ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso Di Laurea Triennale In Informatica

MARKET E TOOLS: UTILITY PER LA PERSONALIZZAZIONE D'APPLICAZIONI ANDROID

Tesi Di Laurea In Architettura Degli Elaboratori

Relatore:
Chiar.mo Prof.
VITTORIO GHINI

Presentata da
ANDREA POLA

II Sessione

Anno Accademico : 2011/2012

«La fortuna è quel momento
in cui la preparazione incontra l'opportunità»
cit: Randy Pausch

Market e tools: Utility per la personalizzazione di applicazioni Android

Studio di fattibilità e delle problematiche relative alle tecniche di personalizzazione di applicazioni Android mediante Webservice o Ide

Sommario

Elenco delle Figure

Introduzione e obbiettivi

Questo testo si pone come obbiettivo l'analisi di fattibilità tecnica e l'introduzione all'implementazione di sistemi che permettano il riutilizzo di codice sorgente di applicazioni con necessità simili su dispositivi Smartphone. In particolare su sistemi Google Android.

Questo è il concetto di personalizzazione di applicazioni, in pratica la costruzione di sistemi che permettano di generare applicazioni mobile attraverso interfacce user-friendly e mediante l'uso di codice modulare.

L'obbiettivo è fornire una visione globale delle soluzioni e delle problematiche di questo campo, poste come linee guida per chi intendesse studiare questo contesto o dovesse sviluppare un progetto, anche complesso, inerente alla personalizzazione di applicazioni.

Non mancherà un'introduzione al contesto del sistema Android, per giustificarne la scelta, al Market e un approfondimento sui Tools per la programmazione Android.

L'idea è che ogni sviluppatore Android (ma non solo) dopo un certo numero di progetti si ritrovi con determinate parti di codice, che potremmo azzardarci a chiamare librerie, che vorrebbe riutilizzare.

Si delinea cosi l'esigenza di trovare un sistema per ridurre i tempi di sviluppo di applicazioni che abbiano necessità comuni. Portando all'estremo questo concetto, si potrebbe realizzare un servizio, un web-service od un ambiente di lavoro per lo sviluppo di applicazioni mobile che faciliti il riuso di librerie o progetti stabili attraverso interfacce intuitive.

Questo sistema potrebbe essere cosi adatto per realizzare differenti versioni di un'applicazione, personalizzandola in base ai dati del cliente. Si possono sviluppare sistemi che permettano di personalizzare e comporre la propria applicazione attraverso interfacce user-friendly direttamente online e senza grosse competenze. Attraverso un servizio come questo è possibile permettere a persone che hanno attività commerciali di rappresentare la propria azienda sul market, rilasciare semplici servizi, senza che abbiano particolari conoscenze di programmazione su ambienti mobile.

Quando parliamo di questo stiamo intendendo la personalizzazione di applicazioni Android, anche se questi sistemi trovano da poco tempo una nicchia di mercato che è stata identificata come "Application Building".

Sarà implementato come esempio, un web service per la personalizzazione di applicazioni Android, soffermandosi in particolare sulle problematiche legate alla paternità del software e delle firme digitali necessarie per la pubblicazione sul market Android. Saranno definite alcune scelte da prendere se si sviluppano applicazioni per terzi che in seguito saranno rilasciate sul market.

Nella ultima parte sarà analizzata una strategia di customizzazione attraverso alcune buone pratiche, che permette attraverso l'uso di progetti libreria e direttamente nell'ambiente di sviluppo, di realizzare codice modulare e pronto per il market Android in diverse versioni.

Felicemente, durante la stesura del preludio alla tesi, partecipando ad una conferenza , la WhyMCA 2012, alcuni dei concetti che stavo riportando erano riposti in alcuni dei Talk della conferenza, confermando l'attualità dei temi riportati in questo testo.

Breve riassunto degli argomenti trattati

Nel primo capitolo saranno introdotte le motivazioni per cui Android è la piattaforma ideale di questa tesi, grazie alla disponibilità sul mercato di numerosi dispositivi e di un market che più dei concorrenti permette la pubblicazione di applicazioni in maniera snella. Inoltre si analizza come l'ambiente mobile e i markets influenzano le aziende e i consumatori.

Di seguito nel secondo capitolo si cerca di chiarire due delle correnti tecniche progettuali a riguardo di applicazioni mobile, mettendole a confronto e valutando se il concetto di personalizzazione di applicazioni possa essere applicato insieme al concetto di portabilità.

Continuando, vengono date le basi tecniche a riguardo di SDK e strumenti di sviluppo Android. Viene spiegata la struttura generica di funzionamento di un progetto di personalizzazione di applicazioni Android mediante WebService .

Successivamente il quarto capitolo è di fatto il cuore della tesi e affronta una problematica tecnica e concettuale che si genera durante l'analisi di fattibilità in Android: la gestione delle firme digitali per il rilascio sul market. Si illustreranno le scelte progettuali adottabili e per ognuna di esse verrà fornita una breve discussione dei pro e dei contro. Concludendo il capitolo con l'analisi dello stato dell'arte dei sistemi commerciali di Application building, soffermandosi in modo particolare all'analisi di due soluzioni: AppInventor e AppsBuilder.

Il quinto capitolo è una Demo di quanto esaminato precedentemente e mostra un esempio didattico di implementazione di Web Service di personalizzazione basato sulla generazione di Webview. E' allegato uno schema di funzionamento, una discussione dell'implementazione, dei punti critici per la sicurezza del sistema oltre che l'elenco dei requisiti del sistema host per ospitare un webservice di questo tipo.

Il sesto capitolo affronta, seguendo la discussione delle possibili ottimizzazioni della Demo, una strategia progettuale per realizzare applicazioni utilizzando progetti libreria. Analizzando quali obbiettivi coinvolge questo mutamento nel modo di personalizzare applicazioni e quali aziende potrebbero adottare questa procedura ed in quali progetti. Il capitolo viene integrato con l'esempio di struttura di un'applicazione pronta per essere collocata sul market in versione lite ed in versione pro, come succede spesso in applicazioni reali sul market Android.

Non manca, nella sezione Possibili Sviluppi un elenco di tematiche che leggendo questa tesi viene naturale voler approfondire, in tema sia di tecnologia che di legalità.

--

NOTA:

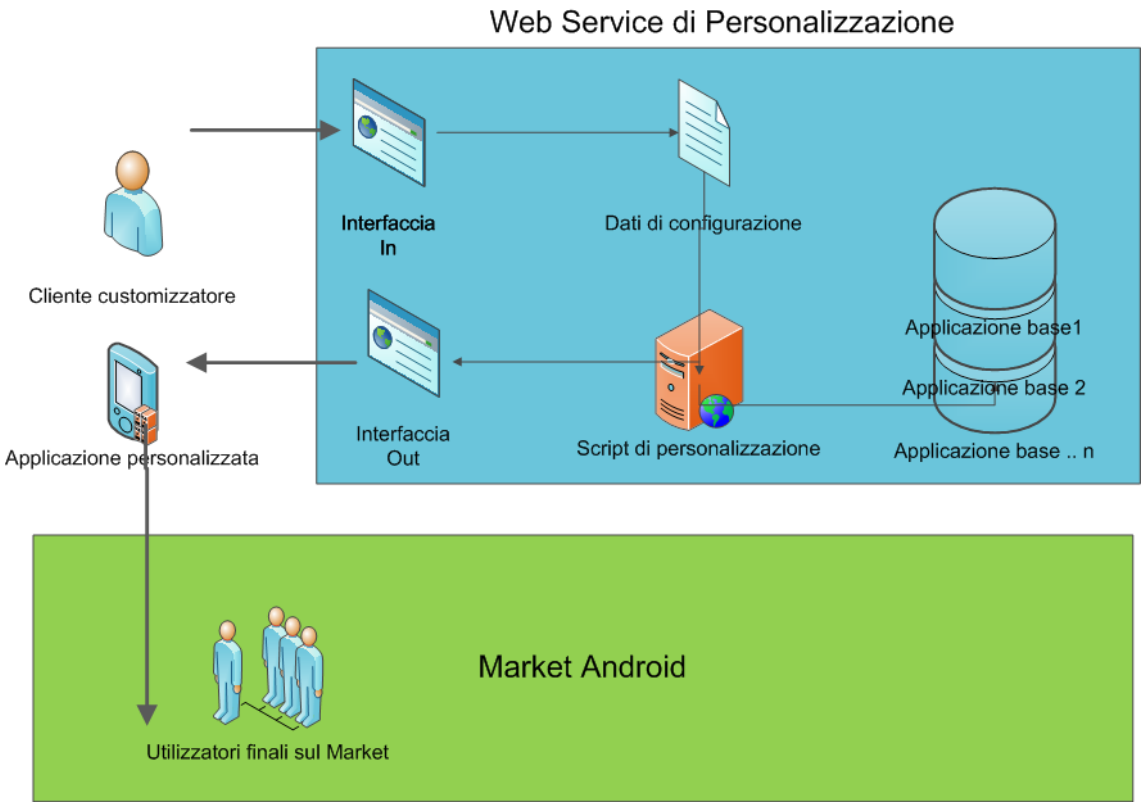


FIGURA 0: SCHEMA GENERALE

Nella tesi si fa molte volte uso della parola cliente, se non specificato diversamente il riferimento sarà un riferimento al cliente customizzatore dello schema sopra. Questa persona è il principale utilizzatore del webservice di personalizzazione e utilizza questo servizio per generare un'applicazione secondo le sue esigenze con l'intenzione di rilasciarla sul market.

Per il riferimento agli utilizzatori finali sul market si è preferita la parola pubblico.

Capitolo 1
Ambiente

1.1 Dispositivi mobile e Markets

In questa prima parte vediamo di giustificare e analizzare il contesto attuale delle piattaforme smartphone e le motivazioni per cui si è scelto Android.

Google Android al momento attuale è la piattaforma miglior candidata per la rappresentazione del più largo settore di utenti e dispositivi. La piattaforma Android infatti conta il maggior numero di dispositivi compatibili sul mercato e lo share di maggioranza assoluta nel mercato statunitense.

Poniamo il caso di dover scegliere una piattaforma tra le disponibili attualmente sul mercato, come Blackberry, Symbian, Apple, Windows Mobile e Google Android. Un'attenta valutazione va fatta sulla politica del market della piattaforma scelta. Và analizzato il funzionamento del market e dei Tool con cui si vuole operare, per vedere se è possibile generare in maniera automatica o guidata diverse istanze di un'applicazione partendo da un codice sorgente comune.

In questo momento, sul Google Market, che prende il nome commerciale di Google Play, è possibile caricare applicazioni in breve tempo e senza controlli manuali da parte del team di Google. Il controllo di qualità su queste applicazioni è lasciato ad un meccanismo di feedback e segnalazione autonomo degli utenti, è possibile infatti votare ogni applicazione e segnalare abusi. Applicazioni meglio valutate avranno migliore visibilità in tutto il market. Non mancano però controlli per la rimozione di applicazioni direttamente da parte di Google, mantainer del Market, che analizza il market alla ricerca di software pericoloso e malaware.

Come i markets influenzano il software?

Con lo sviluppo di dispositivi cellulari con interfacce di rete veloci come Wi-fi e 3G, disponibilità di potenza di calcolo e RAM, si è sviluppato velocemente un nuovo modo di rendere disponibile il software verso i consumatori appunto i Markets di Applicazioni.

I Markets rendono disponibili direttamente sul cellulare attraverso un portale centralizzato e autorizzato, molte applicazioni, con vantaggi in termini di sicurezza e di visibilità dei prodotti pubblicati. Si tratta sicuramente di una grande opportunità per uno sviluppatore, che una volta sviluppato un buon

prodotto deve spendere meno energie e risorse per rendere disponibile il proprio software al grande pubblico.

Con queste opportunità si sono delineati filoni diversi di sviluppatori mobile, gli sviluppatori per il grande pubblico, che rilasciano i loro prodotti direttamente nel market e gli sviluppatori per conto terzi. Proprio questi due filoni di sviluppatori avrebbero necessità diverse in campo di personalizzazione di applicazioni. Il primo sarebbe slanciato verso un versioning della propria applicazione in ,ad esempio: "lite","free" e "pro" oppure se viene realizzata un'applicazione con un grosso successo, il team che sviluppa per il grande pubblico potrebbe pensare di effettuare variazioni su quell'applicazione, cambiarne il tema trattato ma lasciarne la struttura …

Un'azienda invece che sviluppa per conto terzi ha necessità di raccogliere le richieste del cliente, i dati e personalizzare eventualmente un progetto base in funzione dei dati raccolti. Per la maggior parte di questa tesi sarà proprio questa la situazione di riferimento.

Le esigenze di queste due tipologie di aziende, possono essere affiancate ma vista la differenza degli obbiettivi in un caso potrebbe risultare più comoda una soluzione direttamente nell'ambiente di sviluppo (come suggerito nel capitolo 6) e in altri casi lo sviluppo di un progetto autonomo come affronteremo di seguito.

A differenza di altri campi dell'ingegneria del software, di fatto, con le opportunità createsi dai market, la base economica necessaria per uscire al grande pubblico è diventata abbordabile a molti, comprese le piccole medie aziende, tipiche del contesto italiano, favorendo la nascita di nuove e giovani Startup.

Come vedremo, i Markets hanno anche dato vita a nuove (o ripresentate in un nuovo contesto) "Legal Issue" per la gestione di responsabilità e certificati tra sviluppatori, pubblicatori , utenti e aziende , che in determinate situazioni si scontrano ricoprendo lo stesso ruolo in un mercato molto concorrenziale e meritocratico. Proprio questo tema sarà affrontato in maniera approfondita per quanto riguarda la pubblicazione di applicazioni personalizzate sul market Android in quanto punto critico per lo sviluppo del progetto trattato in questo testo.

1.2 Tendenze del mercato

Grazie allo sviluppo dei markets e grazie al notevole successo di queste soluzioni è sempre più evidente la necessità di avere un'applicazione che rappresenti la propria azienda sul market Android e/o sull'App Store. Le statistiche parlano di

un rapido tasso di crescita del numero di applicazioni disponibili sul market Android che non mette in dubbio la crescente importanza del settore mobile.

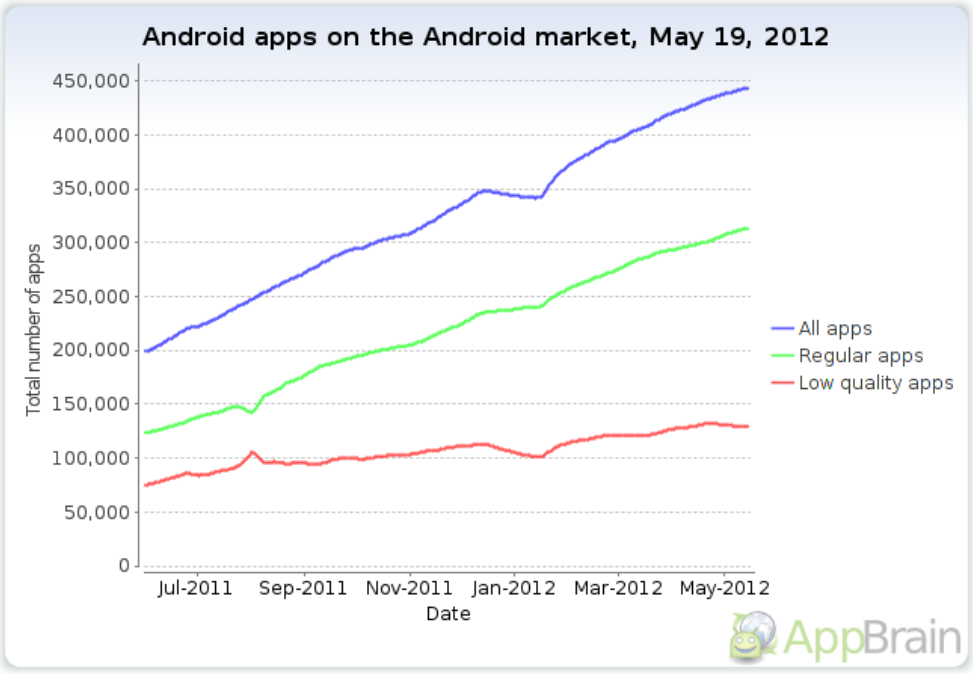


FIGURA 1: CRESCITA APPLICAZIONI ANDROID 2011/2012

Avere una soluzione mobile dei propri servizi rappresenta un grande valore aggiunto per il proprio marchio oltre che un'estensione della propria soluzione web.

Questa tendenza ha accelerato il mercato delle applicazioni mobile, al punto di aumentare l'aspettativa dell'utente, che si aspetterà di trovare, per una grande azienda, soluzioni mobile, sia su Apple App-store che su Google Play, quindi sia per IOS che per Android. Questo da vita ad un tema molto vicino a quello trattato in questo testo, lo sviluppo cross-platform di applicazioni mobile.

Molte testate giornalistiche ad esempio, sono venute incontro a questa esigenza di mercato estendendo i loro quotidiani al settore mobile attraverso le versioni a piccolo schermo dei loro siti e giornali.

Lo sviluppo Cross-platform risulta come una soluzione unificata e a costo ridotto per risolvere lo stesso problema (rendere disponibili i propri servizi) su diverse piattaforme che non condividono strumenti di sviluppo e linguaggi di programmazione.Proprio lo sviluppo di applicazioni cross-platform basate su web (web-app, web-based app, webview) può essere un bell'esempio per trattare di come è possibile sviluppare un'applicazione mobile per renderne il codice sorgente riutilizzabile per diversi clienti, quindi **"personalizzabile"**.

Capitolo 2

Personalizzazione e Portabilità

2.1 Web-app un ottimo esempio

Di fatto una web app mobile viene sviluppata in due componenti principali, un lato web e sullo smartphone un lato client. Lo sforzo in questa situazione si concentra totalmente lato web, sviluppando interfaccia e funzionalità con linguaggi che daranno come output pagine web compatibili con tutti gli smartphones dell'attuale generazione. Esistono frameworks di sviluppo per web app pensati appositamente per questo, che realizzano pagine web per interfacce mobile. Lo schema che ne deriva è il seguente:

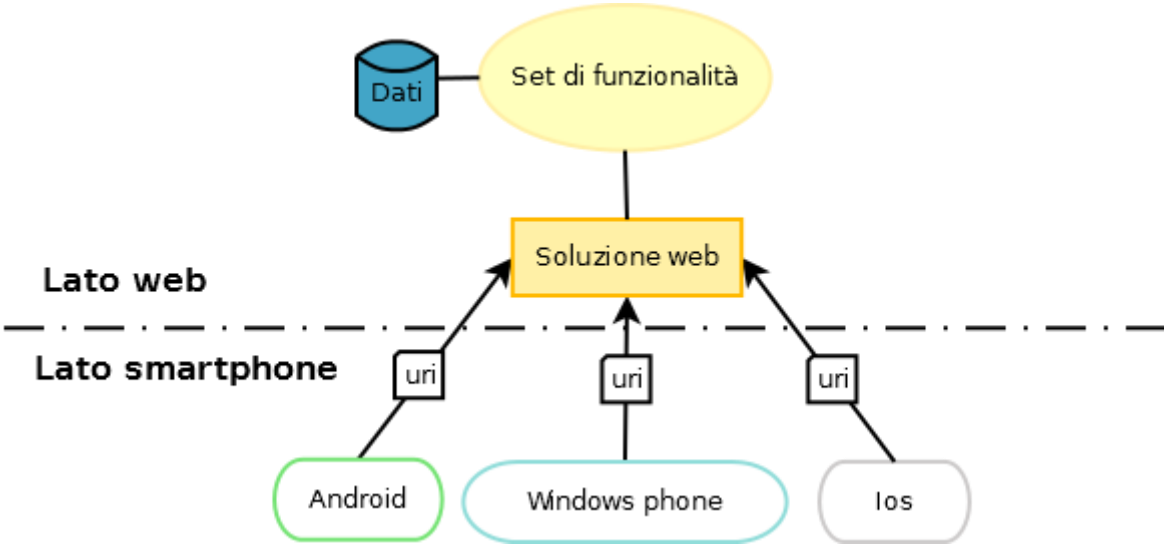


FIGURA 2: ARCHITETTURA APP WEB / CLIENT MOBILE

Questo schema porta essenzialmente un vantaggio:

- Scrivi una volta, usa sempre

In questo modo la parte client delle applicazioni, che sarà installata sullo smartphone risulta molto semplice. In una tecnologia o in un'altra (IOS, Android Windows Phone...), l'essenza del client è una chiamata verso il browser passando come parametro l'url del servizio web, magari con maschere o variazioni sulle funzionalità del browser stesso (Click Handling, Gesture Handling etc...).

Si delinea che la struttura parte client/smartphone di un applicazione di questo tipo può essere banalmente riutilizzata. Il codice funzionale infatti sarà sempre lo stesso, a cambiare sarà l'unica parte che ha necessità di comunicare con sistemi esterni, in questo caso l'url della soluzione web.

