML](#) possono essere visualizzati anche in una sequenza temporale. Per ulteriori informazioni, vedi la sezione [Visualizzazione di una sequenza temporale](#).

- **Dati immagine:** puoi importare dati immagine quali mappe aeree o topografiche e fare in modo che vengano proiettate correttamente sulle immagini di base nel visualizzatore 3D. Perché ciò sia possibile, il file immagine deve essere di un formato specifico. Le immagini di questo tipo sono dette immagini GIS.

Nota: la funzione di importazione di immagini GIS è disponibile unicamente per gli utenti di Google Earth Plus, Pro e EC. Google Earth Plus può importare fino a 100 indirizzi in una volta.

Una volta importati i dati vettore o i dati immagine nell'applicazione Google Earth, puoi salvare i dati modificati procedendo come per qualsiasi altro segnaposto o sovrapposizione.

### **Importazione di dati vettore**

Google Earth supporta i seguenti tipi di dati vettore:

- Punti

- Linee e percorsi
- Poligoni, inclusi i poligoni pieni

Il processo di importazione dei file di dati vettore è semplice.

1. Importa il file vettore procedendo in uno dei seguenti modi:
  - **Utilizzando il trascinamento:** individua il file sul tuo computer o su un server di rete e trascinalo nel visualizzatore 3D di Google Earth.
  - **Tramite l'opzione Apri o Importa del menu File:** selezionando questa opzione puoi specificare il tipo di dati che intendi importare (ad es. TXT, SHP, TAB) oppure scegliere l'opzione Tutti i formati di importazione dati.

Puoi aprire file salvati su computer presenti sulla rete procedendo come per l'apertura di qualsiasi altro file. Per aprire un file situato su un browser web, devi prima scaricare il file stesso e tutti i file correlati sul tuo computer o sulla rete locale.

2. Quando ti viene richiesto, scegli se applicare o meno un modello di stile. Se decidi di applicare un modello di stile, puoi definire sul momento un nuovo modello oppure selezionarne uno esistente già definito per quel file di dati. Per ulteriori informazioni, vedi l'argomento [Uso dei modelli di stile](#).

Al termine dell'importazione, gli elementi vettore appaiono nel visualizzatore 3D e il file importato viene elencato nella cartella Luoghi temporanei. Etichette, icone, colore e descrizione vengono visualizzati come per gli altri tipi di luoghi o cartelle, a seconda di come sono stati definiti attraverso il modello di stile.

Nota: se non utilizzi un modello di stile per modificare l'aspetto dei dati importati, Google Earth cerca un campo Nome da utilizzare come etichetta dei dati. L'etichetta compare nel visualizzatore 3D accanto ai punti e nell'elenco della finestra Luoghi temporanei. Se i dati non contengono un campo Nome, il primo campo disponibile contenente testo viene utilizzato come etichetta dei dati.

Il seguito di questa sezione illustra i seguenti argomenti:

- [Uso di dati vettore di altri fornitori](#)
- [Uso di file di testo generici](#) contenenti dati di punti delimitati da virgole o tabulazioni (solo gli utenti di Google Earth EC e Google Earth Pro possono importare i file di dati vettore GIS; tutte le versioni di Google Earth permettono tuttavia di importare file di testo generici)
- [Importazione di immagini](#)
- [Breve introduzione alle proiezioni e ai datum](#)

### **Uso di dati vettore di altri fornitori**

Gli utenti di Google Earth EC e Google Earth Pro, che dispongono del modulo di importazione dati, sono in grado di importare i seguenti formati di file vettore:

