POLITECNICO DI MILANO
Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

MobiMash
Un framework per la generazione di mobile mashup

Relatore: Prof. **Maristella Matera**

Tesi di Laurea di:
Fabio Cosmai matricola 739947
Lorenzo Di Mulo matricola 735096

Anno Accademico 2010-2011

ai nostri genitori

INDICE

1 Introduzione

Il lavoro sviluppato in questa tesina parte dal concetto di End-user development in ambiente Web 2.0, cioè dall'idea di dare all'utente finale la possibilità di sviluppare da se applicazioni e quindi avvicinarsi sempre più al ruolo di sviluppatore. Una pratica sempre più diffusa nello scenario del Web 2.0 è lo sviluppo di mashup, applicazioni Web basate sulla composizione di contenuti e servizi offerti da terze parti in maniera del tutto aperta. Vi sono molti software che utilizzando processi di sviluppo leggeri permetto anche agli utenti meno esperti di assumere il ruolo di sviluppatore, tra questi vi sono Yahoo! Pipes, IntelMashMaker, IBMLotusMashups o JackBe Presto, e molti altri.

Da queste basi ormai consolidate in ambiente Web 2.0, si è pensato di portare l'EUD[1] in ambiente Mobile e quindi dare la possibilità agli utenti meno esperto e non solo, di sviluppare le proprie applicazioni per poi usufruirne sul dispositivo mobile. Come nell'ambito del Web 2.0, che come detto sopra, vede nei mashup una pratica sempre più crescente, anche in ambito mobile aumenta il concetto di "mobile mashup". Questi ultimi consistono nell'unire il concetto di mashup con i dispositivi mobile così svelando un mondo di nuovi mashup.

Analizzando le diverse applicazioni per smartphone, presenti sui diversi store dei principali sistemi operativi per mobile, si sono riscontrati due tipi di mobile mashup secondo la metodologia con i quali sono sviluppati:

- **Mobile Web Mashup**, quando sono sviluppati con tecnologie Web, cioè eseguiti all'interno del browser internet dello smartphone. Ogni applicazione sviluppata utilizzando le tecnologie Web funziona su tutte le piattaforme mobile perché il browser è l'unico strumento comune a tutti i sistemi operativi per cellulari.
- **Mobile Mashup all'interno di un'applicazione**, si tratta di mashup tradizionali all'interno dell'applicazione senza fare riferimento al browser ma recuperando i dati dalle diverse sorgenti tramite chiamate REST[2] o di altro tipo, e poi visualizzare i dati restituiti sfruttando tutto lo spazio disponibile offerto dal dispositivo mobile.

Al fine di consentire la creazione della seconda modalità di mobile mashup si è sviluppato un framework ad hoc chiamato **MobiMash**, il cui nome deriva dalla fusione delle parole Mobile e Mashup; termini chiave del nostro progetto.

MobiMash, al fine di garantire i risultati per i quali è stato sviluppato, deve soddisfare i requisiti fondamentali brevemente descritti di seguito:

- **EUD:** come già detto sopra, deve permettere all'utente meno esperto di sviluppare le proprie applicazioni.
- **User Friendly:** deve essere intuitivo riuscire a creare la propria applicazione e ritrovare in maniera semplice tutte le funzionalità che questa mette a disposizione, in modo da garantire la migliore esperienza utente.
- **Indipendente dal dominio applicativo:** come vedremo nel capitolo riguardante l'architettura del framework, grazie alla scelta di una struttura fissa per la creazione dell'interfaccia utente delle singole applicazioni, si riesce ad adattarle a qualsiasi contesto; ad esempio passando in maniera semplice da un'applicazione per concerti ad una per la visualizzazione dei film nelle sale cinematografiche.

[1] EUD: End-user development
[2] REST: Representational state transfer

Per ora tale framework è sviluppato per creare applicazione mobile solo per dispositivi con sistema operativo Android, un possibile sviluppo futuro sarà quello di renderlo multi piattaforma così da poter essere utilizzato su altri dispositivi equipaggiati con differenti sistemi operativi.

Il seguente documento raccoglie diverse fasi connesse al processo di sviluppo: analisi dei requisiti, progettazione, architettura, implementazione e validazione. Si è deciso, per motivi di chiarezza di impostare il documento in due parti ben distinte.

La prima, chiamata background, è relativa ad uno studio del progetto dal punto di vista dei concetti base, quali mashup, EUD, informazioni generali sul mondo degli smartphone e presentazione del sistema operativo Android. La seconda parte, chiamata MobiMash, mette in luce i passi che abbiamo compiuto nello sviluppo del progetto approfondendo le nostre scelte progettuali: si tratta quindi di uno sviluppo di fattibilità del framework.

Di seguito riporteremo un sunto dei capitoli che compongono le due parti del documento sopra descritte. Il background è formato dai seguenti capitoli, accompagnati da una breve descrizione:

- Capitolo 2 – Mobile Mashup: tale capitolo è diviso in diversi sottocapitoli che partono a descrivere il mondo degli smartphone e di come l'accesso a internet tramite questi sia sempre maggiore rispetto a quanto fatto con i desktop; per poi parlare dei mashup e come questi si sono introdotti nel mondo mobile, terminando con l'End-user development e come questo si affaccia al mondo degli smartphone.
- Capitolo 3 – Android: descrizione dettagliata del sistema operativo Android, mettendo in luce i concetti principali utili a comprendere il capitolo dell'implementazione esposto nella seconda parte del documento.

La seconda parte del documento, come sopra citato, descrive il framework MobiMash ed è costituita dai seguenti capitoli, anch'essi accompagnati da una descrizione:

- Capitolo 4 – Architettura Framework: argomenta tutte le scelte sull'architettura di MobiMash sia da un punto di vista progettuale sia fisico. Infatti, come vedremo, il framework è costituito da due applicazioni: una lato desktop utilizzata per la fase di impostazione e l'altra rappresentante l'applicazione mobile vera e propria.
- Capitolo 5 – Implementazione: si definisce l'implementazione del framework studiando le classi Java del progetto senza entrare nel merito del codice, ma fornendo solo schemi UML[3] per far comprendere come queste comunichino tra loro. Aspetto più fondamentale trattato in questo capitolo è la descrizione dettagliata dei file XML[4] con i quali avviene la creazione automatica dell'applicazione mobile. Ultimi due argomenti trattati sono le prestazioni e la validazione dell'applicazione ottenuta utilizzando MobiMash.

Nella parte finale del documento sono presentate le conclusioni, in cui si fanno le considerazioni finali sullo sviluppo e sui test del framework con possibili idee sugli sviluppi futuri.

[3] UML: Unified Modeling Language
[4] XML: eXtensible Markup Language

Parte 1: Background

2 Mobile Mashup

In questo capitolo si fornisce un'ampia panoramica del mondo degli smartphone e del maggior utilizzo di questi come porta di accesso primaria a internet, in più si definiscono i concetti chiavi grazie ai quali si è sviluppata l'idea e la successiva implementazione del framework MobiMash.

2.1 L'evoluzione del web e l'approccio al mobile

La rivista BusinessWeek ha denominato l'ultimo decennio (1999-2009) il "decennio perduto". Se è vero che l'ultimo decennio ha visto una turbolenza globale della ricchezza, redditi e quotazioni di borsa, esso coincide anche con la crescita di Internet che fornisce ai cittadini globali una sorta di "potere" del tutto inaspettato. Nonostante la turbolenza economica, Internet e in particolare Internet a banda larga, sono stati i fattori chiave dell'arricchimento sociale ed economico della società coinvolgendo i cittadini comuni di tutte le nazioni. La crescita di Internet a banda larga è stata spettacolare. Per esempio, gli Stati Uniti hanno iniziato con due milioni di connessioni a banda larga, fino ad arrivare alla fine del decennio (2009) con oltre 80 milioni di connessioni a banda larga. Si prevede che entro il 2014, tale numero crescerà fino ad arrivare a 96,4 milioni di connessioni nei soli Stati Uniti. A livello globale, si stima che ci saranno quasi 700 milioni di utenti a banda larga entro il 2013.

La diffusione della banda larga ha arricchito i consumatori grazie alla possibilità di creare contenuti, partecipare a social media e comunicare tra loro attraverso nuovi mezzi, come Instant Messaging (IM), video e VoIP[5]. Tuttavia, mentre la banda larga è stata il fattore abilitante per questi nuovi servizi, la tecnologia stessa (e la sua velocità) significano poco per i consumatori. Piuttosto, questi ultimi erano più preoccupati per l'impatto della connettività di rete, la quale poteva fare la differenza nella loro vita quotidiana. L'adozione del consumatore ha creato un effetto di auto rinforzo poiché più connettività ha portato a più servizi e maggiore innovazione, dall'altra parte la diffusione della banda larga ha coinciso anche con la crescita dei servizi guidata dall'economia.

Con un impatto diffuso sui consumatori e le imprese, i governi hanno riconosciuto la capacità della banda larga di fare la differenza rispetto al PIL, e sono diventati i catalizzatori della banda larga veloce nei diversi paesi. Secondo la Banca Mondiale, vi è una correlazione tra banda larga e il PIL. Con il 10% di aumento delle connessioni Internet ad alta velocità, la crescita economica aumenta dell'1,3%. Inoltre, la diffusione della banda larga è cresciuta oltre le reti di telefonia fissa fino ad arrivare a coinvolgere anche la banda larga mobile.

Il mobile è la nuova frontiera per l'innovazione e la crescita. Stiamo vedendo il coinvolgimento di organismi di regolamentazione, come la FCC (Federal Communications Commission) e l'Unione europea nello spazio della banda larga mobile. Le decisioni dei governi hanno il potenziale per portare a nuove strade, che permetteranno la creazione di ricchezza entro il prossimo decennio sia per i consumatori sia per le imprese.

[5] VoIP: Voice over IP

Che il telefono cellulare stia diventando lo strumento privilegiato per la navigazione su internet, è ormai appurato. Le previsioni vedono il 2011 come l'anno in cui gli smartphones supereranno il numero di PC acquistati e in cui gli accessi a Facebook da telefonino superano quelli da computer fisso.

In questi giorni sono usciti altri risultati che prevedono quale sistema operativo mobile avrà la supremazia. La situazione attuale in Italia vede uno share del 49% per iOS (Apple), 13% Android (Google), 1,45% per Blackberry OS (RIM) e 0,78% per Windows Phone (Microsoft). Esiste ancora una grossa quantità (26%) di utenti di SymbianOS (Nokia) in Italia, ma questi dati stanno per cambiare drammaticamente. Infatti, qualche settimana fa è stata annunciata la fusione di Nokia con Microsoft che porterà non poco scompiglio nel mondo del mobile. Una nuova ricerca dell'agenzia *Gartner* ha previsto uno scenario completamente diverso per il 2015 a livello mondiale: Android sarà il sistema operativo più diffuso con uno share del 49,2% e, dato ancora più clamoroso, Windows Phone supererà iOS portando in casa Microsoft una piccola soddisfazione nella "guerra" eterna con Apple. Sarà interessante vedere come questo cambiamento si rifletterà in Italia, ma sicuramente questa sana competizione fa bene sia all'industria sia all'utente finale.

Che cosa significa questo per la comunicazione?

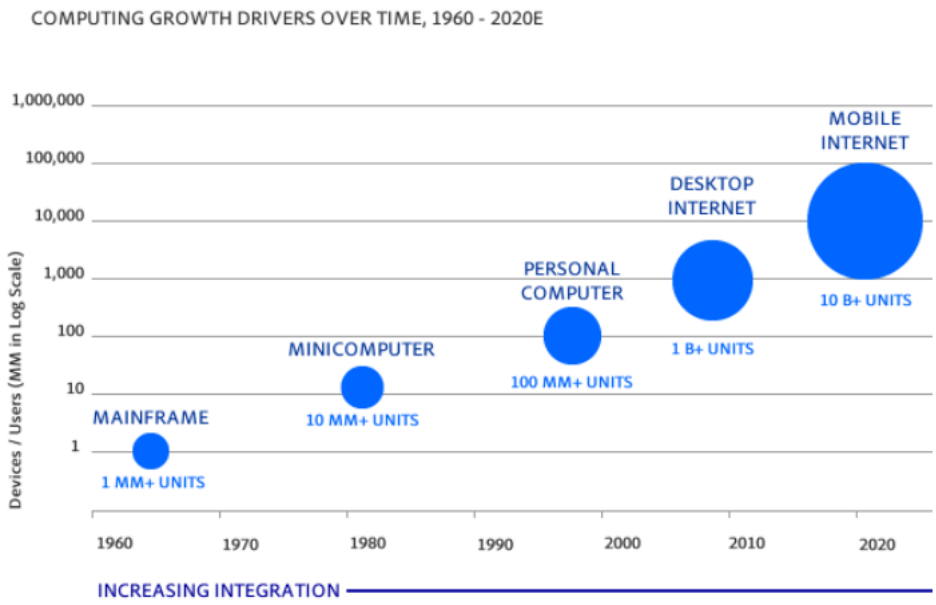
Già da qualche anno, quando si progetta qualsiasi tipo di sito web, non si può fare a meno di tenere in considerazione il fatto che l'utente può accedere al sito anche da telefonino. Bisogna tenere in considerazione che, nella maggior parte dei casi, chi naviga un sito da mobile ha bisogno di trovare informazioni in modo veloce e senza troppe distrazioni. Per questo motivo la versione mobile è una necessità che deve essere progettata con attenzione e cura nei contenuti e navigazione. Il cambiamento dello share previsto per i prossimi anni nel mondo del mobile non influirà troppo sulle tecnologie utilizzate per creare la versione mobile di un sito perché tutti i browser dei telefonini supportano Html e Javascript (linguaggi di programmazione utilizzati per siti web). Piuttosto sarà interessante tenere d'occhio la situazione per capire come cambierà l'utilizzo delle applicazioni dei telefonini.

La grande differenza tra le varie tipologie di sistema operativo mobile, infatti, sta proprio nella creazione di applicazioni dedicate. Le applicazioni sono un grande strumento per creare un valore aggiunto nella comunicazione di un'azienda. Quando queste applicazioni sono progettate come parte di una comunicazione strategica aziendale possono veramente fare la differenza. Queste applicazioni sono però proprietarie a un determinato sistema operativo, quindi sarà importante capire esattamente qual è il target group che si sta cercando di colpire e qual è il loro strumento di comunicazione preferito.

L'industria mobile è in continua evoluzione quindi diventa difficile fare previsioni a lungo termine. L'unico modo per non sbagliare la pianificazione della comunicazione è di comprendere esattamente chi è il proprio target group di riferimento in modo da sviluppare una comunicazione efficace e coerente con la situazione attuale dell'industria mobile e, più in generale, delle tecnologie utilizzate per fare comunicazione.

Oggi, ci sono più computer portatili che ogni altra forma di dispositivo di elaborazione. Come detto sopra, Internet su mobile cresce più veloce di quanto non aveva fatto sui desktop, ciò a causa della convergenza di cinque tendenze:

- Connessione internet 3G
- Social networking
- Video
- VoIP
- Dispositivi mobili di fascia alta



0Figura 2.1: numero dei punti di accesso a internet da diversi dispositivi e la loro variazione nel tempo.

Com'è facile notare dal diagramma di sopra si prevede che i dispositivi mobili diventeranno la "porta" principale di accesso a internet entro i prossimi cinque anni e il loro numero sarà oltre dieci volte quello dei dispositivi fissi che accederanno alla rete. Nonostante i vincoli hardware dei quali i dispositivi mobili soffrono, i progressi in termini di web browser introdotti dalle principali aziende del settore, quali ad esempio Apple, hanno dimostrato di essere user-friendly e allo stesso tempo raffinati nel modo di navigare attraverso le pagine web.

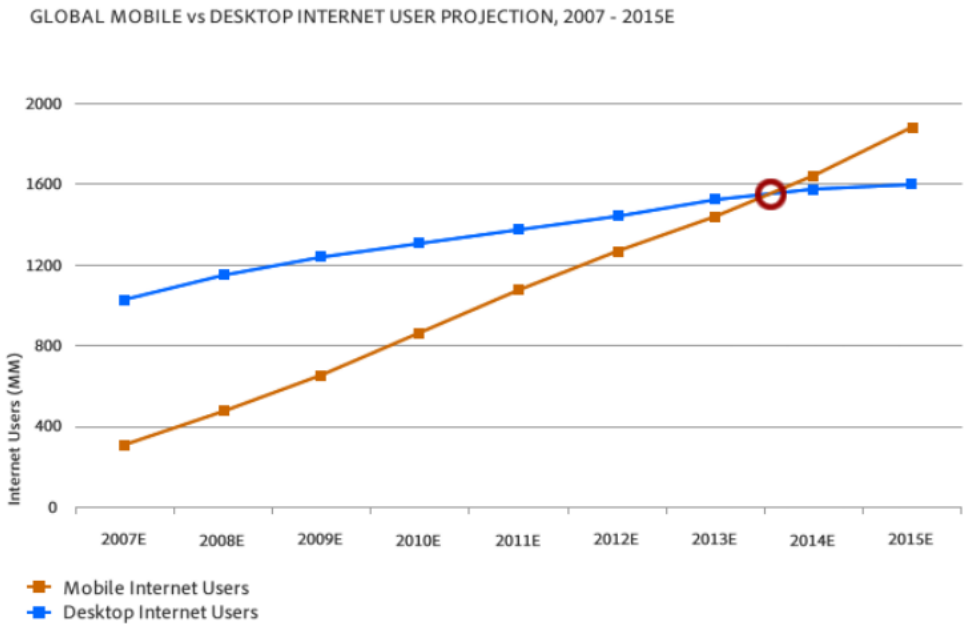


Figura 2.2: rette indicanti l'andamento del numero di utenti che accedono a internet da desktop e da mobile.

Dal grafico sopra riportato si nota come tra il 2013 e il 2014 il numero di utenti che avrà accesso a internet tramite mobile eguaglierà il numero di quelli che vi accederanno da desktop.

2.4 Mashup

Prima di diventare popolari in Internet, mashup era un termine usato nell'industria musicale che stava a indicare la miscela di due o più tracce al fine di creare una nuova canzone. La tecnica è stata chiamata anche registrazione multi traccia e ri-registrazione, dove il famoso gruppo dei Beatles ha fatto notevoli progressi su tale campo.

Nei giorni nostri, nello sviluppo web, un mashup è una pagina web che unisce le risorse da due o più fonti creando un nuovo servizio.

Ora saranno illustrati due esempi di mashup per dare l'idea di come questi si presentano e come interagiscono con l'utente.

Il primo presentato è **LivePlasma**, un grande database di film e di musica tutto da interrogare. Basta inserire nome e cognome di un attore, di un regista, oppure il titolo di un film per ottenere una mappa grafica che comprende tutte le interconnessioni possibili tra film. Cliccando sulle immagini di DVD e CD, si è automaticamente indirizzati su Amazon alle schede corrispondenti.

Figura 2.3: screenshot mashup LivePlasma

Il funzionamento è molto semplice, s'inserisce nel campo di ricerca ciò che si vuole visualizzare (ad esempio attore, regista o film) e in qualche secondo è generata una mappa. Tale mappa comprende al centro l'oggetto della ricerca e attorno tutte le connessioni, quali film con lo stesso regista, con stesso attore o con tematiche affini. Al passaggio del mouse sulle varie celle comparirà un cuore, cliccandoci sopra si aggiungono i titoli alla "my movies map", in altre parole una mappa che comprende tutti i film preferiti. Dopo aver aggiunto un po' di preferiti, cliccando su un collegamento ipertestuale sarà possibile vedere le interconnessioni fra i film selezionati e in più, a titolo di suggerimento, saranno aggiunti altri film affini.

Figura 2.4: screenshot mashup DUIMap.org

Il secondo è **DUIMap.org**, un mashup nel quale sono mostrati su una mappa le zone più soggette a incidenti stradali causati dall'assunzione di alcool.

Ogni icona sulla mappa rappresenta la posizione di un incidente in cui è stato coinvolto un veicolo a motore e il colore dell'icona indica il numero d'incidenti verificatosi entro un miglio di distanza dal precedente.

Cliccando su ogni icona è possibile visualizzare altre informazioni riguardanti il sinistro come ad esempio data e ora dell'accaduto.

Oltre a questi vi sono più di 5000 mashup che possono essere trovati nella sezione corretta del sito ProgrammableWeb, in cui è anche possibile individuare più di 2000 API[6] web che rappresentano i mattoni principali per la creazione di mashup. Ci sono molti tipi di mashup, quali mashup commerciali, mashup di consumo e mashup di dati; tra questi i più comuni sono quelli di consumo destinati al pubblico generale.

- **Mashup commerciali**, generalmente definiscono applicazioni che combinano le proprie risorse e dati con altri servizi web esterni. Essi concentrano i dati in una singola presentazione e permettono azioni collaborative tra le imprese e gli sviluppatori. Questo metodo funziona bene per un progetto di sviluppo agile, che richiede la collaborazione tra gli sviluppatori e clienti per la definizione e l'implementazione dei requisiti di business. I Mashup aziendali sono sicuri e dal punto di vista visivo assomigliano a ricche applicazioni web che espongono le informazioni fruibili da diverse fonti informative sia interne sia esterne.
- **Mashup di consumo,** combinano diversi tipi di dati provenienti da molteplici fonti pubbliche nel browser, e li organizzano attraverso una semplice interfaccia utente; ad esempio Wikipediavision combina Google Map e un'API Wikipedia.
- **Mashup di dati**, opposti ai mashup di consumo, combinano tipi simili di media e informazioni provenienti da più fonti in un'unica rappresentazione. La combinazione di tutte queste risorse creare un nuovo e distinto servizio web, che non era originariamente fornito da nessuna fonte.

Da un punto di vista commerciale, i mashup permettono di utilizzare diversi servizi e rompere i processi in parti più piccole accelerando notevolmente la velocità con cui i nuovi servizi di business sono distribuiti. Secondo Peenikal, questo rappresenterà un sostanziale vantaggio competitivo e le organizzazioni che non sono pioniere nell'utilizzo dei mashup inevitabilmente non sono competitive. Inoltre, il riutilizzo di servizi e dati esistenti riduce l'investimento iniziale di capitale necessario per qualsiasi nuovo progetto.

I mashup non richiedono ampie conoscenze informatiche e possono essere sviluppati in tempi relativamente brevi, migliorando il time-to-market e riducendo i costi di sviluppo delle applicazioni. Infine, i mashup sono più personalizzati per gli utenti finali grazie al fatto di richiedere meno risorse e di stimolare l'innovazione; ampiamente riconosciuti come antidoto all'attuale crisi economica globale in cui ci trova.

I componenti usati nello sviluppo di mashup sono quindi di tre tipi:

- Servizi di dati come i feed RSS (Really Simple Syndication) o Atom, contenuti formattati in JSON (JavaScript Object Notation) o XML oppure semplici file di testo. Per esempio, quasi tutti i giornali pubblicano ormai i titoli delle loro notizie tramite feed RSS che possono essere letti da un cosiddetto RSS reader e permettono all'utente di saltare facilmente il dettaglio delle varie notizie.
- Servizi Web o API accessibili tramite il Web come i servizi SOAP (Simple Object Access Protocol) o REST (REpresentational State Transfer).
- Questi servizi tipicamente non forniscono semplici dati, ma permettono il riuso di logica applicativa come, per esempio, il calcolo del nome di una città a partire dalle sue coordinate GPS.
- Componenti UI (cioè dotati di interfaccia utente) come pezzi di codice HTML o interfacce programmabili in JavaScript (per esempio, i cosiddetti widget – http://www.w3.org/TR/widgets-apis/). Il tipico esempio di componente UI è Google Maps, che fornisce non solo dati in forma di carte geografiche ma anche un'interfaccia utente facilmente integrabile in una pagina Web che permette all'utente di navigare le mappe. Comunque, anche l'estrazione di contenuti da pagine Web tradizionali è ancora una prassi molto diffusa, specialmente in assenza di servizi equivalenti già disponibili e pronti per l'uso.

[6] API: Application Programming Interface

Anche se i mashup non richiedono particolari competenze informatiche, a causa della loro eterogeneità di componenti, e quindi di tecnologie e di logiche applicative diverse da componente a componente, il loro sviluppo è tutt'altro che facile e spesso, ancora una prerogativa di sviluppatori esperti. Sono loro che conoscono i linguaggi di programmazione, i protocolli di comunicazione, e gli standard per implementare la logica d'integrazione necessaria per mettere in comunicazione i vari componenti e l'impaginazione grafica dei componenti all'interno del layout del mashup. La situazione è ancora più complessa qualora i componenti necessari non fossero già disponibili online. In questo caso è lo sviluppatore che deve prima implementare il componente per poi poterlo comporre. Nella maggior parte delle volte questo vuol dire estrarre dati da una o più pagine Web, una pratica che tipicamente richiede l'uso di espressioni regolari complesse e sforzi significativi di programmazione.

Per alleviare tali sforzi, di recente sono emersi strumenti online, che sono in grado di assistere lo sviluppatore nell'estrazione di dati da pagine Web tramite interfacce grafiche e interattive e che permettono di selezionare i contenuti desiderati e di ottenere automaticamente un componente riusabile. Esempi di questi strumenti sono Dapper o Openkapow, entrambi accessibili online e gratuiti.

Comunque, la visione della comunità scientifica e dell'industria del software è di mettere a disposizione dell'utente Web medio degli strumenti di sviluppo e ambienti di esecuzione che siano semplici e intuitivi (per esempio basati su linguaggi di composizione grafici e paradigmi *drag-and-drop*) e che permettano all'utente stesso di sviluppare applicazioni personali da componenti esistenti in maniera assistita e senza avere bisogno dell'aiuto da parte di programmatori esperti.

Strumenti come Yahoo! Pipes, IntelMashMaker, IBMLotusMashups o JackBe Presto rappresentano un primo passo in quella direzione, ma ancora non possiamo parlare di utenti veramente autosufficienti. Spesso tali strumenti forniscono semplificazioni tecnologiche rilevanti, ma quello che manca all'utente Web medio sono le nozioni base dello sviluppo di software. In questa situazione, il design di meccanismi che assistano l'utente tramite raccomandazioni di sviluppo interattive è una delle sfide di ricerca del futuro.

Vantaggi	Svantaggi
Riuso dei servizi e dei dati esistenti.	Affidabilità e qualità dei servizi.
Non richiedono ampie conoscenze informatiche.	Integrità del contenuto non garantita.
Sviluppo veloce.	Problemi di scalabilità.
Ridotti costi di sviluppo.	La maggior parte delle fonti di dati non sono fatti come un servizio.
Miglior adattamento grazie al numero ridotto di risorse.	Non c'è uno standard ed è difficile l'implementazione di meccanismi di sicurezza.

Tabella 2.5: riassunto dei vantaggi e svantaggi nella creazione e utilizzo di mashup.

2.5 End-user development in ambito mashup

End-user development (EUD) è argomento di ricerca in campo informatico e dell'interazione uomo macchina, e descrive le attività o le tecniche che permettono agli utenti finali di creare programmi. Gli sviluppatori possono comunque usare strumenti EUD per creare o modificare software utilizzando oggetti complessi senza conoscenze significative di un linguaggio di programmazione. I primi tentativi di end-user development si sono focalizzati sull'aggiunta di semplici linguaggi di programmazione di scripting per poi estendere e adattare delle applicazioni esistenti, come ad esempio programmi per ufficio.

Cosa spinge gli utenti a sviluppare le proprie applicazioni? Esiste un fattore specifico che guida il fenomeno dei mashup: *la possibilità, offerta agli utenti, di innovare*, cioè il desiderio e la capacità degli utenti di sviluppare "prodotti" propri e di realizzare le proprie idee, esprimendo così le proprie capacità personali.

Per comprendere meglio questo fattore trainante, consideriamo il fenomeno dilagante di Twitter, un servizio di microblogging attraverso cui amici e collaboratori possono rimanere in contatto e scambiarsi opinioni. Circa l'80% del traffico di Twitter parte attraverso la sua API pubblica; ciò vuol dire che il traffico è generato principalmente da applicazioni web che al loro interno integrano tale servizio. I fondatori di Twitter considerano l'API pubblica la loro più importante scelta di innovazione: "*Ci ha permesso in primo luogo di mantenere il servizio semplice e di creare uno strumento di facile utilizzo tramite cui gli sviluppatori possono costruire applicazioni nuove sulla base della nostra piattaforma e venir fuori con idee (come per esempioTwitterific) che sono decisamente migliori, grazie all'innovazione creata dagli utenti, delle nostre*". È così che Twitter è diventata un'infrastruttura per applicazioni Web create dagli utenti del Web, secondo un paradigma che l'economista Thomas Huges ha definito *invenzione conservativa*: l'innovazione generata da terze parti contribuisce alla crescita di un sistema, mentre il sistema stesso continua a rimanere sotto il controllo del suo produttore originale. In un ciclo tradizionale di progettazione-sviluppo-valutazione, il riscontro del cliente che commissiona l'applicazione diventa disponibile ai progettisti solo dopo il rilascio di un prototipo, quando ogni modifica può comportare costi considerevoli.