E' facile in questo modo capire che il sorgente client di un'applicazione di questo tipo è riusabile ogni volta che c'è la necessità di visualizzare una nuovo soluzione web di un nuovo cliente. Una possibile soluzione per il riutilizzo del codice funzionale è l'inserimento di un file di configurazione, che raccoglie i parametri che poi saranno personalizzati.

Durante la personalizzazione di un applicazione di questo tipo, sarà solamente il file di configurazione ad essere variato, facendo rimanere invariata la parte funzionale.
In questo semplice esempio potremmo realizzare un file di configurazione che contenga l'url del servizio web.

Elaborando un po' il concetto appena espresso, possiamo notare che molte applicazioni commerciali per numerosi motivi accollano "la parte difficile" del problema che devono risolvere a sistemi esterni. Nell'esempio precedente tutto il peso dell'implementazione è lato web ed è quindi risolto con tecnologie web.

E' importante che i dati siano elaborati dal sistema esterno, che ha capacità di calcolo adatte a risolvere un problema "difficile" e ospitare grandi quantità di dati. I dati saranno filtrati ed elaborati dal sistema centrale e solo una volta resi presentabili e trasformati in informazioni inviati al dispositivo mobile. Considerando anche l'alta latenza e la banda molto variabile di questi dispositivi è necessario progettare un protocollo (quasi sempre basato su HTTP), da e verso il dispositivo mobile che utilizzi poca banda per le comunicazioni. Nella soluzione precedente, la banda veniva in gran parte utilizzata per l'invio di informazioni stilistiche, di grafica e di impaginazione, cosa che però potremmo evitare con l'uso delle interfacce native di Android oppure IOS oppure Windows Phone…

Possiamo quindi ridurre, molte altre applicazioni presenti sui Market allo schema seguente, molto simile a quello presentato in precedenza:

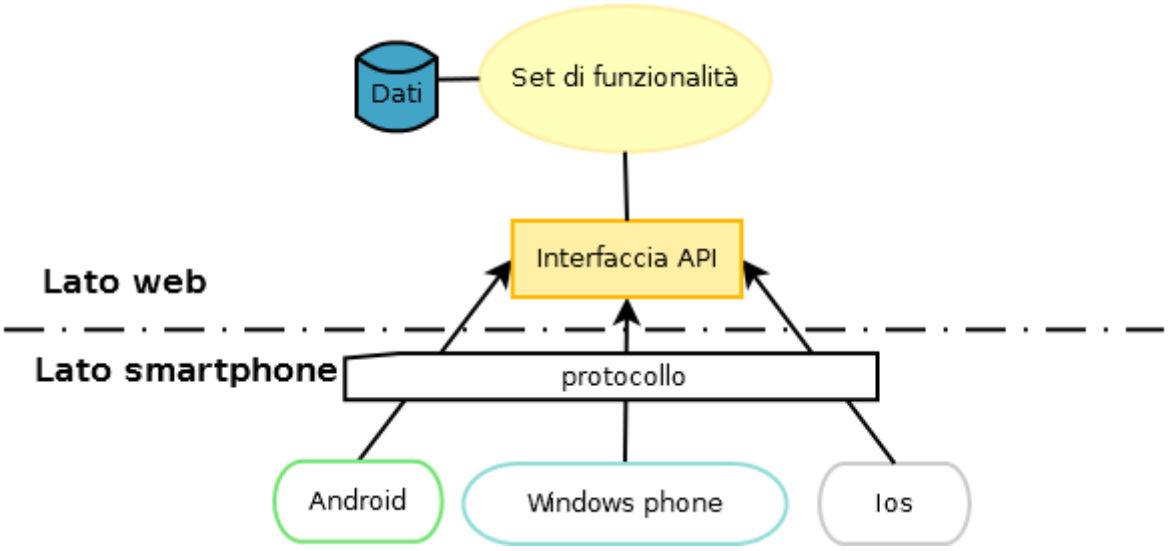


FIGURA 3: ARCHITETTURA APP API / CLIENT MOBILE

Il protocollo è l'insieme di comandi verso l'interfaccia API del Web Server, essa fornirà determinate risposte, che la parte client deve conoscere, saper gestire nonché visualizzare adeguatamente.

Notiamo che in questo schema abbiamo passato un po' della parte funzionale del progetto al lato client, per poter godere di alcuni vantaggi:

- Utilizzo delle interfacce native delle piattaforme
- Riduzione di banda richiesta

Sono sicuramente due punti forti, siccome l'esperienza utente su di un certo dispositivo porta a far preferire all'utente stesso interfacce simili a quelle di uso quotidiano e quindi all'utilizzo delle interfacce native. E' possibile attraverso le interfacce native usare anche sensori e dispositivi integrati sullo smartphone, che con interfaccia completamente web non sarebbero utilizzabili.

Il Secondo punto è chiaramente essenziale per aumentare l'esperienza utente, rendendo più affidabile l'applicazione estendendo il funzionamento anche in zone dove la banda sullo smartphone risulta molto ridotta. E' chiaramente noto come la banda disponibile su queste macchine sia molto variabile.

2.2 Estensione del concetto a multipiattaforma

Personalizzazione su singola piattaforma:

Scelta una singola piattaforma, qualunque essa sia bisogna porsi alcune domande per vedere se è possibile realizzare un sistema basato su WebService che possa personalizzare applicazioni.

Riferendoci agli schemi in Fig 2 e Fig 3, in entrambi i casi un grosso muro da superare è la valutazione di fattibilità rispetto a quello che viene offerto dall'SDK e dagli strumenti di sviluppo della piattaforma scelta, bisogna rispondere ad alcune domande, come:

- E' possibile pacchettizzare dinamicamente un'applicazione ?
- Ci sono problemi di firme/paternità sui market?
- Ci sono restrizioni sull'hardware su cui è possibile far girare gli sdk?
- E' possibile realizzare script che usino l'sdk in modo agile, senza utilizzare tool pesanti che comprometterebbero le prestazioni dell'intero progetto?

Quindi è necessario un attento studio della tecnologia legata alla piattaforma, come viene fatto in questo testo per Android. Nei prossimi schemi e fino alla fine del capitolo, tutto questo sarà assunto come ipotesi ed integrato nei legami tra piattaforme e applicazione.

Se invece si intende personalizzare mediante IDE, allora la questione diventa molto più semplice e non ci sono, almeno a prima vista particolari problemi siccome stiamo facendo la procedura con direttamente sui Tool di sviluppo.

Come cambia il discorso aggiungendo la possibilità di scegliere la piattaforma direttamente al momento della personalizzazione? Quali problemi si pongono se volessi generare una applicazione personalizzata in funzione al cliente e realizzarla contemporaneamente per tutte le principali piattaforme in commercio?

Personalizzazione su più piattaforme:

Con lo schema di Fig 3 possiamo notare come il concetto di Personalizzazione contenga più strettamente rispetto a prima il concetto di Portabilità, poiché ora abbiamo parte delle funzionalità (compatibilità con il protocollo) anche sull'applicazione smartphone.

Mentre nello schema di Fig 2 era intuibile che la generazione delle versioni dell'applicazione per i vari sistemi operativi fosse fattibile, con questo nuovo schema il discorso è un po’ più complicato. Nello schema in Fig 2, l'applicazione

era sostanzialmente realizzata via web e semplicemente visualizzata sul dispositivo mobile. Mentre prima era chiaro che il codice delle varie versioni fosse semplice e per generarlo bastasse un codice di base per ogni piattaforma e un file di configurazione, ora invece bisogna realizzare una rappresentazione del protocollo. E partendo da questa rappresentazione , generare, in funzione delle delle librerie disponibili sulla piattaforma, l'applicazione funzionante. Problema sicuramente molto più difficile rispetto a prima.

Questa osservazione fa notare come torna utile sviluppare la compatibilità per un protocollo noto e diffuso, che potrebbe essere ad esempio RSS senza realizzare una descrizione del protocollo nella configurazione. Restringendo cosi la possibilità di personalizzazione all'utilizzo di fonti di dati che usino RSS.

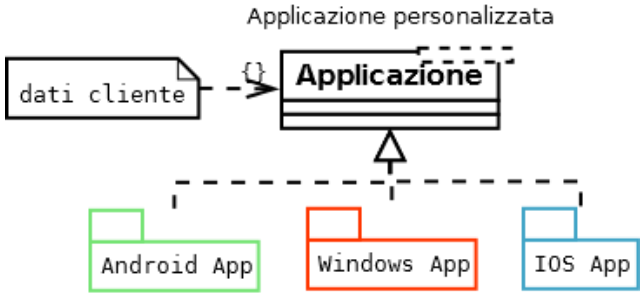


FIGURA 4: GENERAZIONE MULTIPIATTAFORMA CON SCHEMA FIG 2

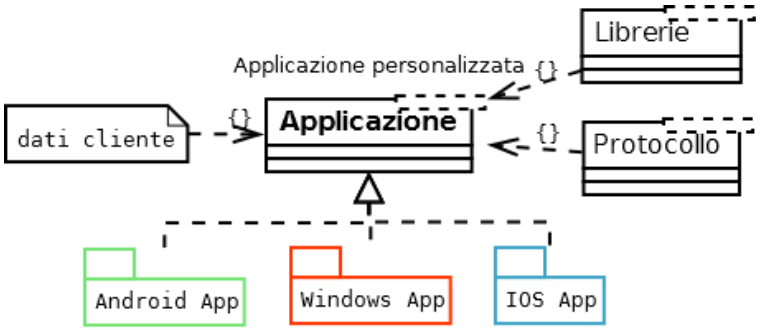


FIGURA 5: GENERAZIONE MULTIPIATTAFORMA CON SCHEMA FIG 3

Sarebbe quindi interessante e purtroppo rimandato per questioni di tempo e risorse a "possibili sviluppi" l'analisi di fattibilità di un servizio che permetta di realizzare la portabilità contemporaneamente su più piattaforme secondo lo schema di Fig 2 e analizzare la fattibilità dello schema di Fig 3. Unendo quindi in un progetto il concetto di Personalizzazione e Portabilità.

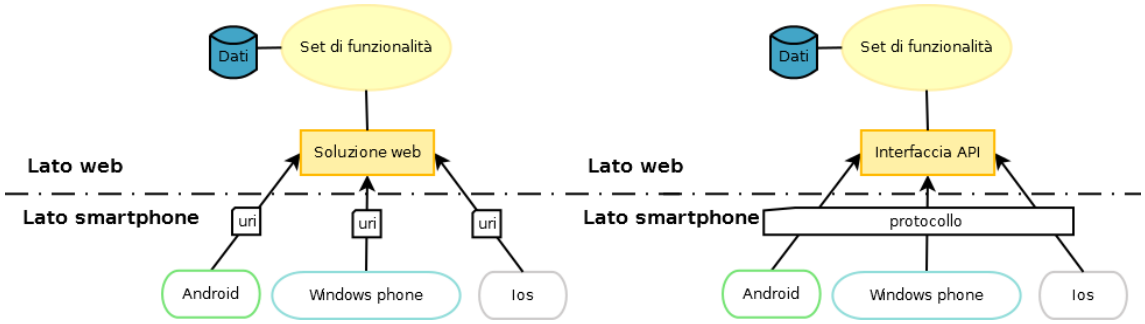


FIGURA 2BIS E FIGURA 3BIS

Ricapitolando:

Schema:	soluzione web (Fig 2)	protocollo + api (Fig 3)
Vantaggi	Sviluppi una volta per sempre	Riduzione di banda richiesta
Svantaggi	Utilizzo di banda per aspetto grafico	Interfacce grafiche diverse su piattaforme diverse
Costo di Manutenzione	Bassa (Una volta per tutti)	Alta (Una per piattaforma)
Portabilità	Alta (la soluzione è già compatibile)	Bassa (di solito il porting è manuale)
Aziende di "Application Building" che usano questa soluzione	Tutte	Nessuna su multipiattaforma
Efficacia di questa soluzione	Media	Alta

*www.**apps-builder**.com/ è un azienda italiana che ha sviluppato portabilità e personalizzazione attraverso un web-service con applicazioni basate sullo schema in Fig 2*

Da questo punto in poi quindi si faranno considerazioni su di un'unica piattaforma, la piattaforma Android.

2.3 Pattern di applicazioni

Da un analisi delle applicazioni pubblicate sul market Android, si può notare in una visione ancora più generale, come gli applicativi mobile, su qualsiasi piattaforma, possano essere raggruppati in pattern o categorie sia riguardanti la struttura di funzionamento che i temi trattati, che in molti casi assumono schemi simili a quelli riportati prima.

Alcune potrebbero essere:

- Gallerie
- Applicazioni che fanno uso di Api Web predefinite
- Applicazioni di offerte commerciali
- Applicazioni di news Rss
- Webview / Web App ...
- Ed altre potrebbero assumere schemi di funzionamento riconducibili ad un pattern ma non esemplificati in questo testo…

Ogni pattern, visto come classe di applicazioni che realizzano la stessa funzionalità, può essere implementato attraverso un codice funzionale comune e poi personalizzato al volo in maniera guidata con l'uso di form o interfacce utente.

La personalizzazione di questi pattern, come nell'esempio delle webview di fig 2 (web-app,etc..), andrà ancora una volta a riguardare il file di configurazione ed eventualmente le risorse grafiche.

Un esempio di dati che potrebbero essere inseriti nel file di configurazione e di volta in volta personalizzati potrebbero essere i dati fiscali i dati di brand come logo dell'azienda del cliente e nome dell'applicazione.

Questo suggerisce che potrebbe essere utile riutilizzare il sorgente di un' applicazione "BASE" rilasciato con istanze diverse del file di configurazione a seconda del cliente richiedente.

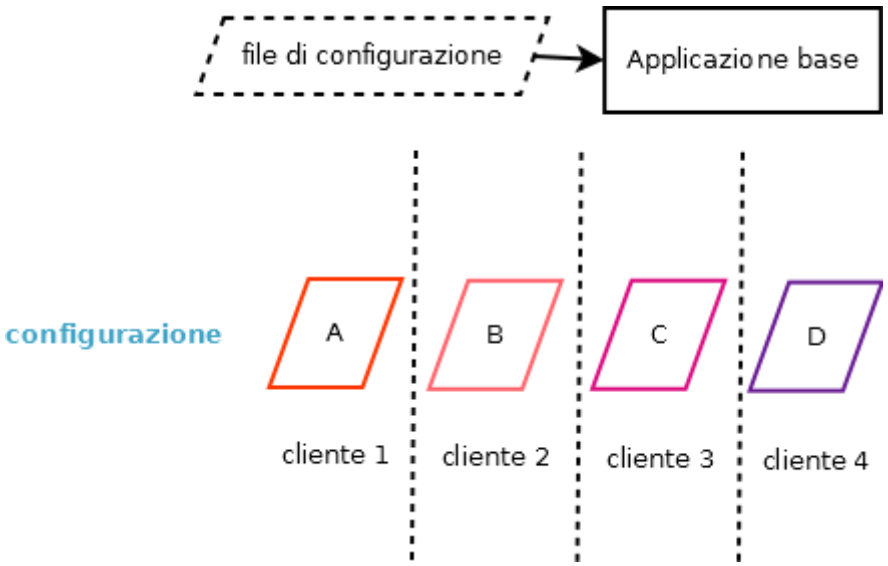


FIGURA 6: SCHEMA PERSONALIZZAZIONE APPLICAZIONE

A questo punto risulta chiaro il concetto di personalizzazione di applicazioni Android: riusare il codice sorgente di un'applicazione per diversi clienti con necessità simili attraverso , ad esempio, un file di configurazione d'istanza.

Questo apre la valutazione e l'analisi di una possibile implementazione.

Capitolo 3

Personalizzazione in Android Mediante WebService

3.1 Principi di funzionamento

Ora che abbiamo chiarito quale sia il significato di personalizzazione di applicazioni in ambiente Android, vediamo quali sono i passi per implementare questa tecnica.

Gli argomenti che vanno affrontati e analizzati per valutare la fattibilità sono:

- Sorvolata generale sul Sdk Android (Cap. 3)
- File di configurazione per applicazioni Android (Cap. 3)
- Signing Issues in Android (Cap. 4)
- Fattibilità su piattaforme web (Cap. 5)

L'idea è di implementare un Web Service che permetta di personalizzare applicazioni Android al volo attraverso un interfaccia grafica.

Il Web Service farà uso di un sorgente di applicazione Android "BASE" preparato in modo da avere un file di configurazione da cui trarre tutte le informazioni per il funzionamento.

L' interfaccia grafica del Web Service permetterà di raccogliere tutte le informazioni necessarie per riempire il file di configurazione.

Una volta riempito il file di configurazione dell'applicazione Android "BASE", questa viene pacchettizzata e rilasciata al richiedente direttamente attraverso l'interfaccia del Web Service.

L'esempio che verrà trattato nella demo è la personalizzazione di una WebView, quindi la personalizzazione di un'applicazione che permetta la visualizzazione di una pagina web, che potrebbe essere una pagina web di un particolare servizio dedicato al mobile (web-app) che si vuole rilasciare sul market oppure una qualsiasi pagina web anche privata.

Proprio le webview /webapp come già detto si addicono molto al tema trattato per il basso numero di parametri e quindi la veloce e semplice implementazione degli script di demo per la gestione del file di configurazione e dell'interfaccia grafica che ne risulta.

Il principio di funzionamento della piattaforma rimane tale anche in caso di applicazioni più complesse, quindi appartenenti ad un'altro pattern. Si potrebbe

quindi estendere il webservice a più classi di applicazioni, permettendo di generare la propria applicazione vetrina, l'applicazione di offerte commerciali della propria azienda, l'applicazione che ti informa di eventi oppure che ti invia informazioni riguardo un determinato tema, semplicemente inserendo la fonte dei dati che appartiene ad un determinato protocollo, ad esempio RSS, tra i parametri di configurazione.

Nei casi più complessi andranno trattate con particolare attenzione la modularità dei sorgenti del Web Service e l'interfaccia grafica, che dovrà essere adatta a raccogliere molte informazioni senza stressare troppo l'utente, dovrà inoltre fornire un'interfaccia per scegliere da quale applicazione base partire oppure poter permettere di comporla tramite moduli disponibili.

Vediamo alcuni schemi generali di funzionamento di web service, basati sulla personalizzazione di applicazioni.