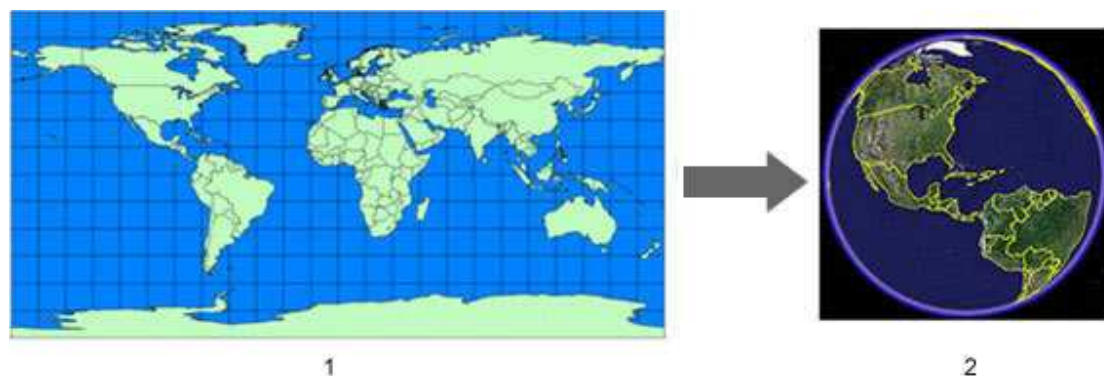
- MapInfo (TAB) - File richiesti:
  - MAP ID
  - DAT
- ESRI Shape (SHP) - Richiede SHX Dati di proiezione, incorporati nel file SHP o definiti in un file a parte con estensione PRJ DBF (per la visualizzazione dei dati dei campi)

La maggior parte dei dati vettore GIS di fornitori terzi consiste in una raccolta di file correlati che interagiscono per produrre tutti i dati vettore visualizzati in Google Earth. Se i dati previsti non compaiono nel visualizzatore 3D, è possibile che manchino file di supporto. I tipi di file vettore che richiedono file di supporto aggiuntivi sono indicati nell'elenco precedente.

Nota: puoi utilizzare file di testo generici per creare i tuoi dati di punti da utilizzare in Google Earth. La funzione è disponibile anche per gli utenti di Google Plus.

### Breve introduzione alle proiezioni e ai datum

Per le sue immagini, Google Earth utilizza la proiezione cilindrica equirettangolare con datum WGS84.



1. Proiezione cilindrica equirettangolare (Plate Carree)
2. Base di immagini di Google Earth

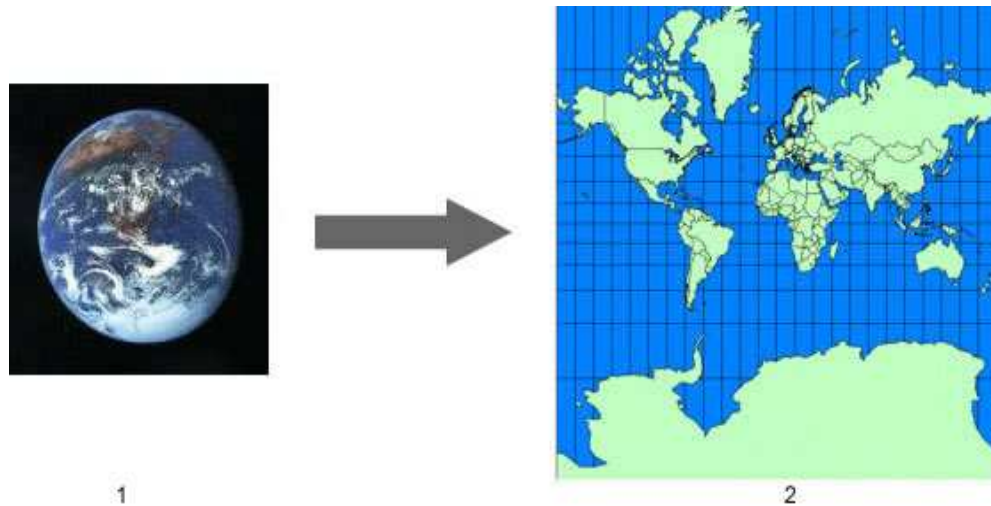
In genere, i dati importati nell'applicazione Google Earth vengono creati in base a un sistema di coordinate geografiche specifico, ad esempio con la proiezione Universale Trasversa di Mercatore (UTM) e datum NAD27 (North American Datum del 1927). Ciascun sistema di coordinate geografiche può assegnare coordinate leggermente diverse alla stessa posizione sul pianeta. Quando importi dati in Google Earth, questi vengono interpretati in base al sistema di coordinate di Google Earth.

Nella maggior parte dei casi la riproiezione funziona come previsto; in altri, invece, non funziona correttamente. In questi casi puoi utilizzare uno strumento di terze parti per trasformare i dati dal sistema di coordinate iniziale a quello adottato da Google Earth.