In un approccio basato sull'innovazione guidata dagli utenti, un'azienda offre un *pacchetto di innovazione* attraverso cui i clienti stessi possono costruire i loro prodotti. Questo pacchetto fornisce un'interfaccia limitata sulle potenzialità della piattaforma dell'azienda, così da assicurare che i nuovi prodotti siano costruiti in modo corretto e appropriato (per esempio senza violazioni d'uso).

L'idea alla base è quindi che la sperimentazione iterativa, necessaria per sviluppare un nuovo prodotto, può essere portata avanti interamente dall'utente stesso del prodotto. Diversi utenti possono lavorare in parallelo alla soluzione di un problema, ognuno concentrandosi su una propria versione. Essi possono creare soluzioni ad hoc per i loro requisiti e possono velocemente verificarne la fattibilità tramite i loro esperimenti di sviluppo. Allo stesso tempo, l'azienda che fornisce il pacchetto di innovazione non deve sostenere il costo di produzioni senza successo. Inoltre, quando un esperimento produce valore nuovo e rilevante, l'azienda ha il diritto di integrare l'innovazione nel suo prodotto originario. Questo è quanto successo nel Web quando alcuni sviluppatori integrarono in un mashup il servizio di Flickr con un servizio di mappe: Flickr ha successivamente incorporato una funzionalità di mappa sia nella sua piattaforma Web, sia nel suo servizio pubblico. Anche Google monitora costantemente l'uso delle sue API pubbliche (come per esempio GoogleMaps e GoogleSearch) per raffinarle ed "appropriarsi" degli usi innovativi migliori.

L'apertura dei servizi affinché terze parti possano utilizzarli nei propri mashup è quindi una scelta strategica, che rivoluziona il modello di business che per anni ha caratterizzato il Web e le sue applicazioni. Piuttosto che continuare a essere ricettori passivi d'innovazione, gli utenti del Web diventano attori attivi nella creazione dell'innovazione.

Il controllo sempre più crescente sulla proprietà intellettuale della soluzione creata fornisce un incentivo per gli utenti affinché essi si assumano le responsabilità dello sviluppo del software. Questo significa anche che il valore creato dall'innovazione è ora condiviso con gli utenti.

Il valore globale è con buona probabilità più grande, così che la condivisione con gli utenti rimanga comunque vantaggiosa. In tale contesto, l'innovazione del software è divenuta sempre più sistemica. Un'azienda dipende sempre più da fornitori esterni per la produzione di servizi che estendono un prodotto base. I fornitori esterni non sono sotto il controllo diretto dell'azienda. Quindi, per contare su di essi, l'azienda deve dimostrare un impegno credibile nell'innovazione sistemica.

I fornitori di servizi aperti, per esempio, cercano di attrarre fornitori esterni offrendo libero accesso ai loro dati e riducendo il controllo sulla proprietà dei dati e dei formati di scambio usati dalle API. I progressi dei mashup sono quindi il risultato logico di una tendenza generale a incrementare la complessità dei sistemi e allo stesso tempo la loro specificità. La complessità è legata al numero di tecnologie e di componenti incorporati in un prodotto. La specificità si riferisce invece al fatto che ogni azienda coinvolta nello sviluppo ha la possibilità di concentrarsi sulle competenze in cui eccelle, aumentando così l'efficienza del processo di produzione.

Il modo in cui i mashup sono sviluppati dipende dal tipo di mashup. In questo momento, i mashup destinati ai "consumatori" (per esempio, tutti i numerosi mashup che integrano mappe) sono perlopiù il risultato di attività di programmazione da parte di sviluppatori esperti. I mashup "aziendali" evidenziano invece diversi scenari di sviluppo, che vedono anche il coinvolgimento di utenti meno esperti. Prenderemo quindi spunto da studi recenti sui mashup aziendali e cercheremo di evidenziare i contributi che diversi attori, con diversi livelli di esperienza tecnica, possono apportare al loro sviluppo.

La figura 2.6 illustra tre diversi scenari, che differiscono per l'eterogeneità dei servizi integrati nei mashup, la diversità dei bisogni degli utenti, e il grado di sofisticazione degli utenti e degli strumenti che supportano il loro lavoro:

A. I mashup possono essere creati da sviluppatori esperti (per esempio sviluppatori di un dipartimento IT o fornitori di servizio) in grado di rilasciare applicazioni in breve tempo. Gli utenti finali non sono coinvolti nella produzione dei mashup ma traggono in ogni caso benefici dai tempi di sviluppo ridotti.
B. Gli sviluppatori esperti creano servizi adatti a essere composti in un mashup e forniscono allo stesso tempo un ambiente ben delimitato (sandbox) dove gli utenti finali possono combinare i servizi. Il ruolo degli sviluppatori è quindi creare servizi, mentre i mashup sono costruiti dagli utenti, i quali sono maggiormente a conoscenza del dominio applicativo in cui i mashup saranno utilizzati.
C. Gli sviluppatori esperti rilasciano un ambiente che permette a ognuno di creare i propri mashup. Questo è analogo al modo in cui i fogli di calcolo sono usati oggigiorno nelle organizzazioni: gli utenti finali, per esempio gli analisti finanziari di un'azienda, creano i propri fogli di calcolo senza dover necessariamente coinvolgere un programmatore esperto. I mashup sono spesso creati per un singolo scopo e per un unico utente e sono legati a una situazione specifica. Queste applicazioni sono, infatti, note come *situational applications*.

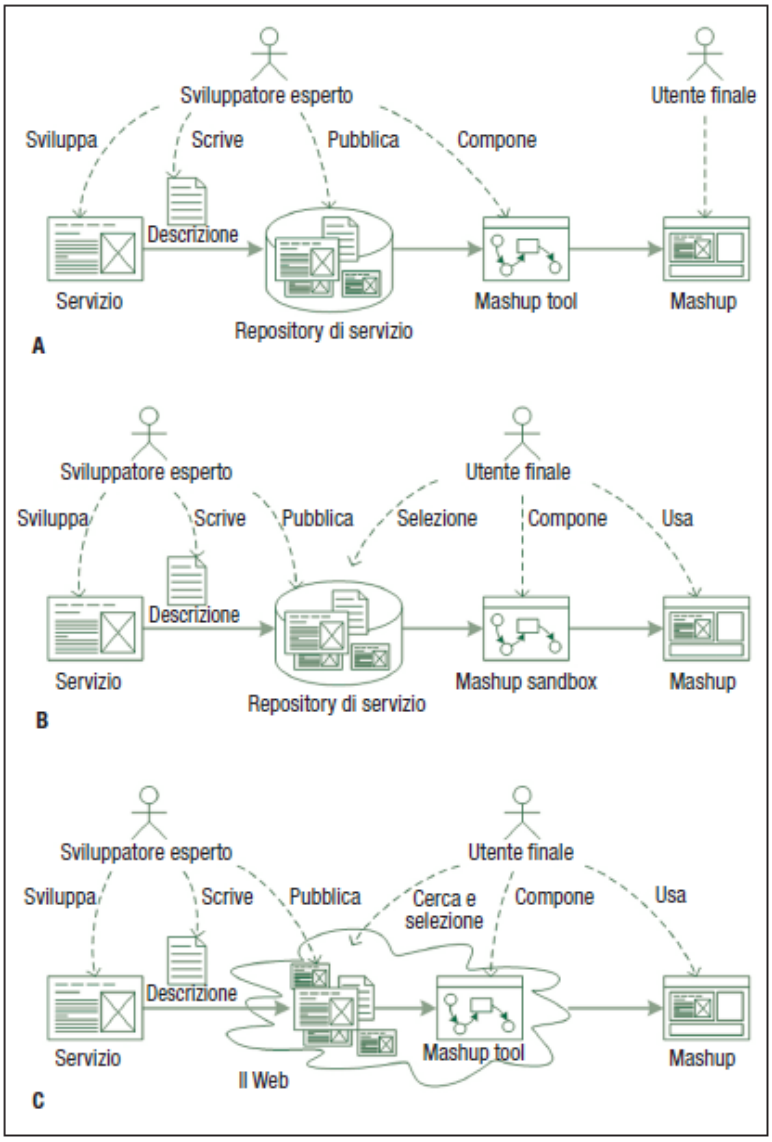


Figura 2.6: scenari che differiscono per l'eterogeneità dei servizi integrati nei mashup, la di versità dei bisogni degli utenti, e il grado di sofisticazione degli utenti e degli strumenti che supportano il loro lavoro.

Quando i mashup sono sviluppati in modo centralizzato (scenario A), lo sviluppo è affidato ai soli sviluppatori esperti del dipartimento IT. Date le risorse limitate di un dipartimento IT, potranno essere sviluppate solo le applicazioni più richieste. Tuttavia, gli sviluppatori esperti hanno l'esperienza e le capacità necessarie a integrare servizi largamente eterogenei a un basso livello di astrazione, e possono quindi essenzialmente produrre applicazioni di ogni tipo. Quando il dipartimento IT concentra le sue risorse sullo sviluppo di componenti e sul supporto necessario agli utenti per usare tali componenti (scenario B), esporrà certi tipi di dati e servizi in un formato facile da usare da utenti non sviluppatori. Rispetto ai programmatori esperti, questi utenti avranno una migliore conoscenza del dominio applicativo. Poiché gli sforzi per lo sviluppo dei mashup non sono totalmente a carico dello staff IT, è possibile soddisfare un maggior numero di richieste degli utenti.

Tuttavia, i mashup che possono essere costruiti sono esclusivamente limitati ai componenti forniti dallo staff IT: la composizione del mashup consisterà in generale nella parametrizzazione dei componenti, poiché gli utenti non hanno la capacità di eseguire un'integrazione più "profonda" e articolata. La produzione di uno strumento per lo sviluppo di mashup (scenario C) è sicuramente la soluzione più impegnativa da realizzare, ma allo stesso tempo la più proficua per i vari attori. Attraverso tale strumento, gli utenti possono integrare servizi e dati per creare i propri mashup. Rispetto ai mashup programmati "a mano", lo strumento limita le possibilità di integrazione al fine di garantire la correttezza della composizione, ma allo stesso tempo concede agli utenti la libertà di integrare qualsiasi tipo di componente disponibile nel Web. Tale strumento fa cioè in modo che gli utenti possano creare le proprie applicazioni di diversi tipi così da assecondare un'ampia diversità di requisiti.

Un'altra differenza tra i tre diversi scenari riguarda il grado di controllo sulla qualità dei mashup creati. Nello scenario A, lo sviluppo da parte dello staff IT è garanzia di qualità. Tuttavia, non tutti i mashup hanno requisiti stringenti in termini di sicurezza, prestazioni, o affidabilità, soprattutto se destinati a essere usati per uno scopo ben specifico, per il quale una soluzione complessa sviluppata da un dipartimento IT sarebbe troppo costosa. Nello scenario B, il dipartimento IT seleziona i componenti che possono essere integrati nei mashup degli utenti e fornisce inoltre una piattaforma per comporre ed eseguire i mashup.

Gli utenti sono quindi abilitati a creare mashup per soddisfare bisogni specifici che difficilmente potrebbero essere previsti e assecondati dal dipartimento IT. I mashup sviluppati dagli utenti nello scenario C possono in seguito essere utilizzati come prototipi per rendere più solide le applicazioni sviluppate dal dipartimento IT, per esempio nel caso in cui dovesse emergere il bisogno di rendere disponibili quei mashup a diversi utenti all'interno dell'azienda o a clienti esterni. Secondo von Hippel, i mashup rendono l'innovazione "democratica", permettendo agli utenti finali di soddisfare quei bisogni cui un dipartimento IT centralizzato, o più in generale un fornitore di servizio, non è sempre in grado di rispondere. Il loro uso riduce i tempi necessari a implementare una certa funzionalità. Uno dei possibili usi dei mashup è, infatti, la prototipazione di soluzioni a problemi di utenti specifici, che possono in seguito essere generalizzate per soddisfare i bisogni di una comunità più ampia. Lo sviluppo dei mashup è quindi simile allo sviluppo open source (che ha ispirato la metafora di vonHippel) secondo due diverse sfaccettature: coloro che contribuiscono allo sviluppo di un progetto open source sono anche utenti del software prodotto; i progetti open source forniscono un meccanismo attraverso il quale, chi contribuisce attivamente al progetto passa dall'essere un utente passivo al fornire utili commenti, richieste di funzionalità aggiuntive e in alcuni casi pezzi di codice. In modo simile, gli sviluppatori di mashup sono spesso anche gli utenti dei mashup (vedi scenari B e C nella Figura 1); non tutti gli utenti dei mashup rivestono necessariamente il ruolo di sviluppatori, ma possono comunque contribuire allo sviluppo fornendo commenti e richiedendo nuove funzionalità durante lo sviluppo dei prototipi.

I concetti sopra esposti, riguardanti l'EUD, continuano a valere anche in ambito mobile e sono alla base dello sviluppo del nostro framework che si colloca nello scenario B, poiché dà all'utente la possibilità di selezionare servizi pre-registrati al fine di utilizzarli per la creazione autonoma dell'applicazione mobile.

2.4 Mobile Mashup

I mobile mashup consistono nel prendere sorgenti di dati libere o a basso costo (quali ad esempio Twitter, segnalazioni meteo, conversioni metriche, eccetera) e/o servizi multipli (mappe, flickr, eccetera) al fine di costruire un'applicazione mobile, quindi si tratta di unire il concetto di mashup con i dispositivi mobili così svelando un mondo di nuovi mashup. Definiamo due tipi di mobile mashup secondo la metodologia con i quali sono sviluppati:

- **Mobile Web Mashup**, quando sono sviluppati con tecnologie Web, cioè sono eseguiti all'interno del browser internet del cellulare. Ogni applicazione sviluppata utilizzando le tecnologie Web funziona su tutte le piattaforme mobili perché il browser è l'unico strumento comune a tutti i sistemi operativi mobili.
- **Mobile Mashup all'interno di un'applicazione**, si tratta di mashup tradizionali all'interno dell'applicazione senza fare riferimento al browser ma recuperando i dati dalle diverse sorgenti tramite chiamate REST o di altro tipo e poi visualizzare i dati sfruttando tutto lo spazio disponibile che il dispositivo mette a disposizione.

Gli ingredienti che stanno alla base per lo sviluppo di buon mobile mashup sono:

- Velocità, deve essere in grado di compiere qualsiasi operazione sia in background sia in foreground in tempi brevi per offrire la migliore esperienza utente.
- Efficienza, deve essere in grado di svolgere le operazioni per le quali è stato creato.
- Grafica accattivante, l'aspetto è uno degli elementi principali della buona riuscita di un'applicazione in generale e lo è ancora di più nel caso di mashup.
- Intrattenimento, deve coinvolgere l'utente il più possibile lasciando in lui il desiderio di riprovarla.

2.5 Sistemi operativi dei dispositivi mobili

Un sistema operativo mobile può essere paragonato in linea di principio ai sistemi operativi come Windows, Mac OS o Linux ma orientato al controllo di un apparecchio mobile. Tuttavia, sono attualmente un po' più semplici e si specializzano maggiormente sulla connettività sia wireless sia a banda larga, sui formati multimediali e diversi metodi di input. Tipici esempi di dispositivi che usano sistemi operativi mobile sono gli smartphone, personal digital assistant PDA, tablet PC, tra questi possono essere anche inclusi i sistemi embedded o altri dispositivi mobile e dispositivi wireless.

2.5.1 La storia

La crescente importanza dei dispositivi mobili ha innescato una forte concorrenza tra i giganti del software come Google, Microsoft e Apple, così come alcuni leader del settore mobile come Nokia, Research In Motion (RIM) e Palm, nel tentativo di accaparrarsi il monopolio sul mercato.

Con il rilascio dell'iPhone nel 2007, Apple ha causato un significativo scompiglio nel settore mobile iniziando così una nuova era di sistemi operativi smartphone che si concentrano sulla user experience e si basano sull'interazione tramite touch.

Nel novembre del 2007 Google ha costituito la Open Handset Alliance con 79 società, tra cui quelle di hardware, software e telecomunicazioni (ad esempio ASUS, HTC, Intel, Motorola, Qualcomm, T-Mobile, e NVIDIA), per farsi strada nel mercato degli smartphone con il suo nuovo sistema operativo Android. Anche se il suo lancio è stato da subito apprezzato sia da parte dei media sia dal pubblico, il rilascio di Android ha creato una rottura tra Apple e Google, portando alle dimissioni del CEO di Google, Eric Schmidt, dal consiglio di amministrazione di Apple.

Dal lancio di entrambi i sistemi operativi Apple iOS e Google Android, il mercato degli smartphone è esploso in popolarità, e nel maggio del 2010 ha raggiunto più del 17.3% di tutti i cellulari venduti. Questo ha portato ad esporre un piano pubblicitario su larga scala coinvolgente i principali produttori di telecomunicazioni formando delle partnership. Dal gennaio 2011, Google detiene il 33.3% del mercato degli smartphone in tutto il mondo, dimostrando una crescita incredibile per Android che deteneva solo il 4.7% un anno prima. Nokia, Apple, RIM e Microsoft hanno rispettivamente il 31%, 16.2%, 14.6% e 3.1% del mercato degli smartphone.

2.5.2 L'evoluzione sul mercato

Le piattaforme mobile sono in fase nascente, le proiezioni per quanto riguarda la crescita nel mercato sono difficili da fare in questo momento. Tuttavia, una chiara tendenza in aumento riguarda la crescita dei sistemi operativi mobili, che sono sviluppati per dispositivi smart, piuttosto che per telefoni con funzionalità di base (chiamate e SMS).

Dal febbraio 2011, Nokia ha annunciato una partnership con Microsoft che chiude l'era del sistema Symbian OS, il più popolare sistema operativo per telefoni con funzionalità basilari, entro la fine del 2011 per fare spazio a Windows Phone. E' interessante notare che Intel sta focalizzando l'attenzione su dispositivi portatili oltre che sugli smartphone, si tratta di Mobile Internet Devices (MID) e Ultra-Mobile PC (UMPC). Nel frattempo Palm ha abbandonato il piano di sviluppo per Foleo, che doveva essere un dispositivo legato al mondo degli smartphone.

2.5.3 Navigazione GPS su mobile

Canalys, una società indipendente di analisi tecnologica, ha stimato che nel 2009 gli smartphone con GPS integrato erano 163 milioni di unità in tutto il mondo, di cui Nokia ha rappresentato più della metà di tali dispositivi fabbricando circa 83 milioni di dispositivi con GPS (circa il 51%). Il 22 gennaio 2010, Nokia ha rilasciato una versione gratuita di Ovi Maps, nel tentativo di aumentare il numero di clienti.

2.5.4 Elenco dei sistemi operativi per smartphone

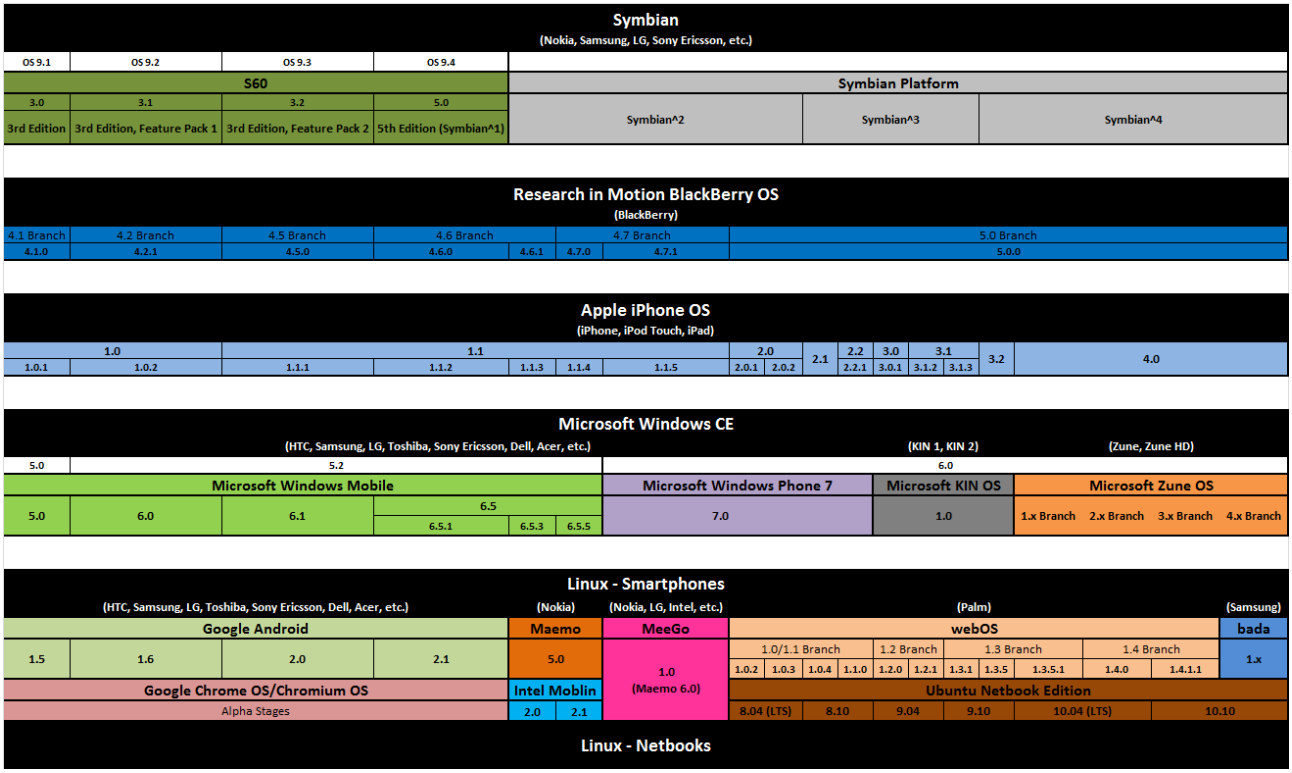


Figura 2.7: mostra la maggior parte degli attuali sistemi operativi mobili per smartphone, PDA e netbook nel 2010.

I sistemi operativi che si possono trovare su smartphone sono Nokia Symbian, Android di Google, Apple iOS, RIM BlackBerry OS, Microsoft Windows Mobile, Linux, HP webOS, Bada Samsung, Nokia Maemo e MeeGo e molti altri. Android, Bada, WebOS e Maemo sono costruite basandosi su Linux, e iOS è derivato dai sistemi operativi BSD e NeXTSTEP, che sono tutti derivati da Unix.
I più comuni sistemi operativi (OS) utilizzati negli smartphone dal 3° trimestre 2010 basandosi sui rapporti di vendita sono:

- Il sistema operativo open source **Symbian** (36.6% delle vendite Q3 2010).
  Symbian ha la maggiore quota di vendita sul mercato mondiale, è stato usato da molti produttori di telefoni tra cui BenQ, Fujitsu, Nokia, Samsung, Sharp e Sony Ericsson. Attuali dispositivi basati su Symbian sono stati fatti da Fujitsu, Nokia, Samsung, Sharp e Sony Ericsson. Già da prima del 2009 Symbian supportava interfacce utente multiple, come per UIQ Technologies, S60 da Nokia e MOAP da NTT DOCOMO. Come parte del nucleo della piattaforma Symbian nel 2009 ha unito queste tre interfacce utente in una singola piattaforma che ora è disponibile a tutti e completamente open source. Recentemente però la quota di mercato per Symbian è scesa da oltre il 50% a poco più del 40% su scala mondiale dal 2009 al 2010. Il sistema operativo fornisce un nucleo di interfacce di programmazione comuni a tutti i telefoni che adottano Symbian OS.

Alcune delle sue caratteristiche principali sono:

- Una ricca suite di applicazioni di base con programmi per la gestione di contatti, appuntamenti, messaggistica, browsing, utilità e controllo del sistema, API per la gestione dei dati, testo, tastiera e grafica.
- Gestione dei messaggi multimediali MMS, EMS, SMS
- Posta elettronica sui protocolli POP3, IMAP4, SMTP e MHTML
- Gestione di file multimediali con supporto per la registrazione video, playback, streaming e conversione delle immagini.
- Grafica con accesso diretto allo schermo ed alla tastiera, API per l'accelerazione grafica.
- Protocolli di comunicazione che includono gli standard TCP/IP (IPv4/IPv6), WAP, IrDA, USB Bluetooth.
- Supporto internazionale conforme allo standard Unicode 3.0
- Sincronizzazione dati con l'uso di OTA, SyncML, Bluetooth, USB, IrDA, cavo seriale.
- Gestione della sicurezza con possibilità di criptatura completa e gestione dei certificati, protocolli sicuri come HTTPS, WTLS, SSL, TLS e WIM framework.
- Linguaggi e tools di sviluppo che includono C++, Java (J2ME) MIPD 1.0 e successivi, PersonalJava con JavaPhone e WAP.
- Input utente con supporto completo della tastiera, voce e riconoscimento della scrittura.

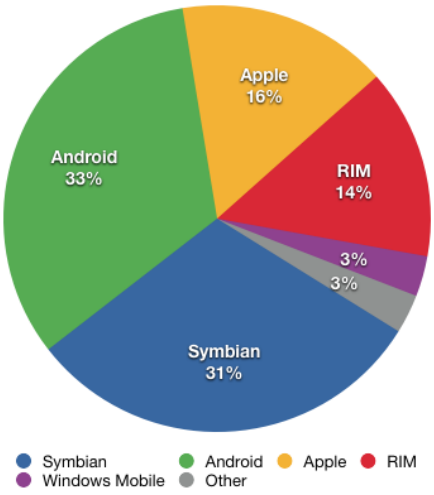


Figura 2.8: distribuzione dei sistemi operativi sul mercato.

- **Android** di Google Inc. con licenza open source Apache (25.5% delle vendite Q3 2010).
  Android è stato sviluppato da una piccola startup poi acquistata da Google Inc. ed è proprio quest'ultima che tiene in costante aggiornamento il software. Android è una piattaforma open source che deriva da Linux, Google insieme con importarti sviluppatori di hardware e software come Intel, HTC, ARM, Samsung, Motorola ed eBay ha creato la Open Handset Alliance. Uscito il 5 novembre 2007, il sistema operativo è già arrivato alla sua settima versione partendo da Android 1.0, 1.5, 1.6, 2.0, 2.1, 2.2, 2.3. Tutte le versioni hanno un soprannome ad esempi Cupcake(1.5) o Frozen Yogurt(2.2). La maggior parte dei principali fornitori di servizi mobili vorrebbe Android sui propri dispositivi. Da quando HTC Dream (T-Mobile G1) è stato introdotto, c'è stata un'esplosione del numero di dispositivi che portano Android OS. Dal secondo trimestre del 2009 al secondo trimestre del 2010, la quota di mercato per Android è salita dell'850% dall'1.8% al 17.2% in tutto il mondo.
  Di seguito sono elencate alcune delle funzionalità del sistema:

  - **Application framework** abilità funzionalità per il riuso e la sostituzione dei componenti.
  - **Dalvik virtual machine** il motore grazie al quale è possibile eseguire le applicazioni in Android.

- **Integrated browser** un browser per la navigazione internet basato sul motore open source WebKit.
- **Optimized graphics** la grafica 2D gestita con librerie personalizzate, mentre la grafica 3D si basa sulle specifiche standard OpenGL ES 1.0. (funzionalità opzionale legata all'acceleratore hardware del dispositivo)
- **SQLite** un motore di database relazionale integrato.
- **Media support** per la gestione dei più diffusi formati audio e video (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF) .
- **GSM Telephony** funzionalità per la gestione della telefonia mobile (dipendente dall'hardware del dispositivo).
- **Bluetooth, EDGE, 3G, and WiFi** gestione dei più comuni protocolli di comunicazione. (dipendenti dall'hardware del dispositivo).
- **Camera, GPS, compass, and accelerometer** gestione di dispositivi per la ripresa fotografica, video e di localizzazione. (dipendenti dall'hardware del dispositivo).
- **Rich development environment** strumenti per lo sviluppo di applicazioni personalizzate che includono emulatori di device, tools per il debugging, profilazione per il monitoraggio delle performance e della memoria utilizzata, plugin per Eclipse.