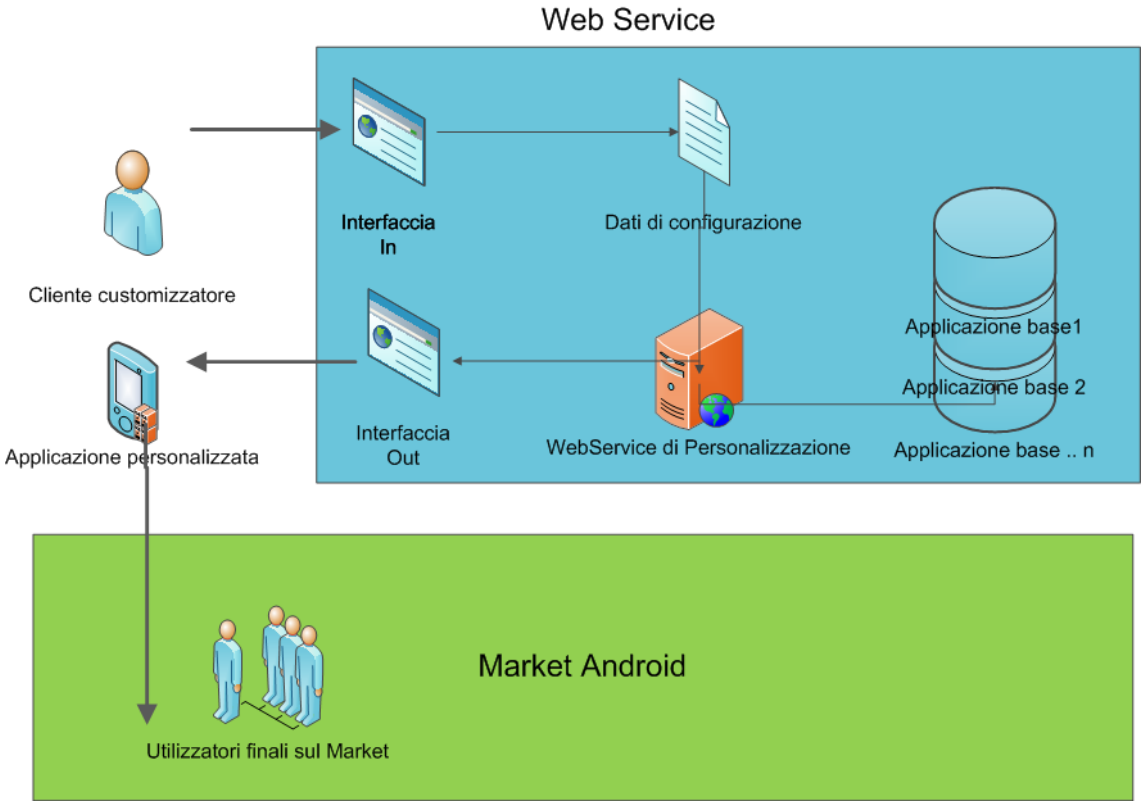


FIGURA 7: SCHEMA AGENTI ED ENTITA WEBSERVICE DI PERSONALIZZAZIONE

Vediamo che ci sono 3 principali blocchi,

1. **Il cliente customizzatore**, che usa il web service inserendo dati e costruisce cosi l'applicazione più vicina alle sue esigenze
2. **Il webservice**, che raccoglie i dati forniti dal cliente e in funzione di questi personalizza l'applicazione base e la rilascia tramite un'interfaccia di output
3. **Gli utilizzatori finali** sul market, che utilizzeranno l'applicazione personalizzata per usufruire dei servizi e delle informazioni del Cliente customizzatore

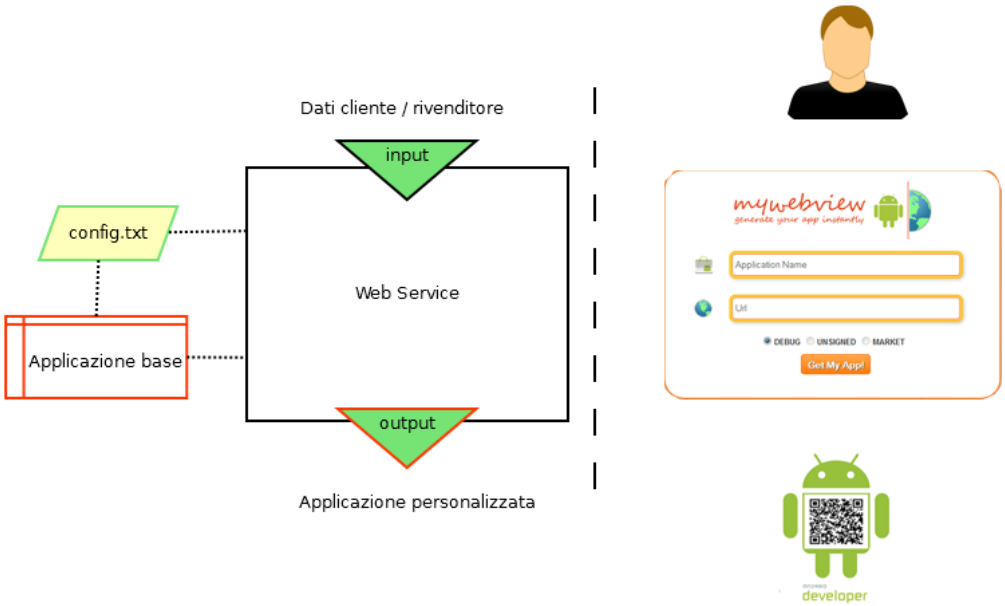


FIGURA 8: FUNZIONAMENTO WEB SERVICE DI PERSONALIZZAZIONE

Caso d'uso Esempio 1 WebView (implementato nella demo):

Utente A ha un'azienda web e vuole pubblicare un suo servizio web mobile sul market Android a costo ridotto senza riscrivere l'applicazione in modo nativo. Si reca sul Web Service, inserisce l'url del suo servizio web e il nome dell'applicazione. A questo punto il sistema Personalizza l'applicazione Base (Webview in questo caso) e Utente A ottiene senza sforzo un modo per essere nel market Android senza nuovi costi.

Caso d'uso Esempio 2 Applicazione Offerte Commerciali:

Utente A vuole sviluppare un'applicazione che mostri offerte commerciali ai suoi clienti a basso costo: si reca sul Web Service , utilizza l'interfaccia grafica, inserisce i suoi dati commerciali, il feed rss del suo suo sito e in pochi istanti genera l'applicazione di cui aveva necessità senza conoscere Eclipse, Tool di sviluppo o contattare Aziende Specializzate.

--

L'esempio potrebbe estendersi per altri tipi di applicazioni, cambierà come detto in precedenza l'interfaccia grafica del Web Service e l'applicazione base.

3.2 Tools e Sdk android

Risulta necessario analizzare, almeno in una visione generale, come funzionino i Tools e gli strumenti di sviluppo per applicazioni Android.

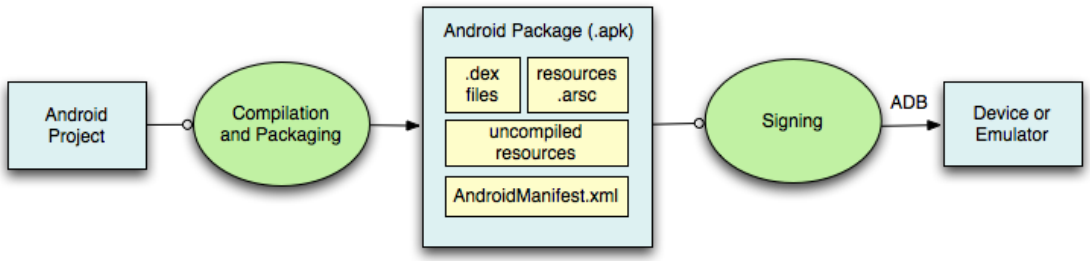


FIGURA 9: OPERAZIONI E STRUTTURA APK

Le applicazioni Android vengono sviluppate in linguaggio Java e con descrittori xml per utilizzati per dati di installazione e i metadati riguardanti l'applicazione stessa.

Lo sviluppo del sorgente può essere fatto attraverso l'uso di IDE come Eclipse, oppure senza IDE. Per quanto riguarda la parte di compilazione e generazione dei pacchetti invece si fà riferimento agli strumenti rilasciati da Google stesso, l'Android SDK.

L'android SDK ha, tra i vari compiti, l'obbiettivo di generare una variante del pacchetto JAR preparato per sistemi Android, denominato APK. In questo pacchetto, come nel progetto sorgente risiedono le risorse grafiche, alcune risorse testuali, certificati e i sorgenti compilati per la JVM presente su sistema Android, una versione riprogettata di JVM denominata Java Dalvik Virtual Machine. L'APK è un archivio frutto di tutto lo sforzo, andrà installato sui device e conterrà la nostra applicazione pronta all'uso.

Il Web Service dovrà utilizzare questi tool in modo molto simile a quello che viene fatto da Eclipse all'atto della generazione del package.

Un tool che è utilizzato per la generazione dei pacchetti APK è ANT, strumento utilizzato anche per lo sviluppo di pacchetti JAR o WAR in caso di sviluppo in piattaforma JEE o Tomcat. Sarà necessario preparare adeguatamente anche questo strumento sulle macchine che ospiteranno il Web Service.

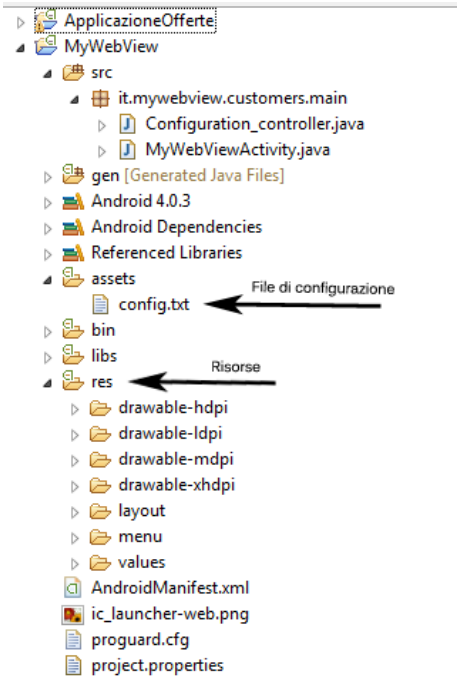
La necessità di usare questi tool, chiarisce che nel caso di sviluppo di un Web Service come quello di questo testo, non può essere usata una normale piattaforma di hosting, ma la scelta deve ricadere, per numerosi motivi su una macchina virtuale su cui è possibile installare questi servizi e configurarli in modo autonomo.

3.3 File di configurazione - Assets

Ora bisogna vedere se è possibile e se è presente una soluzione comoda, possibilmente supportata dalle librerie Android per utilizzare file di configurazione nell' applicazione base.

I dati devono essere presenti e coerenti al momento della compilazione, la soluzione più immediata, almeno per la demo in queste pagine è quindi costruire un file di testo con un formato definito ad hoc.

Le librerie Android supportano questo modo di operare attraverso il concetto di Assets.



Ecco quanto riportato dalla documentazione Android Developer:

"*Resources are an integral part of an Android application. In general, these are external elements that you want to include and reference within your application, like images, audio, video, text strings, layouts, themes, etc. Every Android application contains a directory for resources (`res/`) and a directory for assets (`assets/`). Assets are used less often, because their applications are far fewer. You only need to save data as an asset when you need to read the raw bytes. The directories for resources and assets both reside at the top of an Android project tree, at the same level as your source code directory (`src/`).*"

FIGURA 10: PROGETTO CON ASSETS

"The difference between "resources" and "assets" isn't much on the surface, but in general, you'll use resources to store your external content much more often than you'll use assets. The real difference is that anything placed in the resources directory will be easily accessible from your application from the `R` *class, which is compiled by Android. Whereas, anything placed in the assets directory will maintain its raw file format and, in order to read it, you must use the to read the file as a stream of bytes. So keeping files and data in resources (*`res/`*) makes them easily accessible."*

I progetti di applicazioni Android hanno un'organizzazione ben definita delle risorse, che sono organizzate e visibili direttamente durante lo sviluppo di un progetto, in questo caso dell'applicazione Base. E' possibile quindi astrarre dal filesystem ed utilizzare direttamente le librerie Android per accedere al file di configurazione come fosse una qualsiasi risorsa del progetto, ma utilizzando i dati che ci sono all'interno in formato grezzo.

L'applicazione Base quindi, oltre a realizzare la propria semantica, dovrà integrare alcune funzionalità per leggere i dati dal file di configurazione e renderli disponibili al resto del pacchetto. Và implementata quindi un'interfaccia per la lettura dei dati dal file di configurazione.

Avere un file di configurazione su cui è definita un'interfaccia di lettura ed essere un'applicazione Android sono i requisiti perchè l'applicazione sia quindi personalizzabile.

3.4 Problematiche di sicurezza su file di configurazione

Come scopriremo tra poco il file di configurazione, se inserito negli Assets una volta pacchettizzato sarà visibile in chiaro, anche se non modificabile poiché tutto il pacchetto andrà firmato. Si pone quindi un problema: non è possibile usare gli Assets se nei dati di configurazione sono presenti dei dati sensibili che non possono essere mostrati in chiaro.Il file di configurazione inoltre, per progetti grossi, diventa una soluzione un po' grezza e poco prestante, diventano naturali candidati i file di configurazione xml delle risorse di Android (vedi Fig 11) o l'utilizzo di un database SQLite.

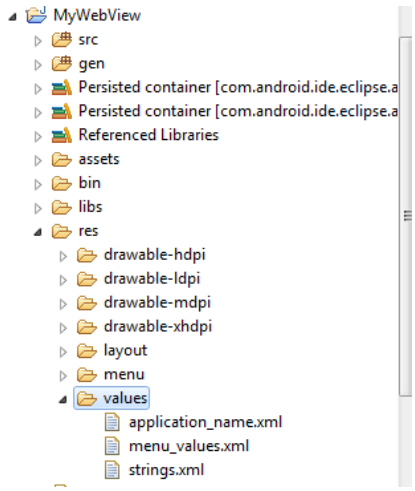


FIGURA 11: RES/VALUES DIRECTORY ESEMPIO

Se si sceglie di utilizzare i file xml presenti in /Res, una volta ottenuto il pacchetto apk, i dati contenuti in questi file non sono facilmente interpretabili, siccome la gestione delle risorse xml è interamente affidata all SDK Android che compilerà questi dati insieme al resto del codice Java.

Utilizzando invece un database SQLite dipende dall'implementazione che si intraprende, in un caso si converge alla soluzione fatta con gli assets e nell'altro alla soluzione degli xml delle risorse.

E' da verificare inoltre se la sicurezza di questi dati venga mantenuta una volta installata l'applicazione sul dispositivo mobile, valutando quindi se non sia una soluzione migliore utilizzare un protocollo sicuro e mantenere i dati sensibili su server remoto.

Di default nei sistemi Android le applicazioni girano con privilegi inferiori al livello di Root, questo meccanismo permette di implementare livelli di riservatezza sui database privati delle singole applicazioni una volta che queste sono installate. Ovviamente nel caso di privilegi di Root l'efficacia di questi meccanismi decade. E' chiaro quindi come la presenza di dati sensibili sia da valutare attentamente e a prima vista porti a concludere che una soluzione web+protocollo sicuro sia da preferire.

L'utilizzo di file Xml o Database SQLite locali non introduce nuove problematiche su come lavora il servizio web di personalizzazione fintanto che questi dati siano accessibili all'atto della pacchettizzazione. Il servizio al momento dell'immisione dei dati di personalizzazione invece che scrivere sul file di configurazione testuale userà una di queste due nuove soluzioni. Ne risulta che si complica un po' il codice inerente alla gestione della scrittura di questi file e dovrà essere adattato anche il modulo che sull'applicazione mobile legge i dati dalla configurazione.

Un punto importante è invece la scelta di gestire i dati sensibili attraverso web e sistemi esterni. Se ciò fosse scelto si dovrebbe modificare profondamente l'architettura di tutto il progetto. L'analisi della gestione di dati sensibili è un argomento così vasto, anche se limitato alla piattaforma Android che necessiterebbe di un'equivalente tesi in proposito. Questo è lasciato come possibile sviluppo.

Capitolo 4

Paternità e firme di applicazioni personalizzate

4.1 Packaging e signing dell'apk

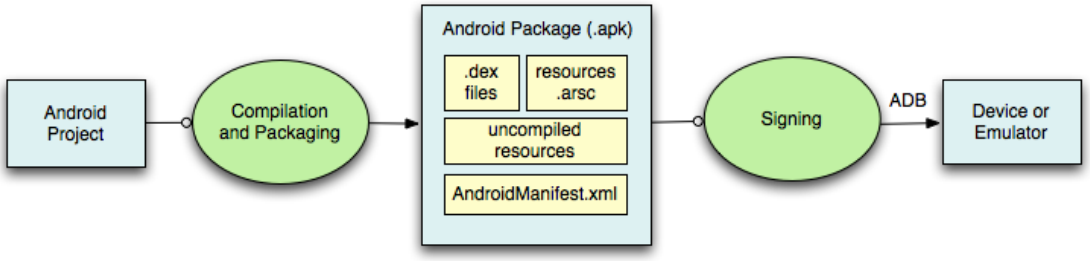


FIGURA 12: OPERAZIONI E STRUTTURA APK 2

Nell'introduzione ai Tool Android è stato presentato il formato di packaging che viene utilizzato per le applicazioni Android: il formato APK. Ora Analizziamo più in dettaglio quali sono le caratteristiche dei file Apk da curare per il rilascio di un'applicazione personalizzata utilizzando la soluzione di questo testo.

I file APK sono il risultato della compilazione e del packaging da parte dell'SDK Android attraverso un processo un pò complicato la cui descrizione lascio alla guida ufficiale: (Rif 2)

I sorgenti Java del progetto vengono compilati e offuscati per essere eseguiti dalla JDVM, nell'Apk finale quindi non saranno visibili in chiaro, mentre altre risorse come gli Assets e le risorse grafiche (immagini,loghi etc) rimangono in chiaro e visibili.

Dopo il packaging dell' Apk, è necessaria una procedura di firma. Questa procedura di firma è il punto critico di tutto il progetto. Banalmente si potrebbe pensare di poter compilare una volta per tutte il progetto base e di personalizzarlo aprendo il file APK e cambiando i parametri del file di configurazione, visto che questa risorsa è in chiaro. Questa procedura non è possibile ed è sconveniente per due motivi:

1. Il pacchetto deve essere firmato per poter essere installato su dispositivi Android. Quindi la modifica dopo la firma non è possibile.
2. Ogni applicazione, deve avere un package unico in tutto il Market e unico sul dispositivo Android. Questo rende necessaria la generazione in maniera dinamica e automatica del nome del package dell'applicazione personalizzata .Questa procedura a questo livello di analisi è possibile solo prima della compilazione dei sorgenti Java.

Detto questo è necessario personalizzare il package name, il file di configurazione e solo in seguito compilare. Questo intende che il processo di generazione dell'applicazione personalizzata costa di una compilazione per istanza. Costo affrontabile per un'applicazione di piccole dimensione, ma potenzialmente un problema nel caso di applicazioni di grandi dimensioni, questo discorso sarà affrontato nelle Ottimizzazioni Possibili alla fine del 5 capitolo.

4.2 Signing Issue

Ora analizziamo i punti critici della procedura di firma. La firma delle applicazioni Android è obbligatoria per poter installare l'applicazione.

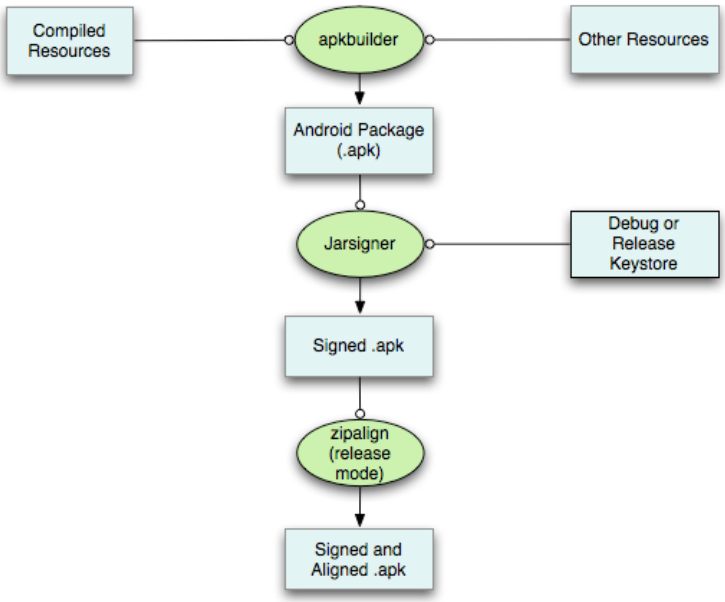


FIGURA 13: PROCEDURA DI FIRMA

La procedura di firma viene portata a termine da un tool integrato nell'sdk che si chiama Jarsigner. Questo tool prende in input una chiave Privata e l'apk non firmato, firma con la chiave privata il pacchetto che poi potrà essere rilasciato al pubblico sul market o attraverso web.

L'SDK usa un meccanismo automatico di generazione di chiavi, le chiavi generate da questo meccanismo vengono chiamate chiavi di Debug (Debug Keystore). Queste chiavi hanno dati fittizi e vengono generate per poter testare la propria applicazione durante la fase di test.

Con le chiavi di Debug non è possibile rilasciare alcuna applicazione sul market. A questo punto per poter caricare l'applicazione sul market è necessario generare una chiave privata reale (Release Keystore) con cui firmare le applicazioni personalizzate.

Tipo di chiave	Modalità di pubblicazione
Debug Keystore	Web e Diretta
Release Keystore	Market, Web e Diretta

* Con web si intende la pubblicazione dell'applicazione personalizzata direttamente dal sito dell'utente, tramite url o qr-code.

Questo è il principale problema da affrontare per questo tipo di sistema, con conseguente scelta progettuale.

Gestione di chiavi private

La principale scelta del progetto sta quindi nella soluzione a questo problema. Si tratta di riuscire a trovare una soluzione elegante che permetta all'utente del Web Service di poter personalizzare l'applicazione e poterla pubblicare sul market, in modo semplice ma senza violare le procedure di sicurezza da cui è nata l'idea di firma e crittografia asimmetrica.

4.3 Soluzioni e scelte progettuali

Detto quindi che con le chiavi di Debug, generate automaticamente non è possibile pubblicare nulla sul market, vediamo quali sono le possibili scelte progettuali per poter procedere.

1.	Generazione al volo di una Release Key
2.	Upload della Release Key
3.	Rilascio dell'Apk non firmato e rilascio di un tool per la firma assistita
4.	Rilascio dell'Apk firmato dal proprietario del Web Service

1. Generazione al volo di una Release Key *

Si tratta di raccogliere altri dati oltre a quelli necessari per la compilazione del file di configurazione. I dati minimi in questione sono una password e un nome. Questi dati verranno utilizzati per generare una Release Keystore dell'utente con cui verrà firmato il pacchetto Apk.

Pro	Contro
• Semplice da implementare • Non richiede conoscenze avanzate all'utente • Accounting dell'utente • Riuso chiave per altre applicazioni	• Ci si accolla la gestione di chiavi private di terzi. (Legal Issue?) • E' concettualmente sbagliato generare chiavi private di terzi

* possibile implementazione con chiave privata usa e getta?