Il seguito di questo argomento fornisce una breve panoramica su proiezioni e datum.

### **Che cos'è una proiezione cartografica?**

Una proiezione cartografica è un'espressione matematica utilizzata per rappresentare la superficie sferica tridimensionale del pianeta su una carta bidimensionale.



1. Terra in 3D
2. Proiezione di Mercatore


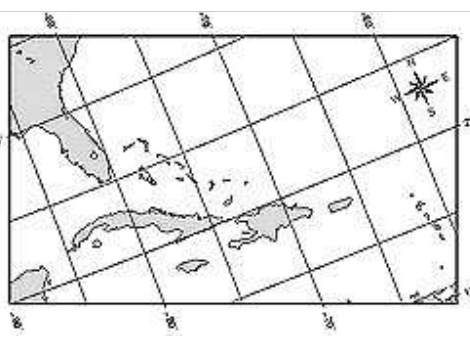
Questo processo produce sempre una distorsione di una o più proprietà cartografiche quali area, scala, forma o orientamento. Per ovviare a questa distorsione, sono stati sviluppati centinaia di tipi di proiezioni, al fine di rappresentare correttamente un particolare elemento o un certo tipo di mappa.


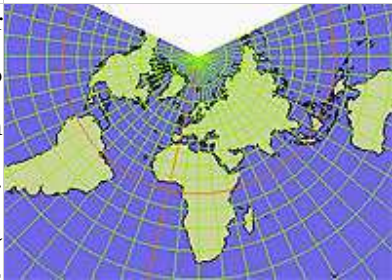
Le fonti di dati delle mappe vengono espresse in varie proiezioni a seconda della caratteristica che il cartografo sceglie di rappresentare in modo più accurato (a discapito di altre caratteristiche). Nell'esempio in alto, la proiezione di Mercatore preserva gli angoli retti delle intersezioni tra latitudine e longitudine a discapito



dell'area, che risulta distorta ai poli, mostrando masse di terra di dimensioni maggiori rispetto alla realtà.

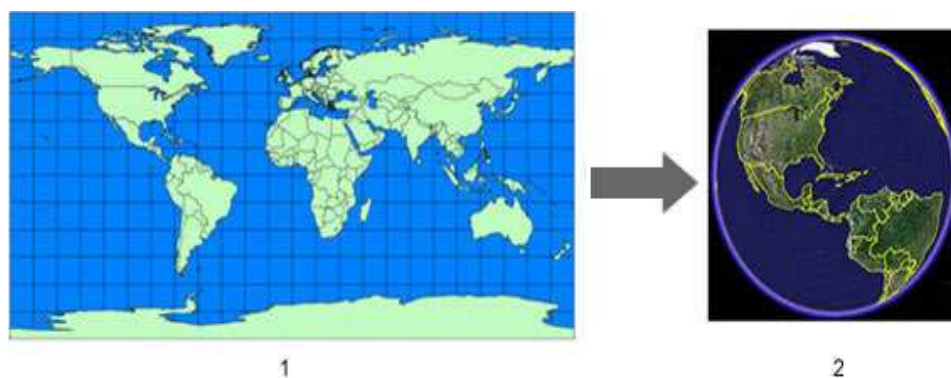
Ecco alcuni esempi delle proiezioni cartografiche più diffuse:

(Plate Carree)	Descrizione	Esempio
Conica equivalente di Albers	Utilizzata in genere per regioni o Paesi di piccole dimensioni con estensione est-ovest, ma non per continenti. Preserva gli angoli tra i meridiani e i paralleli. Tenta di ridurre al minimo la distorsione di forma e scala lineare, ma nessuna delle due risulta perfettamente corretta. L'esempio mostra la proiezione applicata all'intero globo.	
Obliqua di Mercatore (Hotine)	Proiezione cilindrica simile alle proiezioni di Mercatore, in cui però il cilindro viene allineato a una regione obliqua e non segue gli assi nord-sud o est-ovest. La regione che	