- **iOS** di Apple Inc.con licenza proprietaria (16.7% delle vendite Q3 2010).
iOS (fino a giugno 2010 **iPhone OS**) è il sistema operativo sviluppato da Apple per iPhone, iPod touch e iPad. Come Mac OS X è una derivazione di UNIX (famiglia BSD) e usa un microkernel XNU Mach basato su Darwin OS. iOS ha quattro livelli di astrazione: il Core OS layer, il Core Services layer, il Media layer e il Cocoa Touch layer. Il sistema operativo occupa meno di mezzo gigabyte della memoria interna del dispositivo. Il sistema operativo non aveva un nome ufficiale fino al rilascio della prima beta dell'iPhone SDK il 6 marzo 2008; prima di allora, il marketing Apple affermava che "iPhone usa OS X". Attualmente Apple detiene con iOS il 2,63% del mercato, risultando il terzo sistema operativo più diffuso al mondo dopo Microsoft Windows e Mac OS X ed il primo sistema operativo mobile. L'SDK è diviso nei seguenti set:

Figura 2.9: iPad Apple

  - Cocoa Touch che comprende: Multi-touch, supporto all'accelerometro, gerarchia di View, localizzazione (i18n) e supporto alla fotocamera.
  - Media che comprendono: OpenAL, Audio mixing e recording, Video playback, supporto a vari formati d'immagini, Quartz, centro delle animazioni e OpenGL ES.
  - Servizi principali tra cui vi sono: Networking, database SQLite incorporato, Geolocalizzazione e Threads.
  - OS X Kernel che comprende: TCP/IP, sockets, power management, file system e sicurezza.

  All'interno dell'SDK è contenuto l'iPhone Simulator, un programma usato per emulare il "look and feel" dell'iPhone nel desktop dello sviluppatore. Originariamente chiamato Aspen Simulator, è stato rinominato con la beta due dell'SDK. Da notare che l'iPhone Simulator non è un emulatore ed esegue codice generato per un target x86. L'SDK richiede un Mac Intel con Mac OS X Leopard. Altri sistemi operativi, inclusi Microsoft Windows e vecchie versioni di Mac OS X, non sono supportati.

- **RIM BlackBerry OS** con licenza proprietaria (14.8% delle vendite Q3 2010).
Questo sistema operativo si concentra sulla facilità d'uso ed è stato originariamente progettato per le imprese. Recentemente si è visto un aumento di applicazioni di terze parti e il sistema operativo è

stato migliorato per offrire pieno supporto multimediale. Attualmente Blackberry App World ha oltre 15000 applicazioni scaricabili.

- **Windows Phone** di Microsoft con licenza proprietaria(trascurabili vendite Q3 2010).
  Il 15 febbraio 2010, Microsoft ha presentato la nuova versione del sistema operativo,Windows Mobile 7. Il nuovo OS include un'interfaccia utente completamente nuova, inoltre comprende la piena integrazione dei servizi Microsoft come Windows Live, Zune, Xbox Live e Bing, ma integra anche servizi non Microsoft come Facebook e i vari servizi Google. La nuova piattaforma, non permette retro compatibilità con le applicazioni Windows Mobile e alcune caratteristiche della vecchia versione non saranno presenti fino al prossimo aggiornamento.

- **Linux** sistema operativo con licenza open source, GPL (2.1% delle vendite Q3 2010).
  Linux è una grande soluzione per il mercato Cinese, dove è utilizzato dalla Motorola mentre in Giappone è utilizzato da CoMoDo. Invece di essere una piattaforma a se stante, Linux è utilizzato come base per una serie di diverse piattaforme tra cui Android, LiMo, Maemo, Openmoko e Qt Extended, che sono per lo più incompatibili. PalmSource ora Access si sta muovendo verso un'interfaccia di esecuzione su Linux. Un'altra piattaforma basata su Linux sta per essere sviluppata da Motorola, NEC, NTT DoCoMo, Panasonic, Samsung e Vodafone.

Figura 2.10: smartphone HP che esegue webOS.

- **webOS** da HP (parzialmente open source).
  Palm webOS deriva da Palm sistema operativo di nuova generazione, ed è stato acquistato da HP. Diversi dispositivi saranno pronti per il 2011.

- **bada** da Samsung Eletronics (licenza proprietaria).
  Si tratta di un sistema operativo mobile sviluppato da Samsung Eletronics. Samsung sostiene che bada sarà presto il sostituto delle sue piattaforme telefoniche con funzionalità basilari. Il nome bada deriva dal termine coreano che significa oceano o mare. Il primo dispositivo sui cui sarà montato bada si chiamerà "Wave" ed il prodotto è stato presentato al pubblico durante il Mobile World Congress del 2010. Wave è un cellulare touchscreen che gira perfettamente con il nuovo sistema operativo. Oltre all'acquisto del cellulare Samsung ha anche rilasciato il suo app store chiamato Samsung Apps. Attualmente offre circa 3000 applicazioni. Samsung ha affermato che Bada non è una piattaforma per smartphone, ma si avvicina di più ad una piattaforma con un'architettura kernel configurabile. Anche se Samsung prevede di installare bada su molti suoi cellulari, la società ha ancora un'ampia gamma di prodotti Android.

- **MeeGo** di Nokia e Intel (licenza open source, GPL).
  In occasione del Mobile World Congress 2010 di Barcellona, Nokia e Intel hanno svelato "MeeGo" un nuovo sistema operativo mobile che unisce il meglio di Moblin e Maemo per creare un qualcosa personalizzabile dagli utenti e disponibile per tutti i dispositivi. Dal 2011, Nokia ha annunciato che non seguirà più Meego per concentrarsi sull'adottare Phone Windows 7 come sistema operativo mobile primario.

- **OS Brew** di Qualcomm.
  Brew OS è utilizzato da diversi produttori di cellulari, ma il più delle volte l'utente finale non lo sa perché spesso il telefono su cui è caricato Brew è privo di marchio Brew OS. Esso è eseguito in background e l'interfaccia è personalizzabile in base al produttore di telefonia. BrewOS è utilizzato da Sprint Nextel, MetroPCS, US Cellular e Verizon negli Stati Uniti e dalla rete Tre in gran parte d'Europa. Produttori come Huawei, INQ Mobile, Amoi Samsung Mobile. Solo pochi telefoni HTC utilizzano questa piattaforma, ad esempio, HTC Freestyle.

Anno	Symbian	Android	RIM	iOS	Microsoft	Altri OSs
2011	27.4%	36.0%	12.9%	16.8%	3.6%	3.3%
2010	37.6%	22.7%	16.0%	15.7%	4.2%	3.8%
2009	46.9%	3.9%	19.9%	14.4%	8.7%	6.1%
2008	52.4%	0.5%	16.6%	8.2%	11.8%	10.5%
2007	63.5%	N/A	9.6%	2.7%	12.0%	12.1%

Tabella 2.11: riassunto delle percentuali di diffusione dei sistemi operativi dal 2007 al 2010.

3 Android

In questo capitolo sarà presentato il sistema operativo Android descrivendone le caratteristiche principali e gli strumenti messi a disposizione del programmatore al fine di realizzare applicazioni accattivanti. La scelta di utilizzare Android come strumento di sviluppo per MobiMash è fondamentalmente dettata dalla sua natura open source.

3.1 Cos'è Android?

Android è un insieme di software per dispositivi mobili che include un sistema operativo, un middleware ed applicazioni di base. L'Android SDK (Software development kit) fornisce gli strumenti e le APIs necessarie per iniziare a sviluppare applicazioni per la piattaforma utilizzando il linguaggio di programmazione Java.

Android è stato sviluppato inizialmente da Google e in seguito da Open Handset Alliance (OHA). Questa è un'alleanza di aziende composta fra le altre da: Google, Intel, HTC, Motorola, Samsung, LG, Nvidia, Qualcomm e T-Mobile, che ha il fine di sviluppare applicazioni e hardware per smartphone.

Il sistema operativo è stato annunciato il 5 novembre 2007 e rilasciato il 12 dello stesso mese. La prima versione commerciale di un cellulare fornito del sistema operativo Android, è stata rilasciata il 22 ottobre del 2008 negli Stati Uniti, sotto il nome di T-Mobile G13. L'hardware del telefono è stato realizzato dalla compagnia di Taiwan High Tech Computer (HTC).

La caratteristica che rende particolarmente interessante Android è che è stato rilasciato sotto licenze open - source: Apache 2.0 e GPLv2. É quindi disponibile in rete anche il codice sorgente del sistema operativo, il quale può essere modificato e ridistribuito a piacere. Questa scelta è stata presa poiché Android è pensato per rendere facile lo sviluppo libero di applicazioni che sfruttino appieno le funzionalità.

Caratteristiche:

- **Framework dell'applicazione** permette il riuso e la sostituzione di componenti.
- **Dalvik virtual machine** ottimizzata per dispositivi mobili.
- **Browser integrato** basato sull'engine open source WebKit.
- **Grafica ottimizzata**, data da librerie personalizzate di grafica 2D, e grafica 3D basata sulle specifiche OpenGL ES 1.0.
- **SQLite** permette l'archiviazione di dati strutturati.
- **Supporto media**, per i comuni formati audio e video, formati di immagine (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF).
- **Telefonia GSM** (dipendente dall'hardware).
- **Bluetooth, EDGE, 3G, e Wi-Fi** (dipendente dall'hardware).
- **Camera, GPS, bussola ed accelerometro** (dipendente dall'hardware).
- **Ricco ambiente di sviluppo**, include un emulatore del dispositivo mobile, strumenti per il debugging, profilazione di memoria e prestazioni, e un plugin per Eclipse IDE.

Come i sistemi operativi per calcolatori fissi, anche Android è organizzato secondo il paradigma a pila. Uno schema della sua struttura è rappresentato nella figura seguente in cui è riportata una descrizione dettagliata delle principali componenti.

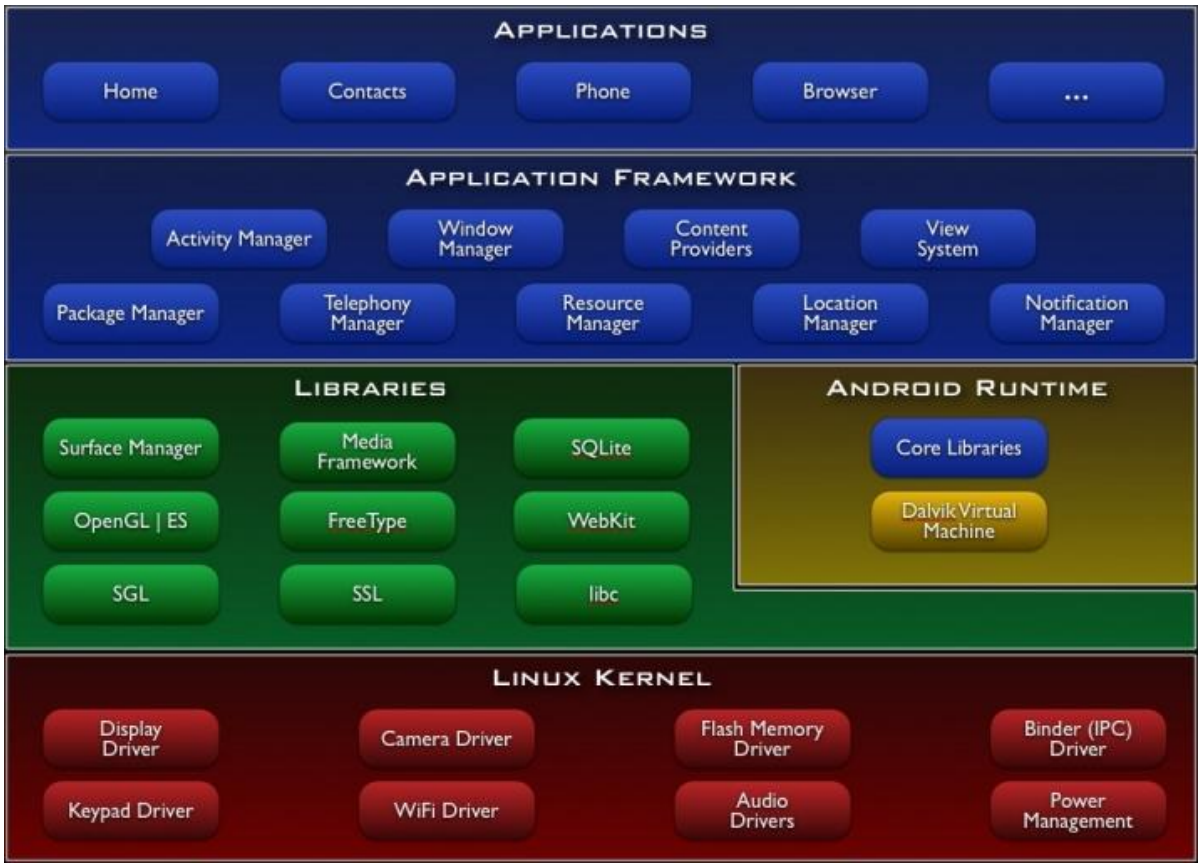


Figura 3.1: struttura dello stack software del sistema operativo Android.

- **Applications**: la versione base di Android non presenta solamente il sistema operativo, ma anche applicazioni base di frequente utilizzo, come un browser, un client di posta elettronica, un'agenda per la gestione dei contatti eccetera, a esse possono essere aggiunti altri programmi non appartenenti alla distribuzione ufficiale.
  La filosofia della OHA riguardo lo sviluppo di nuove applicazioni può essere riassunto con la frase: "*All applications are created equal*". Ovvero: non c'è differenza fra i prodotti ufficiali e quelli da sviluppare in modo indipendente. Tutti si basano sulle stesse librerie e hanno pari privilegi nell'accesso alle risorse messe a disposizione dal sistema operativo.
- **Application Framework**: questo livello dello stack è necessario per rendere disponibili i servizi del sistema operativo alle applicazioni di livello superiore. Essendo Android una piattaforma di sviluppo aperta offre agli sviluppatori la possibilità di creare applicazioni estremamente ricche ed innovative. Questi ultimi sono liberi di sfruttare i vantaggi dell'hardware del dispositivo, le informazioni sulla posizione di accesso, eseguire servizi in background, impostare allarmi, aggiungere notifiche alla barra di stato, e molto, molto di più. Gli sviluppatori hanno pieno accesso allo stesso framework utilizzato dalle applicazioni principali.
  L'architettura applicativa è stata progettata per semplificare il riutilizzo dei componenti; ogni applicazione può pubblicare le proprie capacità e qualsiasi altra applicazione può quindi farne uso. Questo stesso meccanismo consente ai componenti di essere sostituiti dall'utente.
  Gli elementi più importanti del framework sono:
  - *View System*: ovvero l'insieme di oggetti grafici che permettono la costruzione dell'interfaccia con l'utente.
  - *Content Providers*: un content provider è un elemento che permette la condivisione di dati fra applicazioni. La gestione dei dati memorizzati è lasciata all'implementazione del singolo

provider mentre tutti devono implementare interfacce comuni per permettere l'esecuzione di query e l'inserimento di dati a ogni applicazione.
    - *Resource Managers*: permettono l'accesso a tutti gli oggetti non costituiti da codice sorgente, quali immagini, file di configurazione dell'interfaccia, stringhe eccetera.
    - *Notification Managers*: permettono alle applicazioni di presentare all'utente notifiche di eventi asincroni che avvengono in background.
    - *Activity Managers*: gestiscono il life-cycle delle activity, le quali sono oggetti che rappresentano una singola operazione che l'utente può eseguire. Le activity, essendo la base di ogni programma sviluppato in Android, saranno descritte in modo approfondito in seguito.
- **Libraries**: le funzionalità messe a disposizione dall'application framework sono implementate da una serie di librerie scritte in C/C++; le più importanti sono:
    - Un'implementazione della libreria standard C (libc) specificatamente implementata per i sistemi Linux di tipo embedded.
    - Librerie per la gestione di materiale multimediale basate su PacketVideo's OpenCORE, supportano la produzione e la registrazione di molti formati audio e video, così come file di immagini statiche, tra cui MPEG4, H.264, MP3, AAC, AMR, JPG e PNG.
    - LibWebCore, un moderno motore del browser web che incrementa le capacità sia del browser Android e della vista web incorporata.
    - SQLite: potente e leggero motore per database relazionale a disposizione di tutte le applicazioni.
    - FreeType, bitmap e font rendering vettoriale.
    - Gestione della grafica 2D e 3D attraverso la libreria OpenGL.
    - Android Runtime: oltre a altre librerie è presente in questo sottoinsieme la Dalvik Virtual Machine, ovvero una macchina virtuale all'interno della quale sono eseguite le applicazioni. Essa esegue classi java che sono state compilate in uno specifico bytecode chiamato dex.
- **Linux Kernel**: Android si basa sulla versione 2.6 del kernel Linux per i servizi di base del sistema, quali sicurezza, gestione della memoria, gestione dei processi, stack di rete e driver. Il kernel funziona anche come un livello di astrazione tra l'hardware e il resto dello stack software.

3.2 Nozioni Fondamentali

Le applicazioni Android sono scritte nel linguaggio di programmazione Java e gli strumenti messi a disposizione dall'SDK compilano tale codice insieme a tutte le risorse, dati e file, producendo un pacchetto con estensione .apk. Tutto il codice in un unico file .apk è considerato come una sola applicazione ed è il file che i dispositivi Android utilizzano per installare l'applicazione stessa. Una volta installata su un dispositivo, ogni applicazione Android vive nella sua area di sicurezza:

- Il sistema operativo Android è un sistema Linux multi-utente in cui ogni applicazione è un utente differente.
- Per impostazione predefinita, il sistema assegna a ogni applicazione un ID univoco rappresentante un utente Linux (l'ID è utilizzato solo dal sistema e non è riconosciuto dalle applicazioni).
- Il sistema imposta le autorizzazioni per tutti i file di un'applicazione in modo che solo l'ID utente assegnato a tale applicazione può accedervi.
- Ogni processo possiede la sua macchina virtuale (VM), per cui il codice dell'applicazione viene eseguito in modo isolato rispetto alle altre applicazioni.

- Per impostazione predefinita, ogni applicazione è eseguita nel proprio processo Linux.
- Android inizia il processo quando uno dei componenti dell'applicazione deve essere eseguito, di contro chiude il processo quando tale componente non è più necessario o quando il sistema deve recuperare la memoria per altre applicazioni.

In questo modo, il sistema Android attua il principio del privilegio minimo. Cioè, ogni applicazione, di default, ha accesso solo ai componenti di cui ha bisogno per svolgere il suo lavoro e non di più.
Questo crea un ambiente altamente sicuro, in cui un'applicazione non può accedere a parti del sistema per le quali non ha il permesso.

Tuttavia, ci sono modi per un'applicazione di condividere i dati con altre applicazioni e di accedere ai servizi di sistema:

- E' possibile organizzare due applicazioni in modo da condividere lo stesso ID rappresentante l'utente Linux, nel qual caso esse sono in grado di accedere una ai file dell'altra.
- Per risparmiare risorse di sistema, applicazioni con lo stesso ID utente possono essere eseguite nello stesso processo Linux e condividere la stessa VM.
- Un'applicazione può richiedere il permesso di accedere a dati del dispositivo come i contatti dell'utente, messaggi SMS, la memoria esterna (SD), fotocamera, Bluetooth e altro ancora.
- Tutti i permessi dell'applicazione devono essere concessi dall'utente al momento dell'installazione.

Quanto detto sopra copre le basi di come un'applicazione Android esiste all'interno del sistema. Il resto del capitolo presenta:

- I componenti centrali del framework che definiscono l'applicazione.
- Il *Manifest*, file in cui si dichiarano i componenti e le caratteristiche richieste del dispositivo necessarie per l'applicazione.
- Risorse che sono separate dal codice dell'applicazione e consentono all'applicazione stessa di ottimizzare il suo comportamento per una varietà di configurazioni di dispositivi.

3.2.1 Componenti di un'applicazione

Si presentano ora i concetti e i tipi di dato astratto fondamentali per lo sviluppo di un'applicazione Android.
I blocchi essenziali di un'applicazione sono i componenti e ognuno di questi è un punto diverso attraverso il quale il sistema può accedere all'applicazione.

Non tutti i componenti sono punti di ingresso effettivo per l'utente e alcuni dipendono l'uno dall'altro, ma ognuno di essi esiste come elemento unico e con un ruolo specifico, contribuendo a definire il comportamento generale dell'applicazione.

Ci sono quattro diversi tipi di componenti per le applicazioni, ed ogni tipo ha uno scopo e un ciclo di vita diverso che definisce come il componente viene creato e distrutto.

- **Activity**
  Un'attività rappresenta una singola schermata nell'interfaccia utente. Per esempio, un'applicazione di posta elettronica potrebbe avere un'attività che mostra un elenco di nuove email, e un'altra attività per comporre un'e-mail, nonché un'altra attività per la lettura dei messaggi di posta elettronica. Anche se le attività lavorano insieme per creare un'esperienza utente coesa, all'interno dell'applicazione di posta elettronica, ognuna è indipendente dalle altre. Come tale, una diversa applicazione può iniziare una qualsiasi delle attività dell'esempio sopra (se l'applicazione di posta elettronica lo permette).

Per esempio, un'applicazione che utilizza la telecamera può iniziare l'attività dell'applicazione di posta elettronica che compone la nuova posta, in modo tale che l'utente possa condividere una foto. Le attività sono probabilmente il modello più diffuso in Android e si realizzano estendendo la classe base *android.app.Activity*. Nella prossima sezione si approfondiranno gli aspetti principali delle attività.

- **Service**
  Un Service è un componente dell'applicazione che viene eseguito in background per un periodo di tempo indefinito, e che non ha quindi interazione diretta con l'utente. É importante notare che il codice contenuto in una classe che estende Service è eseguito nel thread principale dell'applicazione. Nel caso in cui siano eseguite operazioni che utilizzano molto la CPU o la connessione a una rete è quindi consigliabile la creazione di un nuovo thread. Come per le activity anche nei Service sono presenti metodi di callback eseguiti in momenti specifici dell'esecuzione. Essi sono ad esempio onCreate(), onDestroy(), dal significato affine a quello specificato per la classe Activity. Vi sono due modi per avviare un service:

  - All'invocazione del metodo Context.startService() una nuova istanza di un service è creata (con il metodo onCreate()) e lanciata (con il metodo onStart(Intent, int)). L'esecuzione prosegue fino alla chiamata del metodo Context.stopService().
  - Se il metodo Context.onBind() è invocato, è instaurata una connessione persistente a un o specifico servizio il quale, se non esiste già, può esser creato. In questo caso il metodo onStart() non è chiamato. Il chiamante otterrà così un identificativo del Service, cosa che gli permetterà di compiere delle richieste. Un Service può essere al tempo stesso avviato con il metodo onStart() e oggetto di connessioni. Esso potrà essere terminato solo quando non ci saranno più connessioni attive e il metodo Context.stopService() non sarà chiamato.

  Un esempio di servizio è la riproduzione di musica in background mentre l'utente è in un'altra applicazione, oppure il recupero di dati attraverso la rete senza bloccare l'interazione dell'utente con un'altra attività. Un servizio si realizza estendendo la classe *android.app.Service*.
- **Content providers**
  Gli oggetti di tipo ContentProvider hanno la funzione di garantire l'accesso a dati condivisi da più applicazioni. L'effettiva implementazione del salvataggio dei dati in memoria di massa non è specificata ed è lasciata al programmatore. Tutti i content provider però devono implementare interfacce predefinite che specificano il modo col quale le query possono essere effettuate e la modalità di presentazione del risultato. I tipi di dati salvati possono essere di vario tipo: contatti telefonici, file audio e video. Ogni ContentProvider fornisce un URI ai client che vogliono utilizzare i suoi dati; è attraverso questa stringa che è possibile effettuare richieste o inserire dati.
  Esempio: L'URI content://contacts/people/ permette di ottenere tutti gli elementi della tabella people; invece l'URL content://contacts/people/ ritorna la singola persona identificata dal codice.
  L'inserimento di dati invece avviene specificando oltre a un URI anche una mappa che fa corrispondere alle colonne della tabella relazionale i campi della riga da inserire.
  Per esempio, il sistema Android fornisce un content provider che gestisce le informazioni di contatto dell'utente. Come tale, qualsiasi applicazione con le autorizzazioni appropriate può interrogare parte del content provider (come ad esempio ContactsContract.Data) per leggere e scrivere informazioni su una particolare persona. I content providers sono utili anche per la lettura e la scrittura dei dati che sono privati e non condivisi con l'applicazione.
  Un content providers è implementato estendendo la classe astratta *android.content.ContentProvider* e deve implementare un insieme di API standard per permette alle altre applicazioni di eseguire le transazioni di richiesta dati.

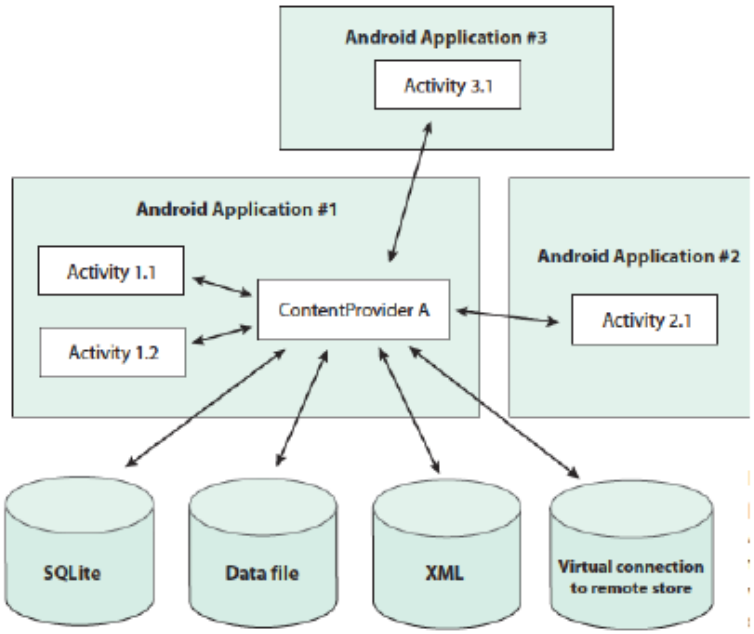


Figura 3.2: schema di utilizzo dei Content Provider all'interno delle applicazioni.

- **Broadcast receivers**
  Un Broadcast Receiver è utilizzato quando si intende intercettare un particolare evento, attraverso tutto il sistema. Ad esempio lo si può utilizzare se si desidera compiere un'azione quando si scatta una foto o quando parte la segnalazione di batteria scarica. La classe da estendere è *android.content.BroadcastReceiver*.

Un aspetto unico del design del sistema Android è che qualsiasi applicazione può iniziare un componente di un'altra applicazione. Per esempio, se si desidera che l'utente scatti una foto con la fotocamera del dispositivo, probabilmente esiste già un'altra applicazione che svolge tale attività e che può essere richiamata senza dover scrivere da zero un'attività. Per fare questo non c'é bisogno di includere o di far riferimento al codice dell'applicazione della fotocamera, ma basta semplicemente richiamare tale attività. Quando il sistema avvia un componente, esso avvia il processo per quell'applicazione (se non è già in esecuzione) e istanzia le classi necessarie per il componente.

Per esempio, se un'applicazione avvia l'attività fotocamera per la cattura di una foto, tale attività è eseguita nel processo che appartiene all'applicazione fotocamera, non nel processo dell'applicazione che l'ha richiamata.

Pertanto, a differenza di applicazioni su molti altri sistemi, le applicazioni Android non hanno un unico punto di ingresso (non c'è funzione main (), per esempio).

Poiché il sistema esegue ogni applicazione in un processo separato con i propri permessi che limitano l'accesso alle altre applicazioni, esse non possono attivare i componenti direttamente. Tale compito spetta al sistema che svolge la funzione di intermediario, cioè per attivare un componente in un'altra applicazione è necessario consegnare al sistema un messaggio che specifica l'intenzione di avviare un particolare componente.

3.2.2 Attivazione dei componenti

Tre dei quattro componenti, attività, servizi e broadcast receivers, sono attivati da un messaggio asincrono chiamato *Intent*. Gli *Intents* legano i singoli componenti gli uni agli altri in fase di esecuzione, sia se il componente appartiene alla stessa applicazione sia se appartiene ad un'altra.

Per le attività e i servizi, un *Intent* definisce l'azione da eseguire (ad esempio, "vedere" o "inviare" qualcosa) e può specificare le URI dei dati sui quali agire. Per esempio, un *Intent* potrebbe trasmettere una richiesta di attività per mostrare un'immagine o per aprire una pagina web. In alcuni casi, è possibile avviare un'attività per ricevere un risultato, in questo caso, l'attività restituisce anche il risultato in un *Intent*; ad esempio è possibile emettere un *Intent* per consentire all'utente di scegliere un contatto personale e rispedirlo al mittente, in tal caso l'*Intent* di ritorno include un URI che punta al contatto scelto.

Per i broadcast receivers, l'*Intent* definisce semplicemente l'annuncio da trasmettere; ad esempio, la comunicazione per indicare che la batteria del dispositivo è bassa, include solo una stringa nota che indica "batteria scarica".

L'altro componente, il content provider, non è attivato tramite un *Intent*, ma è attivato quando è raggiunto da una richiesta dal Content Resolver. Il content resolver gestisce tutte le operazioni dirette con il content provider così che il componente che esegue le transazioni con il provider non abbia bisogno di particolari metodi, ma invoca direttamente i metodi sull'oggetto ContentResolver. Questo lascia uno strato di astrazione tra il content provider e il componente che richiede le informazioni.