2.Upload della Release Key

Simile alla soluzione precedente, ma sarà l'utente a caricare la propria chiave

Pro	Contro
• Semplice da implementare • Riuso chiave per altre applicazioni	• Condividere una chiave privata è sbagliato • Richiede conoscenza all'utente di gestione chiavi

3. Rilascio dell'Apk non firmato e rilascio di un tool per la firma assistita

Il Web Service rilascia l'Apk non firmato e un tool con interfaccia grafica in grado di assistere alla creazione della firma. Senza memorizzare dati personali su sistemi esterni da quello dell'utente.

Pro	Contro
• Rispetta tutti i principi della crittografia Asimmetrica • Riuso della chiave privata	• Soluzione non immediata • Dimensione del tool accettabile? • Manutenzione del tool accettabile?

4. Rilascio dell'Apk firmato dal proprietario del Web Service

Il Web Service utilizza una Release Key generata dal proprietario del Web Service, ad esempio Andrea Pola. Firmando le applicazioni personalizzate dei clienti con questa chiave.

Pro	Contro
• Semplice da implementare • Semplice da mantenere	• Non rispetta i principi della crittografia asimmetrica

4.3 Approfondimento su Android market e firme

Come detto all'inizio ecco un'approfondimento sui pregi e difetti della pubblicazione su Google Play. Senza scendere in confronti con la piattaforma Apple, vediamo quali sono i punti che ci interessano per l'implementazione di servizi di personalizzazione di applicazioni Android.

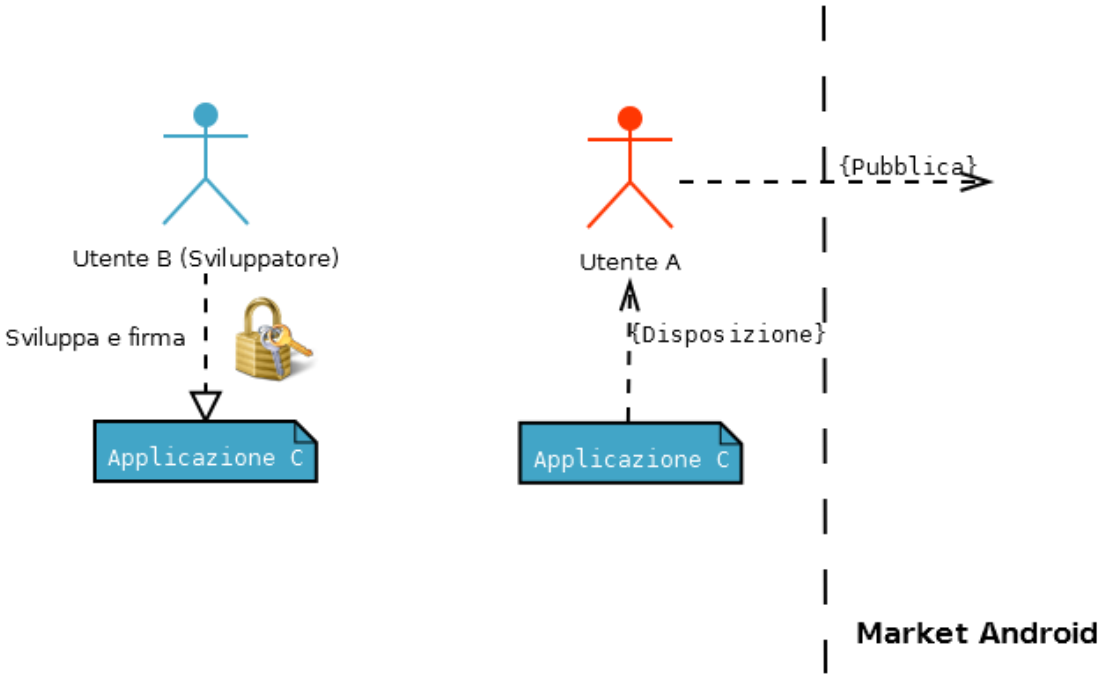


FIGURA 14: STRANEZZA DI MARKET ANDROID SU CONTROLLO FIRME

Il market Android permette, allo stato attuale delle cose, la pubblicazione da parte di un registrato al Market di applicazioni firmate da terzi. Per essere chiari, l'applicazione C sviluppata da B e firmata da B può essere pubblicata dall' utente A sul market senza problemi. Questa scelta a prima vista non rispetta i principi che stanno alla base delle firme digitali. Probabilmente questa scelta risulta una scelta commerciale, pensata per far accedere al market figure che non sono direttamente gli sviluppatori delle applicazioni.

Questa politica quindi non scarta nessuna delle scelte progettuali presentate prima anche se queste non rispettano i principi per cui è nata la firma digitale.

Ma approfondiamo la scelta progettuale n.4:

Poniamo il caso che un utente del Web Service di Demo voglia generare un'applicazione di un sito web di cui lui non è il proprietario, ad esempio una testata giornalistica. L'utente va sul Web Service che personalizza WebView (il caso studio) e genera un'applicazione di questo sito web pronta per il market,

secondo la scelta progettuale 4, firmata dal proprietario del Web Service. A questo punto l'utente pubblica sul market.

In caso di denuncia, che responsabilità ha il firmatario?

Il firmatario in questo caso è il proprietario del Web Service, che è una persona diversa dalla persona che pubblica sul market l'applicazione.

E' chiaro come questo processo sia in contraddizione con i principi legali della firma digitale. E' evidente inoltre che questo processo e i servizi di personalizzazione delle applicazioni debbano essere sostenuti da una "licenza d'uso" e un comparto legale accurato.

Anche se formalmente e tecnicamente la soluzione più pulita è la soluzione n.3 la presenza di un Tool per la firma assistita potrebbe complicare la procedura di rilascio sul market, spaventare l'utente customizzatore che potrebbe non essere abituato a procedure di questo tipo. Le soluzioni commerciali purtroppo non adottano scelte progettuali di questo tipo, preferendo la n.2 o la n.4

4.4 Stato dell'arte

Ora vediamo quali sono le soluzioni commerciali presenti sul web che forniscono servizi di personalizzazione di applicazioni. Dal punto di vista commerciale queste soluzioni vengono nominate come "Application Builders" o nomi simili. Soluzioni di questo tipo stanno comparendo e sviluppandosi molto velocemente al momento della scrittura di questo documento.

Valutiamo quali scelte progettuali sono state applicate e come è stato affrontato il problema precedente, limitandoci al contesto Android in caso di soluzioni multipiattaforma.

App Inventor

http://appinventor.mit.edu/

Ritornato da poco, dopo una storia un pò difficoltosa. Servizio lanciato ufficialmente da Google tramite i Google Labs, poi chiusi, ora è disponibile grazie al suo rilascio Open Source e grazie al Mit che ha preso in mano il progetto e rilanciato da poco come Beta. Si tratta di un progetto per lo sviluppo di applicazioni, è stato il primo nel suo genere. E' un sistema un pò diverso da quello analizzato in questo testo, è di fatto un tool per la programmazione visuale ad alto livello.

Pro	Contro
• Potente • Tendenzialmente un Tool per lo sviluppo con interfaccia grafica • Alte potenzialità didattiche	• Non permette il rilascio di applicazioni nel market* • Servono conoscenze di programmazione

*alla prima stesura di questo testo non era permesso il rilascio su market, ora è stata implementata una gestione delle firme secondo soluzione n.2

Buzz Touch

http://www.buzztouch.com/

Progetto per il building di applicazioni, semplice, multipiattaforma, ma che rilascia il codice sorgente e non l'eseguibile. Il sorgente viene lasciato con istruzioni per firma compilazione e pubblicazione. Il nome dell'applicazione deve essere disponibile, in sostanza è presente un meccanismo di mutua esclusione sul nome dell'applicazione (dovuto alla gestione del package name probabilmente).

Pro	Contro
• Implementazione semplice • Multipiattaforma	• Non risolve il problema • Mutua esclusione sul nome dell'applicazione

The AppBuilder.com

http://www.theappbuilder.com

La versione per Store/Market è a pagamento quindi non ho verificato quali sono meccanismi di gestione delle chiavi siano presenti. L'applicazione generata viene pubblicata attraverso il loro Account nel market. Probabilmente viene fatto un controllo manuale dell'applicazione finale. La firma ipotizzo sia a nome di The AppBuilder. Molto potente è il sistema di gestione degli aggiornamenti dell'applicazione. In sostanza l'applicazione è sempre aggiornata grazie al loro sistema "In the Cloud". Probabilmente le implementazioni sono attraverso Web App.

Di fatto stai acquistando un'applicazione loro tramite abbonamento.

Pro	Contro
• Potente • Applicazioni Sempre aggiornate • Multipiattaforma • Soluzione commerciale valida	• Poca autonomia • Aggira il problema • Pubblicazione sul loro Account • Controlli manuali

Apps-Builder

http://www.apps-builder.com/

Apps Builder è sicuramente la soluzione più interessante, inoltre è italiana, sviluppata al politecnico di Torino e poi diventata azienda. E' multipiattaforma e ha un bel set di pattern a disposizione, con interfaccia grafica intuitiva e potente. Il punto critico di questa realizzazione è che nel reparto Android l'applicazione viene firmata a nome dell'intestatario dell'azienda.

Questa azienda adotta la scelta progettuale n.4* .

E' stata lanciata una soluzione per partner che fa uso un CMS per la gestione delle applicazione, siccome è un progetto per partner Business to Business con contratto non è stata avanzata l'analisi.

Utilizza soluzioni web-app based.

Pro	Contro
• Multipiattaforma • Ampia scelta • Interfaccia grafica ben realizzata • Primo a lanciare una soluzione B2B	• Firma dell'intestatario dell'azienda / a nome dell'azienda

*alla prima stesura di questo testo era rilasciato con scelta progettuale n.4 ora è stata cambiato in scelta progettuale n.2

Concludendo questa parte sullo Stato dell'arte, possiamo vedere che le soluzioni in commercio utilizzano diverse modalità per aggirare o risolvere il problema di firma trattato. Sarebbe interessante realizzare una soluzione che risolva il problema delle firme dal punto di vista tecnico e nel rispetto dei principi della firma digitale. Anche in vista di possibili nuove politiche del market Android.

4.5 Approfondimento su AppInventor

AppInventor merita un paragrafo dedicato, poiché si tratta di un progetto molto interessante, anche se dal punto di vista commerciale non ha ricevuto successo e probabilmente rimarrà un progetto Beta per tutta la sua vita. Si tratta di un sistema on-line (alcuni moduli vengono avviati sulla macchina locale) per il design e lo sviluppo di applicazioni mobile esclusivamente Android che utilizza tutte le tecnologie e le tecniche di cui abbiamo parlato fino ad ora. Possiamo assimilare questo progetto al tema trattato soprattutto per il discorso di pacchettizzazione e gestione firme.

Mentre l'interfaccia di front-end è molto complessa, permette infatti di comporre la propria applicazione con moduli in corrispondenza uno ad uno alle librerie e le classi del SDK Android, la parte conclusiva del workflow di questo progetto può assomigliare alle implementazioni commerciali. Ma questo è ovvio siccome se deve essere generata un'applicazione per terzi è necessario affrontare i temi trattati nei precedenti paragrafi quali building , firma...etc

AppInventor è un progetto, come già detto, gestito dal MIT e sviluppato inizialmente da Google che lo ha reso Open Source. E' utilizzato per i primi approcci alla programmazione grazie alle sue interfacce che assimilano la programmazione alla costruzione di schemi a blocchi.

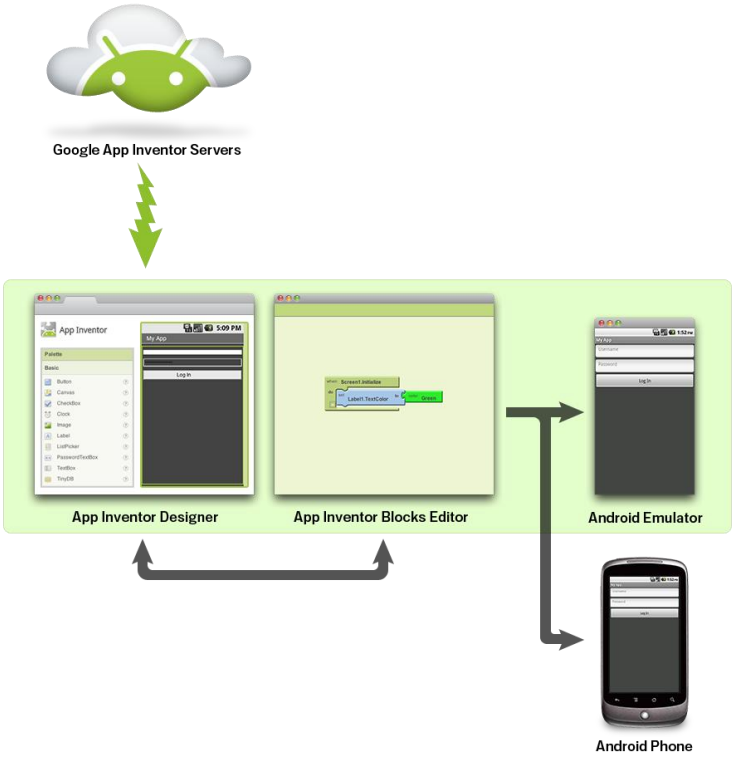


FIGURA 15: SCHEMA MODULI APPINVENTOR

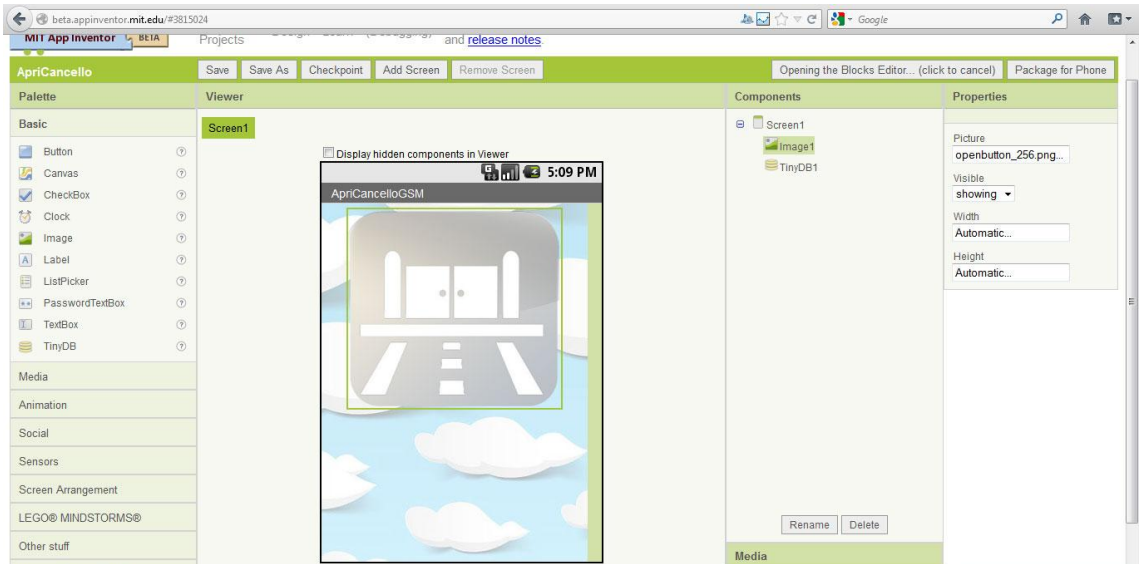


FIGURA 16:INTERFACCIA DESIGN APPINVENTOR

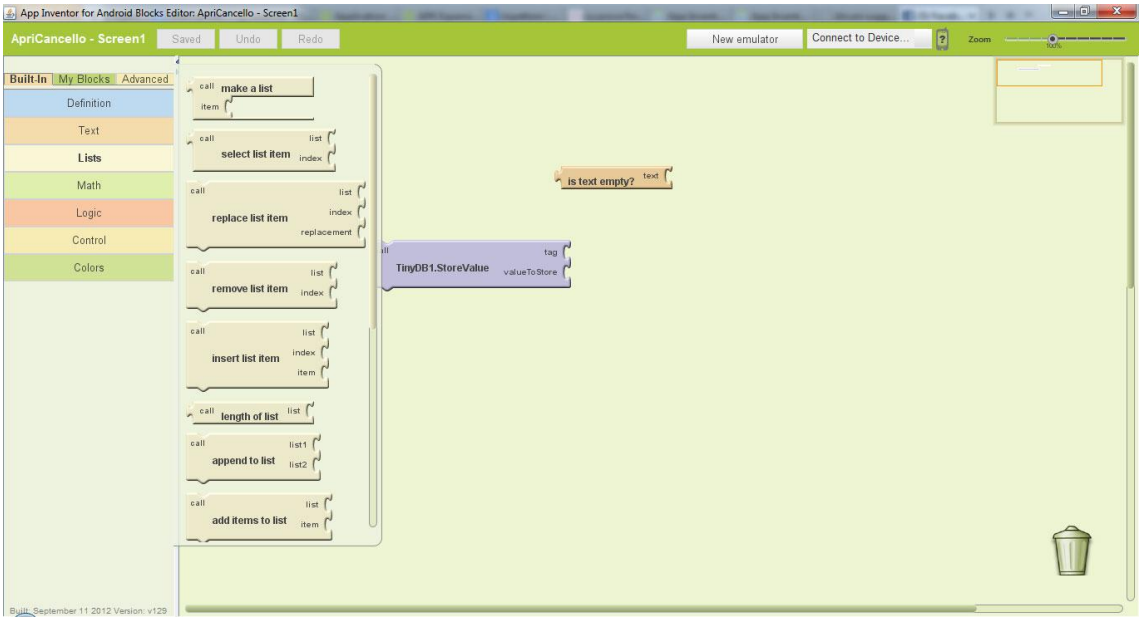


FIGURA 17: INTERFACCIA MODULI/CODICE APPINVENTOR

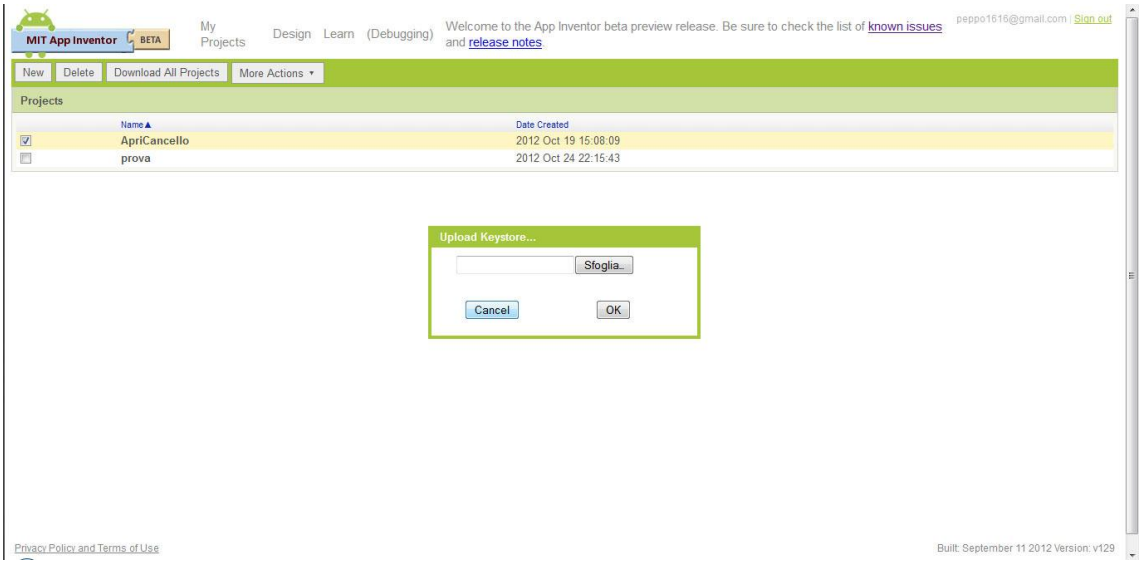


FIGURA 18:SCHERMATA DI RICHIESTA KEYSTORE

Le interfacce di AppInventor sono in pratica quelle di uno strumento di sviluppo ma con Editor **WYSIWYG (What You See Is What You Get)**. AppInventor realizza un obbiettivo di fatto superiore alla personalizzazione di applicazioni Android. La personalizzazione utilizza un'idea simile ma costruita con moduli che abbiano già una semantica apprezzabile ad un commerciante, un obbiettivo utile al cliente che vuole realizzare la propria applicazione. Parliamo appunto di cliente e non di programmatore.