	viene mappata è generalmente una piccola porzione lungo il meridiano e ad esso affiancata. Nell'esempio, la proiezione è stata sviluppata in origine per la mappatura della penisola malese.	
Trimetrica di Chamberlin	Utilizzata dalla National Geographic Society per la mappatura della maggior parte dei continenti. Si tratta di una proiezione equidistante a tre punti che mira a preservare la distanza tra tre punti di riferimento rispetto a qualsiasi altro punto.	
Conica Lambert	Proiezione ideale per latitudini intermedie e/o per territori con orientamento est-ovest. Questa proiezione si trova spesso nelle mappe USGS (United States Geological Survey) create dopo il 1957. La scala è più	

	accurata a discapito dell'area.	
--	---------------------------------	--

Nella creazione di database relativi all'intero pianeta, un'unica proiezione globale rappresenta la soluzione più idonea. Per la sua base di immagini Google Earth utilizza la proiezione cilindrica equirettangolare. Si tratta di una semplice proiezione cartografica in cui i meridiani e i paralleli sono rappresentati da linee rette equidistanti, che si intersecano ad angolo retto. Questo tipo di proiezione è conosciuto anche come sistema di riferimento globale WGS84.

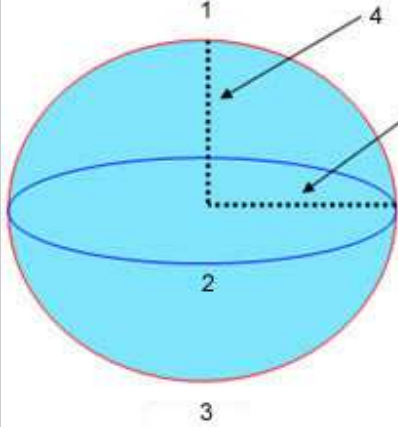


1. Proiezione cilindrica equirettangolare (Plate Carrée)
2. Base di immagini di Google Earth

### Che cos'è un datum?

Mentre la proiezione è utilizzata nella mappatura per rappresentare il globo su una superficie piana, il datum è utilizzato per descrivere la forma reale del pianeta in termini matematici. Questo perché la superficie del nostro pianeta non è perfettamente sferica, ma piuttosto ellissoide. Un datum definisce inoltre l'associazione di coordinate di latitudine e longitudine a punti sulla superficie della Terra e definisce la base per la misurazione delle elevazioni.

Come avviene per le proiezioni, c'è più di una interpretazione matematica della forma della Terra. Google Earth utilizza il datum WGS84.



1. Polo Nord  
 2. Equatore  
 3. Polo Sud  
 4. Semiasse minore o raggio polare  
 5. Semiasse maggiore o raggio equatoriale

		Semiasse maggiore	Semiasse minore
	NAD83	6,378,137.0	6,356,752.3141
	WGS84	6,378,137.0	6,356,752.3142
	Clark 1866	6,378,206.4	6,356,583.8
	Airy 1830	6,377,563.4	6,356,256.9

## *A p p e n d i c e B*

### IL PARADIGMA REST

REST descrive uno stile di architettura per i sistemi che operano attraverso l'uso di reti. Questo termine è frutto degli studi di Roy Fielding (uno dei principali autori delle specifiche http) ed è stato espletato, nell'anno 2000, nel suo documento "[Architectural Styles and the Design of Network-based Software Architectures](#)". Qui di seguito se ne riporta un paragrafo <sup>\*\*\*\*\*</sup>:

"Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

Attraverso i prossimi paragrafi sono tratti da <http://yeahnowow.com/2007/rest/> e danno una descrizione, poco formale, ma abbastanza chiara del paradigma.

---

\*\*\*\*\* Il documento completo è disponibile all'indirizzo "[Architectural Styles and the Design of Network-based Software Architectures](#)".

### *REST: Mettere al centro le risorse*

REST si basa sul concetto di risorsa, che diventa l'elemento principale in un network di servizi. Ad esempio si può pensare a quando avviamo un browser con l'intenzione di visitare una pagina Internet. Ad esempio tramite l'url <http://www.apple.com> si richiede la pagina descritta da questo indirizzo, la richiesta arriva fino al server che risponde fornendo il contenuto della pagina al browser.

Fino a qui abbiamo assunto che per visualizzare una pagina internet sia necessario inserire l'indirizzo di quella pagina, e che quindi a ogni pagina internet corrisponda uno e un solo indirizzo internet. La risorsa pagina internet è quindi identificabile univocamente dal suo indirizzo internet.