Ci sono metodi separati per l'attivazione di ogni tipo di componente:

- È possibile avviare un'attività, o darle qualcosa da fare, passando un *Intent* al metodo startActivity() o al metodo startActivityForResult() quando si desidera che l'attività restituisca un risultato.
- È possibile avviare un servizio o dargli nuove istruzioni passando un *Intent* al metodo StartService(), oppure è possibile vincolare un servizio passando un *Intent* a bindService().
- È possibile avviare un broadcast passando un *Intent* a metodi come sendBroadcast(), sendOrderedBroadcast() o sendStickyBroadcast().
- È possibile eseguire un'interrogazione a un content provider chiamando il metodo query() su un ContentResolver.

3.2.3 Il file manifest

Prima che il sistema Android possa avviare un componente dell'applicazione deve sapere che esiste, attraverso la lettura del file AndroidManifest.xml dell'applicazione (il file "manifesto"). L'applicazione deve dichiarare tutti i suoi componenti in questo file, che deve essere alla radice della cartella del progetto.

Oltre a dichiarare i componenti dell'applicazione, il *manifest* svolge un certo numeto di compiti quali:

- Identificare eventuali autorizzazioni utente che l'applicazione richiede, come l'accesso a internet o l'accesso in lettura dei contatti dell'utente.
- Dichiarare il livello minimo di API richiesto dall'applicazione, in base alle API utilizzate.
- Dichiarare le caratteristiche hardware e software utilizzate o richieste dall'applicazione, come ad esempio la fotocamera, servizi bluetooth o uno schermo multitouch.
- Librerie alle quali l'applicazione deve fare riferimento, come ad esempio quelle di Google Maps.
- E altro ancora.

Dichiarazione dei componenti

Il compito primario del *manifest* è di informare il sistema sui componenti dell'applicazione. Ad esempio, un file *manifest* può dichiarare un'attività come segue:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
  <application android:icon="@drawable/app_icon.png" ... >
    <activity android:name="com.example.project.ExampleActivity"
              android:label="@string/example_label" ... >
    </activity>
    ...
  </application>
</manifest>
```

Nell'elemento <application>, l'attributo android:icon punta alle risorse per il recupero dell'icona che identifica l'applicazione. Nell'elemento <activity>, l'attributo android:name specifica il nome completo della classe dell'activity e l'attributo android:label specifica la stringa utilizzata come nome visibile all'utente dell'activity.

È necessario dichiarare tutti i componenti dell'applicazione in questa maniera:

- <activity> per le attività
- <service> per i servizi
- <receiver> per i broadcast receivers
- <provider> per i content providers

Attività, servizi e content providers che si includono nel sorgente ma non si dichiarano nel *manifest* non sono visibili al sistema e, di conseguenza, non potranno mai funzionare. Tuttavia, i broadcast receivers possono essere dichiarati nel *manifest* o creati dinamicamente nel codice e registrati dal sistema chiamando il metodo registerReceiver().

Dichiarazione delle azioni dei componenti

Come detto sopra, per l'attivazione dei componenti, è possibile utilizzare un *Intent* per avviare attività, servizi e broadcast receiver; ciò è possibile farlo esplicitando, all'interno dell'*Intent*, il nome del componente di destinazione identificato dalla classe relativa al componente stesso.

Tuttavia, il vero potere degli *Intent* risiede nel concetto di azioni, con questo concetto è sufficiente descrivere il tipo di azione che si desidera eseguire (e, facoltativamente, i dati sui quali si desidera eseguire l'azione) e consentire al sistema di trovare un componente sul dispositivo in grado di eseguire l'operazione specificata ed avviarlo. Se ci sono più componenti in grado di eseguire l'azione descritta dall'*Intent*, allora l'utente deve specificare quale vuole eseguire.

Il modo in cui il sistema identifica i componenti che possono rispondere ad un *Intent* è di confrontare l'*Intent* ricevuto con i cosidetti *Intent filters*, forniti nel file *manifest*, di altre applicazioni sul dispositivo.

Quando si dichiara un componente nel *manifest* dell'applicazione è possibile includere alcuni *Intent filters*, che dichiarano le capacità del componente in modo che possa rispondere agli *Intent* provenienti da altre applicazioni. È possibile dichiarare un *Intent filter* per un componente con l'aggiunta di un elemento, <intent-filter>, come figlio dell'elemento dichiarativo del componente stesso.

Per esempio, un'applicazione di posta elettronica con un'attività per la composizione di una nuova email può dichiarare un *Intent filter* nel *manifet* per rispondere agli *Intents* di "invio".

Una qualsiasi altra attività può quindi creare un *Intent* con l'azione "invia" (ACTION_SEND), e il sistema farà corrispondere tale richiesta con l'attività di "invio" dell'applicazione email e la lancierà quando si invoca il metodo startActivity() che prende come argomento l'*Intent*.

Dichiarazione dei requisiti dell'applicazione

C'è una varietà di dispositivi che installano Android e non tutti offrono le stesse caratteristiche e capacità. Al fine di impedire che le applicazioni siano installate su dispositivi che non hanno le caratteristiche necessarie per eseguirle, è importante definire chiaramente un profilo dei dispositivi che le supportano, dichiarandoli insieme al software all'interno del *manifest*. La maggior parte di queste dichiarazioni ha valore informativo e il sistema non le legge, ma i servizi esterni come il Market Android fanno uso di tali informazioni per offrire dei termini di filtraggio quando l'utente effettua una ricerca tra le applicazioni per il proprio dispositivo.

Per esempio, se l'applicazione richiede una fotocamera e usa le API introdotte in Android 2.1(Livello 7), si dovrebbero dichiarare questi requisiti nel file *manifest*. In questo modo, i dispositivi che non hanno una fotocamera e dispongono di una versione diversa dalla 2.1 non saranno in grado di installare l'applicazione dal Market.

Ecco alcune delle più importanti caratteristiche del dispositivo che si dovrebbe considerare nella progettazione e sviluppo dell'applicazione:

- Dimensioni dello schermo e la densità
  Al fine di classificare i dispositivi dal loro tipo di schermo, Android definisce due caratteristiche per ciascun dispositivo: dimensioni dello schermo (le dimensioni fisiche dello schermo) e la densità dello schermo (la densità fisica dei pixel sullo schermo, o dpi - punti per pollice). Per semplificare tutte le diverse tipologie di configurazioni dello schermo, il sistema Android li generalizza in gruppi selezionati che li rendono più facile da identificare. Le dimensioni dello schermo sono: piccolo, normale, grande ed extra large. La densità dello schermo può essere: bassa densità, a media densità, alta densità, ed extra alta densità. Per impostazione predefinita, l'applicazione è compatibile con tutte le dimensioni dello schermo e densità, perché il sistema Android effettua le regolazioni adeguate sia per il layout dell'interfaccia utente sia per le altre risorse quali le immagini. Tuttavia si dovrebbero creare layout specializzati per determinate dimensioni di schermo e fornire immagini specializzati per determinate densità, utilizzando layouts alternativi, e dichiarando nel *manifest* esattamente quali dimensioni dello schermo l'applicazione supporta con l'elemento <supports-screens>.
- Configurazioni di ingresso
  Molti dispositivi forniscono un diverso tipo di meccanismo di input dell'utente, come ad esempio una tastiera hardware, una trackball, o un pad a cinque direzioni di navigazione. Se l'applicazione richiede un particolare tipo di input hardware lo si dovrebbe dichiarare nel *manifest* con l'elemento <uses-configuration>, in ogni caso è raro che un'applicazione dovrebbe richiedere una certa configurazione di ingresso.

- Componenti del dispositivo
  Ci sono molte componenti hardware e software che possono esistere o non in un dato dispositivo Android, come ad esempio una fotocamera, un sensore di luce, bluetooth, una certa versione di OpenGL o la fedeltà del touchscreen. Non si dovrebbe mai pensare che una certa componente sia disponibile su tutti i dispositivi Android, perciò si dovrebbe dichiarare qualsiasi funzionalità utilizzata dall'applicazione con l'elemento <uses-feature>.
- Versione della piattaforma
  Diversi dispositivi Android eseguono diverse versioni della piattaforma, come ad esempio Android 1.6 o 2.3, ed ogni versione successiva spesso include API che non sono disponibili nella versione precedente. Al fine di indicare quale insieme di API sono utilizzate, ciascuna versione della piattaforma specifica un livello di API (ad esempio, Android 1.0 è di livello 1 ed Android 2.3 è il livello 9). Se si utilizzano le API che sono state aggiunte alla piattaforma dopo la versione 1.0, si dovrebbe dichiarare il livello minimo di API utilizzando l'elemento <uses-sdk>.

3.2.4 Risorse di un'applicazione

Un'applicazione Android non è composta soltanto da semplice codice ma richiede risorse che sono separate dal codice sorgente, come immagini, file audio e qualsiasi oggetto relativo alla presentazione visiva dell'applicazione. Ad esempio, si dovrebbero definire le animazioni, menu, gli stili, i colori e il layout dell'interfaccia utente attraverso i file XML. L'utilizzo delle risorse dell'applicazione rende facile l'aggiornamento di varie caratteristiche senza modificare il codice e fornendo un insiemi di risorse alternative permettendo di ottimizzare l'applicazione per una varietà di configurazioni dei dispositivi (come le lingue differenti e dimensioni dello schermo).

Per ogni risorsa che si include nel progetto Android, gli strumenti dell'SDK definiscono un intero utilizzato come ID univoco, che può essere utilizzato per fare riferimento alla risorsa dal codice dell'applicazione o da altre risorse definite in XML. Per esempio, se l'applicazione contiene un'immagine di nome logo.png, salvata nella cartella res/drawable/, gli strumenti dell'SDK genereranno un ID denominato R.drawable.logo, che si potrà utilizzare per fare riferimento all'immagine ed inserirla nell'interfaccia utente.

Uno degli aspetti più importanti nel fornire le risorse separate dal codice sorgente è la possibilità di offrire risorse alternative per diverse configurazioni dei dispositivi. Per esempio, definendo le stringhe dell'interfaccia utente in un file XML, è possibile tradurre le stringhe in altre lingue e salvarle in un file separato.

Poi sulla base di un qualificatore che si aggiunge come suffisso al nome della cartella contenente le risorse ed in base alle impostazione della lingua sul dispositivo dell'utente, il sistema Android applicherà le stringhe appropriate per l'interfaccia utente. Android supporta molti qualificatori diversi per le risorse alternative. Il qualificatore è una breve stringa che si include al nome della cartella delle risorse al fine di definire la configurazione dei dispositivi per i quali queste risorse dovrebbero essere utilizzate. Come altro esempio, è spesso necessario creare diversi layout per le attività, a seconda dell'orientamento dello schermo del dispositivo e delle dimensioni. Per esempio, quando lo schermo del dispositivo è orientato in verticale (altezza), si potrebbe desiderare un layout con bottoni a orientamento verticale, ma quando lo schermo è orientato in orizzontale (larghezza), i pulsanti devono essere allineati orizzontalmente.

Per modificare il layout a seconda dell'orientamento, è possibile definire due diversi layout ed applicare il qualificatore adeguato per ogni nome della cartella di ogni layout. Poi, il sistema applica automaticamente il layout appropriato a seconda dell'orientamento del dispositivo.

3.3 **Activity, il componente principale delle applicazioni**

Un'attività è una componente dell'applicazione che fornisce una schermata con cui l'utente può interagire al fine di svolgere un'azione, come ad esempio effettuare una chiamata, scattare una foto, spedire una mail o visualizzare una mappa. Ad ogni attività è associata una finestra in cui disegnare l'interfaccia utente; tale finestra di solito riempie lo schermo, ma può essere più piccola e fluttuare sopra le altre finestre.

Solitamente un'applicazione è formata da più attività che sono debolmente legate tra loro, tra queste ve n'è una definita "principale" perché è presentata all'utente quando l'applicazione è lanciata per la prima volta. Ogni attività può lanciare altre attività al fine di compiere differenti azioni ed ogni volta che una nuova attività è eseguita quella precedente, è interrotta, ma il sistema preserva quest'ultima salvandola in uno stack chiamato "back stack".

Il back stack è gestito tramite una coda LIFO (last in first out), così quando l'utente ha finito di eseguire l'attività corrente e preme sul tasto indietro; essa è estratta dallo stack e distrutta, mentre l'attività precedente riprende l'esecuzione. Quando un'attività è arrestata perché una nuova attività è eseguita, si verifica un cambio di stato il quale è segnalato attraverso i metodi di callback del ciclo di vita delle attività.

Ci sono diversi metodi che un'attività potrebbe ricevere a causa di un cambiamento del suo stato, ad esempio se il sistema l'ha sta creando, arrestando, riprendendo o distruggendo; ed ogni metodo di callback fornisce l'opportunità di svolgere un lavoro specifico che è opportuno per quel cambiamento di stato. Per esempio, quando un'attività è arrestata, dovrebbe rilasciare qualsiasi oggetto di grandi dimensioni come connessioni ad internet o a database; e quando l'attività riprende la sua esecuzione, è possibile riacquistare le risorse necessarie e riprendere le azioni che si erano interrotte.

Il resto di questa sezione descrive i principi fondamentali su come creare ed utilizzare un'attività, includendo una discussione completa di come funziona il suo ciclo di vita così da poter comprendere come gestire le transizioni tra i vari stati.

3.3.1 Creazione di un'attività

Per creare un'attività bisogna utilizzare una sottoclasse di Activity o una sua sottoclasse già esistente. In tale sottoclasse bisogna implementare i metodi di callback che il sistema chiamerà quando l'attività effettuerà le transizioni tra i vari stati del suo ciclo, come quando è creata, arrestata, ripresa o distrutta. I due metodi di callback più importanti sono:

- onCreate()
  Tale metodo deve essere implementato ed il sistema lo richiamerà quando l'attività è creata.
  L'implementazione consiste nell'inizializzazione dei componenti essenziali dell'attività ed ancor più importante, questo è il luogo in cui si deve richiamare il metodo setContentView() per definire il layout dell'interfaccia utente dell'attività.
- onPause()
  Il sistema chiama questo metodo come la prima indicazione che l'utente sta lasciando l'attività, anche se non sempre ciò significa che l'attività sarà distrutta. Tale metodo è, di solito, il luogo dove si dovrebbero sottoscrivere tutte le modifiche che devono essere mantenute al di là della sessione corrente dell'utente.

Ci sono molti altri metodi di callback facenti parti del ciclo di vita, che si dovrebbero utilizzare per fornire un'esperienza utente fluida tra le attività e per gestire le interruzioni inaspettate che causano il fermo o addirittura la distruzione dell'attività. Tutti i metodi di callback saranno discussi in dettaglio più avanti.

3.3.2 Implementazione dell'interfaccia utente

L'interfaccia utente per un'attività è fornita da una gerarchia di views, oggetti derivati dalla classe View. Ogni view controlla un particolare spazio rettangolare all'interno dalla finestra dell'attività e in più può rispondere all'interazione con l'utente; ad esempio, una view potrebbe essere un bottone che avvia un'azione quando è schiacciato. Android fornisce un numero di view già definite che è possibile utilizzare per progettare ed organizzare il layout.

I "Widgets" sono views che offrono elementi visibili ed interattivi per lo schermo, come bottoni, campi di testo, checkbox o semplicemente un'immagine.

I "Layout" sono views derivate da ViewGroup che forniscono un modello unico di layout per le views che sono al loro interno, ad esempio layout lineare, layout a griglia o layout relativi.

È anche possibile creare una sottoclasse delle classi View e ViewGroup per creare widgets e layouts personalizzati ed applicarli al layout dell'attività.

Il modo più comune per definire un layout usando views è con un file XML salvato nelle risorse dell'applicazione, in questo modo è possibile mantenere il design dell'interfaccia utente separato dal codice sorgente che defisce il comportamento dell'attività. È possibile impostare il layout come interfaccia utente per l'attività attraverso il metodo setContentView(), passandogli l'id della risorsa relativa al layout.

Tuttavia, è anche possibile creare nuove views nel codice sorgente dell'attività e costruirne una gerarchia attraverso l'inserimento di quest'ultime in un ViewGroup, e poi richiamare il metodo setContentView() passandogli la root del layout.

3.3.3 Dichiarazione dell'attività nel Manifest

Affinché un'attività possa essere accessibile al sistema bisogna dichiararla all'interno del manifest, e per farlo basta aggiungere l'elemento <activity> come figlio dell'elemento <application> come mostrato nel frammento di codice riportato di seguito.

```
<manifest ... >
  <application ... >
      <activity android:name=".ExampleActivity" />
      ...
  </application ... >
  ...
</manifest >
```

Ci sono diversi altri attributi che possono essere inclusi in questo elemento per definire proprietà come l'etichetta dell'attività, un'icona identificativa dell'attività o un tema per lo stile dell'interfaccia utente.

Un elemento <activity> può anche specificare vari intent filters utilizzando l'elemento <intent-filter>, al fine di dichiarare come gli altri componenti dell'applicazione possano attivare l'attività corrispondente.

Quando si crea una nuova applicazione utilizzando gli strumenti dell'SDK Android, questi creano automaticamente un'attività "madre" che include un intent filter il quale dichiara che l'attività risponda all'azione "main" e che dovrebbe essere posta nella categoria "lancher".

L’intent filter è simile a quanto segue.

```
<activity android:name=".ExampleActivity" android:icon="@drawable/app_icon">
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
```

L’elemento <action> specifica che questo è il punto di accesso principale all’applicazione, mentre l’elemento <category> specifica che l’attività dovrebbe essere elencata nell’applicazione di avvio del sistema per consentire agli utenti di lanciarla.

Se si intende creare un’applicazione autosufficiente e non consentire ad altre di attivare le sue attività, allora non è necessario dichiarare alcun intent filters. Soltando un’attività all’interno di un’applicazione deve avere associata l’azione “main” e la categoria di “lancher”, come nell’esempio precedente. Le attività che non si desiderano rendere disponibili ad altre applicazioni non dovrebbero avere intent filters e possono essere eseguite utilizzando intent espliciti come discusso nella prossima sezione, tuttavia, se si vuole fare rispondere l’attività ad intent impliciti che sono consegnati da altre applicazioni è necessario definire intent filters aggiuntivi per l’attività. Per ogni tipo di intent al quale si vuole rispondere, è necessario includere un <intent-filter> che includa un elemento <action> ed opzionalmente un elemento <category> e/o un elemento <data>, tali elementi specificano il tipo di intent a cui l’attività può rispondere.

3.3.4 Avvio e arresto di un’attività

È possibile avviare un’altra attività chiamando il metodo startActivity() passandogli un intent che descrive l’attività che si desidera avviare. L’intent specifica o l’attività esatta da avviare o descrivere il tipo di azione che si desidera eseguire e il sistema seleziona l’attività appropriata, che può anche essere di un’altra applicazione.

Un intent può anche trasportare piccole quantità di dati per essere utilizzate dall’attività cui farà riferimento l’intent. Quando si lavora all’interno dell’applicazione, si ha spesso l’esigenza di lanciare in modo semplice un’attività nota, ciò è possibile farlo creando un intent che definisce in modo esplicito l’attività da lanciare usando il nome della classe.

Per esempio, ecco come un’attività ne inizia un’altra di nome SignInActivity:

```
Intent intent = new Intent(this, SignInActivity.class);
startActivity(intent);
```

Tuttavia, l’applicazione potrebbe anche voler eseguire alcune azioni come ad esempio inviare una mail, messaggi di testo o aggiornare lo stato usando i dati dell’attività. In tal caso, l’applicazione potrebbe non avere tra le proprie attività una che possa eseguire tali azioni, così si potrebbero sfruttare le attività fornite dalle altre applicazioni sul dispositivo in grado di eseguire le azioni richieste, questo è l’aspetto più prezioso degli intent. Se ci sono molteplici attività in grado di gestire l’intent, allora l’utente può scegliere quale usare.

Ad esempio, se si desidera consentire all'utente di inviare un messaggio di posta elettronica, è possibile creare il seguente intent:

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.putExtra(Intent.EXTRA_EMAIL, recipientArray);
startActivity(intent);
```

Il parametro EXTRA_EMAIL aggiunto all'intent è un array di stringhe rappresentante gli indirizzi email cui il messaggio deve essere recapitato. Quando un'applicazione di posta elettronica risponde a tale intent, legge l'array di stringhe fornite nell'extra e le mette nel campo "to" del modulo di composizione email.

A volte, si potrebbe desiderare ricevere un risultato dall'attività che si è avviata, in tal caso bisogna richiamare il metodo startActivityForResult() al posto di startActivity(). Per ricevere i risultati dall'attività successiva bisogna implementare il metodo onActivityResult(), così quando l'attività è terminata, restituisce il risultato in un intent a tale metodo.

Dall'altra parte è possibile arrestare un'attività chiamando il suo metodo finish(), o in alternativa è possibile invocare il metodo finishActivity() per arrestare un'attività distinta precedentemente iniziata.

Nella maggior parte dei casi non si deve esplicitamente terminare un'attività con i metodi sopra elencati, infatti, come verrà discusso nella sezione seguente sul ciclo di vita delle attività, il sistema Android gestisce tale ciclo di vita in modo tale da non esserci bisogno di terminare un'attività.

In più chiamare questi metodi potrebbe influenzare negativamente l'esperienza degli utenti e dovrebbero essere utilizzati solamente se non si vuole che l'utente ritorni all'istanza dell'attività appena interrotta.

3.3.5 Gestione del ciclo di vita delle attività

La gestione di vita di un'attività, implementando i metodi di callback, è un passo cruciale per lo sviluppo di un'applicazione forte e flessibile. Tale ciclo è direttamente influenzato dalla sua associazione con le altre attività, dal suo compito e dal back stack. Tre sono le modalità principali nelle quali un'Activity si può trovare:

- **Running**: l'activity è visualizzata sullo schermo e l'utente interagisce con essa.
- **Paused**: l'utente non può più interagire con l'attività ma il suo layout è ancora visibile (ad esempio sopra di esso appare una finestra di dialogo). Le activity in questo stato mantengono tutte le informazioni riguardanti il loro stato e possono essere terminate dalla DVM nel caso di scarsità di risorse.
- **Stopped**: quando l'utente passa da un'Activity a un'altra, la prima è posta in stato di stop.
  Ciò significa che una nuova activity sarà posta sulla cima della pila. Le attività in stop sono le prime a essere terminate in caso di necessità di ulteriori risorse.

Se l'attività è sospesa o interrotta, il sistema può rilasciarla dalla memoria sia chiedendo che finisca, chiamando il suo metodo finish(), o semplicemente uccidendo il suo processo. Quando l'attività è aperta di nuovo deve essere ricreata da zero.

L'uscita e l'entrata di un'attività in diversi stati del ciclo di vita è notificata tramite vari metodi di callback. Tutti i metodi di callback possono essere implementati in modo il giusto lavoro quando lo stato dell'attività cambia.

Il seguente frammento di codice include tutti i metodi fondamentali del ciclo di vita di un'attività.

```
public class ExampleActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
    }
    @Override
    protected void onStart() {
        super.onStart();
        // The activity is about to become visible.
    }
    @Override
    protected void onResume() {
        super.onResume();
        // The activity has become visible (it is now "resumed").
    }


    @Override
    protected void onPause() {
        super.onPause();
        // Another activity is taking focus (this activity is about to be "paused").
    }
    @Override
    protected void onStop() {
        super.onStop();
        // The activity is no longer visible (it is now "stopped")
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // The activity is about to be destroyed.
    }
}
```

Con l'implementazione di questi metodi, è possibile monitorare tre cicli annidati nel ciclo di vita di un'attività:

- **Ciclo di vita**: ricadono in questo insieme tutte le operazioni comprese fra le chiamate ai metodi onCreate(), nel quale si allocano tutte le risorse necessarie, e onDestroy(), nel quale le risorse sono rilasciate. Per esempio, se l'attività ha un thread in esecuzione in background per scaricare dati dalla rete, si potrebbe creare tale thread nel metodo onCreate() e poi fermarlo nel metodo onDestroy().
- **Ciclo di visibilità**: compreso fra i metodi onStart() e onStop(). In questo periodo di tempo l'interfaccia dell'activity è sì visibile all'utente ma può non essere possibile l'interazione con essa a causa della presenza di altre activity in primo piano.
  Ad esempio, è possibile registrare un BroadcastReceiver in onStart() per monitorare i cambiamenti che impattano sull'interfaccia utente, e annullare la registrazione in onStop() quando l'utente non può più vedere ciò che si sta visualizzando. Il sistema potrebbe chiamare onStart() e OnStop()più volte durante l'intera durata dell'attività, come se l'attività si alterna tra l'essere visibile e non all'utente.
- **Ciclo di vita in primo piano**: in questo insieme sono comprese tutte le operazioni fra i metodi onResume() ed onPause(). L'Activity è qui completamente disponibile all'utente il quale può interagire con essa, ad esempio riempiendo dei form di input. Dato che questa transizione appare

spesso, il codice in questi due metodi dovrebbe essere abbastanza leggero in modo da evitare transizioni lente che costringono l'utente ad aspettare.

La figura sottostante illustra questi cicli e i percorsi che un'attività potrebbe prendere tra gli stati. I rettangoli rappresentano i metodi di callback che è possibile implementare per eseguire le operazioni quando l'attività esegue una transizione tra gli stati.

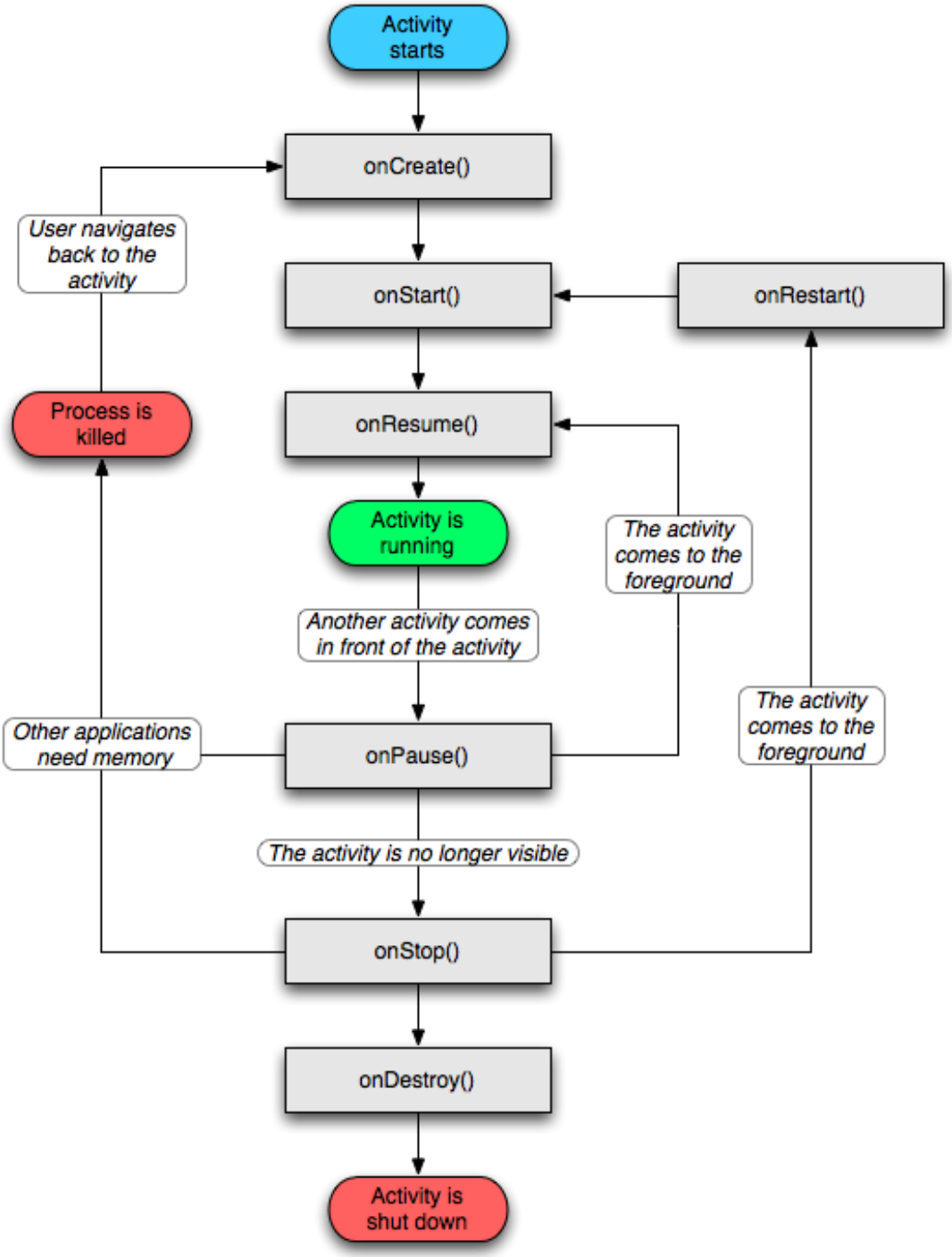


Figura 3.3: ciclo di vita delle attività all'interno di un'applicazione.