Le potenzialità del progetto permettono di sviluppare applicazioni che con uno strumento di Application Building commerciale non si potrebbero fare, il problema è che la libertà di soluzioni realizzabili è strettamente legata dalle conoscenze e dall'esperienza in programmazione del cliente che sta realizzando la propria applicazione. Il livello di conoscenze richiesto è troppo elevato per un target commerciale, portando una persona con basse conoscenze di programmazione a realizzare un'applicazione scadente o non funzionante, cosa che in contesto a pagamento deve essere comunque garantita.

Le soluzioni generate su AppInventor sono totalmente native .

In Fig 19 è mostrata la finestra di upload della propria chiave privata, in conformità della scelta progettuale n.2.

4.6 Approfondimento su Apps-Builder

Come rappresentante delle soluzioni commerciali invece è doveroso analizzare, almeno superficialmente la soluzione di personalizzazione di applicazioni torinese Apps-Builder. Citata da importanti testate giornalistiche italiane e riviste del settore tecnologico/informativo utilizza un'interfaccia grafica molto pulita e funzionale, che permette di costruire la propria applicazione utilizzando moduli con una semantica apprezzabile ad un 'utente che vuole realizzare un servizio partendo dalle proprie risorse web.

Siamo ad un livello di astrazione dalla piattaforma mobile molto più elevato rispetto ad AppInventor. Questo webservice permette di dare in output la propria applicazione sia per Iphone che Android che Windows Phone, estendendo oltretutto il servizio con la possibilità di pubblicare l'applicazione anche su facebook, Chrome Store etc.. Per fare questo si è spostata l'implementazione dalla parte mobile alla parte web, come abbiamo già analizzato nel Capitolo 2 (vedi fig 2).

FIGURA 19:PREZZI,FUNZIONALITÀ E PIATTAFORME APPS-BUILDER

Ecco di seguito alcuni screenshot effettuati sul sito apps-builder.com.

E' da notare come l'interfaccia non sia molto diversa da App-inventor, è presente anche qui infatti un editor **WYSIWYG** che utilizza però moduli con obbiettivi già definti, ad esempio: Modulo Rss, Shop, Sito Web, Galleria online, Immagini. E' possibile anche inserire codice come soluzione opzionale, il codice può essere javascript,html o css, sottolineando l'implementazione web dell'applicazione personalizzata.

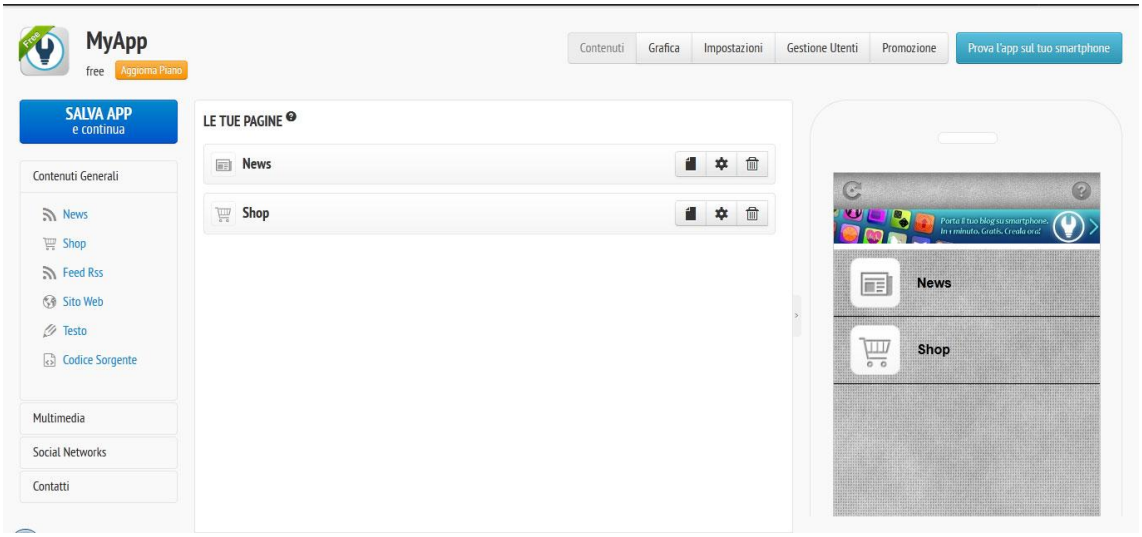


FIGURA 20: INTERFACCIA 1 APPS-BUILDER

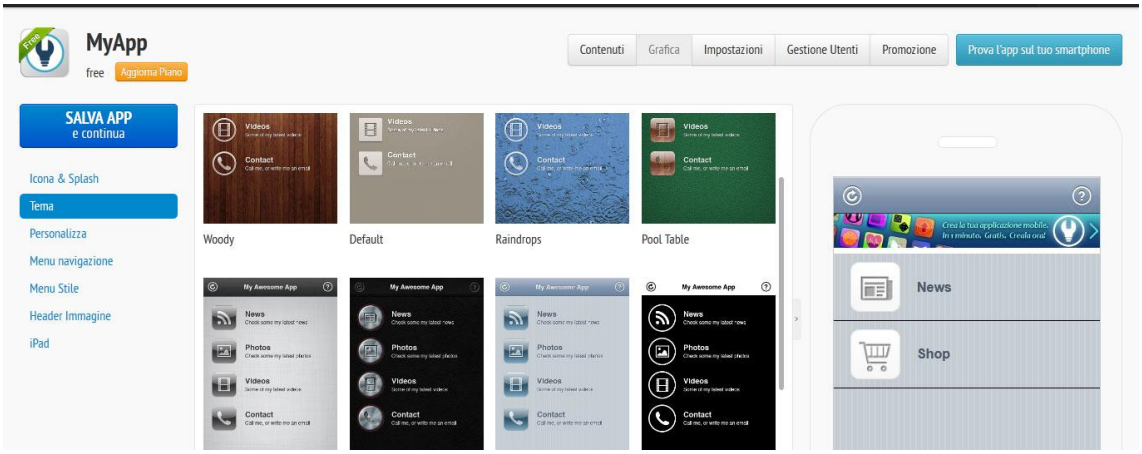


FIGURA 21:INTERFACCIA 2 APPS-BUILDER

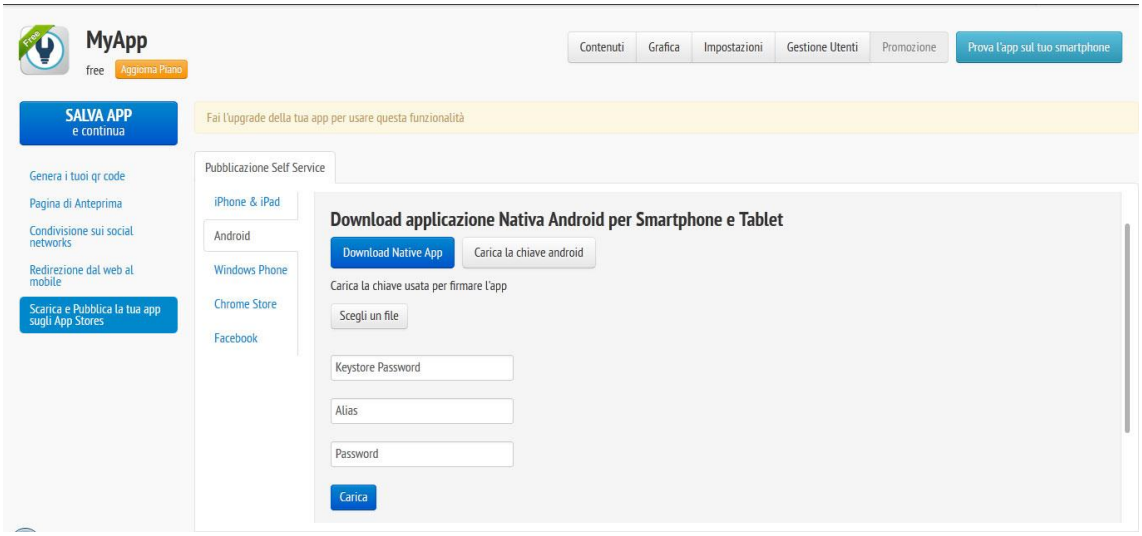


FIGURA 22: SCHERMATA INSERIMENTO KEYSTORE APPS-BUILDER

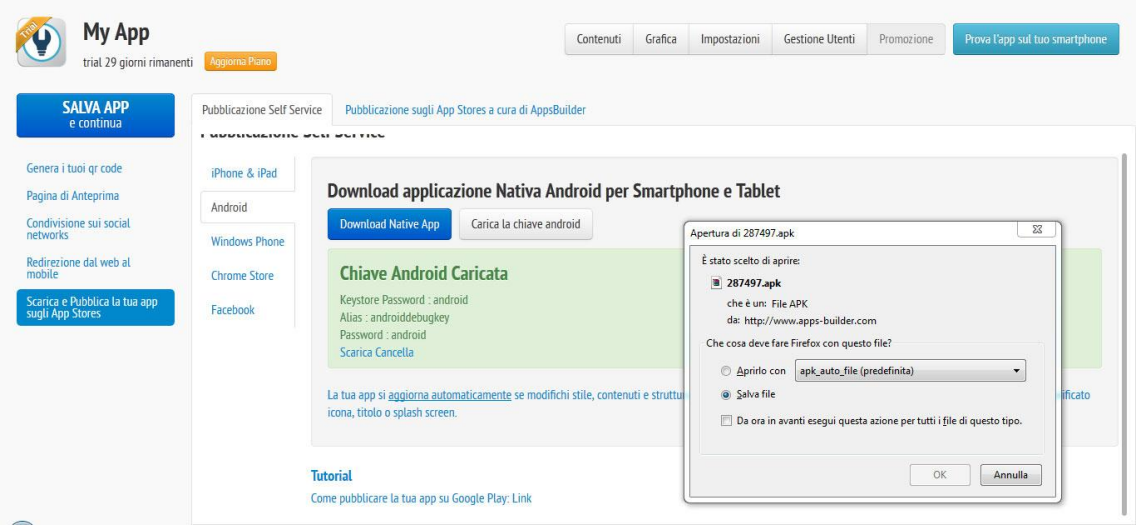


FIGURA 23: KEYSTORE ANDROID E DOWNLOAD APK FIRMATO

Notiamo nella Fig 24 che è stata implementata la scelta progettuale n.2. Non è ben chiaro perché in AppInventor la stessa scelta progettuale non necessiti della password di keystore, dell'alias e di un'ulteriore password e in questo caso invece si …

In ogni caso questa soluzione rimane il riferimento più efficacie per quanto riguarda la personalizzazione di applicazioni.

Aggiornamento politiche di Apps-Builder

E' infine interessante vedere che pochi mesi prima dalla stesura di questa tesi, durante l'analisi fatta per il preludio a questo testo (il documento di progetto curriculare), l'azienda torinese adottava una politica di gestione delle firme diversa da quella attuale, rilasciando l'applicazione firmata con una firma a nome dell'azienda.

Probabilmente è stato analizzato il rischio della politica adottata e la non coerenza con i principi della firma digitale e di conseguenza adottata una politica meno rischiosa per l'azienda ma... ad una nuova analisi c'è qualcosa che non và.

Ecco lo screenshot che mostra l'adozione della politica n.4 sul pacchetto scaricato quando era attiva questa scelta progettuale nella versione precedente di Apps-Builder . I dati sono reperiti attraverso il Tool Jarsigner che permette di vedere i metadati del firmatario di un'applicazione Android.

Il download del pacchetto di dimostrazione è presubilmente da datare tra Aprile e Maggio 2012. In dettaglio il pacchetto è del 18 Aprile 2012 come si nota dalla

data di ultima modifica del pacchetto, la data appartiene al, periodo in cui stavo facendo l'analisi delle piattaforme commerciali disponibili.

Firma Aprile 2012: Adozione scelta progettuale n.4

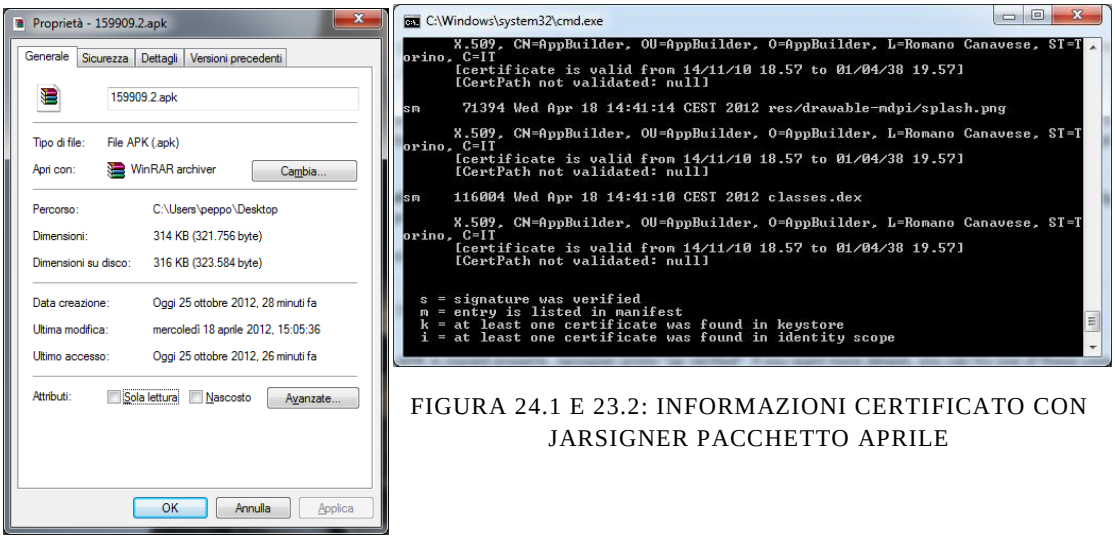


FIGURA 24.1 E 23.2: INFORMAZIONI CERTIFICATO CON JARSIGNER PACCHETTO APRILE

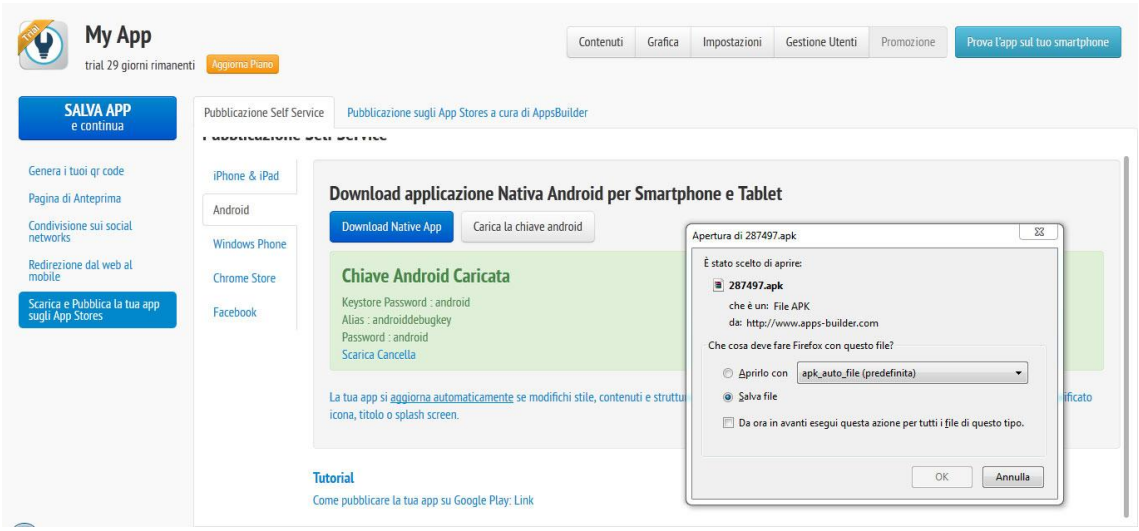
Il comando dato è jarsigner -verify -verbose -certs sul file 159909.2.apk

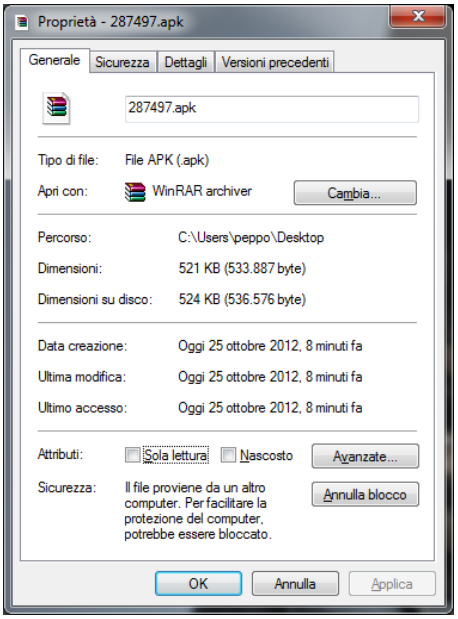
Si vede che il certificato utilizzato per la firma è di tipo X509 e sono mostrati i dati del firmatario, in questo caso AppsBuilder con sede in provincia di Torino, appunto Romano Canavese confermando l'adozione della scelta progettuale n.4 E' presente anche il tempo di validità del certificato. Succede che i certificati X509 abbiano di solito una Certification Autority abilitata a rilasciare certificati per terze parti, attuando cosi una catena di certificazione. Nel caso di Android parlare di catene è ancora prematuro, sarebbe interessante estendere questo discorso, non verrà fatto in questo testo ma lasciato come possibile sviluppo. Come vediamo anche nel prossimo screenshot questi APK sono verificati ma non mantengono la catena di certificazione.

Come le documentazioni ufficiali riportano, ovviamente (lo abbiamo già fatto in tutti i momenti in cui parliamo di firma dell'apk) è possibile la realizzazione di autocertificati.

*«The certificate does not need to be signed by a certificate authority: it is perfectly allowable, **and typical**, for Android applications to use self-signed certificates.»*

Firma Ottobre 2012: Adozione scelta progettuale n.2, ma non funzionante?





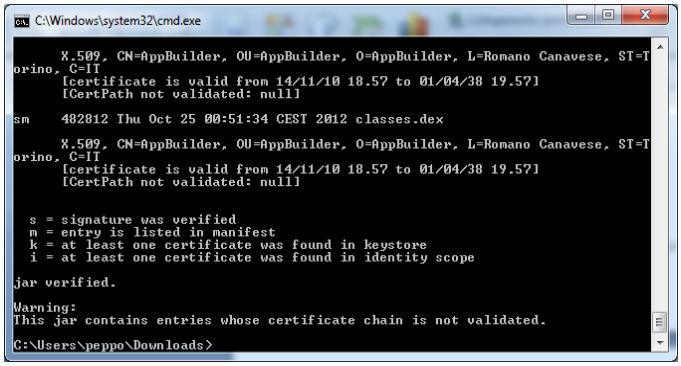


FIGURA 25:INFORMAZIONI CERTIFICATO JARSIGNER PACCHETTO OTTOBRE

Il comando dato è jarsigner -verify -verbose -certs sul file 287497.apk

Anche se sul sito ho caricato la keystore di debug, questa chiave non è stata applicata durante la firma. Il risultato infatti doveva mostrare dati a riguardo di Google e Debug key ma cosi non è stato.

Concludiamo che o sono sorti problemi durante le mie verifiche oppure alcune delle funzionalità a riguardo della gestione delle firme su AppsBuilder sono ancora da sistemare, mostrando come le tematiche affrontate fino ad ora siano fresche e ancora in evoluzione.

Capitolo 5
WebService di demo

E' stato analizzato lo stato dell'arte delle piattaforme commerciali e abbiamo visto che sono presenti molte soluzioni che fanno uso appunto di webapp/webview per realizzare portabilità, facilità di aggiornamento e manutenibilità delle applicazioni personalizzate dei clienti.

Il progetto di Demo è la realizzazione di un webservice per la generazione di webview come già detto nel capitolo 3. Si tratta in pratica di sviluppare un sottoinsieme delle funzionalità dei sistemi commerciali. Le realizzazioni commerciali infatti realizzano grandi e potenti interfacce per la personalizzazione delle applicazioni, offrendo molte scelte tra i moduli. In questo testo invece, ci concentriamo su quella che è l'implementazione del modulo di generazione dell'apk per Android e implementando un'interfaccia molto semplice per la raccolta di un url e il nome dell'applicazione. Lo schema seguente aiuta a capire dove si sono concentrati gli studi , mostrando un diagramma a blocchi di un progetto commerciale e mostrando in arancione il blocco analizzato in questa tesi.