Per comunicare fra di loro il nostro browser e il server che custodisce la pagina internet che ci interessa hanno utilizzato per comunicare il protocollo HTTP. In particolare il browser ha inviato la richiesta HTTP "GET <http://www.apple.com>". GET è uno dei metodi previsti dal protocollo HTTP ed è utilizzato per il reperimento delle risorse web.

Sulla pagina visualizzata dal browser e corrispondente all'indirizzo <http://www.apple.com> è attualmente presente una immagine cliccabile che permette di visualizzare informazioni relativamente all'iPod Touch. Cliccando sulla immagine, nella barra degli indirizzi del browser viene visualizzato un nuovo indirizzo, <http://www.apple.com/ipodtouch/guidedtour/>, il browser invia una nuova richiesta HTTP al server "GET <http://www.apple.com/ipodtouch/guidedtour/>" e quindi procede a visualizzare la nuova pagina.

Oltre a digitare gli indirizzi internet e cliccare sui link, è possibile interagire con siti internet e applicazioni Web per mezzo dei form, ad esempio quando dobbiamo inserire i propri dati per registrarci ad un novo servizio. A differenza della semplice navigazione, quando si compila un form e lo si invia (di solito per mezzo di un apposito pulsante sistemato in fondo al form), la nostra operazione modifica lo stato del server, ad esempio aggiungendo il nostro nominativo alla lista delle persone iscritte ad un determinato servizio. La nostra operazione inserisce nuovi dati, e per indicare questa esigenza viene utilizzato non il metodo GET, ma il metodo POST.

Utilizzando i metodi GET e POST e l'indirizzo di una determinata risorsa web siamo quindi in grado da un parte di reperire una risorsa da un sistema remoto, nel nostro esempio una pagina internet, dall'altra di fornire una nuova risorsa ad un sistema remoto, nel nostro esempio il nominativo introdotto nel form.

Il protocollo HTTP prevede inoltre la possibilità di operare sulle risorse con due altri metodi: PUT per modificare e DELETE per cancellare una risorsa remota. Questi due metodi non sono disponibili per mezzo dei comuni browser, ma fanno parte del protocollo HTTP alla pari dei metodi GET e POST.

Per riuscire a visualizzare il significato dei differenti metodi previsti dal protocollo HTTP è possibile fare un'analogia con il "copia e incolla" sui programmi di composizione testo (Microsoft Word, Bocco Note); quando si seleziona una porzione di testo e si esegue il "copia", è come eseguire il metodo GET per una risorsa remota; l'"incolla" corrisponde al POST, l'"incolla" su una porzione di testo già esistente corrisponde al PUT e la cancellazione di una parte di testo al DELETE.

Una seconda analogia più precisa è con le operazioni CRUD (CREATE, READ, UPDATE, DELETE) del database (il progetto [CouchDB](#) ad esempio è un sistema di persistenza distribuito che espone le operazioni CRUD per mezzo di API REST).

A loro modo anche le operazioni CRUD e di "copia e incolla" aderiscono al paradigma REST.

8 Dicembre 2007





## INDICE DELLE FIGURE

Figura 1 esempio di mappa raster .....	7
Figura 3 esempio di mappa raster con sovrapposizione di dati vettoriali.....	8
Figura 4 interfaccia Google Maps .....	13
Figura 5 Localizzazione di un punto su una mappa tramite Google Maps. ....	14
Figura 6: architettura del servizio Google Maps.....	15
Figura 7 ambiente di Google Earth .....	16
Figura 8: formati di input/output Geoserver .....	18
Figura 9 I diversi layers di una stessa mappa per la gestione dello zoom.....	24
Figura 10 Caricamento di mappe da Google Maps .....	25
Figura 11 Localizzazione di un punto su Google Maps .....	26
Figura 12 Il pacchetto mapviewer.....	31
Figura 13 DefaultTileFactory.....	32
Figura 14 simulazione della geometria circolare del pianeta. ....	39
Figura 15 ProviderBuilder .....	40
Figura 16 WMSService .....	41
Figura 17 BBox.....	44
Figura 18: esempio di pannello con tre LegendComponent.....	51
Figura 19 Screen Mock-up .....	63

## BIBLIOGRAFIA

Martin C. Brown. Hacking  
“Google Maps and Google Earth”  
Wiley Publishing, Inc 2006.