Lo stesso ciclo di vita dei metodi di callback è elencato nella tabella sottostante, che descrive ciascuno dei metodi di callback in modo più dettagliato e ne individua ognuno all'interno del ciclo di vita complessivo dell'attività, includendo se il sistema può uccidere l'attività dopo che il metodo di callback è completato.

Metodi	Descrizione	Killable after?	Next
onCreate()	Richiamato non appena l'attività è creata. L'argomento savedInstanceState serve per riportare un eventuale stato dell'attività salvato in precedenza da un'altra istanza che è stata terminata. L'argomento è null nel caso in cui l'attività non abbia uno stato salvato. Sempre seguito da onStart().	No	onStart()
onRestart()	Chiamato dopo che l'attività è stata arrestata, appena prima che sia ripresa. Sempre seguito da onStart()	No	onStart()
onStart()	Chiamato poco prima che l'attività diventa visibile all'utente. Seguito da onResume() se l'attività viene in primo piano, o da onStop()se diventa nascosta.	No	onResume() or onStop()
onResume()	Chiamato appena prima che l'attività inizi a interagire con l'utente. Sempre seguito da onPause().	No	onPause()
onPause()	Chiamato quando il sistema è in procinto di iniziare a riprendere un'altra attività. Questo metodo è generalmente utilizzato per salvare le modifiche in dati persistenti, fermare animazioni e altre cose che possono consumare CPU e così via. Tale metodo dovrebbe eseguire le operazioni di sopra in fretta poiché la prossima attività non sarà ripresa fino a che quest'ultimo non ha terminato. Seguito da onResume() se l'attività ritorna in primo piano, o da onStop()se diventa invisibile per l'utente.	**Yes**	onResume() or onStop()

Metodi	Descrizione	Killable after?	Next
onStop()	Chiamato quando l'attività non è più visibile per l'utente. Questo può accadere perché è distrutta, o perché un'altra attività, sia un'esistente o una nuova, è stata ripresa. Seguito da onRestart()se l'attività è tornare a interagire con l'utente , o da onDestroy() se l'attività termina definitivamente.	**Yes**	onRestart() or onDestroy()
onDestroy()	Chiamato prima che l'attività è distrutta. Questa è l'ultima chiamata che l'attività riceverà. Potrebbe essere chiamato sia perché l'attività sta finendo, qualcuno ha chiamato finish(), o perché il sistema è ha bisogno di spazio in maniera urgente. Si può distinguere tra questi due scenari con il metodo isFinishing().	**Yes**	*Nessuno*

Tabella 3.4: descrizione dettagliata dei metodi che formano il ciclo di vita dell'attività.

La colonna denominata "killable after?" indica se il sistema può uccidere il processo che ospita l'attività in qualsiasi momento dopo la restituzione del metodo, senza eseguire nessun'altra riga di codice dell'attività. Tre metodi sono contrassegnati con "Yes": onPause(), OnStop() e OnDestroy(). I metodi che sono contrassegnati con "No" proteggono il processo che ospita l'attività dall'essere ucciso. Quindi, un'attività è killable dal metodo onPause() fino alla chiamata di onResume(), quindi non sarà possibile ucciderla fino a che il metodo onPause() non verrà di nuovo chiamato.

L'introduzione alla gestione del ciclo di vita di un'attività accenna brevemente al fatto che quando un'attività è in pausa o arrestata, lo stato di quest'ultima è mantenuto dal sistema. Questo è vero perché l'oggetto Activity è ancora conservato in memoria e tutte le informazioni sui suoi componenti e lo stato corrente sono ancora vivi. Così, tutte le modifiche fatte dall'utente all'interno dell'attività sono salvate nella memoria, in modo che quando l'attività torna in primo piano, quando si "riprende", tali modifiche sono ancora presenti.

Tuttavia, quando il sistema distrugge un'attività al fine di recuperare la memoria, l'oggetto Activity è distrutto, così che il sistema non può semplicemente riprendere l'attività con il suo stato originale. Per far ciò, il sistema deve ricreare l'oggetto Activity per far si che l'utente ritrovi l'attività come l'ha lasciata prima di cambiare attività. In questa situazione, è possibile garantire che le informazioni importanti sullo stato dell'attività siano conservate mediante l'implementazione di un ulteriore metodo di callback che consente di salvare le informazioni sullo stato dell'attività e quindi ripristinarlo quando il sistema ricrea l'attività.

Il metodo di callback in cui è possibile salvare tali informazioni è onSaveInstanceState(). Il sistema chiama questo metodo prima che l'attività diventa vulnerabile all'essere distrutta e gli passa un oggetto Bundle. Il Bundle è, dove è possibile memorizzare informazioni di stato sull'attività nel formato nome-valore, utilizzando metodi quali putString().

Poi, se il sistema uccide il processo contenente l'attività e l'utente torna a quell'attività, il sistema passa il Bundle al metodo onCreate() in modo da poter ripristinare lo stato dell'attività che è stato salvato durante la chiamata al metodo onSaveInstanceState(). Se non ci sono informazioni sullo stato da ripristinare, il Bundle passato a onCreate() è nullo.

Alcune configurazioni dei dispositivi possono cambiare durante l'esecuzione, come ad esempio l'orientamento dello schermo, la disponibilità di tastiera e la lingua. Quando un tale cambiamento si verifica, Android riavvia l'attività in esecuzione chiamando il metodo onDestroy()seguito immediatamente dalla chiamata al metodo onCreate(). Il comportamento di riavvio è progettato per aiutare l'applicazione ad adattarsi alle nuove configurazioni ricaricando automaticamente l'applicazione con le risorse alternative che si sono fornite.

Se si progetta l'attività in modo da gestire correttamente questo evento, sarà più resistente agli eventi imprevisti nel suo ciclo di vita. Il modo migliore per gestire un cambiamento di configurazione è preservare semplicemente lo stato dell'applicazione utilizzando onSaveInstanceState() e onRestoreInstanceState(), come discusso in precedenza.

3.4 Lo sviluppo di un'applicazione per Android

Il linguaggio di programmazione scelto dagli sviluppatori della OHA per la realizzazione di applicazioni in Android è Java. Questa scelta è particolarmente significativa in quanto Java è il linguaggio più diffuso al mondo. L'obiettivo della Open Handset Alliance è quindi quello di rendere accessibile al maggior numero di programmatori la possibilità di realizzare software per Android. È da leggere in quest'ottica la scelta di Google di indire nel maggio del 2008 un concorso (Android Developer Challenge) volto a premiare le applicazioni più innovative. Le semplificazioni introdotte permettono al programmatore di concentrarsi sulla sua applicazione tralasciando le problematiche riguardanti l'interazione con l'hardware, gestite dal sistema operativo.

3.4.1 Il plugin per Eclipse e l'emulatore

La realizzazione di applicazioni per Android è possibile attraverso il Software Development Kit (SDK), in altre parole l'insieme di tutte le classi Java necessarie allo sviluppo. La ricerca da parte della OHA della massima semplicità nella realizzazione di nuove applicazioni ha portato alla realizzazione di un'IDE (Integrated Development Environment) apposita per Android. Essa consiste in un'estensione dell'IDE preesistente Eclipse attraverso un plugin. Anche la scelta di Eclipse non è casuale poiché esso è uno degli ambienti di sviluppo gratuiti più diffusi per la programmazione in java. Il plugin ADT (Android Developement Tools) non è parte integrante dell'SDK ma è estremamente utile in fase di sviluppo. Esso, ad esempio, aggiunge a Eclipse la perspective chiamata DDMS (Dalvik Debug Monitor Service) che permette di lanciare l'emulatore in modalità di debug e avere una visione di insieme sui processi attivi, l'occupazione della memoria e le prestazioni della CPU. Vi sono anche view per il controllo dei dati sul file-system dell'emulatore, dell'heap e dei singoli thread di ogni processo. Di grande importanza per la rilevazione degli errori è la view chiamata LogCat la quale presenta allo sviluppatore messaggi di log inviati dall'emulatore.

Attraverso la classe Log è possibile programmare l'invio di messaggi con il fine di verificare lo stato dell'applicazione e quindi determinare eventuali errori.

Le applicazioni sviluppate con il plugin possono essere testate in modo realistico attraverso un emulatore: un telefonino virtuale eseguito sul computer. L'emulatore, come si vede nella figura nella pagina seguente, presenta la riproduzione di un telefonino a fianco di una di una tastiera con la quale è possibile interagire con l'emulatore. Tutti i bottoni visibili sono funzionanti e hanno un effetto sull'emulatore.

L'unico limite dell'emulatore consiste nel fatto che non può eseguire vere telefonate. Si possono anche simulare eventi esterni come l'arrivo di un SMS, di una chiamata o il cambiamento della posizione nel mondo del telefono. La presenza del dispositivo GPS è molto importante per applicazioni di tipo location-based. Se non si usa il plugin di Eclipse è possibile comandare l'emulatore attraverso comandi telnet inviati alla porta sulla quale è in esecuzione l'emulatore Nel caso in figura il comando per connettersi è il seguente: telnet localhost 5554.

Figura 3.5: emulatore Android per il test dell'applicazione in fase di sviluppo.

3.4.2 Struttura progetto Android in Eclipse

La struttura di un progetto Android sviluppato con il plugin di Eclipse è qui riportata nell'immagine a fianco. Nella cartella “src” sono contenute tutte le classi java utilizzate per lo sviluppo del progetto. Molto interessante è la cartella chiamata “res”. In essa sono contenute tutte le risorse non costituite da codice sorgente utilizzate dall'applicazione. Esse sono suddivise in cartelle:

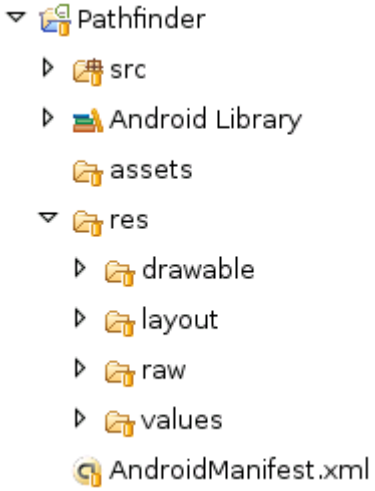


Figura 3.6: struttura di un progetto Android in Eclipse.

- **Drawable** raccoglie tutti gli elementi che possono essere utilizzati come icone o in generale disegnati sullo schermo. Sono accettati tre tipi di oggetti: immagini (di solito in formato png o jpg), definizioni di colori attraverso file xml e immagini di tipo NinePatch (file png cui è associato un xml che ne definisce proprietà riguardanti la deformazione che può subire).
- **Layout** contiene tutti i file per la definizione delle interfacce utente delle differenti activity specificate in formato xml. Ogni file contiene la definizione di una schermata o di una sua parte. Essi sono associati alla corrispondente activity attraverso l'invocazione del metodo Activity.setContentView() il quale prende come parametro un riferimento all'xml desiderato. Attraverso l'utilizzo dei tag propri di xml è possibile creare una struttura ad albero che rappresenta l'interfaccia grafica. A ogni elemento proprio di un'interfaccia corrisponde un tag. Ad esempio esistono i tag <TextView> e <Button> ma anche elementi che ne raggruppano di altri al loro interno come <LinearLayout>, ordinandoli secondo un criterio prestabilito. Per l'elenco completo degli elementi che possono essere utilizzati nella definizione di un layout e dei corrispondenti tag xml consultare l'indirizzo: http://code.google.com/android/devel/ui/layout.html.
- **Values** raccoglie valori semplici che sono utilizzati da molti elementi all'interno dell'applicazione. Si possono inserire tre tipi di oggetti: stringhe definizioni di colori e di misure di lunghezza.

Se si usa il plugin di Eclipse tutte le classi sviluppate e tutte le risorse saranno automaticamente raccolte in un singolo file di estensione .apk (Android PacKage). Esso sarà l'unico file necessario per l'installazione dell'applicazione su un emulatore come quello descritto in precedenza.

Questa soluzione rende estremamente semplice la condivisione di programmi. Prima di essere inseriti nel file .apk tutte le risorse utilizzate dall'utente e inserite nella cartella “rsc” saranno prima compilate dal compilatore aapt (Android Asset Packaging Tool) in file binari. Ancora una volta ADT si rivela estremamente comodo giacché invoca in modo automatico il tool in fase di deploy dell'applicazione.

L'ultimo file rimasto da descrivere è quello chiamato AndroidManifest.xml. Esso deve essere specificato in ogni programma e fornisce informazioni di carattere generale. Per esempio devono qui essere dichiarati le Activity e i Service utilizzati, i permessi che devono essere garantiti all'applicazione per l'esecuzione dei suoi compiti, oppure riferimenti a librerie necessarie per specifiche funzionalità. Sono qui definiti anche gli IntentFilter spiegati nello specifico nei paragrafi successivi.

Parte 2: MobiMash

4 Architettura Framework

4.1 L'architettura

Lo scopo di questo lavoro è portare il concetto di end-user development, in altre parole dare all'utente la possibilità di creare programmi e quindi avvicinarsi al ruolo di sviluppatore, in ambito mobile. L'utente sarà quindi in grado di comporre i propri mashup selezionando semplici componenti grafici. L'ambiente di sviluppo supporta fondamentalmente due modi di integrare servizi; il primo riguarda un mashup di dati, che presi diversi fornitori di informazione espone in modo omogeneo il risultato, il secondo è un "view mashup" grazie al quale è possibile visualizzare i dati attraverso diversi servizi forniti da terze parti.

Il framework che andremo a presentare raccoglie tutte queste caratteristiche e il suo nome è **MobiMash**, che deriva dalla fusione delle parole Mobile e Mashup; termini chiave del nostro progetto.

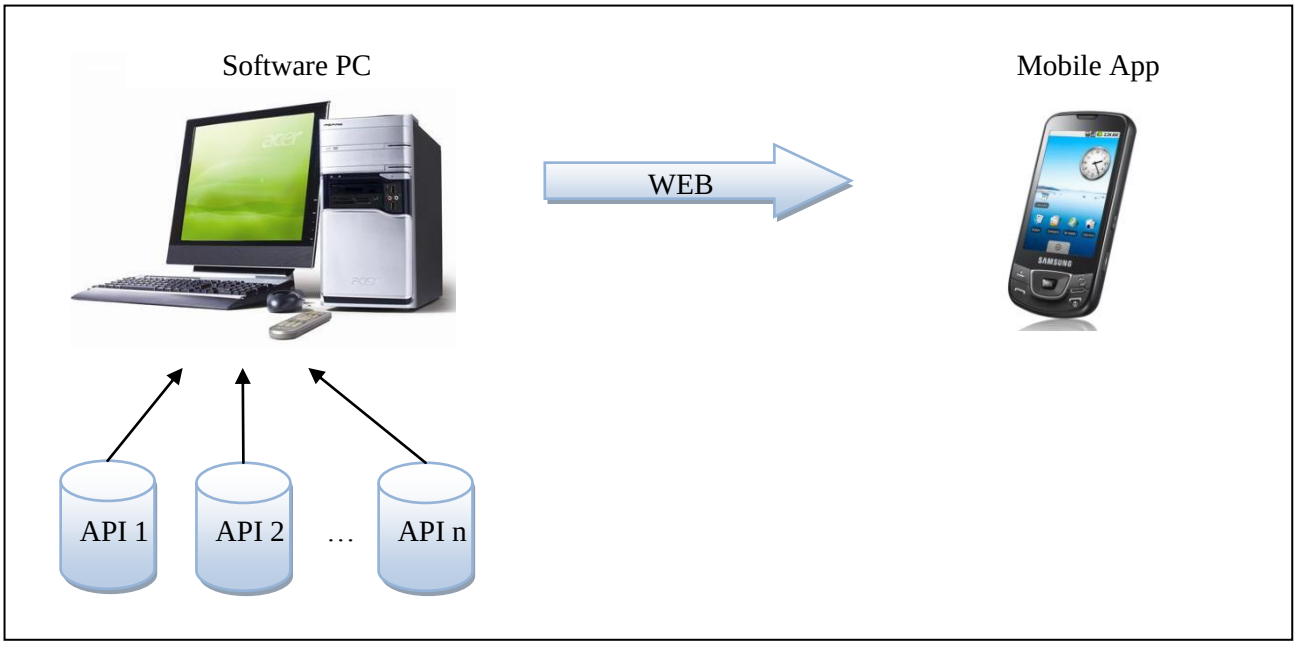


Figura 4.1: architettura fisica framework MobiMash.

L'architettura è divisa in due applicazioni, quella desktop per la definizione della struttura di ciò che si visualizzerà su mobile e l'applicazione mobile vera e propria che permetterà all'utente di fruire dei contenuti.

Questa scelta è stata dettata dal fatto che i dispositivi mobile non sono sufficientemente potenti, sia dal punto di vista hardware sia software, da poter supportare un elevato carico di operazioni pertanto si è introdotto nell'architettura un server che si occupa delle operazioni più onerose. Un'altra motivazione dell'aggiunta di un'applicazione lato server è di permettere all'utente di maneggiare i componenti grafici in uno schermo più ampio e di avere un dispositivo di input più veloce come ad esempio la tastiera. In parole povere l'architettura fisica sulla quale il framework si appoggia, altro non è che una semplice architettura client-server; nella quale i singoli dispositivi mobile svolgono il ruolo di client.

4.2 MobiMash (lato desktop)

MobiMash desktop ha lo scopo di creare dei file di setting, che saranno poi letti dall'applicazione mobile, la quale si adatterà in base al loro contenuto. L'applicazione desktop avrà una serie di servizi pre-registrati che esporranno i contenuti presi dalle varie API. Le sorgenti sono tutte raccolte all'interno di un repository ed ordinate per categorie ad esempio sotto la categoria "Eventi Musicali" vi potrebbero essere le API più famose che forniscono questo tipo di servizio (ad esempio Last.fm, Eventful, Upcoming). L'utente-sviluppatore potrà quindi scegliere il dominio dell'applicazione che andrà a creare, selezionando da una lista i componenti corrispondenti.

Per ora il sistema supporta solo sorgenti dati con una particolare struttura, ovvero file XML in cui il contenuto è all'interno dei tag e non come valori degli attributi (ad esempio JSON o file XML il cui contenuto è specificato come valore degli attributi). Questo vincolo può essere facilmente risolto lato desktop trasformando le sorgenti in ingresso in una struttura ad albero con contenuto sui nodi.

Le operazioni che l'utente può eseguire per estrapolare i dati che saranno poi visualizzati nell'applicazione mobile, vanno ad agire non sulla singola istanza del file contenente la sorgente dati, ma sulla sua struttura così evitando la ridondanza dei dati, poiché le modifiche sulla struttura descrittiva si ripercuotono sull'intera istanza.

Per ogni dominio applicativo il software fornisce la possibilità di "taggare" gli elementi dell'albero che descrive la struttura dati. Tramite un semplice drag&drop si va ad affiancare una stella rossa per selezionare gli elementi che andranno a comporre la lista, in blu per il dettaglio, in arancione per i filtri e in verde per le views (vedi paragrafo 4.3 su lista, dettaglio, filtri e view). Grazie a questo sistema di Tag e drag&drop l'applicazione risulta molto semplice ed intuitiva anche per utenti che non sono necessariamente sviluppatori.

Cliccando su ogni singola stella, si accede ad una schermata che permette di definire il ruolo che quel tag assumerà nell'applicazione mobile o di specificare meglio altre impostazioni, se l'utente non dovesse specificare nessuna impostazione, sarà attribuito un valore di default all'elemento in base all'ordine di aggiunta delle stelle agli elementi descrittivi della struttura dati.

Vediamo ora un esempio, come mostrato in figura 4.2 l'elemento dell'albero "title" sarà visualizzato sia nella lista che nel dettaglio, quindi cliccando sulla stella blu si potrà scegliere se il contenuto di "title" sarà visualizzato come intestazione nel dettaglio o come semplice elemento al suo interno. L'unica eccezione è l'elemento filtro che può essere applicato sia ad un singolo tag della struttura sia a livello di applicazione.

L'ultima fase di design consiste nel cliccare il bottone "Crea Mobile App", con il quale il software in background creerà dei file di configurazione che saranno poi letti dall'applicazione mobile al fine di auto crearsi. I file generati sono:

- **category.xml**: che conterrà la lista delle sorgenti dati selezionate, e per ognuna il collegamento al corrispettivo file setting.xml e data.xml.
- **setting.xml**: il file di setting conterrà i filtri le views e le impostazioni per la creazione automatica dell'interfaccia grafica mobile, come icone, layout e sfondi.
- **data.xml**: conterrà i dati veri e propri che andranno a riempire i contenuti dell'applicazione. Al file data sono stati aggiungi degli attributi che permetteranno il famoso binding "azione-evento" con il file di setting.

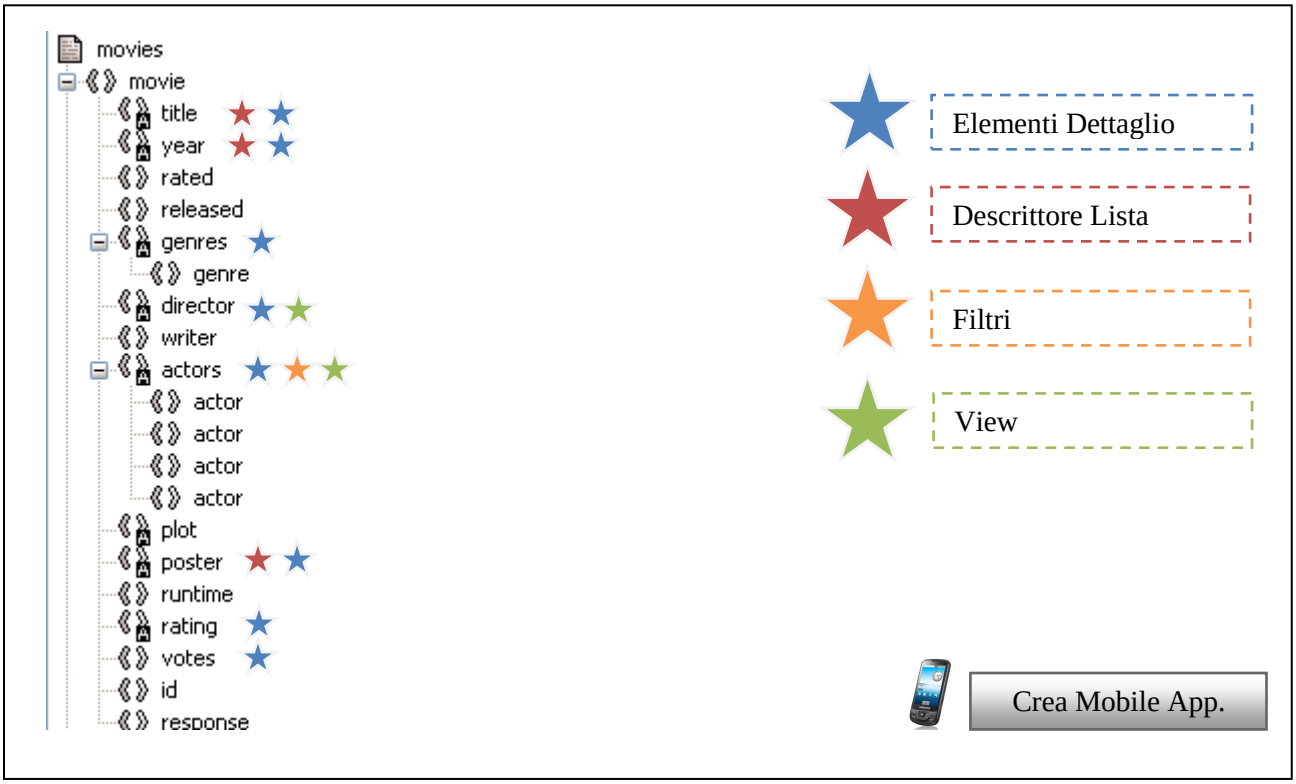


Figura 4.2: applicazione MobiMash lato desktop.

4.3 MobiMash (lato mobile)

MobiMash lato mobile è un'applicazione che gira su sistema operativo per cellulari Android, essa ha la peculiarità di essere versatile ed adattarsi a qualsiasi dominio, inoltre offre la possibilità sia di creare mashup di dati, unendo i risultati provenienti da diverse sorgenti, sia di estendere le informazione visualizzate richiamando una ricerca sulle API più utilizzate come Youtube, Flickr, Twitter e Wikipedia.

Analizzando diverse applicazioni di informazione trovate sulla rete, ci siamo resi conto che la maggior parte di queste sono strutturate nella stessa maniera definendo così una costante da poter sfruttare per la creazione automatica di applicazioni . Tale struttura può essere riassunta in due schermate, la lista contenente tutti i risultati di una ricerca, e il dettaglio relativo ad ogni elemento della lista. Prendiamo ad esempio l'applicazione per vedere gli ultimi film in uscita nelle sale cinematografiche, questa avrà la lista con tutti i film disponibili e per ognuno la locandina con la trama e varie informazioni di contorno.

Come si vede in figura 4.3 gli elementi della lista hanno a loro volta una struttura fissa formata da un'immagine, un titolo e un sottotitolo. Questa è una costante che abbiamo imposto, ma chiaramente si possono aggiungere o togliere elementi in base alle esigenze dell'utente nella costruzione della propria applicazione. Il dettaglio ha una struttura meno rigida dato che a priori non si può sapere il numero di elementi che lo compongono, quindi sarà descritto da una lista di elementi, in cui ognuno di questi avrà un'intestazione e un contenuto. L'utente-sviluppatore avrà la possibilità di scegliere quali contenuti e in che ordine visualizzarli grazie all'applicazione desktop descritta nel paragrafo precedente.

La necessità di trovare delle costanti e quindi di mostrare i contenuti in una determinata maniera, nasce dal fatto che MobiMash è un'applicazione che si adatta ad ogni contesto, quindi se come dominio applicativo avessimo avuto la lista dei concerti o le news del giorno, l'applicazione si sarebbe adattata ai contenuti mantenendo la stessa struttura di base.

Figura 4.3: descrizione Lista, Dettaglio

La composizione delle schermate di figura 4.3 avviene eseguendo il parsing dei file data.xml e setting.xml, nel capitolo successivo descriveremo in dettaglio la fase di implementazione di questo meccanismo.

In figura 4.4 si mostra la schermata iniziale dell'applicazione mobile. Essa è creata dinamicamente andando a leggere il contenuto del file category.xml creato dall'applicazione desktop, in cui sono descritte tutte le sorgenti dati raggruppate per categoria. Nell'esempio abbiamo due categorie di sorgenti "Movies" e "Concert" e all'interno di ogni singola categoria vi è la lista di sorgenti che è possibile selezionare compiendo in questo modo il mashup di dati, che unirà in un'unica lista i risultati proveniente dalle diverse sorgenti.

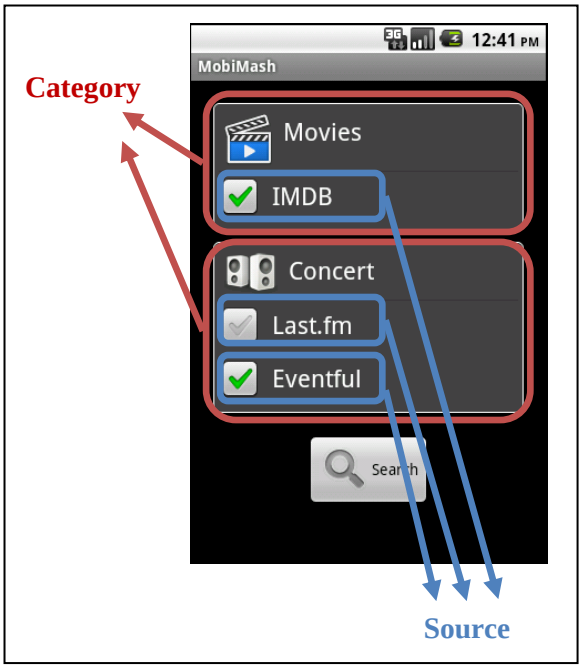


Figura 4.4: selezione sorgenti.

Va fatto notare che ci sono due livelli di selezione delle sorgenti, uno a monte effettuato all'interno di MobiMash lato desktop, dove l'utente-sviluppatore seleziona un sottoinsieme di sorgenti per creare una specifica applicazione mobile, mentre la seconda selezione è fatta dall'utente mobile che a partire dal sottoinsieme che gli arriva dal programma desktop sceglie solo le fonti che per lui sono rilevanti in quel momento.