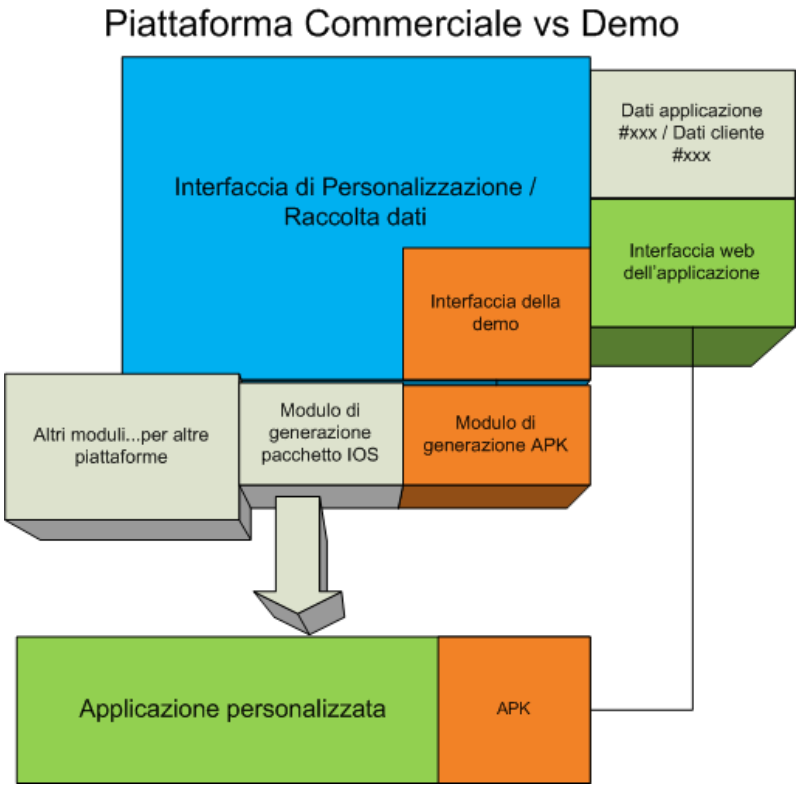


FIGURA 26: MODULI DI UN PROGETTO COMMERCIALE DI APPLICATION BUILDING

Riprendendo un esempio fatto nel capitolo 3 ecco il caso d'uso della piattaforma di demo:

Utente A ha un'azienda web e vuole pubblicare un suo servizio web mobile sul market Android a costo ridotto senza riscrivere l'applicazione in modo nativo. Si reca sul Web Service, inserisce l'url del suo servizio web e il nome dell'applicazione. A questo punto il sistema Personalizza l'applicazione Base (Webview in questo caso) e Utente A ottiene senza sforzo un modo per essere nel market Android senza nuovi costi.

Possiamo guardare il progetto di demo come un modulo implementato in tutte le soluzioni commerciali.

Se le soluzioni commerciali utilizzano un web service per la personalizzazione sicuramente il problema è affrontabile in ambiente web ma vediamo quali requisiti deve avere il sistema su cui andremo a ospitare il webservice di demo.

5.1 Requisiti dell'ambiente web

Requisiti sistema host

Ambiente JDK	OK
ANT	OK
Android SDK	OK
LAMP	OK

Tutti i requisiti sono installabili senza problemi. La demo è stata preparata su ambiente debian. Unico dettaglio và detto su l'ambiente JDK, poiché vanno configurate le variabili d'ambiente per il funzionamento dell'android SDK.

Problematiche tecniche di programmazione

Generazione dinamica di package name	OK
Compilazioni concorrenti della stessa applicazione base	OK
Modalità di firma	DEBUG

Generazione dinamica di package name: questa problematica è stata risolta facendo "search and replace" tra i sorgenti dell'applicazione base. Parte del package name base (che è costante) viene sostituito con una stringa generata dai dati forniti dall'utente, l'id dell'applicazione.

Compilazioni concorrenti della stessa applicazione base: questa problematica è stata risolta con l'uso di directory temporanee generate in funzione dei dati forniti dall'utente. La versione di demo del progetto, personalizza un'applicazione fermandosi alla firma in modalità Debug.

5.2 Implementazione della demo

Brevemente, il cliente customizzatore arriva sul sito e riempie la semplicissima interfaccia di personalizzazione, inserendo nome del sito web e nome dell'applicazione. Il sistema scrive il file di configurazione memorizzando l'url, compila il sorgente e restituisce l'apk firmato in modalità debug. L'applicazione personalizzata sarà una webview verso l'indirizzo indicato al momento della personalizzazione.

Ecco l'intefaccia di raccolta dati e l'interfaccia di output.

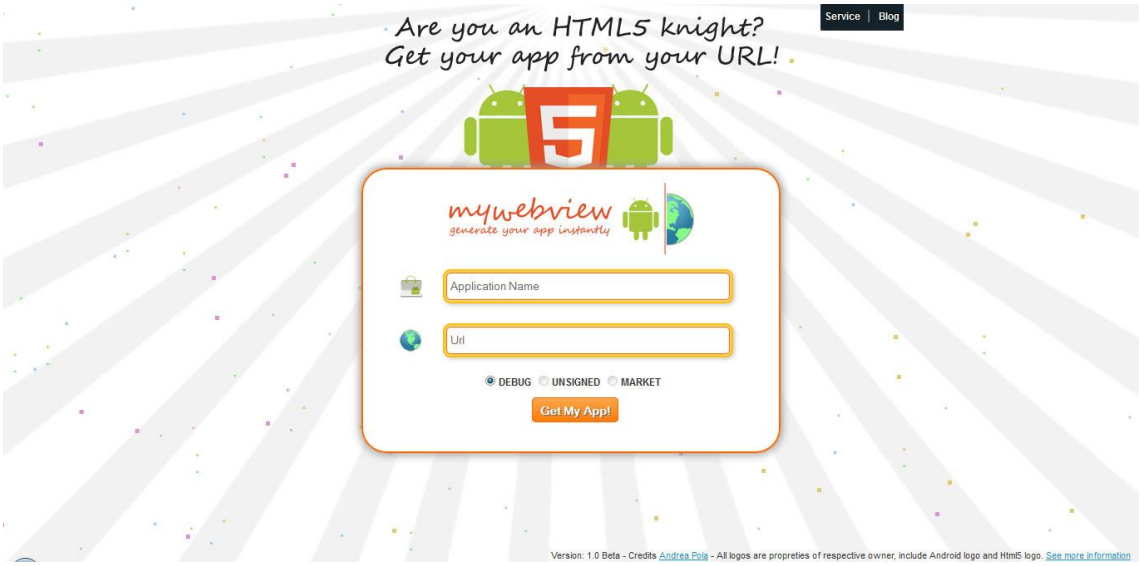


FIGURA 27: INTERFACCIA DI INPUT PROGETTO DI DEMO

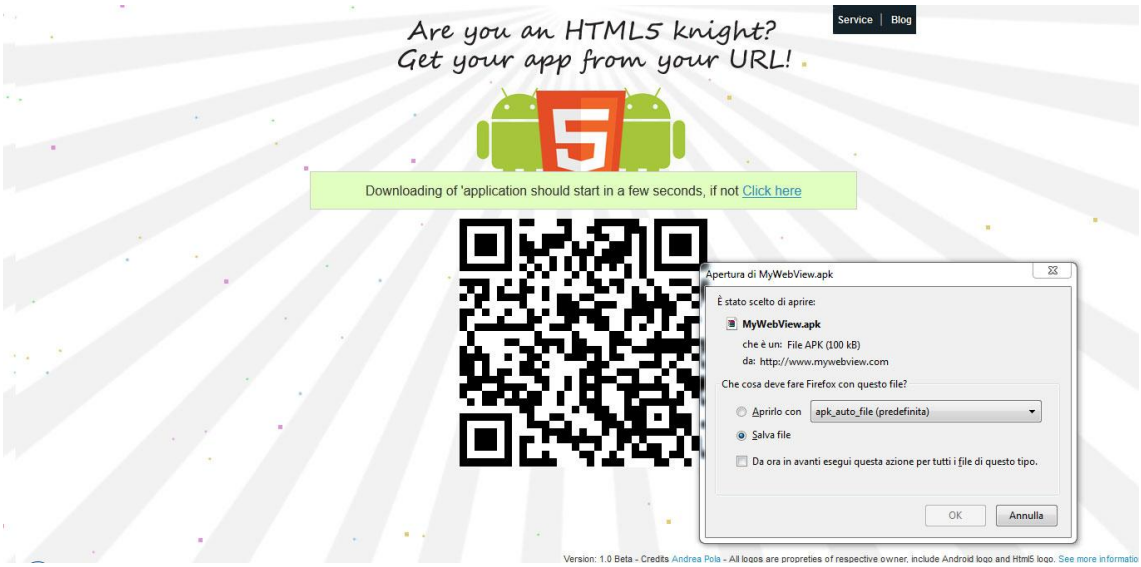


FIGURA 28:INTERFACCIA DI OUTPUT PROGETTO DI DEMO

Il Web Service si mostra con un'interfaccia grafica molto semplice, si tratta di un form con 2 input, uno per il nome dell'applicazione e l'altro per l'url. Questi due dati vengono inviati tramite chiamata POST ad uno script PHP che genererà un nome temporaneo dai due dati in ingresso, questo sarà l'identificativo dell'applicazione e della directory temporanea ad essa assegnata. L' id è generato con funzione md5 a partire da dagli input e per evitarne la previsione è stata inserita anche una stringa in modo da realizzare una semplice funzionalità di Salting.

Nell'attesa di una risposta da parte del server, il form viene sostituito da una pagina di attesa.

Questi 3 dati: url, nome e directory temporanea verranno passati ad uno script shell, che gestirà le directory per la mutua esclusione e riempirà il file di configurazione dell'applicazione base. Fatto questo lo script chiama ANT per la generazione dell'apk. ANT e l'sdk android ,chiamato a sua volta, utilizzano una chiave di Debug precostruita e depositata in una directory predefinita per la firma dell'Apk.

Una volta concluso il processo di compilazione e packaging, l'Apk generato viene depositato in una directory di download.

A questo punto lo script Shell ha concluso, l'esecuzione ritorna allo script PHP che restituirà risposta alla chiamata POST avviata dal form. La risposta conterrà il codice identificativo dell'applicazione appena generata. Nella pagina di attesa sarà inserito un link ad uno script che si occuperà di permettere il download l'APK. Infine viene visualizzato un qr-code che punta all'applicazione appena creata per il download diretto su cellulare.

Per la comunicazione script PHP -> Shell è stata utilizzata una chiamata exec. Questo deve essere fatto con cura, poichè i dati passati alla shell sono variabili inserite dall'utente, questo significa che potrebbe essere inserito codice malevolo o comandi shell non previsti. Per questi motivi sono stati usati filtri forniti dalle librerie PHP per l'Excaping di caratteri.

Input Utente	Url, Nome Applicazione
Input PHP	Url, Nome Applicazione
Input shell script	Url,Nome Applicazione, Directory Temporanea
Output Shell Script	Stringa di conferma
Output PHP Script	APP ID
Output Utente	APK URL, QR-CODE

Schema di funzionamento tecnico:

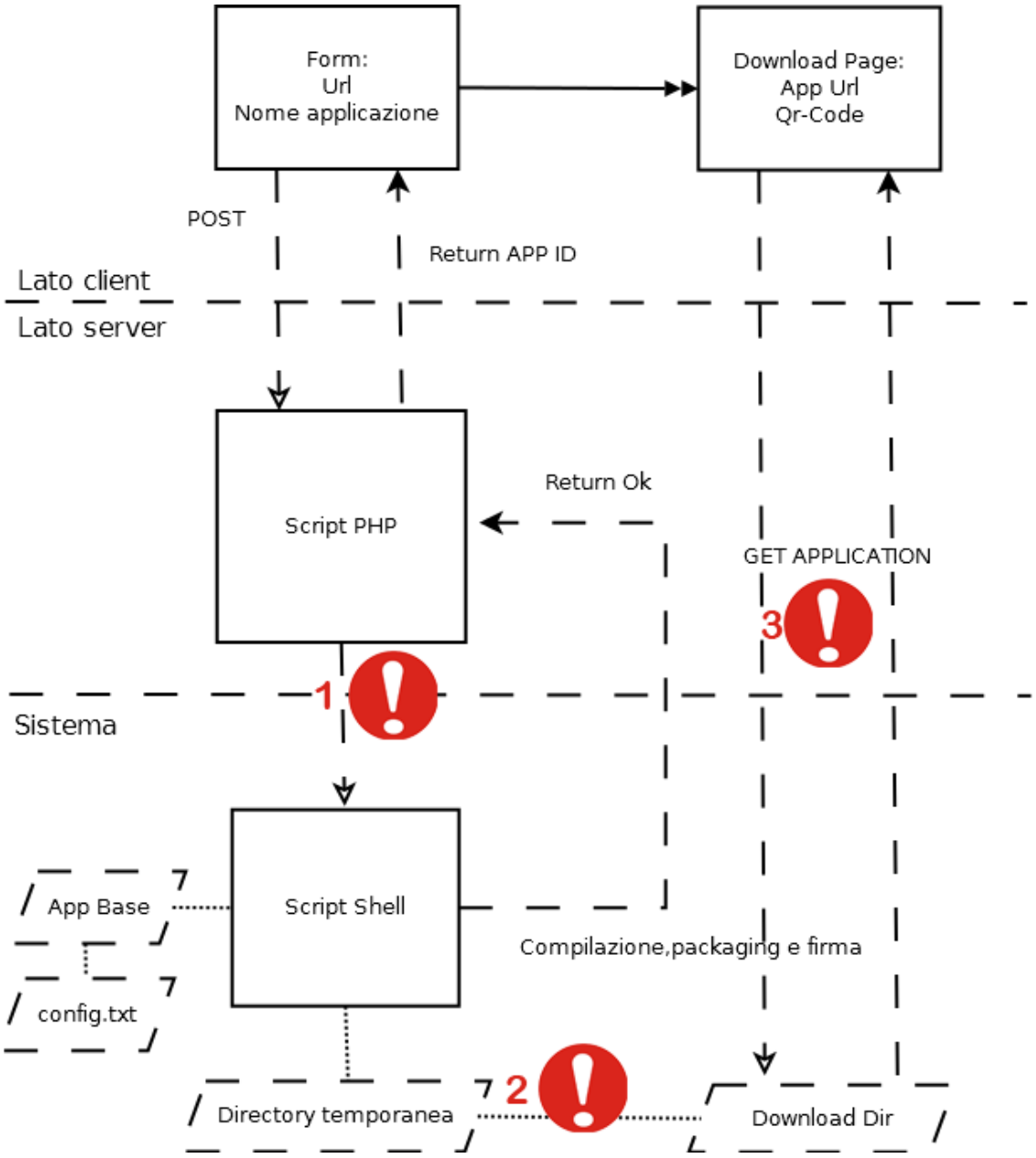


FIGURA 29: SCHEMA IMPLEMENTAZIONE WEBSERVICE DI DEMO

Dal basso verso l'alto si passa da una visualizzazione di interfacce utente fino a una visione sul file system della macchina che ospita il webservice.

- Lato client: inteso visualizzazione delle pagine web lato browser
- Lato server: siamo sulla macchina che ospita il webservice e stiamo eseguendo codice php in ambiente Web Server Apache
- Sistema: stiamo eseguendo script sul file system della macchina host

5.3 Dettagli sul codice

Come è evidente nello schema sono stati inseriti dei punti esclamativi che identificano i punti critici dell'implementazione a riguardo di sicurezza del sistema host e sicurezza dei dati.

I prossimi punti sono associati ai rispettivi punti esclamativi sullo schema di funzionamento

1) **Controllo stringhe verso shell**

Date come assodate le problematiche riguardo le Signing Issue in Android è interessante notare come un'implementazione di questo genere, che fa uso della shell con parametri provenienti da un form compilato dall'utente debba curarsi di analizzare la struttura di queste stringhe in input. Poiché i dati vengono dall'esterno, potrebbero essere costruiti da un malintenzionato in modo da compromettere la sicurezza del sistema host. E' possibile che un malintenzionato inserisca, conoscendo la struttura di funzionamento del progetto, codice maligno o comandi arbitrari per danneggiare la struttura informatica. Per far fronte a questa potenziale e classica vulnerabilità sono stati inseriti controlli che effettuano l'escape dei possibili comandi shell all'interno delle stringhe.

Inoltre è stato implementato anche un controllo di formato dell'input lato client tramite una semplice funzione javascript che poi nella versione pubblicata è stata offuscata.

Allego il codice riguardante:

```
if (name.length<1 || name == $("#Nome").defaultValue) {
    errors++;
    var valid = '<small><br /> Nome: non valido'+isr;
    $(".Nome-wrapper .alert").html(valid);
    $("#Nome").addClass("wrong_input");
}else{
    $("#Nome").removeClass("wrong_input");
    $(".Nome-wrapper .alert").empty();
}

var sitoweb = $("#SitoWeb").val();
if (sitoweb.length<1 || !sitoweb.match(/(ftp|http|https):\/\/(\w+:{0,1}\w*@)?(\S+)(:[0-9]+)?(\/|\/([\w#!:.?+=&%@!\-\/]))?/)) {
    errors++;
    var valid = '<small><br />Sito web: non valido'+isr;
    $(".SitoWeb-wrapper .alert").html(valid);
    $("#SitoWeb").addClass("wrong_input");
}else{
    $("#SitoWeb").removeClass("wrong_input");
    $(".SitoWeb-wrapper .alert").empty();
}
```

FIGURA 30: CONTROLLI LATO CLIENT, JS

```php
/*Pulisco i caratteri non attesi*/
$nome = escapeshellarg($nome);
$sito = escapeshellarg($sito);
$indirizzo = escapeshellarg($indirizzo);

/*Compilo da shell e passo i parametri di personalizzazione*/
$mycmd = SHELL_PATH." ".$nome." ".$sito." ".$tmp_dir;

/*Pulisco i comandi non attesi*/
$last_line = exec(escapeshellcmd($mycmd));
```

FIGURA 31:CONTROLLI LATO SERVER, PHP

2) **ID variabile per directory temporanea e applicazione**

Dobbiamo risolvere due problemi,

- la mutua esclusione sul codice dell'applicazione base per permettere a più utenti di generare la propria webview contemporaneamente
- garantire che qualcuno non possa ricostruire la path temporanea (o di download) e poter scaricare le applicazioni personalizzate di altri.

Il secondo punto è interessante se supponiamo ci siano dati privati all'interno dell'applicazione oppure qualche specie di esclusiva. Ovviamente per il caso della demo non ci sono dati personali, siccome gli unici dati raccolti sono url e il nome dell'applicazione, ma per casi reali è sicuramente un dettaglio da tenere in considerazione.

Ecco il codice che riguarda la non ricostruibilità della path temporanea/download. Si tratta della generazione dell'id unico dell'applicazione. Ne deriverà una path temporanea unica non ricostruibile a meno di sapere le sequenti informazioni

- Nome dell'applicazione che si vuole recuperare
- Url visualizzato dall'applicazione
- Frase di salting
- Path sul sistema dove è presente la directory temporanea

```php
/*genero il nome unico per l'applicazione*/
$tmp_dir = md5($nome.SALTPHRASE.$sito);
```

FIGURA 32:SALTING E GENERAZIONE DELL ID DELL'APPLICAZIONE

```
39
40  ######################/CONFIG#########################
41
42  #making an istance for this customization
43  cp -rf ${MYDIR}${SOURCE} ${MYDIR}${DEST}
44
45  # for custom package name
46  mv ${MYDIR}${DEST}${PACKAGE} ${MYDIR}${DEST}"src/it/mywebview/customers/app_${3}"
47  find ${MYDIR}${DEST} -name "*.xml" -o -name "*.java" -o -name "*.cfg" | xargs sed -ri '
48
49  rm ${MYDIR}${DEST}"assets/config.txt"
50  rm ${MYDIR}${DEST}"res/values/application_name.xml"
51
52  #write custom data
53  echo -e '<?xml version="1.0" encoding="utf-8"?>\n<resources>\n<string name="app_name">'
54  echo "URL = "${URL}";" >> ${MYDIR}${DEST}"assets/config.txt"
55
56
57  #compile project
58  android update project --target android-15 --path ${MYDIR}${DEST}
59
60  ant debug -buildfile ${MYDIR}${DEST}"build.xml"
61  cp ${MYDIR}${DEST}${APK_PATH} ${MYDIR}${DEST}
62  if cp ${MYDIR}${DEST}${APK_PATH} ${MYDIR}${DOWNLOAD_DIR}"MyWebView_"${3}".apk"; then
63     rm -rf ${MYDIR}${DEST}
64     echo "finish"
65  else  echo "error"
66     exit 1
67  fi
68
69
```

FIGURA 33: SCRIPT SHELL E MUTUA ESCLUSIONE

In fig 34 vediamo come è gestita la mutua esclusione della applicazione base per istanze diverse di personalizzazione. La soluzione è molto banale, prima della scrittura del file di configurazione e del building vero e proprio viene fatta una copia del progetto originale nella directory temporanea e si lavora su quella directory. Riga 43.

Per quanto riguarda lo script shell inoltre: nel caso di file di configurazione più complessi di quello della demo, dovrà essere messo appunto uno script shell, in grado di prendere in input altri parametri di personalizzazione, sarà dunque da valutare se passarli come parametri o attraverso altre soluzioni. L'alto numero di parametri per lo script shell renderebbe difficoltosa la lettura e contorto il codice…

3) **Controllo richieste di download applicazioni**

Nell'ultimo punto abbiamo uno script che data in input una stringa va analizzare un path e ritorna un file. In questo contesto è importante analizzare la stringa in input siccome stringhe ben costruite potrebbero fornire ad un attaccante informazioni sulla struttura delle directory del progetto. Si potrebbe ripresentare inoltre il problema al punto 2.