Jeff de La Beaujardière  
“WEB Map Service Implementation Specification”.  
Open GIS Consortium Inc. 2002

Lisa Calderoni.  
“KML: una sintesi”.  
[www.freegis-italia.org](http://www.freegis-italia.org) – GI&GIS in Italia. 2007

Charlie Savage.  
“Google Maps Deconstructed”.  
Internet , 2006.

Ahmet Sayar  
“Architecture of the integration of Google and OGC WMS Web services”.  
Indiana University. 2005

Lorenzo Dal Col  
“Trasformazione coordinate su Google Maps”.  
Internet, 2007.

David Maguire  
“Ubiquitous Gis”  
Interner, Gis Matters, 2005

Sergio Maistrello  
“L’era della geografia demografica”.  
Monthly Vision, 2006.

Jeff McKenna  
“WMS Servers with MapServer”  
Internet, 2006

[http://code.google.com/apis/kml/documentation/kml\\_tags\\_21.html](http://code.google.com/apis/kml/documentation/kml_tags_21.html)  
“KML 2.1 Reference”  
Google, 2008.

[http://code.google.com/apis/kml/documentation/kml\\_tut.html](http://code.google.com/apis/kml/documentation/kml_tut.html)  
“KML Tutorial”  
Google, 2008.

<http://it.wikipedia.org/>

Vari autori  
Documentazione di librerie  
<http://swinglabs.org>

Joshua Marinacci,

“Building Maps into Your Swing Application with the JXMapView”,

<http://www.java.net/>, 2007.

Joshua Marinacci,

“Mapping Mashups with the JXMapView”,

<http://www.java.net/>, 2007.

Mark McLaren

“Unmarshalling KML with XStream and Polycythemmic Data Models”

<http://cse-mjmc.cse.bris.ac.uk/blog/2007/02/28/1172675719703.html>, 2007.

Andrea Aime,

“Geoserver, introduzione”,

Internet.

## INDICE

### 1

- 1. Obiettivi generali; 11
- 1. Rappresentazione di Mappe; 4

### 2

- 2. Scopo del progetto; 8
- 2. Sistemi correnti; 12
  - 2.1 GeoServer; 17; 10; 18; 23; 24; 26; 29; 30; 31; 32; 33; 34; 36
  - 2.1 Google Earth; 15
  - 2.1 Google Maps; 12

### 3

- 3. Sistema proposto; 18
- 3. Tempificazione delle attività; 9
  - 3.1 Overview; 19
  - 3.2 Le sorgenti di informazione; 19
  - 3.3 Requisiti funzionali; 19
  - 3.4 Requisiti non funzionali; 20
  - 3.6 Modello dei casi d'uso; 37
  - 3.7 Interfaccia utente; 41

### 4

- 4. Struttura del documento; 10

### A

- Appendice A; 1
- Appendice B; 1
- approfondimenti su google earth; 1

### C

- Capitolo 1; 1
- Capitolo 2; 11
- Capitolo 4; 22
- Capitolo 5; 8
- Capitolo 6; 37
- Capitolo 7; 42
- conclusioni; 42
- Conclusioni; 7

### D

- DefaultLegendPane;** 30
- DefaultScreenPolygon,**
- DefaultScreenLine,**
- DefaultScreenPoint;** 34
- DefaultTileFactory** (pacchetto mapviewer); 10

### G

- GeoMapView;** 24

## I

il paradigma REST; 1  
Indice; 8  
introduzione; 1  
Introduzione; 11  
INTRODUZIONE; 1

## J

**JXMapView** ( pacchetto  
org.jdesktop.swingx.mapviewer);  
23

## L

**LegendComponent**; 29

## M

**MapLegend**; 30  
**MapObject**; 31  
**MapPaintableElement**; 32

## O

**OverlayMap**; 26

## P

**ProviderBuilder**; 18

## R

requisiti; 1; 11  
REST: Metodi di accesso alle risorse;  
3  
REST: Mettere al centro le risorse; 2

## S

**ScreenShape**; 33  
studio di fattibilità; 22

## U

Un prototipo; 37  
una libreria per lo sviluppo; 8