I filtri, che in fase di design si sono indicati con una stella arancione, saranno visualizzati quando l'utente si trova sulla lista e clicca sul tasto "menù". A tale proposito ricordiamo che tutti i cellulari con sistema operativo Android hanno due tasti di default; il tasto "menù" per accedere ad eventuali impostazione proprie delle applicazioni, e il tasto "back" che permette di ritornare alle schermate precedenti. Come avviene per l'intera applicazione e per tutte le schermate che la compongono, anche il menù per la selezione dei filtri viene costruito dinamicamente a partire dai file setting.xml generati dall'applicazione desktop.

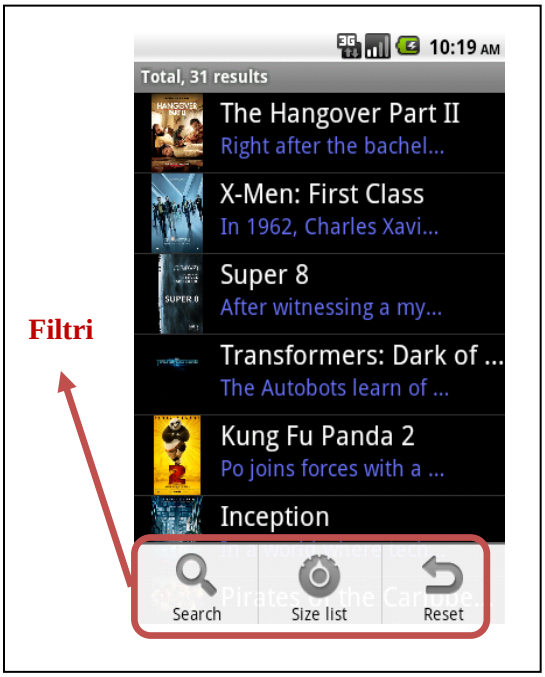


Figura 4.5: Filtri su lista

I filtri per ora disponibili sono:

- Size list, che visualizza solo i primi n elementi della lista.
- Search, che cerca una parola chiave all'interno di tutte le informazioni disponibili.
- Date, che sarà collegato ad un elemento della struttura dati che contiene una data e come risultato restituisce solo gli elementi della lista che hanno la data compresa tra un range.
- Reset, resetta l'effetto di tutti i filtri riportando la lista di default.

In futuro si potranno aggiungere filtri che interagiscono con il GPS e quindi eseguono una selezione sulla lista in base alla distanza, restituendo solo i risultati che rientrano in un certo raggio. Ad esempio prendendo un caso d'uso in cui si è scelto come dominio applicativo i concerti si potrà filtrare la lista visualizzando solo quelli nel raggio di 10Km dall'attuale posizione dello smartphone.

L'ultima caratteristica dell'applicazione MobiMash lato mobile che andremo a descrivere riguarda la selezione delle views sugli elementi del dettaglio, e più precisamente su quegli elementi del dettaglio che sono stati "taggati" nell'applicazione desktop con una stella verde (vedi figura 4.2). Quest'ultimi saranno visualizzati come mostrato in figura 4.5, schermata creata dinamicamente prendendo le informazioni sia dal file data.xml sia dal file setting.xml

L'utente-sviluppatore lato desktop potrà scegliere una configurazione di default per le varie API di visualizzazione (Youtube, Flickr, Twitter e Wikipedia) in base ad una logica da lui definita, ad esempio se il tag fosse relativo al nome di un pittore avrà più senso selezionare di default Flickr (che mostrerà le immagini dei quadri del pittore) e Wikipedia (dove si potrà vedere la scheda del pittore), invece potrebbe sembrare meno utile vedere su Youtube video relativi ad un pittore o commenti su Twitter. Nonostante questi settaggi di default provenienti dall'applicazione desktop, lasciamo la libertà all'utente mobile di selezionare o deselezionare tutte e quattro le view disponibili.

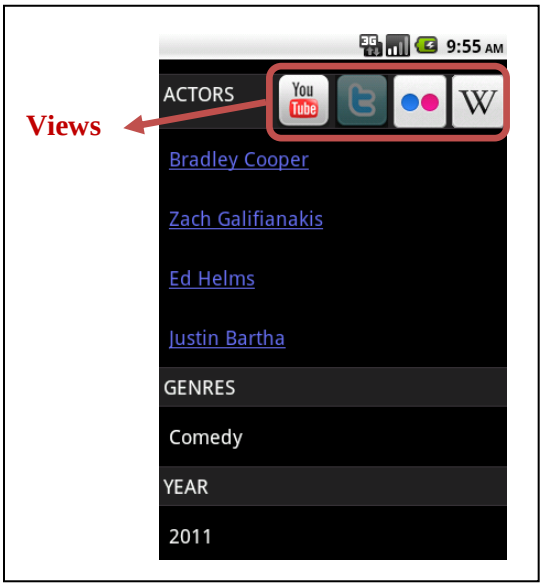


Figura 4.6: selezione views.

L'esempio di figura 4.5 è relativo all'applicazione che mostra i film nelle sale cinematografiche, quindi una volta selezionate le views desiderate, e cliccando su ogni attore si aprirà la

schermata contenente le view che hanno come termine di ricerca il nome dell'attore. Per ora le views implementate sono quattro ma in futuro si potranno aggiungere facilmente altre API di visualizzazione al fine di arricchire l'esperieza utente.

In figura 4.7 vediamo un esempio del risultato delle views, ottenute cliccando su un attore considerando l'applicazione sui film. Commentiamo ora le funzionalità che offre ogni singola view e come l'utente interagisce con esse:

- **YouTube**, richiama le API ufficiali di YouTube creando una lista con i primi 10 video in relazione alla keyword passata come parametro (ad esempio il nome dell'attore). Qui, l'utente può cliccare un elemento della lista che richiama l'applicazione di YouTube presente di default su tutti i dispositivi Android.
- **Flickr**, visualizza una galleria di immagini scaricate attraverso le API messe a disposizione del sito e, cliccando su ogni singola immagine l'utente potrà vederla ingrandita al centro dello schermo.
- **Twitter**, con lo stesso stile a lista di youtube, vengono visualizzati i primi venti commenti scaricati tramite le API di Twitter sempre inerenti alla keyword. Spesso i commenti di Twitter includono link a pagine web, questi sono cliccabili e aprono la pagina con il browser di default dello smartphone Android.
- **Wikipedia**, è richiamata la pagina di Wikipedia settando nel link la keyword corrispondente alla selezione effettuata nella schermata di dettaglio. Il tutto è possibile grazie all'utilizzo di un componente messo a disposizione da Android chiamato WebView, un semplice browser integrato nell'applicazione.

Per passare da una view all'altra si è implementata un'interfaccia molto smart che consiste nel far scorrere la schermata corrente a destra o a sinistra in modo da caricare la view adiacente.

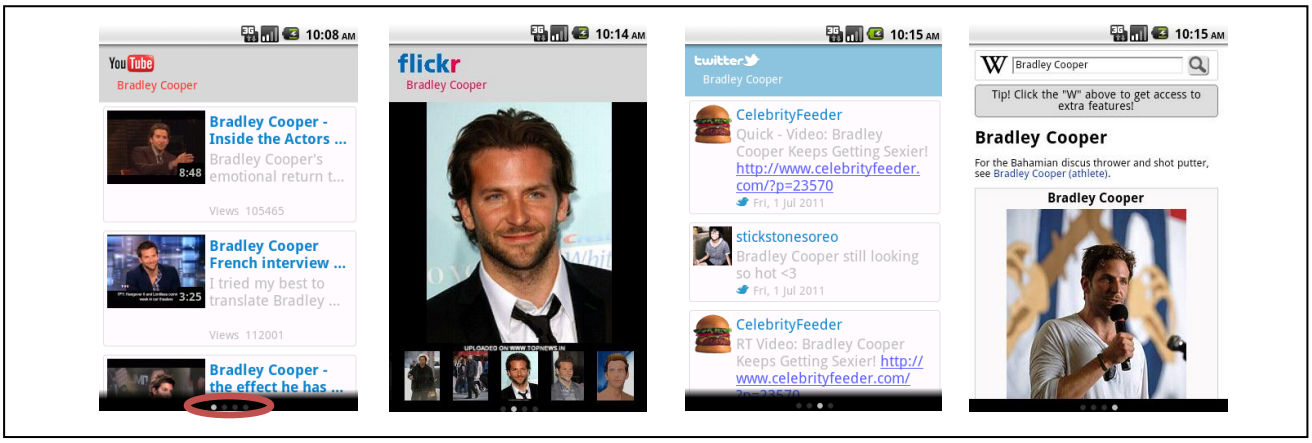


Figura 4.7: esempio di views risultanti dalla selezione di un'attore.

In questo capitolo si è descritta l'architettura del framework MobiMash, analizzando in dettaglio sia l'applicazione desktop che serve a costruire il design, sia l'applicazione mobile che sfruttando l'output fornito dall'applicazione desktop mette a disposizione dell'utente un'interfaccia grafica user friendly ricca di opzioni e personalizzazioni. MobiMash è un applicazione che si adatta in base ai domini e da la possibilità di creare mashup.Nel prossimo capito, riguardante gli aspetti implementativi, discuremo più approfonditamente su come comunicano MobiMash desktop e MobiMash mobile attraverso i file di setting.

5 Implementazione

Con il precedente capitolo abbiamo dato un'ampia panoramica dell'applicazione da un punto di vista progettuale e di com'è suddivisa la sua architettura fisica, in questo capitolo entreremo più in dettaglio nello spiegare come e perché sono state implementate le funzionalità principali al fine di far comprendere al lettore i meccanismi di comunicazione all'interno del software.

Il capitolo si compone di quattro sezioni:

- **File di configurazione**: in questa sezione si definiscono gli aspetti implementativi dei file XML generati dall'applicazione desktop e i binding che li legano al fine di costruire l'applicazione mobile.
- **Aspetti principali dell'implementazione**: qui si analizzeranno le scelte nel realizzare MobiMash lato mobile, illustrando lo schema UML dell'intero progetto con la spiegazione di alcune classi di maggior rilievo.
- **Prestazioni**: in tale sezione si elencheranno i risultati ottenuti da simulazioni dell'applicazione sia recuperando dati direttamente dal dispositivo mobile sia utilizzando programmi specifici nello svolgere tali raccolte di dati.
- **Validazione e sperimentazione**: questa sezione raccoglie le impressioni degli utenti che hanno testato l'applicazione attraverso un semplice mock-up sviluppato in Power Point, e in seguito compilato un breve questionario.

5.1 File di configurazione

I file di configurazione che andremo a descrivere in questa sezione sono il nucleo di tutto il framework MobiMash. Ricordiamo che questi file raccolgono e descrivono sia il layout che la logica dell'applicazione mobile che l'utente-sviluppatore avrà composto grazie all'applicazione desktop.

I file di configurazione sono:

- **category.xml**: che conterrà la lista delle sorgenti dati selezionate, e per ognuna il collegamento al corrispettivo file setting.xml e data.xml.
- **setting.xml**: il file di setting conterrà i filtri le views e le impostazioni per la creazione automatica dell'interfaccia grafica mobile, come icone, layout e sfondi.
- **data.xml**: conterrà i dati veri e propri che andranno a riempire i contenuti dell'applicazione. Al file data.xml sono stati aggiungi degli attributi che permetteranno il famoso binding "azione-evento" con il file di setting.

Andremo ora ad analizzare con un linguaggio più tecnico, ogni singolo file xml, aiutandoci con una serie di screenshot che mostrano tutte le opzioni disponibili.

category.xml

Il file category.xml è formato da un elemento root <categories> che raccoglie n elementi <category> ognuno dei quali rappresenta una singola categoria che riunisce i servizi di quel tipo, l'attributo "name" contiene il nome della categoria mentre "icon" contiene il riferimento ad un immagine che rappresenta quella categoria (ad esempio per la categoria movies l'icona corrispondente potrebbe essere un ciak).

Ogni elemento <category> include una serie di <source> che rappresentano i servizi selezionati dall'utente nell'applicazione lato desktop (ad esempio IMDB, Last.fm, Eventful); inoltre ciascuna singola source possiede due attributi quello "data" contenente il link alle informazioni vere e proprie e l'attributo "setting" contenente il link al file che definisce le opzioni dell'applicazione e il suo layout.

Vediamo ora la proiezione del file sull'applicazione mobile; infatti, lanciando MobiMash mobile, il software scarica tale file ed esegue il parsing creando in automatico la prima schermata che permette all'utente la possibilità di scegliere una sotto selezione delle source mostrate.

```
<?xml version="1.0" encoding="utf-8"?>
<categories>
    <category name="Movies" icon="https://sites.google.com/site/frameworkandroid/movie.png">
        <source data="http://sites.google.com/site/frameworkandroid/movies.xml"
            setting="http://sites.google.com/site/frameworkandroid/movies_setting.xml">IMDB</source>
    </category>
    <category name="Concert" icon="https://sites.google.com/site/frameworkandroid/speakers.png">
        <source data="http://sites.google.com/site/frameworkandroid/lastfm.xml"
            setting="http://sites.google.com/site/frameworkandroid/lastfm_setting.xml">Last.fm</source>
        <source data="http://sites.google.com/site/frameworkandroid/lastfm.xml"
            setting="http://sites.google.com/site/frameworkandroid/lastfm_setting.xml">Eventful</source>
    </category>
</categories>
```

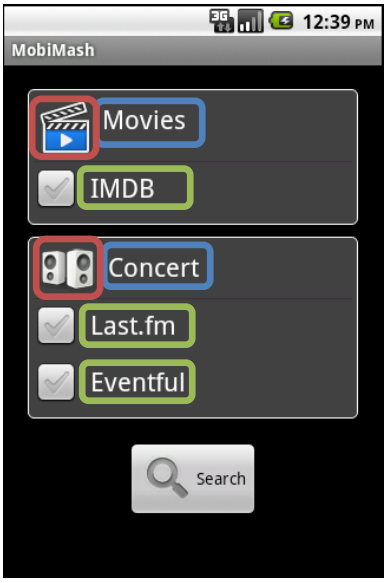


Figura 5.1: category.xml e sua proiezione su mobile.

setting.xml

Il file setting.xml ha come root il tag "setting" esso include quattro elementi fondamentali per la creazione dell'applicazione mobile che sono:

- **<base_element>**, che indica l'elemento di base su cui sarà iterato il file data.xml.
- **<list>**, che contiene i sotto elementi <title>, <subtitle> e <image>, il cui contenuto rappresenta il valore dell'attributo "class" associato a quegli elementi del file data.xml che l'utente ha ritenuto necessari per popolare il contenuto della lista. Con questa tecnica riusciamo a selezionare i veri contenuti da rappresentare nella lista, creando un join tra il file setting.xml e data.xml.
  In figura 5.2 presentiamo un chiaro esempio del flusso con cui la schermata contenente la lista è creata dai file di configurazione.

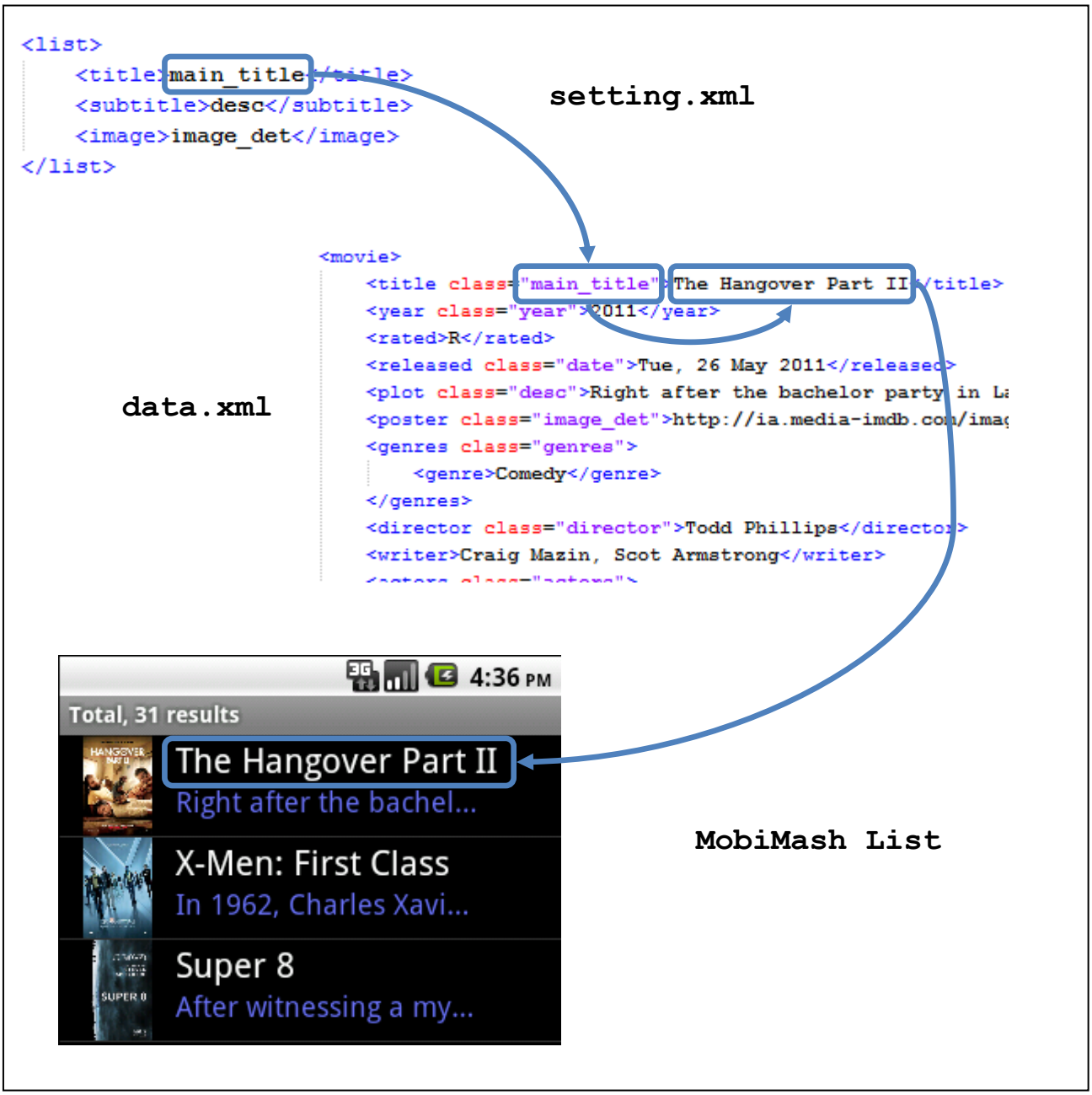


Figura 5.2: meccanismo di binding tra i file setting.xml e data.xml.

Dando una breve descrizione dell'immagine riportata sopra è possibile notare che dal contenuto dell'elemento <title> si crea un legame con il valore dell'attributo degli elementi nel file data.xml, dal quale si preleva l'informazione utile da visualizzare in un secondo momento all'interno della lista. La scelta di creare il join utilizzando l'attributo "class" piuttosto che il nome del tag è dovuta al fatto che a priori non si è a conoscenza dell'intera struttura del file data.xml, che potrebbe contenere al suo interno più tag con lo stesso nome, così da non creare una corrispondenza uno a uno tra gli elementi. Per fissare meglio l'idea sopra esposta conviene introdurre un piccolo esempio: nel caso in cui il tag <title> dovesse identificare il titolo di un libro e allo stesso tempo vi fosse un altro tag <title> rappresentante il titolo di un capitolo, a un qualsiasi livello di annidamento, parsando il file si presenterebbero problemi su quale dei due tag selezionare al fine di accedere all'informazione utile da visualizzare nell'applicazione mobile.

- **<detail>**, che contiene gli elementi che andranno a rappresentare la schermata di dettaglio dell'applicazione. Tale tag sarà composto da un elemento <photo> e da n elementi di tipo <item>. Quest'ultimi sono caratterizzati da tre attributi:
    - "priority": definisce la posizione all'interno del dettaglio che ogni elemento assumerà, l'attributo è obbligatorio e i suoi valori sono numeri interi che vanno da 0 a n, dove con priority 0 si indica il titolo del dettaglio.
    - "action": definisce i servizi aggiuntivi legati a un elemento all'interno del file data.xml con la stessa tecnica di join mostrata per la lista. Tale attributo è opzionale e i possibili valori possono essere *youtube, twitter, flickr e wiki* separati dal carattere pipe "|".
    - "all_child": descrive se l'elemento da parsare è semplice, cioè contenente solo testo o se è di tipo complesso, quindi avente all'interno altri tag. Se tale attributo è presente, allora il suo valore indica il nome dei tag figli da analizzare nel file data.xml, mentre il valore all'interno del tag <item> andrà a definire il tag padre da cui far partire il parsing.

  Come si può notare in figura 5.3 i valori degli elementi <item> sono utilizzati con il ruolo di header, linea di colore verde, mentre l'informazione analizzata nel file data.xml è mostrata come contenuto nel dettaglio, linea di colore rosso.

- **<filter>**, che contiene i tag rappresentanti le tre tipologie di filtro fino ad ora implementate, che compariranno nel menù della lista. I filtri sono:
    - **Find**: tale filtro permette di compiere una ricerca su tutte le informazioni analizzate attraverso una keyword. Il tag è obbligatorio e se si vuole attivare, dovrà avere "true" come valore, o "false" in caso contrario.
    - **Size**: tale filtro permette di eseguire una selezione sul numero degli elementi da visualizzare nella lista, come nel caso precedente il tag è obbligatorio e se si vuole attivare, dovrà avere "true" come valore, o "false" in caso contrario.
    - **Date**: il filtro date, esegue una selezione della lista in base ad un range di date, per capire su quale elemento eseguire la selezione si ha come valore del tag <date> un identificativo che servirà poi per effettuare il join con il quale recuperare la data vera e propria. Il filtro non è obbligatorio e ha un attributo "format" che indica il pattern con cui è rappresentata la data. Per ora i valori che "format" può assumere sono RFC-822 (pattern EEE, dd MMM yyyy HH:mm:ss) e ISO-8601 (pattern yyyy-MM-dd HH:mm:ss).

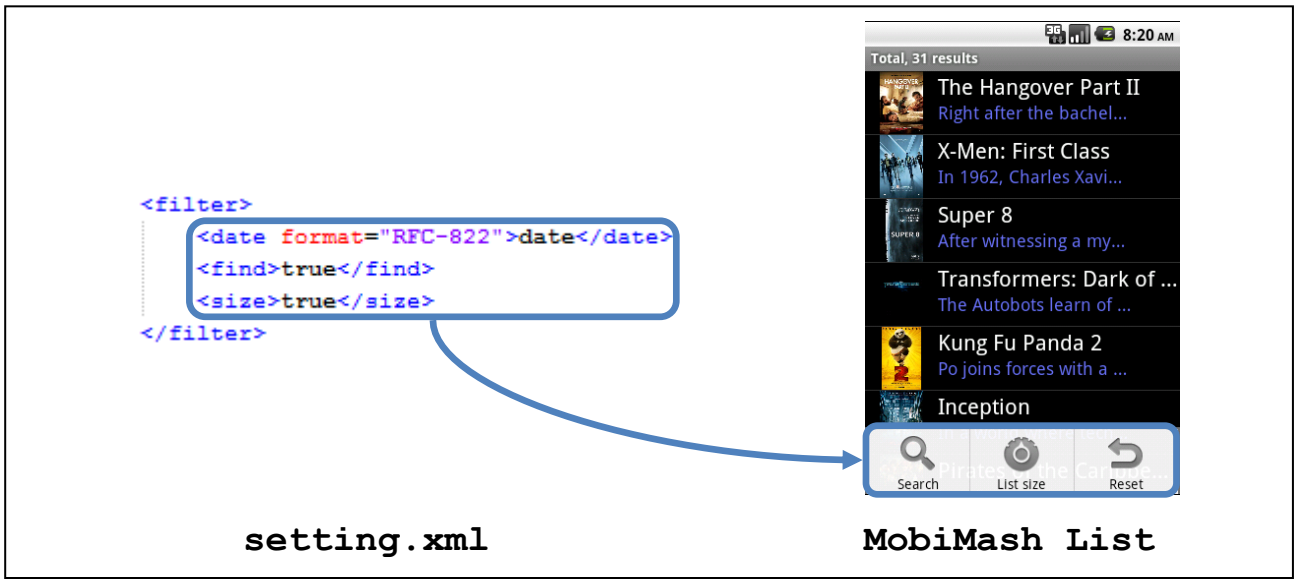


Figura 5.3: filtri sulla lista.

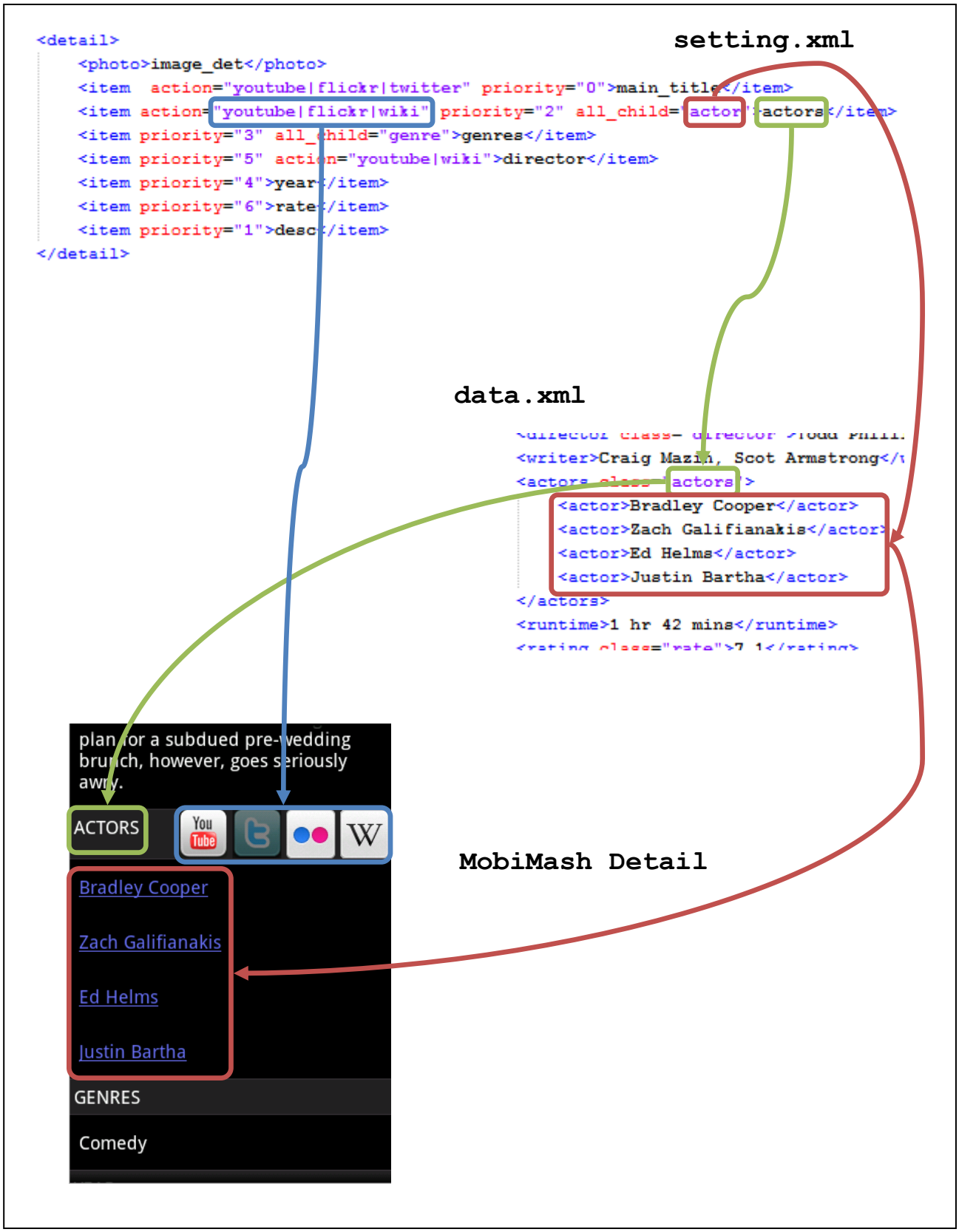


Figura 5.4: meccanismo di binding tra i file setting.xml e data.xml.