Lo script è stato costruito in modo che accetti solamente strighe md5, quindi id validi di applicazioni.

```
*/
    require_once("configuration.php");
    $code =  $_GET["code"];

    /*Controllo se effettivamente non è vuoto*/
    if(!filter_has_var(INPUT_GET, "code")){
        die();
        exit();
    };

    /*pattern per md5*/
    $pattern = "/^[0-9a-f]{32}$/i";

    /*Se è un codice md5 valido procedo per reperirlo*/
    if(preg_match($pattern, $code)){
```

FIGURA 34:SCRIPT PHP PATTERN DI RICONOSCIMENTO ID APPLICAZIONI - MD5

La variabile pattern è la regular expression che permette di definire se l'id richiesto è di un'applicazione oppure no. Se lo è viene restituito il file, altrimenti viene interrotta l'esecuzione.

5.4 Applicazioni base

Per completare la visione del codice, ecco una breve descrizione dell'applicazione base, visualizzata in Eclipse come progetto MyWebView.

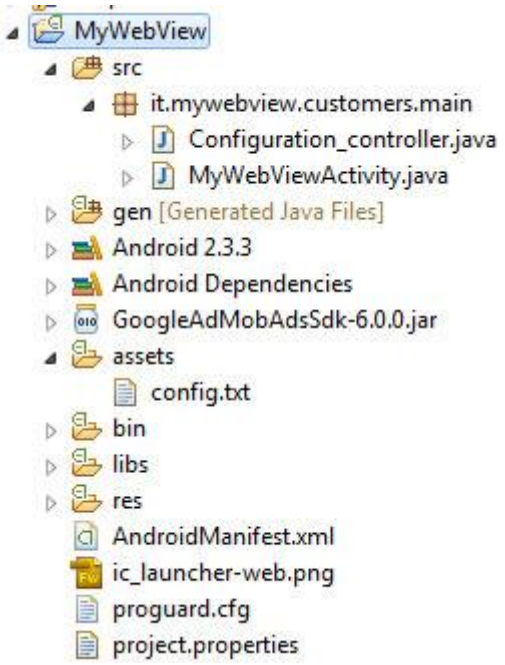


FIGURA 35:PROGETTO MYWEBVIEW SU ECLIPSE

Nel progetto sono presenti due classi

- Configuration_controller.java
- MyWebViewActivity.java

La prima recupera i dati dal file di configurazione config.txt e li restituisce in output, la seconda implementa la webview verso l'url dato in output dalla prima classe. Quindi una classe implementa le funzionalità e l'altra l'interfaccia verso la configurazione. Questa modalità è naturale candidata per l'implementazione di qualsiasi applicazione base del progetto.

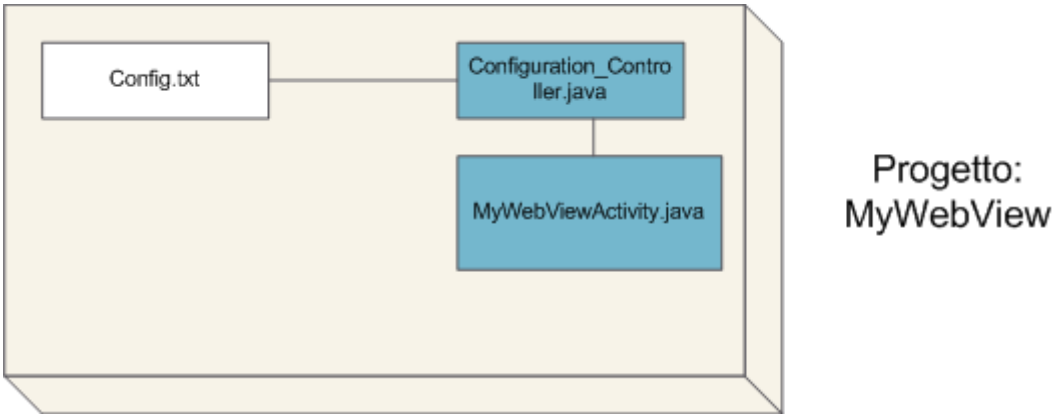


FIGURA 36:SCHEMA PROGETTO APPLICAZIONE BASE MYWEBVIEW

Con questo ultimo punto abbiamo analizzato le parti critiche del codice e la struttura nella sua interezza, per una visione più approfondita dell'implementazione consiglio di vedere direttamente il codice che è rilasciato su GITHUB il cui link che sarà indicato alla fine di questo testo.

La demo è visibile all'indirizzo: www.mywebview.com

5.5 Ottimizzazioni possibili

Le ottimizzazioni possibili su questo progetto possono vertere su diversi punti, avendo già detto che questo è un sottoinsieme delle funzionalità di un progetto commerciale è ovvio che è possibile aggiungere molte funzionalità, espandere l'applicazione base e l'interfaccia di raccolta dati …

Tempo di building

Ma vorrei guidare il discorso verso le prestazioni del servizio, è stato detto durante il capitlo 3 che per alcuni motivi è possibile personalizzare la propria applicazione solo prima della compilazione. Questo obbliga ad una compilazione live. Anche per i progetti commerciali di App Building è così, il punto è che questo costo è accettabile per applicazioni con codice nativo di piccole dimensioni, ma per grosse applicazioni i tempi potrebbero diventare lunghi e non sostenibili, anche per un discorso di risorse di calcolo necessarie. E' noto come la versione Beta attuale di AppInventor abbia grossi rallentamenti che a volte ne impediscono il corretto funzionamento, forse proprio a causa di questi motivi.

Nelle soluzioni commerciali come AppsBuilder, la scelta di sposare l'implementazione tramite tecnologie web mantiene le dimensioni dell'Apk molto basse, limitando così i tempi e le risorse necessarie per il building di questo.

Dimensione Apk Demo	Dimensione Apk AppsBuilder
100Kb	512kb

La dimensione dei due Apk è comparabile mostrando che l'insieme delle funzionalità è simile.

Nella macchina su cui è installata la demo il build impiega circa 5 secondi. L'hardware a disposizione è una VPS con le seguenti caratteristiche

Cpu Virtuale	Ram Virtuale	Hard disk Virtuale
1 core	1GB	60GB

Sarebbe interessante ma per motivi di tempo non affrontata, l'analisi della crescita del tempo di building all'aumentare della dimensione del progetto. Per i progetti disponibili il tempo di building è rimasto al di sotto dei 10 secondi.

Possibili vie di uscita per limitare i tempi di building sono meccanismi di caching di due tipi:

1. Non ricompilare applicazioni uguali (possibile nella demo, ma nella realtà difficilmente applicabile)
2. Compilare solo il necessario attraverso l'uso di pacchetti Jar binari usati come librerie nelle applicazioni base.
3. Inserimento di immagini di piccole dimensioni nell'applicazione base

Il primo punto è applicabile nella demo poiché con stesso "url" e stesso "nome applicazione" ottengo un identificativo univoco di un'applicazione e posso verificare se questa è già stata compilata in precedenza o meno, risparmiandomi di ricostruirla se è già disponibile nella directory di download.

Il secondo punto è più massiccio e più applicabile in casi reali. In caso di applicazioni con codice nativo di grandi dimensioni, possiamo preparare e spezzare il codice in moduli jar contenenti file .class già compilati e risparmiarci una parte del tempo del building dell'applicazione personalizzata.

Il terzo punto è molto semplice da introdurre, basta usare immagini compresse e di piccola dimensione, le immagini vengono infatti compresse durante il processo di build. Questa raccomandazione diminuisce il tempo necessario per questa attività. Ci sono linee guida direttamente sul sito ufficiale developer android.

Questo è l'inzio per un discorso più ampio che sarà fatto nel prossimo capitolo, in cui si va a valutare come creare progetti libreria da importare in altri progetti in modo compatibile con l'sdk Android per godere di vantaggi prestazionali, di

modularità e per poter usare una procedura di riutilizzo di codice sorgente comoda e direttamente in un IDE come ad esempio nella piattaforma Eclipse.

Manutenibilità, Installazione e Aggiornamenti

Fermiamo il discorso sui progetti libreria per chiudere il paragrafo sulle ottimizzazioni parlando della manutenibilità del progetto. Gli Sdk Android sono in continua evoluzione. E' interesse di Google mantenere la retro compatibilità degli SDK in modo tale che progetti ad esempio sviluppati per Android 2.3.3 funzionino anche sull'ultima versione del sistema operativo Android. Quindi se questa politica viene mantenuta, un'applicazione personalizzata può godere di una vita duratura. E' importante però per sollecitare l'evoluzione del progetto e offrire sempre un set di applicazioni efficaci mantenendo aggiornate le applicazioni di base, per fare questo è necessario rinfrescare periodicamente i moduli ed anche l'Sdk. Detto questo è importante realizzare qualche script che faciliti l'installazione e la configurazione iniziale della macchina host per ospitare il Webservice, per facilitare la migrazione da un server all'altro e per facilitare i lavori di trasferimento dal server di test al server di produzione.

Altrettanto importante è realizzare uno script che operi sul webservice stesso nei momenti di manutenzione programmata, in grado di avviare l'aggiornamento degli Sdk e assistere l'inserimento delle nuove versioni delle applicazioni base.

Queste sono tipicamente le ultime fasi del progetto , in cui ci si preoccupa della qualità del servizio offerto e del mantenimento di questo.

Concludendo questo paragrafo concludiamo anche il capitolo e la parte a proposito di personalizzazione per terze parti attraverso l'uso di WebService. Si spera di aver analizzato nella sua completezza, senza entrare troppo nel dettaglio di temi non essenziali tutto ciò che riguarda la progettazione di un WebService di personalizzazione di applicazioni Android, dando anche un'occhiata agli sviluppi multipiattaforma che conseguono le scelte progettuali. Si spera quindi di aver costruito un documento utile e riassuntivo per tutti coloro che abbiano intenzione di implementare, continuare o discutere a proposito dei temi trattati, fornendo ispirazione per entrare nei dettagli di alcune discussioni rimaste aperte in questo testo.

Capitolo 6
Personalizzazione di Applicazioni mediante IDE

6.1 Progetti libreria in Android SDK

Alla fine del precedente capitolo è stato chiarito che in caso di applicazioni con codice nativo di grandi dimensioni i tempi di building possono crescere, soprattutto considerando richieste concorrenti sul servizio di personalizzazione, trasformando i pochi secondi necessari al build in tempi meno accettabili. L'obbiettivo è lasciare il minimo lavoro da compiere alla build live. L'idea quindi è di precompilare tutte le parti di codice che non devono essere personalizzate e ottenere archivi jar binari prima del building live.

Un file jar è un archivio compresso (ZIP) usato per distribuire raccolte di classi Java. Tali file sono concettualmente e praticamente come i package, e quindi talvolta associabili al concetto di libreria.

Questo approccio è molto utile per realizzare e rilasciare librerie, riutilizzabili in progetti aziendali o in progetti di sviluppatori partner senza rilasciare direttamente il codice sorgente.

I noti banner AdMob di pubblicità all'interno di applicazioni Android sono rilasciati sotto forma di librerie binarie jar, pronte da scaricare dal sito AdMob e disponibili allo sviluppatore dopo l'importazione senza che il codice sorgente sia visibile e facilmente manomettibile.

Viene naturale porsi la domanda: come è possibile generare progetti libreria con l'SDK Android?

Dalla versione 14 dell'SDK sono stati introdotti i Library Project, come riportato dalla pagina ufficiale

"An Android *library project* is a development project that holds shared Android source code and resources. Other Android application projects can reference the library project and, at build time, include its compiled sources in their `.apk` files. Multiple application projects can reference the same library project and any single application project can reference multiple library projects.

Note: You need SDK Tools r14 or newer to use the new library project feature that generates each library project into its own JAR file. "

Riportata sulla pagina riguardante i progetti libreria è presente anche questa nota:

"A library cannot be distributed as a binary file (such as a JAR file). This will be added in a future version of the SDK Tools."

Il che significa che alla versione attuale degli SDK Android non è possibile ufficialmente esportare un progetto libreria come Jar, questo renderebbe difficile la procedura di "spezzare" l'applicazione base in vari blocchi precompilati, limitando a prima vista l'aumento di prestazioni a build time.

Esistono vie non ufficiali per realizzare questa cosa, utilizzabili ma non raccomandate come soluzione conclusiva per la bassa affidabilità in prospettiva dei futuri sviluppi dell'SDK Android. Anche dalla documentazione ufficiale, riportata prima appare esserci un controsenso: non è possibile esportare i progetti libreria in formato jar ma i progetti libreria sono utilizzati da altri progetti in formato jar ...

Ecco una versione per costruire un Jar da un progetto libreria in maniera molto veloce e intuitiva, sfruttando il plugin ADT per Eclipse (Android Development Tools).

6.2 Realizzazione di progetti libreria

Partendo dal progetto dell'applicazione base della demo su Eclipse e applicando le regole riportate sul sito ufficiale per la creazione di progetti libreria, ho spezzato il progetto MyWebView (fig) creando due sottoprogetti

1. MyWebViewMain (codice funzionale)
2. MyWebViewLibrary (interfaccia verso il file di configurazione)

in modo tale da inserire le parti funzionali del progetto in MyWebViewMain e l'interfaccia di lettura del file di configurazione nel progetto MyWebViewLibrary.

Un progetto Android diventa progetto libreria quando è inserito un flag particolare nelle proprietà del progetto. Si tratta del flag "android.library = true".

Questo flag permette all'sdk e ad ADT di capire che il progetto è libreria, generarne un file jar e rendere disponibile questo jar nelle librerie di tutto l'ambiente di lavoro.

Questo è chiaramente un pattern di lavoro utilizzabile per progetti di più grandi dimensioni, visto che l'applicazione base di questo testo è molto semplice è del tutto didattico spezzare il progetto in progetto funzionale e libreria.

Ecco alcuni screen e schemi per chiarire le modifiche effettuate.

Prima : Progetto unico.

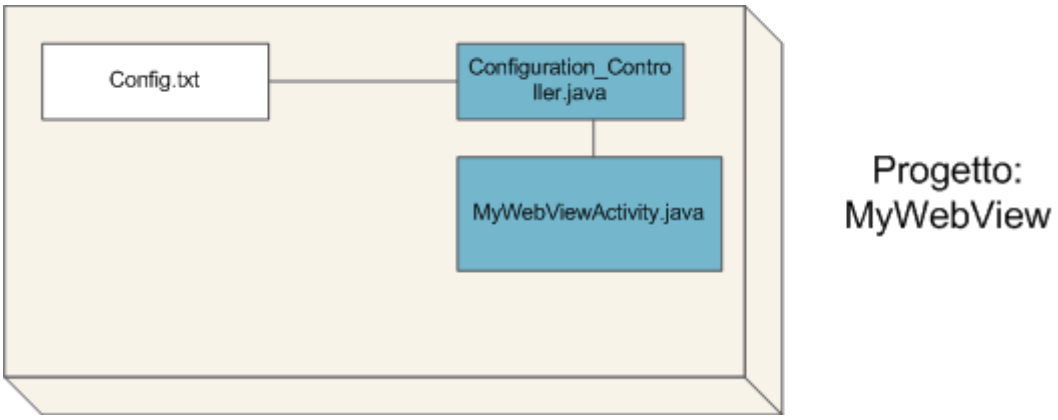


FIGURA 37:PROGETTO APPLICAZIONE BASE MYWEBVIEW CON PROGETTO UNICO

Questo pacchetto conteneva file di configurazione, interfaccia di lettura verso config.txt e semantica dell'applicazione.

Dopo : Progetto + Progetto libreria.

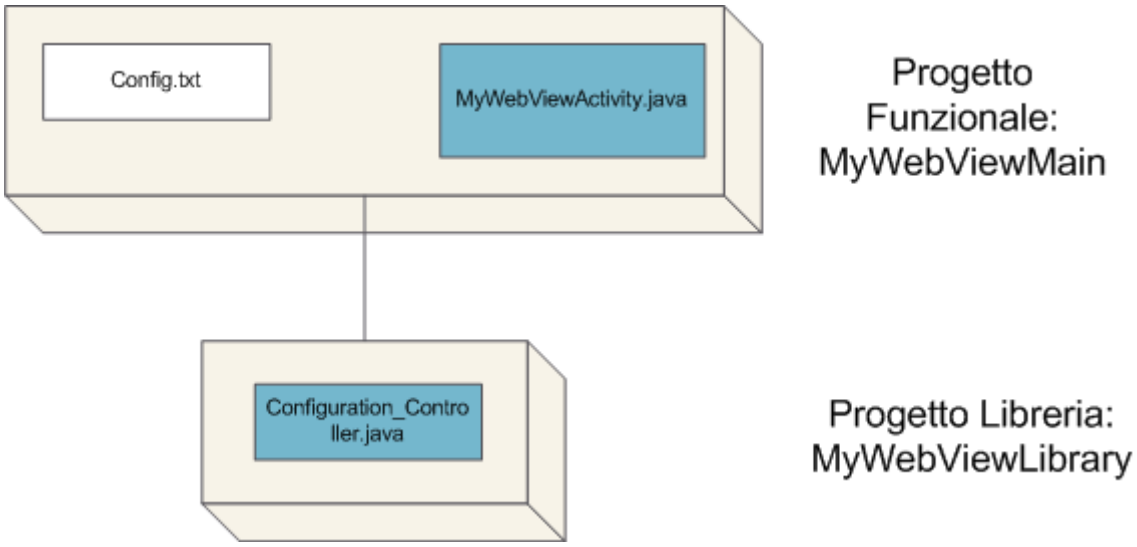


FIGURA 38: PROGETTO FUNZIONALE + PROGETTO LIBRERIA

Ora un pacchetto è adibito ad implementare l'interfaccia verso il file di configurazione e l'altro è adibito a realizzare la semantica dell'applicazione.

La cosa interessante è che sarà proprio la libreria a mettere a disposizione le funzionalità per la lettura del file di configurazione. MyWebViewActivity non è nemmeno a conoscenza della presenza di un file di configurazione nel proprio pacchetto. Il file di configurazione rimarrà nel progetto funzionale, realizzando

esattamente ciò che ci necessitava per poter realizzare librerie binarie per i webservice di personalizzazione. Il file di configurazione quindi rimmarà nel sorgente dell'applicazione base e non nella libreria jar, permettendone la modifica e la personalizzazione .

Diamo un'occhiata con alcuni screenshot di quello che avviene in ambiente Eclipse durante la configurazione della libreria.

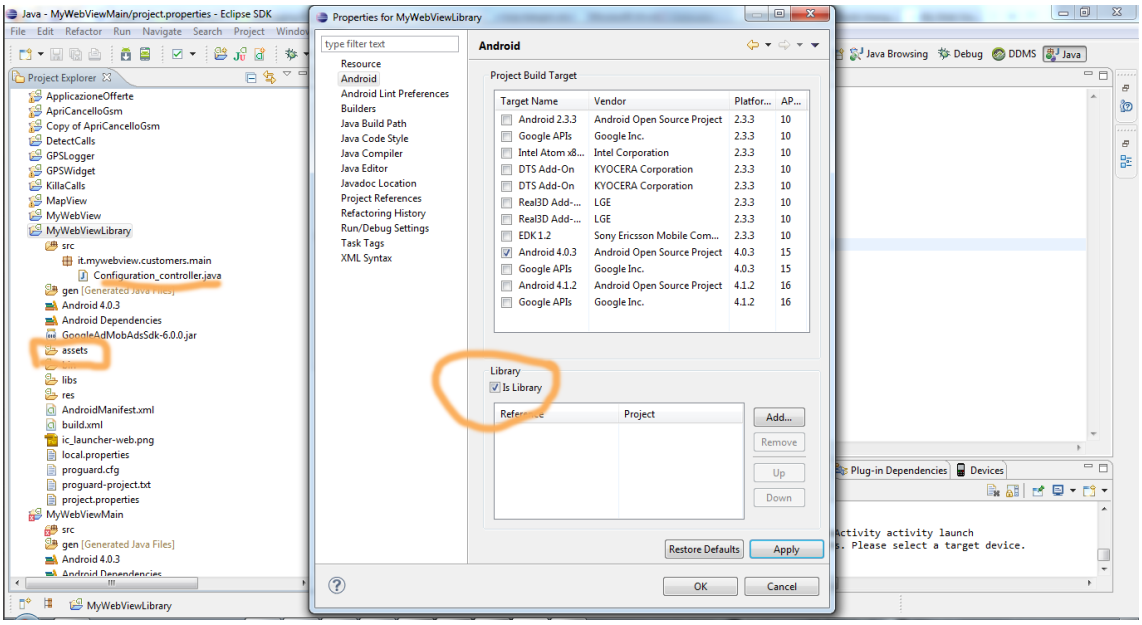
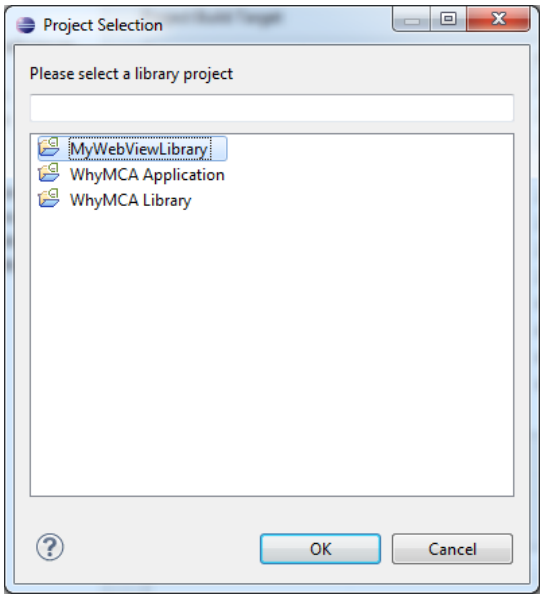


FIGURA 39:SETUP DEL PROGETTO LIBRERIA SU ECLIPSE

Il progetto MyWebViewLibrary è etichettato come libreria e vediamo che contiene solamente il Configuration_controller.java e gli assets sono vuoti.