5.2 Aspetti principali dell'implementazione

In questa sezione, come spiegato sopra, si approfondisce la struttura dell'intero progetto dal punto di vista dell'implementazione facendo luce sulle classi sviluppate senza entrare nel merito del codice Java, poiché è un aspetto superfluo e di poca importanza al fine di comprendere la logica dell'applicazione. L'applicazione è divisa in quattro package ognuno con un ruolo ben preciso:

- com.polimi.mobimash.handler: contiene l'insieme degli handler SAX[7] per il parsing dei file XML generati dall'applicazione desktop.
- com.polimi.mobimash.bean: insieme dei Java Bean popolati dagli handler e quindi rappresentanti il contenuto dei file XML in formato Java, così da permettere alle Activity Android di poter realizzare l'interfaccia utente e le funzionalità necessarie per rendere utilizzabile l'applicazione.
- com.polimi.mobimash.android: package contenente l'insieme delle Activity Android, descritte in dettaglio nel capitolo 3, ed altre classi di supporto come ad esempio gli Adapter per la creazione dinamica delle liste.
- com.polimi.mobimash.utils: contiene un gruppo di classi di appoggio per operazioni secondarie come il confronto tra date, creazione di date a partire da un pattern, download di immagini in modalità lazy load e gestione della connessione con il server sul quale è eseguita l'applicazione MobiMash lato desktop.

Come spiegato nell'elenco puntato riportato sopra, gli handler utilizzano **SAX**, tecnologia che implementa cinque interfacce di programmazione, o per meglio dire API, di cui quattro, di tipo Handler, si occupano dell'operazione di parsing, e una, di tipo Reader, dell'operazione di lettura del documento XML. Ogni interfaccia gestisce le proprie operazioni o può far in modo che il programmatore le gestisca attraverso i metodi, funzioni che operano su un oggetto specifico dichiarato attraverso l'implementazione dell'interfaccia stessa. XMLReader è l'interfaccia di tipo Reader che si occupa di processare e leggere il documento. Durante la lettura del documento XML, il parser SAX è in grado di rilevare alcuni eventi, che corrispondono alla lettura di un elemento tipo l'apertura e la chiusura di un documento XML o l'apertura e la chiusura di un tag. numerosi linguaggi di programmazione che permette di leggere ed elaborare dei documenti XML.

Contrariamente al DOM[8], il SAX processa i documenti linea per linea. Il flusso di dati XML è unidirezionale, così che dati cui si è acceduto in precedenza, non possono essere riletti senza la rielaborazione dell'intero documento.

Infatti, questa differenza è il punto chiave della scelta effettuata tra i due strumenti di parsing, poiché ciò che ci serve è analizzare per intero tutto il documento e nello stesso tempo popolare i Java Bean a seconda dei tag aperti e chiusi.

Nelle prossime pagine approfondiremo i primi tre package focalizzandoci maggiormente su quello contenente gli handler SAX, poiché è proprio a questo livello che avviene tutta la logica di lettura ed interpretazione dei file XML che portano alla creazione dell'applicazione voluta dall'utente.

[7] SAX: Simple API for XML
[8] DOM: Document Object Model

Iniziamo ad analizzare il package contenente gli handler e i Java Bean, ma prima di addentrarci nei particolari è riportato di seguito lo schema UML in modo da capire le classi coinvolte.

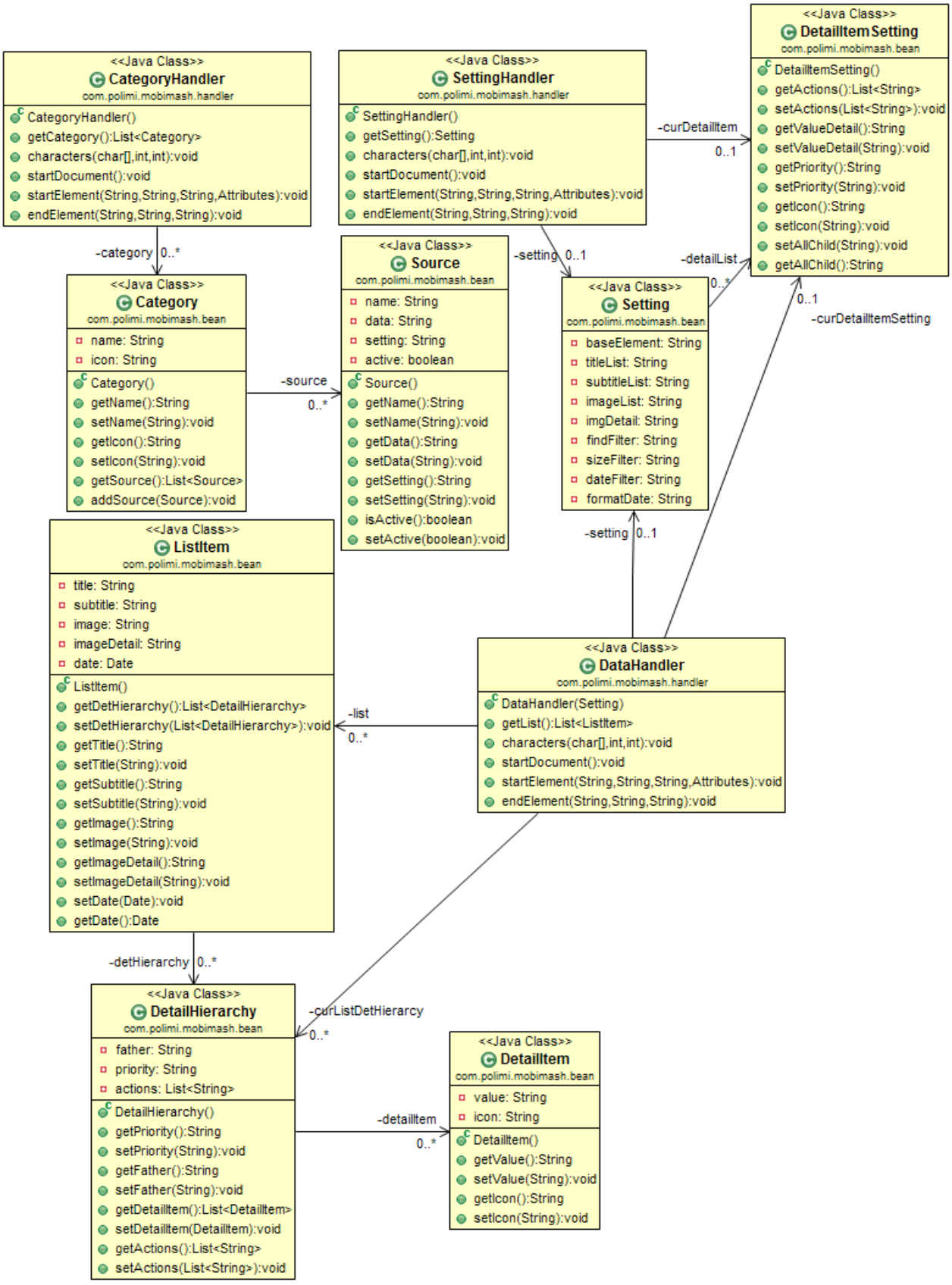


Figura 5.5: schema UML del package com.polimi.mobimash.handler e com.polimi.mobimash.bean.

Com'è possibile notare dallo schema sopra riportato si sono creati tre handler uno per ogni tipologia di file XML prodotti dall'applicazione desktop. Tali file (category.xml, setting.xml, data.xml) sono stati descritti in maniera dettagliata nella sezione precedente in modo che a questo livello i concetti siano chiari per il lettore.

Gli handler creati a tale scopo sono:

- **CategoryHandler**: ha il compito di analizzare il file category.xml, che si occupa di definire quali servizi, raggruppati per categoria, l'utente ha selezionato e dai quali vuole recuperare i dati da vedere in un secondo momento nell'applicazione mobile.
  Contemporaneamente alla scansione del file, l'handler popola i corrispondenti JavaBean Category e Source, che sono legati dal fatto che all'interno di una stessa categoria é possibile avere uno o più servizi (sorgenti) che forniscono i contenuti richiesti dall'utente.
  Ad esempio, nel caso di concerti in una determinata città l'utente avrà la possibilità di decidere quali servizi (sorgenti) includere per tale categoria e, possibili esempi di servizi pre-registrati per questo dominio possono essere Last.fm, Eventful o Upcoming.
  I Java Bean sopra citati permettono alle activity Android di creare l'interfaccia iniziale dell'applicazione già vista nel capitolo precedente.
- **SettingHandler**: tale handler si occupa di analizzare il file setting.xml, nel quale si rappresentano tutte le informazioni riguardanti le impostazioni grafiche dell'applicazione mobile ed in più si definisce la possibilità di includere filtri sia su determinati elementi sia sull'intera struttura del file data.xml.
  Come nell'handler precedente, durante il parsing del file si popolano i seguenti Java Bean: Setting e DetailItemSetting. Il primo dei due mantiene tutte le informazioni di base del file analizzato, tra queste vi è l'elemento base della struttura XML sul quale eseguire l'iterazione, i descrittori degli elementi che comporranno la lista, la presenza di quanti e quali filtri associare all'applicazione e per ultimo contiene una lista di DetailItemSetting. Quest'ultima è formata da elementi che descrivono il singolo dettaglio all'interno della schermata dettaglio vera e propria, in tale oggetto si salvano informazioni quali la priorità che questo assume nel dettaglio completo, l'insieme delle views associate ad esso per fornire una visualizzazione personalizzata ed altre informazioni utili alla creazione dell'interfaccia utente
- **DataHandler**: si occupa di compiere il parsing del file data.xml, il quale contiene i dati veri e propri che andranno a popolare l'applicazione lato mobile.
  Tale file è stato modificato, dal file originale ricevuto dalle sorgenti, attraverso l'albero grafico discusso nel capitolo precedente, al fine di garantire il binding con gli elementi definiti nel file di setting.xml. Tale binding è stato spiegato nella prima sezione di questo capitolo, ma per ricordare al lettore il suo significato si richiama in breve tale meccanismo.
  Alla base della creazione automatica dell'applicazione mobile vi è l'aggiunta di attributi agli elementi XML costituenti il file data.xml, in modo da identificare in modo univoco un determinato elemento della struttura. Così facendo dal file setting.xml ci si potrà riferire a tali elementi impostandogli eventuali proprietà sia grafiche sia di logica.

Ora sarà analizzato lo schema UML riguardante il package com.polimi.mobimash.android contenente le Activity Android dedite alla creazione della grafica e all'aggiunta delle funzionalità in base al contenuto dei Java Bean popolati come spiegato sopra con l'utilizzo degli handle.

Per questioni di spazio e comodità le classi java rappresentanti i Java Bean sono stata ridotte eliminando gli attributi e i metodi al loro interno, poiché già presentati nello schema precedente.

Di seguito è riportato lo schema UML.

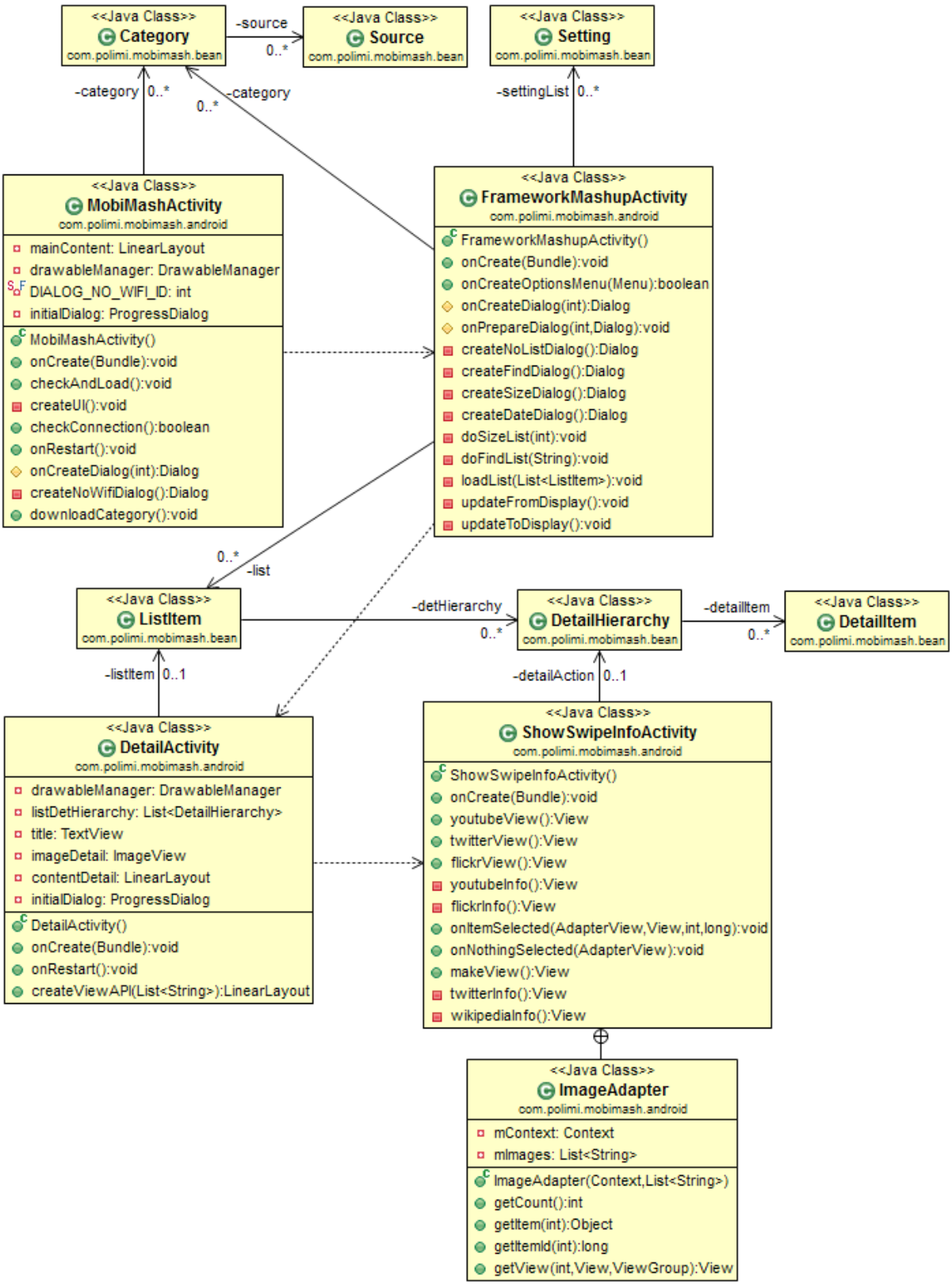


Figura 5.6: schema UML del package com.polimi.mobimash.android.

Per definizione un'activity Android rappresenta ogni singola operazione che l'utente può svolgere all'interno dell'applicazione ed è legata molto al concetto di schermata come descritto nel capitolo 3 riguardante il sistema operativo Android.

Dalla figura 5.2 si nota la presenza di quattro activity che corrispondono alle schermate principali dell'applicazione, che come descritto nel capitolo dell'architettura rappresentano una costante che permette la costruzione di applicazioni su diversi contesti semplicemente modificando i file XML di configurazione tramite l'interfaccia utente di MobiMash lato desktop.

Di seguito è riportata la lista delle activity con la descrizione del loro ruolo.

- **MobiMashActivity**: è l'attività principale con la quale l'applicazione parte e mostra all'utente-sviluppatore la schermata contenente i servizi che forniscono i dati, dando così la possibilità di compiere una successiva scelta delle fonti rispetto a quella già eseguita nell'applicazione lato desktop; così definendo il cosiddetto mashup di dati. Una volta stabiliti i servizi da integrare si passa il controllo all'attività successiva che è FrameworkMashupActivity.
- **FrameworkMashupActivity**: in questa attività avviene il mashup dei dati leggendo il contenuto dei Java Bean Category e Source, che sono stati popolati con i dati presenti nel file category.xml dall'handler CategoryHandler, si avrà accesso ai file di setting.xml e data.xml che sono specificati all'interno di ogni source.
  Quindi si procederà al download di tali file al fine di accedere alle informazioni con le quali creare sia la schermata della lista di risultati ordinati in base alla distanza del singolo item rispetto alla posizione corrente del dispositivo sia di dettaglio.
- **DetailActivity**: questa activity si occupa della creazione della schermata di dettaglio dell'applicazione ordinando gli elementi al suo interno in base al contenuto dei Java Bean DetailItem e DetailItemSetting i cui contenuti sono stati popolati dagli handler DataHandler e SettingHandler.
- **SnowSwipeInfoActivity**: questa è l'activity dove avviene il secondo dei due mashup e cioè quello a livello di views, in cui selezionando il singolo elemento del dettaglio è possibile ottenere informazioni aggiuntivi su di esso attraverso servizi offerti da terze parti, quali ad esempio YouTube, Twitter, Flickr e Wikipedia.
  Il binding che lega il singolo elemento del dettaglio all'insieme di possibili servizi offerti avviene a livello di file setting.xml, in cui per ogni elemento <item> relativo al dettaglio vi è un attributo action il cui valore indica quale servizi sono abilitati nel fornire ulteriori informazioni da mostrare al momento in cui l'applicazione viene testata.
  In questa fase l'utente può ulteriormente definire i binding con i servizi esterni andando a selezionare le icone corrispondenti ai servizi stessi, quindi come nel caso della selezione dei servizi per il recupero dei dati, anche qui l'utente-sviluppatore può agire in due diversi istanti per definire i binding che più preferisce.

Questo è tutto per quanto riguarda l'aspetto implementativo di MobiMash poiché si è voluto fornire solo gli aspetti salienti per non entrare nei dettagli del codice Java. La prossima sezione si occupa di analizzare le prestazioni dell'applicazione mobile prodotta dal framework.

5.3 Prestazioni

In questa sezione ci occupiamo di mettere in luce gli aspetti legati ai tempi di esecuzione di MobiMash lato mobile sia attraverso il programma di test messo a disposizione da Android sia analizzando alcuni dati forniti dal dispositivo stesso durante l'esecuzione dell'applicazione. Le performance sono state valutate su un campione di cinque test per ogni caso d'uso per poi eseguirne la media, e quindi avere dei risultati più accurati.

Per quanto riguarda i dati ottenuti usufruendo del programma di test fornito da Android, si è deciso di eseguire le prove utilizzando due differenti tipi di connessione, cioè wi-fi e 3G che sono ormai accessibili alla maggior parte degli utenti e rappresentano le tecnologie di ultima generazione.

Diamo ora una breve panoramica su queste due forme di connessioni totalmente differenti, per introdurre il lettore a comprendere meglio ciò che i grafici successivi rivelano.

- Nell'ambito della telefonia cellulare, il termine **3G** (acronimo di 3rd Generation) indica le tecnologie e gli standard di terza generazione. Esso sfrutta le antenne per i cellulari, cioè è possibile usarlo ovunque ci sia una copertura telefonica adatta a tale tecnologia e il traffico è pagato direttamente al gestore telefonico.
  I servizi abilitati dalle tecnologie di terza generazione consentono il trasferimento sia di dati "voce" (telefonate digitali) che di dati "non-voce" (ad esempio, download da internet, invio e ricezione di email ed instant messaging anche se la killer application utilizzata come traino dal marketing degli operatori 3G per l'acquisizione di nuova clientela è la videochiamata).
- Il **wi-fi** permette al cellulare di collegarsi ad una rete senza fili, in pratica il cellulare si può collegare ad un access point posto nelle immediate vicinanze (nell'ordine delle decine di metri). E' utile nel caso di una rete wi-fi casalinga, sul luogo di lavoro, o in alcuni luoghi pubblici.

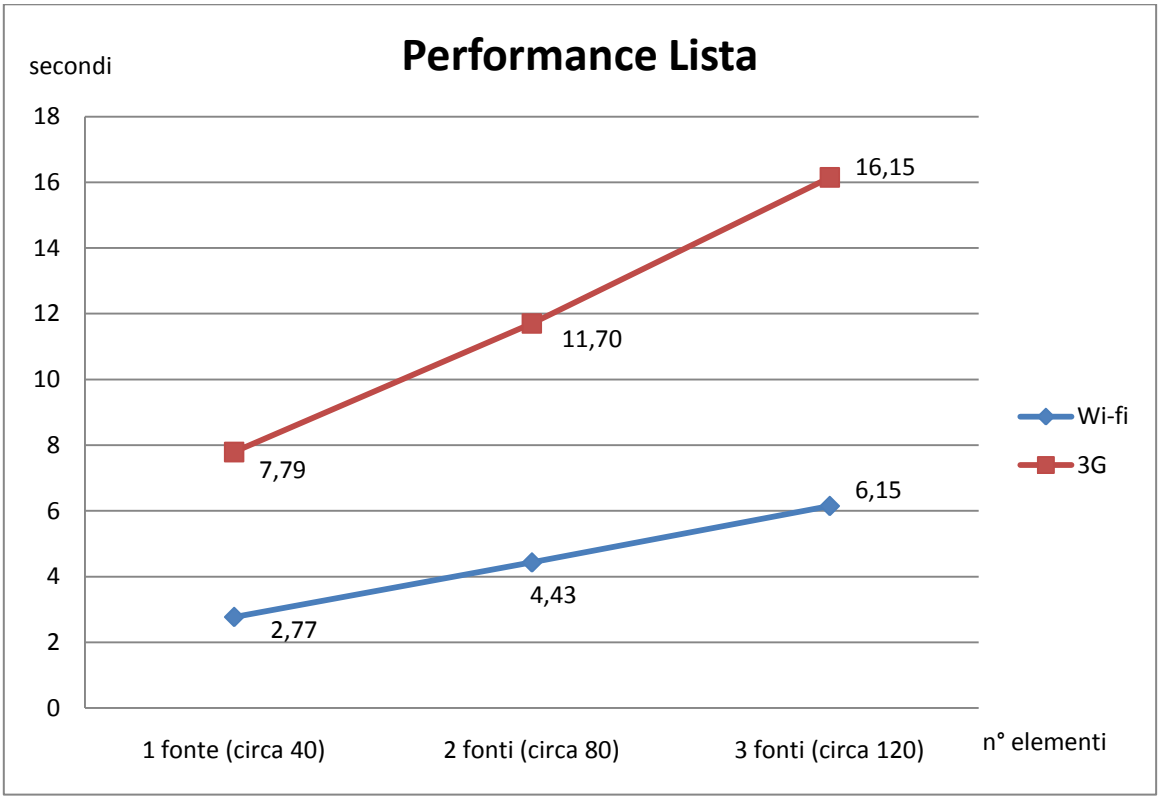


Figura 5.7: grafico delle performance sul caricamento della lista.

La Figura 5.3 mostra il grafico riguardante le performance sul caricamento della lista, in cui l'asse delle ascisse mostra il numero degli elementi che saranno scaricati dai diversi servizi selezionati nella schermata iniziale, mentre sull'asse delle ordinate sono riportati i tempi di download in secondi.

In media un'applicazione che mostra una lista ha non più di 30 elementi, i nostri test sono stati eseguiti con un numero di elementi che varia da 40 a 120, questo per valutare le performance anche in situazioni di stress per l'applicazione.

Com'é facile notare dal grafico dsopra riportato, i tempi crescono linearmente con l'aumentare degli elementi della lista, questo vale per entrambi i tipi di connessione wi-fi e 3G. I tempi di attesa che intercorrono tra la richiesta di ricerca e la visualizzazione della lista corrispondente vanno dai 2 ai 6 secondi circa per la connessione wi-fi e dai 7 ai 16 per la connessione 3G, chiaramente questo è dovuto alla maggiore velocità della connessione wi-fi rispetto a quella 3G come era facile intuire.

Da statistiche reperite sul web la stima del tempo che un'utente mobile è disposto ad attendere prima di avere la risposta ad una sua richiesta è di circa dieci secondi, quindi per il download della lista con connessione wi-fi i tempi sono di gran lunga inferiori a tale soglia infatti l'applicazione è molto fluida e reattiva. Di contro per quanto riguarda la connessione 3G i tempi oltrepassano i dieci secondi nel caso di download sia di 80 elementi sia di 120, ma nonostante questo l'applicazione fornisce un'esperienza utente gradevole e fluida allo stesso tempo.

Ora analizzeremo le performance che si riferiscono al caricamento delle informazioni fornite dalle varie API al momento della selezione di un determinato elemento dalla schermata di dettaglio, riportiamo nella figura 5.4 il grafico che descrive tali performance.

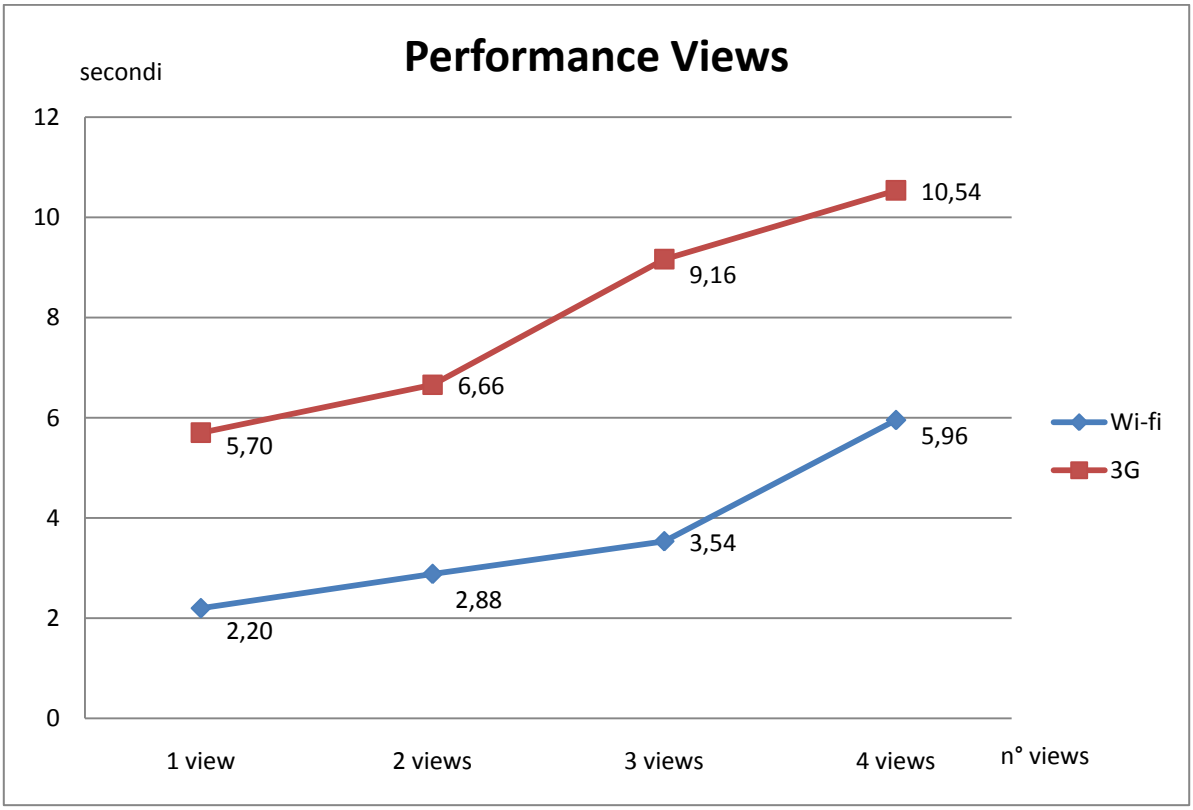


Figura 5.8: grafico delle performance sul caricamento delle diverse views.

Nel grafico riportato sopra l'asse delle ascisse mostra il numero di views (YouTube, Twitter, Flickr e Wikipedia) che l'utente ha abilitato nella schermata del dettaglio, mentre sull'asse delle ordinate sono riportati i tempi di download in secondi. Le valutazioni riguardanti la crescita proporzionale dei tempi in base al tipo di connessione rimangono valide anche in questo caso, infatti, all'aumentare del numero di views selezionate aumentano i corrispondenti tempi di attesa.

Inoltre si è notato che i tempi di download che si riferiscono alla visualizzazione delle views sono molto variabili, e tale variabilità è dovuta principalmente a due aspetti:

- La combinazione delle views scelte dall'utente, poiché la quantità di informazioni scaricare da ognuna di esse differisce sensibilmente.
- Creazione dell'interfaccia utente: le informazioni scaricate dai servizi esterni dovranno essere caricate in un'interfaccia utente all'interno dell'applicazione, ciò richiederà tempi di creazione differenti secondo la complessità di tali interfacce.

Per rimarcare quest'aspetto è ora riportato l'ultimo grafico nel quale si valutano i singoli tempi d'impiego nel caricare i dati rispetto alle singole API di visualizzazione, in modo da individuare quali tra queste rappresentano un "collo di bottiglia" per le prestazioni dell'intera applicazione.

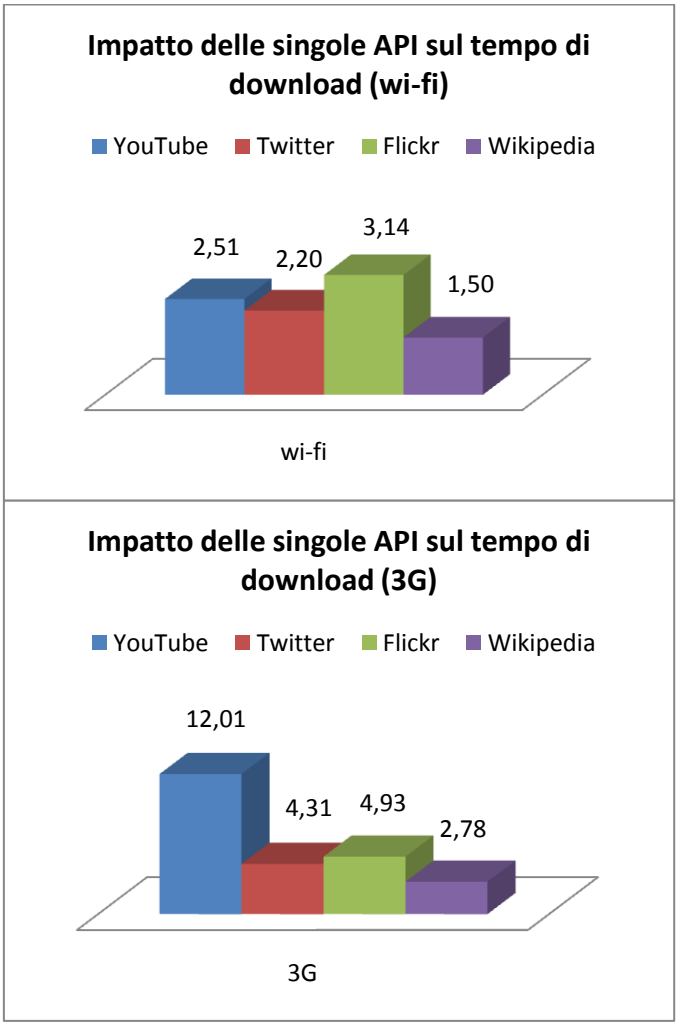


Figura 5.9: tempi di download in secondi per ogni API.

Dal primo grafico di figura 5.5 si nota che i tempi di download su wi-fi sono compresi tra i due e tre secondi per tutte le API, quindi nessuna di queste, influenza in modo significativo i tempi di download dell'applicazione.

Mentre dal secondo grafico di figura 5.5, fra tutte le API, quella di YouTube comporta un aumento sui tempi totali di download con connessione 3G, questo perché ha un'interfaccia utente più complessa e scarica informazioni maggiori rispetto alle altre. Al contrario il servizio di Wikipedia è il più veloce, poiché la pagina di visualizzazione non è creata "al volo" da MobiMash ma è un componente offerto da Android che permette l'utilizzo di un browser incorporato all'interno dell'applicazione nel quale caricare le informazioni recuperate dal servizio.

Per tutti i grafici riportati in questa sezione, c'è da considerare che i tempi registrati sono fortemente influenzati sia dai fornitori dei servizi sia dalle prestazioni della rete 3G, in quanto quest'ultima dipende dalla potenza del segnale ed è più soggetta a variazioni rispetto alla connessione wi-fi.

La seconda modalità con cui poter accedere ad alcuni dati di valutazione delle performance è il dispositivo stesso, che nella schermata applicazioni mostra l'occupazione di memoria e la percentuale di CPU utilizzata durante l'esecuzione dell'applicazione. Nel caso della nostra applicazione l'occupazione di memoria RAM ha valori compresi tra i dieci e i venti megabyte, questo a causa dell'elevato numero di immagini scaricate dai vari servizi, mentre la percentuale di CPU utilizzata è vicino al 2%.

Questo chiude la valutazione delle prestazioni con esito positivo sia per quanto riguarda i tempi di attesa per l'utente sia per l'utilizzo delle risorse interne al dispositivo mobile.

5.4 Validazione e sperimentazione

Quest'ultima sezione del capitolo è dedicata alla validazione e sperimentazione di MobiMash lato mobile per capire quanto gli utenti apprezzino il fatto di poter creare in maniera autonoma le proprie applicazioni sfruttando al tempo stesso il concetto di mashup sia a livello di dati sia di visualizzazione.

Tale sperimentazione è composta di tre parti legate l'una alle altre, di cui diamo una breve spiegazione nell'elenco sotto riportato.

- **Demo:** è stata creata una demo in Power Point di una possibile applicazione mobile creata dal framework MobiMash, e si è data la possibilità all'utente di testare tutte le funzionalità che questa mette a disposizione a partire dai filtri fino ad arrivare all'utilizzo dei due mashup possibili.
- **Questionario:** consiste in una serie di domande con le quali si vuole comprendere sia quanto l'utente abbia trovato semplice l'applicazione sia capire le sue esigenze e le ulteriori funzionalità che aggiungerebbe.
- **Applicazione su mobile:** dov'è stato possibile al posto della demo è stata fatta provare l'applicazione vera e propria, al fine di colmare le inevitabili lacune che la demo Power Point fornisce a causa delle limitazioni del programma stesso.

Di seguito sono riportate le domande del questionario accompagnate dai corrispondenti grafici utilizzati per visualizzare e interpretare le risposte degli utenti.

I due grafici riportati di seguito servono a inquadrare la tipologia di utente che ha eseguito il test, e com'è possibile notare la maggior parte sono studenti con età compresa tra i 20 e i 30 anni.

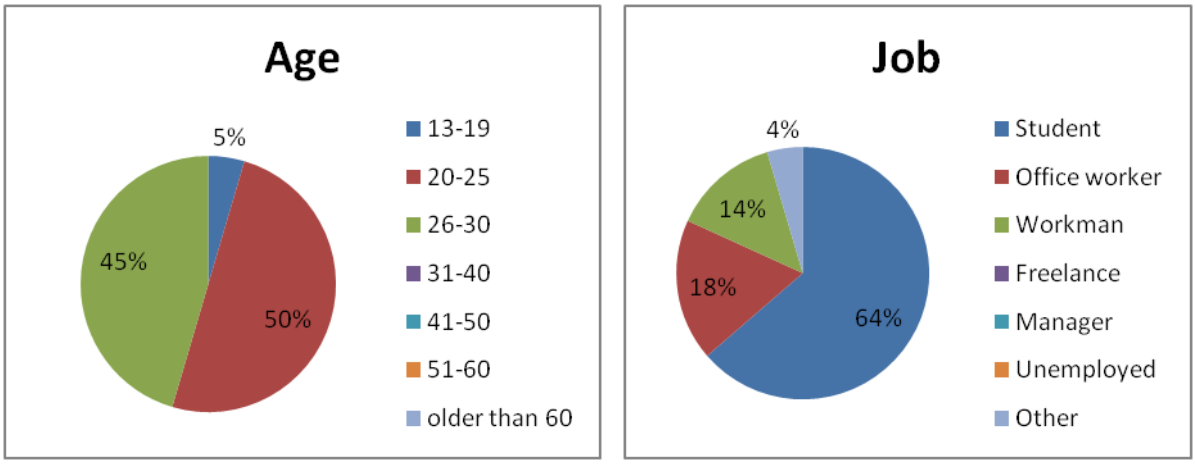


Figura 5.10

I prossimi due grafici mostrano come gli smartphone assumono un ruolo sempre più centrale nella vita quotidiana delle persone, infatti, la maggioranza degli intervistati possiede uno smartphone e lo utilizza sia per uso personale sia in ambito lavorativo.

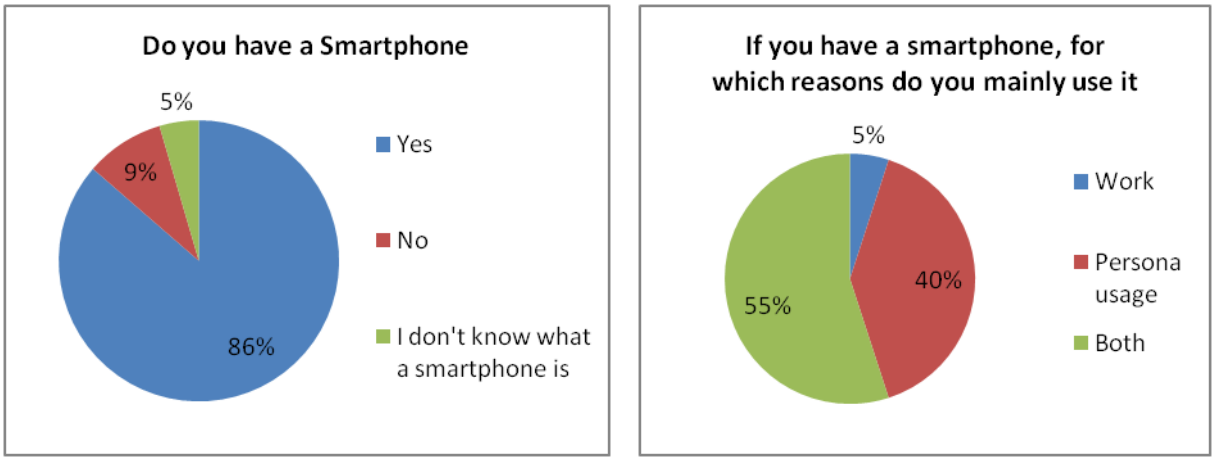


Figura 5.11

Dalla figura 5.12 si capisce come i punti di forza di uno smartphone siano legati alla connettività e alla geolocalizzazione, infatti, oltre all'utilizzo delle semplici funzionalità di base, quali chiamate e SMS, i possessori di smartphone utilizzano il loro device per navigare su internet, leggere email, collegarsi ai social network e utilizzare il GPS integrato.

Anche se ormai il numero di applicazioni sui market è cresciuto in maniera esponenziale, secondo la nostra indagine risulta che più della metà degli intervistati scarica da zero a cinque applicazioni al mese.

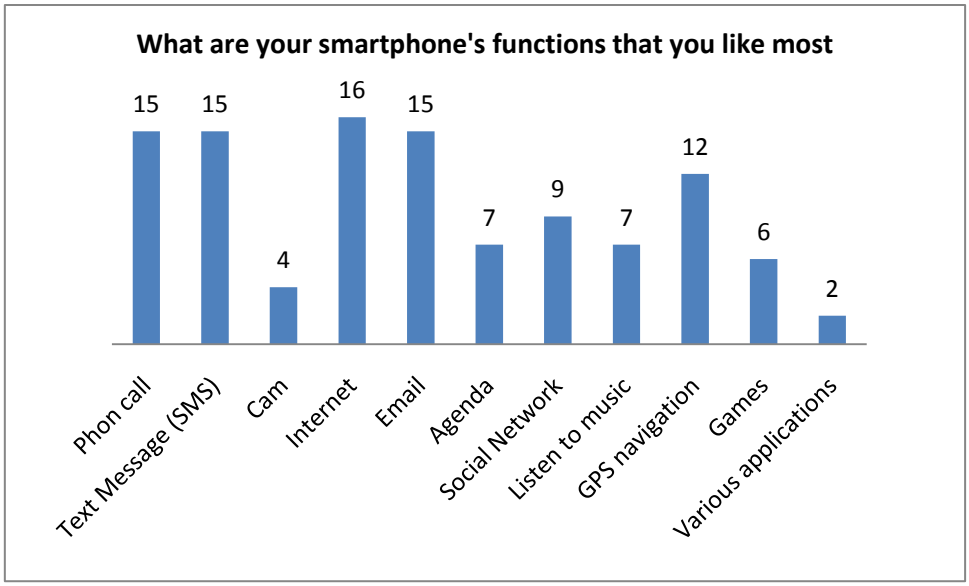


Figura 5.12

Passando ora alle domande sulla demo che mostrava le funzionalità di MobiMash, si può notare che oltre l'80% degli intervista, alla domanda "quanto è stato facile ed intuitivo usare la nostra applicazione" ha dato un riscontro positivo. Il secondo grafico di figura 5.13 rende evidente la risposta positiva degli utenti alla domanda riguardante la personalizzazione dell'applicazione mobile, questo rappresenta un punto a favore del framework in quanto, uno degli obiettivi prefissati è quello di permettere all'utente-sviluppatore di creare applicazioni lato mobile in diversi ambiti semplicemente modificando file di configurazione come spiegato nei capitoli precedenti.

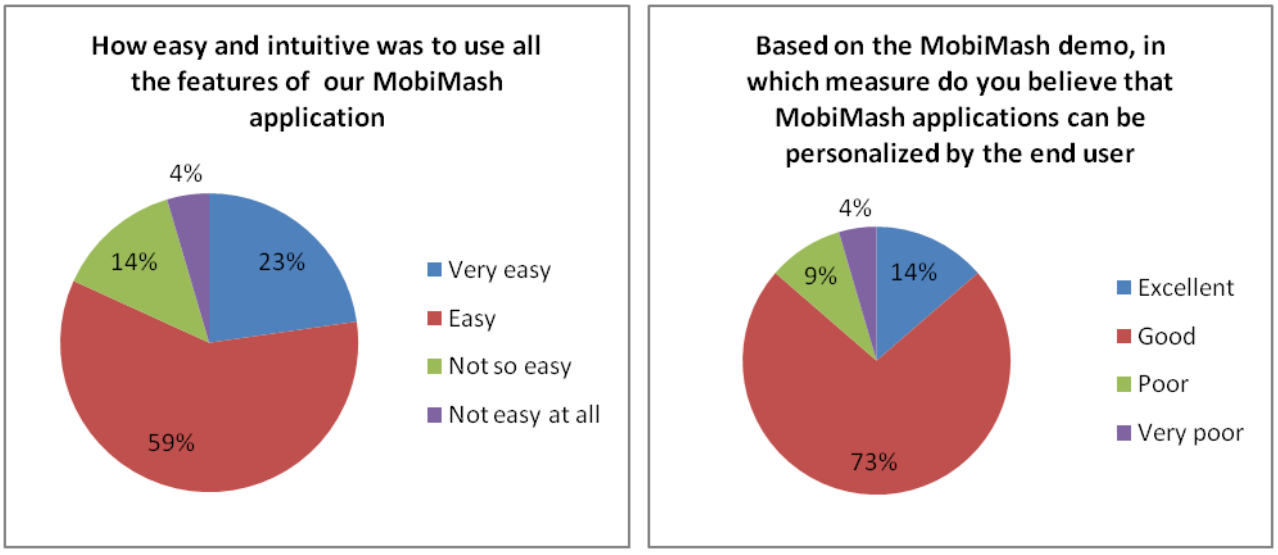


Figura 5.13

I grafici di figura 5.13 mostrano più nel dettaglio quanto è stata apprezzata la possibilità di personalizzare l'applicazione mobile, infatti, la scelta delle fonti ha dato un riscontro positivo per l'80% degli intervistati; quindi il mashup di dati all'inizio dell'applicazione può essere ritenuto valido. Con il secondo grafico si vuole analizzare quanto il mashup di visualizzazione sia gradito dagli utenti, le risposte affermative a questa domanda sono in totale 90%.

Questi due risultati danno un senso all'utilità dei mashup su mobile.

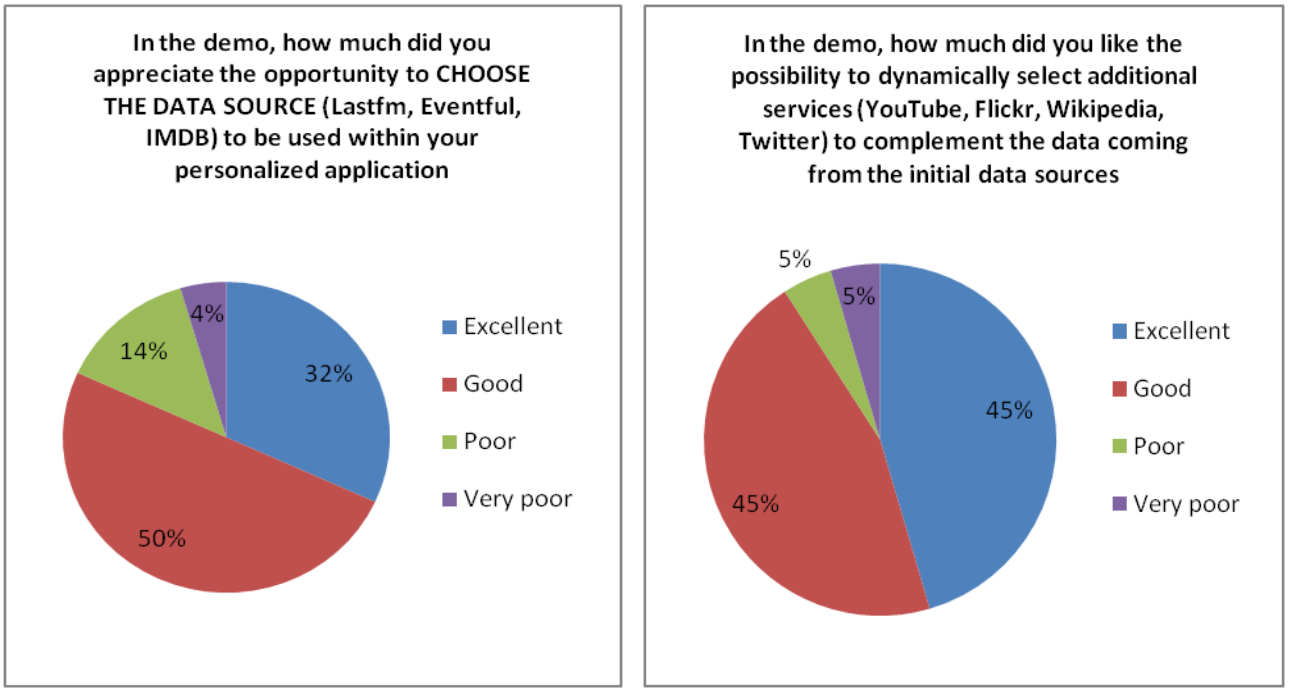


Figura 5.13

L'applicazione ottenuta utilizzando MobiMash ha tre filtri da applicare alla lista dei risultati. E' stato chiesto agli intervistati quali erano i filtri più utili da poter aggiungere all'applicazione,dai risultati spiccano i seguenti filtri

- *Distance*, che permette di settare un raggio in km su cui ottenere i risultati.
- *Date*, che in base all'inserimento di due date restituisce solo i risultati compresi in quel range.
- *Price*, che in base all'inserimento di due prezzi restituisce gli elementi compresi in quel range.

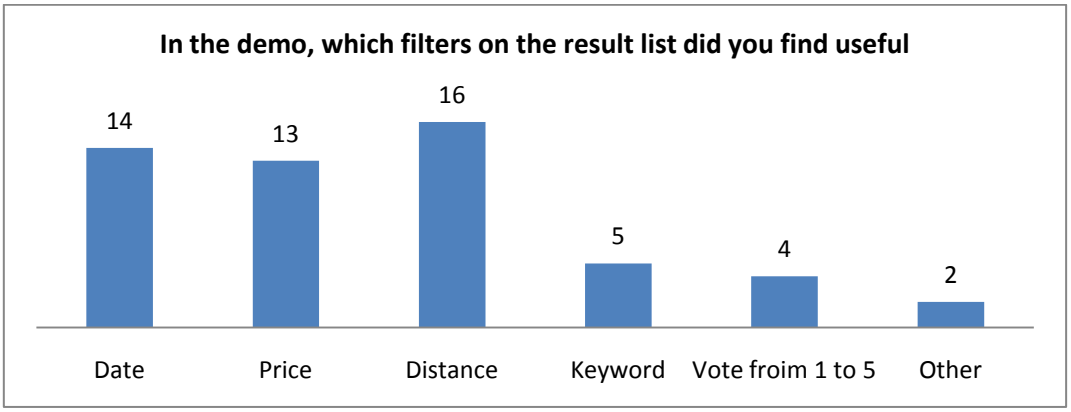


Figura 5.14

Nella voce other di figura 5.14 due intervistati hanno consigliato di aggiungere dei filtri in base alla città.

I domini applicativi che gli utenti vorrebbero poter selezionare per la creazione dei propri mashup, riguardano tutti i domini proposti nella domanda di figura 5.15, tutti questi domini sono stati apprezzati perché riescono ad incastrarsi perfettamente nel nostro paradigma lista, dettaglio, views. E' anche stato consigliato da alcuni intervistati di aggiungere come possibili domini applicativi mostre e individuazione di parcheggi.

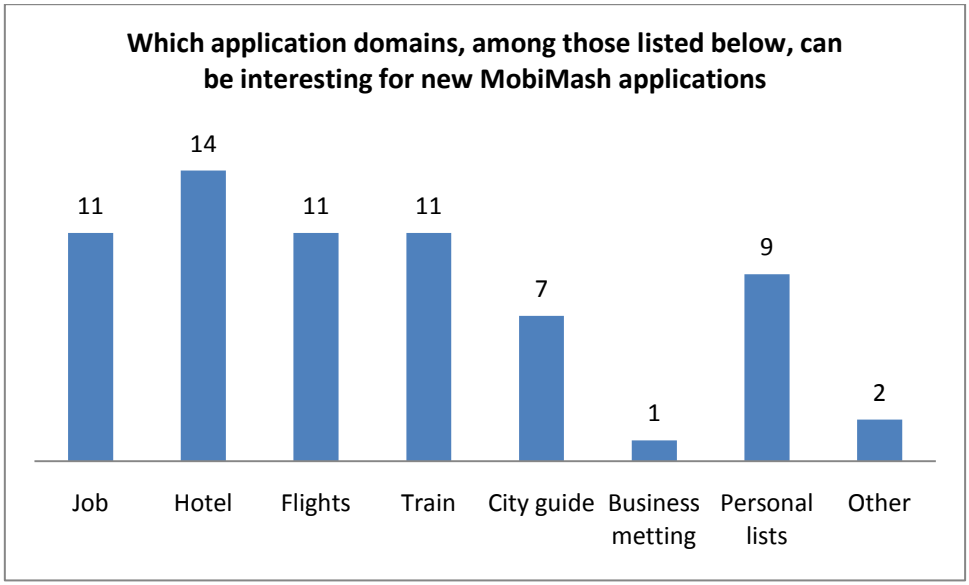


Figura 5.15

Il 41% degli intervistati ha affermato di non conoscere applicazioni simili a MobiMash, metre il 32% pensa di averne già viste e il rimanente 27% non sa rispondere alla domanda.

Tra quel 32% degli intervistati alcuni pensano che MobiMash sia solo una semplice applicazione che mostra i film usciti nelle sale, è chiaro che di applicazioni di questo genere ce ne sono a centinaia, ma in realtà l'esempio sui film era solo un caso d'uso del nostro framework che è in grado di adattarsi a qualsiasi dominio applicativo.

6 Conclusioni

Inizialmente non si aveva un'idea nitida del punto d'arrivo di questo progetto. Si è partiti con l'idea generale di avvicinare il mondo dei mashup a quello dei dispositivi mobile, prendendo come riferimento il framework DashMash basato sul paradigma source, filter e views. Altri due punti chiave che ci eravamo prefissati in fase di descrizione dei requisiti, riguardavano il concetto di end-user development e quello di generalizzazione dell'applicazione finale.

Fatta questa premessa, si può solo dire che il risultato raggiungo supera di gran lunga le aspettative iniziali, quindi riportiamo qui di seguito un resoconto del lavoro svolto a cui si è giunti :

- **Raggiungimento dei requisiti (storia di MobiMash):** in fase iniziale, ci siamo soffermati su analizzare le varie tecnologie disponibili per lo sviluppo mobile, e successivamente capire se queste potessero risolvere vari vincoli di prestazioni, infatti non avendo esperienza in ambito mobile non si poteva sapere quali erano i limiti di questi dispositivi.
  In seguito, scelta la tecnologia da utilizzare ovvero Android, ci siamo imbattuti nello sviluppo di un'applicazione mobile riguardante uno specifico dominio applicativo, sempre seguendo il paradigma source, filter e views. Questa applicazione era ricca di funzionalità come ad esempio la geolocalizzazione o l'integrazione con vari servizi come Youtube, Flickr, Twitter e Wikipedia ma non risolveva ancora i requisiti di generalizzazione.
  Con l'esperienza acquisita nello sviluppo dell'applicazione appena descritta, non è stato difficile staccarsi dal dominio applicativo e rendere quindi il risultato molto parametrico, questo è stato possibile introducendo un'applicazione lato desktop per il design e dei file di configurazione per far adattare sia l'interfaccia sia i contenuti lato mobile.
  Detto questo, si può affermare che con MobiMash si è creato uno dei primi framework per la generazione di applicazioni mobile da parte dell'utente finale.

- **L'utilizzo di Android:** la scelta di orientare lo sviluppo dell'applicazione verso questo sistema operativo, è dettata dal fatto che attualmente ad esso appartiene la fetta più grande della divisione tra i sistemi operativi per smartphone, inoltre l'SDK permette di scrivere applicazioni in codice Java, linguaggio a noi molto familiare. Ricordiamo anche che Android è un sistema open e quindi non richiede dei costi per licenze d'uso.
  In origine pensavamo di poter avere dei problemi tra le varie versioni del sistema operativo o con le dimensioni degli schermi che sono differenti da modello a modello, in realtà Android non risolve completamente questi problemi ma aiuta lo sviluppatore, ad esempio adattando automaticamente l'applicazione in base alla dimensione dello schermo.
  L'utilizzo di questo framework si è rivelato un'ottima scelta per la realizzazione di MobiMash, infatti, essendo uscito da ormai qualche anno si è giunti a una versione stabile del codice e sul web si possono trovare infiniti topic che aiutano lo sviluppatore a risolvere problemi di svariato tipo.

- **Alto livello di personalizzazione:** un altro punto chiave di questo elaborato è di lasciare un'ampia libertà all'utente finale di poter personalizzare la propria applicazione mobile. Questo requisito è stato pienamente soddisfatto, infatti, il concetto di personalizzazione si può notare in MobiMash lato desktop, dove l'utente-sviluppatore è libero di creare la propria applicazione mobile andando a selezionare un insieme di servizi da un repository e poi decidere come visualizzare i contenuti forniti da questi servizi. Infine l'utente dell'applicazione mobile ha la possibilità di fare un mashup sia a livello di dati sia di visualizzazione dei servizi aggiuntivi.

Non solo con l'implementazione di MobiMash si è risposto esaustivamente ai requisiti che ci eravamo prefissati, ma si è anche dato un punto di partenza per un progetto più grande. Elenchiamo qui di seguito dei possibili sviluppi futuri o miglioramenti che potrebbero essere apportati al framework:

- *Implementazione lato desktop:* l'applicazione MobiMash lato desktop è ancora in fase di implementazione anche se si è già definita l'intera architettura del software.

- *Registrazione di servizi:* per ora i servizi disponibili riguardano solo le categorie Musica e Film, in base a quanto espresso dal questionario valutativo, saranno presto aggiunti servizi relativi a Hotel, Treni, Voli, Lavoro e servizi personalizzabili dall'utente.

- *View alternativa sulla lista:* a breve sarà implementata una seconda vista per la lista contenente i risultati di ricerca, questa mostrerà i risultati all'interno di una mappa così da geolocalizzare le informazioni, chiaramente la lista sarà centrata in base alla posizione attuale dell'utente.

- *Altri servizi aggiuntivi:* si sta pensando a quali servizi aggiuntivi oltre a Youtube, Flickr, Twitter e Wikipedia potrebbero essere aggiunti per approfondire una ricerca in base ad una keyword.

- *Miglioramento delle prestazioni:* alcune parti del codice potrebbero essere ottimizzate ad esempio riducendo la complessità di alcuni algoritmi o introducendo il concetto di thread, così da far eseguire in parallelo diverse operazioni.

- *Multi piattaforma:* attualmente il framework è stato sviluppato per piattaforma Android, in futuro lo sviluppo sarà ampliato sulle piattaforme di maggior successo, ad esempio iOS e Windows Phone 7.

MobiMash è un framework per la generazione di mobile mashup che ha già scaturito l'interesse di diverse aziende dell'ICT, nel frattempo stiamo ancora raccogliendo i dati del questionario per capire l'opinione degli utenti, e quindi analizzare gli aspetti più apprezzati e quelli ancora da migliorare.

Bibliografia e riferimenti

Android

http://developer.android.com/guide/basics/what-is-android.html

http://en.wikipedia.org/wiki/Android_(operating_system)

Android in Action, Second Edition - autori: W. Frank Ableson, Robi Sen, and Chris King

Android. Guida per lo sviluppatore (Guida completa) – autore Massimo Carli

Mobile Mashup, End-user development

http://www.mobilemashups.com/

http://www.slideshare.net/avinash.raghava/trends-and-evolution-of-mobile-internet-by-naresh-gupta-adobe

http://media2.telecoms.com/downloads/mobile-internet-traffic-trends.pdf

http://www.slideshare.net/aseidman/building-mobile-mashups

http://www.crunchgear.com/2010/05/29/the-age-of-the-mobile-mash-up/

http://en.wikipedia.org/wiki/Mobile_operating_system

http://mobile.html.it/guide/lezione/966/principali-sistemi-operativi/

Quando L'utente Guida L'innovazione Il Web Mashup – autori Florian Daniel e Maristella Matera