A questo punto tra le librerie disponibili per Il progetto MyWebViewMain cè anche MyWebViewLibrary.

FIGURA 40:LIBRERIE DISPONIBILI PER MYWEBVIEWMAIN

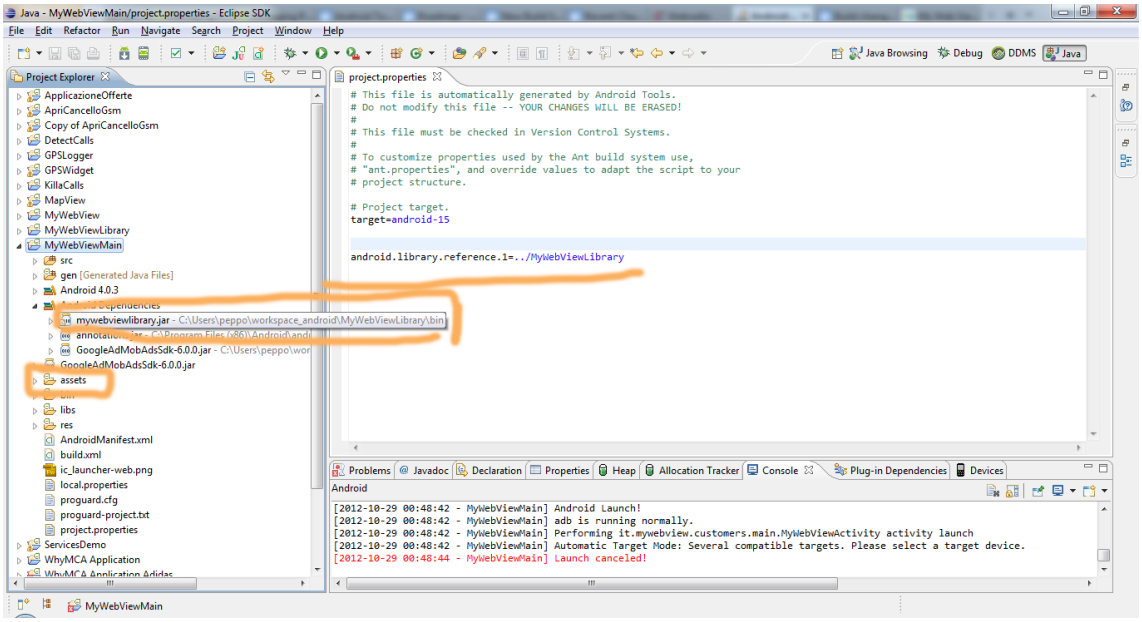


FIGURA 41:PROGETTO MYWEBVIEWMAIN CON IMPORTAZIONE AUTOMATICA JAR LIBRERIA

Una volta aggiunto MyWebViewLibrary come libreria di MyWebViewMain è visibile tra le Android Dependecies un pacchetto jar, che è esattamente il progetto libreria preparato prima ed ora compilato. Possiamo andare a recuperare quel pacchetto, rimuovere i riferimenti nel file project.proprieties, inserirlo nella directory libs del progetto MyWebViewMain e realizzare così una soluzione funzionante di MyWebView con l'uso di librerie jar binarie come prefissato all'inizio di questo capitolo e in modo compatibile alla progettazione della demo.

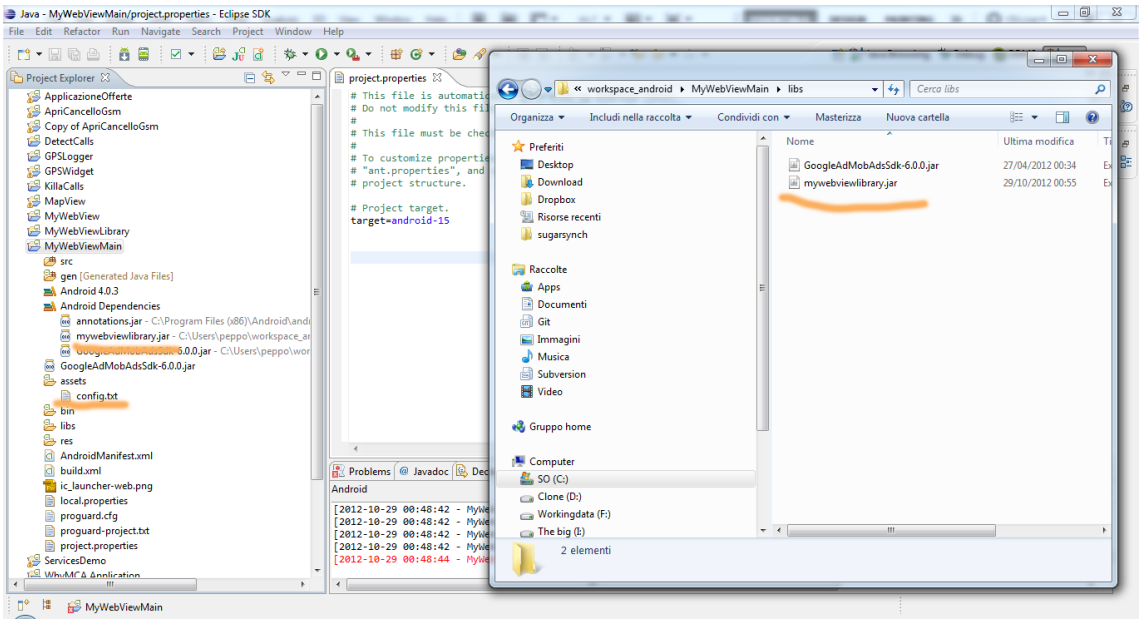


FIGURA 42: PROGETTO MYWEBVIEWMAIN SENZA RIFERIMENTI AL PROGETTO LIBRERIA. JAR INTEGRATO NELLE DIPENDENZE

6.3 Personalizzazione di applicazioni attraverso progetti libreria

Il modo di operare affrontato nel paragrafo precedente oltre ad essere una politica di ottimizzazione dell'applicazione base in prospettiva del build live e della realizzazione di un webservice, ci suggerisce un nuovo modo di procedere per la personalizzazione di applicazioni.

Nell'introduzione alla tesi sono state delineate due tipologie di aziende, quelle che lavorano conto terzi e quelle che lavorano direttamente per il grande pubblico sul market.

Immaginiamo un azienda che ha realizzato un videogioco mobile o un'applicazione di svago, in questo contesto non c'è grossa necessità di personalizzare un grande numero di progetti da giustificare la realizzazione di un webservice. Inoltre ora pubblichiamo diretti sul market, non ci sono terzi che devono usufruire del servizio o generare una loro versione dell'applicazione. Insomma il problema è totalmente da affrontare in azienda.

Il prodotto è praticamente unico e funziona proprio per quel motivo, e non viene rilasciato a terzi. L'esigenza cosi muta e ci saranno molte revisioni nel codice dello stesso progetto, frutto di un ciclo sviluppo-test-codifica, da distribuire sul market in poche versioni, ad esempio lite e pro.

Ne risulta cosi la necessità di organizzare il sorgente in vari moduli, alcuni consolidati, altri no e in conclusione risulta utile avere progetti libreria ed una gestione diretta sul tool di sviluppo delle versioni, come ad esempio Eclipse.

Per lo sviluppo verso il grande pubblico il concetto di personalizzazione di applicazioni android è un po’ deviato, risulta più come una gestione facilitata dello stesso progetto in modo che sia semplice escludere od integrare funzionalità e caratteristiche ed aggiornare il codice.

Mentre per la soluzione con webservice i Library Project venivano utili come pacchetti jar, ora siccome usiamo direttamente Eclipse conviene mantenere direttamente il progetto cosi com'è come codice sorgente!

Parliamo quindi **di personalizzazione di applicazioni mediante IDE**, appunto mediante Eclipse.

6.4 Organizzazione di un'applicazione in versioni lite e pro

Ecco una proposta di organizzazione di un'applicazione in modo che il ciclo di sviluppo sia già orientato al rilascio in versioni lite e pro.

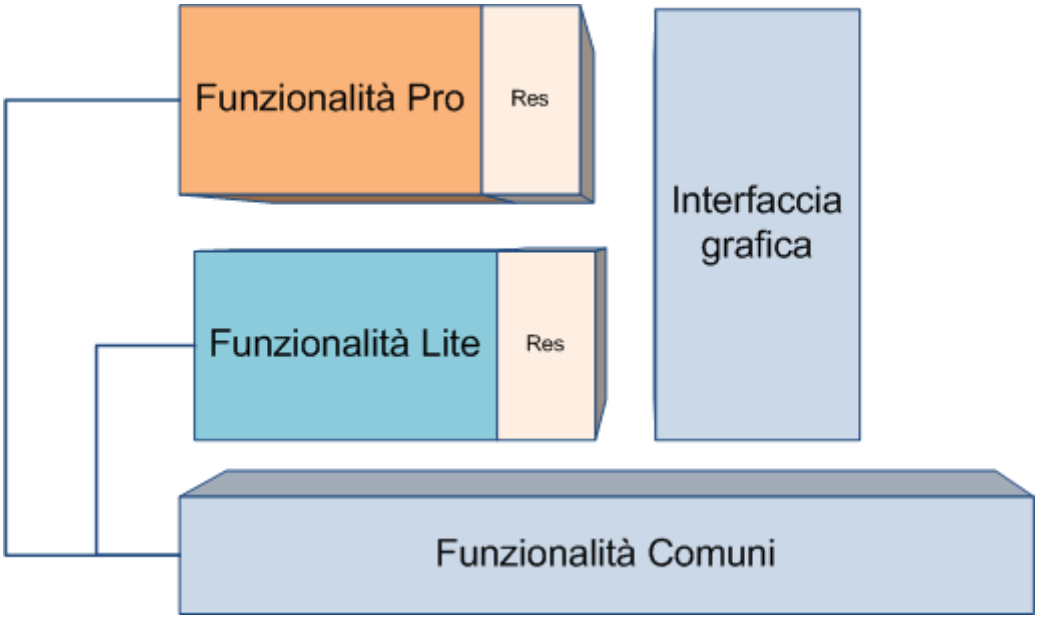


FIGURA 43: SCHEMA PER APPLICAZIONE LITE E PRO CON LIBRERIA

L'implementazione avviene tramite l'ereditarietà dei progetti realizzata dal SDK Android, in modo molto simile a quello analizzato per MyWebView. Si dichiarano i progetti libreria e si importano dove devono essere utilizzati.

In questo schema:

1. Funzionalità comuni: Questo è un progetto libreria che fornisce un set di funzionalità base comuni a entrambe le versioni, viene importato da entrambi i progetti Lite e Pro
2. Funzionalità Lite: Progetto standard che contiene l'insieme di classi disponibili solo in versione lite
3. Funzionalità Pro: Progetto standard che contiene l'insieme di classi disponibili solo in versione pro
4. Res: risorse grafiche, Assets o quant'altro sia raggiungibile astraendo dal file system utilizzando le api dell'sdk Android
5. Interfaccia grafica: E' volutamente scollegata siccome in funzione dei contesti può essere utile farla comune o con alcune variazioni.

E' importante che le risorse non siano integrate nei progetti libreria, poiché allo stato delle cose non è possibile riferirsi a risorse in modo dinamico dai progetti libreria con un metodo ufficiale e supportato da Google.

Google sta volgendo parte dei futuri sviluppi proprio per facilitare e migliorare la gestione dei progetti libreria con il proprio Sdk e il proprio plugin per Eclipse ADT.

Riportando dalla pagina dei tool di Google Android:

Our roadmap is a non-exhaustive list of what is to come to the Android Tools. It is mostly a list of things we are currently focusing on, or plan to focus in the near future.
Please note that we make no guarantee on actually shipping features listed in the roadmap.

More information to be added shortly.

- Eclipse plug-ins
  - Visual layout editor
  - Resource manager
  - Theme editor
  - Better refactoring support
- Build support
  - Library support: merge manifest of libraries into application's manifest
  - Better Proguard support.
  - Build-time Debug flag.

Come si vede alcuni degli sforzi riguarderanno l'approccio alla personalizzazione e supporto alla programmazione (Theme editor, Better refactoring support) e supporto alle librerie. Temi vicini a quelli trattati in questo testo.

6.5 Altri esempi

Si può fare un discorso simile al rilascio di versioni pro e lite anche per le applicazioni che hanno strutture consolidate e di successo, ad esempio videogiochi. In questo campo si può rilasciare una variante del videogioco che volga ad un tema nuovo ma con la medesima struttura di gioco. Ottenendo novità ma mantenendo familiarità al pubblico.

Per chiarire si pensi a tutti quei giochi di tipo TowerDefense, Quiz in cui mantenendo stesse funzionalità e cambiando aspetto grafico e temi trattati si può costruire un gioco nuovo al pubblico ma con il costo di sviluppo molto basso. Casi di successo che hanno utilizzato questo approccio sono ad esempio la serie Angry Birds, CityVille, FarmVille … Zombie and Plants etc… sia su piattaforma mobile che web.

Le aziende che approcciano in questo campo potrebbero quindi usare strategie di riuso del codice attraverso progetti libreria come descritte in questo testo.

6.6 Soluzioni di personalizzazione e obbiettivi aziendali

Concludiamo cercando di associare le diverse strategie ad obbiettivi aziendali. Quali aziende dovrebbero preferire una soluzione di personalizzazione tramite webservice e quali tramite IDE?

Webservice

La soluzione mediante Webservice è utile ed è da seguire se l'azienda è specializzata nella produzione di un set ristretto, stabile di applicazioni, su cui mediamente si è già testato, in modo positivo il feedback dell'utente. Questa azienda quindi si può permettere di mettere a disposizione la propria esperienza in un servizio e permettere a terzi (clienti customizzatori) di personalizzare applicazioni di buona qualità a costi bassi e con conoscenze limitate. In sostanza è da scegliere questa soluzione se si vuole diventare un azienda di Application Building come in questo momento è identificata nel mercato.

Requisiti per la scelta di implementazione di un webservice:

Applicazioni base stabili
Applicazioni graficamente personalizzabili in funzione del cliente
Punti di personalizzazione dell'applicazione ben definiti
Applicazioni Base con grande richiesta

Obbiettivi aziendali

Realizzazione applicazioni per terze parti
Rilasciare i propri servizi in modo che i clienti siano autonomi

Questi termini riducono molto l'insieme di aziende che possono permettersi di generare un servizio di questo tipo. La migliore rappresentante di queste aziende è AppsBuilder, diventata punta di diamante a livello internazionale nel settore.

Sarebbe interessante vedere un progetto di personalizzazione Open Source gestito da community, un po’ come è successo nel tempo per i CMS web e come sta provando a fare, in maniera non OpenSource anche AppsBuilder rivolgendosi ai rivenditori di applicazioni attraverso un CMS per la realizzazione di Apps.

IDE

Per quanto riguarda invece la struttura di personalizzazione mediante IDE parliamo di un contesto molto più comune, a basso impegno, orientato alla pubblicazione diretta sul market di progetti ancora non maturi. In sostanza questa procedura può essere attuata da qualsiasi azienda di sviluppo applicazioni mobili che voglia lavorare fin da l'inizio del ciclo di sviluppo considerando l'obbiettivo di pubblicare la propria applicazione in versioni differenti, ad esempio pro e lite. Questo approccio è da associare naturalmente alle piccole startup.

L'azienda vuole mantenere l'esclusiva sull'applicazione e vuole pubblicare direttamente sul market, non per conto terzi. In questo contesto l'azienda è nuova, i progetti sono novità e ancora poco testati.

Una volta ottenuto feedback positivo poi si può evolvere la situazione riutilizzando il codice prodotto per cambiare tematiche sull'applicazione ma preservare la struttura funzionante come detto nel precedente paragrafo.

Requisiti per un uso di personalizzazione mediante IDE

Applicazione in via sviluppo
Ciclo di build,debug e correzione molto frequente
Necessità di fare poche versioni (Lite,Pro,Develop..)

Obbiettivi aziendali

Sviluppare applicazioni direttamente per il pubblico del market
Minimizzare lo sforzo per l'introduzione di nuove applicazioni

E chiaro come questi requisiti siano laschi e come tutte le aziende che producono in modo professionale applicazioni godano di vantaggi ad utilizzare progetti libreria per "personalizzare" applicazioni direttamente nel loro ciclo di sviluppo.

Capitolo 7
Conclusioni

Gli obbiettivi di questo testo sono la realizzazione di una completa analisi delle problematiche relative alla personalizzazione di applicazioni e all'utilizzo di progetti libreria durante lo sviluppo di un'applicazione, con particolare riferimento ad Android. L'obbiettivo principale è mostrare come partire da un progetto base e fornire un sistema "self-service" web in grado di realizzare applicazioni in funzione ai dati inseriti dal cliente.

Sono emersi discorsi molto importanti riguardo la firma necessaria per la pubblicazione di applicazioni Android sul Market, completati dalle scelte progettuali possibili con relativi punti di forza e svantaggio. Queste scelte sono state poi messe a confronto con le scelte progettuali adottate in commercio, ottenendo interessanti risultati, facendo emergere come le scelte progettuali siano discutibili in funzione della definizione di firma. E' stato implementato un progetto di Demo per la generazione di applicazioni Android webview, partendo da un'applicazione base e personalizzandola in funzione di alcuni dati inseriti in un form on-line. Durante la descrizione dell'implementazione della demo sono stati fatti notare i punti critici per la sicurezza dei dati e dell'host che devono essere affrontati in tutti i tipi di webservice di questo tipo. Concludendo è stata proposta una metodologia di realizzazione di applicazioni che sfrutta il concetto di progetto libreria implementato dall'SDK Android per la realizzazioni di applicazioni modulari, pensate per essere predisposte al rilascio in versione lite e pro. Questo ultimo tema è stato concluso con una comparativa in funzione degli obbiettivi aziendali tra personalizzazione mediante webservice e personalizzazione mediante IDE.

Concludendo, spero di aver fatto un lavoro interessante e utile per chi, sviluppatore di applicazioni mobile si voglia togliere la curiosità sul funzionamento dei progetti di personalizzazione applicazioni. Dando la visione globale dei problemi intrinseci del tema e delle scelte discutibili sia dei progetti commerciali che di Google stesso nel Market.

Cosa possiamo concludere da quello che è stato studiato?

I servizi di personalizzazione di applicazioni attualmente in commercio sono basati su tecnologie web potendo in questo modo offrire anche servizi di porting su diverse piattaforme. Le aziende sul mercato hanno scelto giustamente di risolvere questo problema in maniera semplice e orientata a vantaggi per l'utente piuttosto che affrontare un problema di porting su codice nativo che probabilmente si sarebbe tradotto in una soluzione difficile da usare per i clienti.

Il sistema di firma di applicazioni mobile è ancora immaturo. Chiaramente dalle situazioni analizzate non si è individuata una scelta incontestabilmente corretta.

La scelta di firma dovrà essere fatta in funzione del contratto di utilizzo che si allega al servizio e al valore che si vuole attribuire alla firma stessa, mettendo ad un estremo la praticità commerciale e all'altro la formalità di firma.

L'introduzione di Certification Autorithy, ancora in fase embrionale in questo campo, per ora non costituisce una priorità sia di sviluppatori che di aziende che di Google stesso. Bisognerà cambiare qualcosa nella catena di sviluppo-pubblicazione perché la firma di queste applicazioni abbia un carattere più restrittivo. Per ora posso concludere che l'unico obbiettivo di queste firme è determinare sommariamente una delle entità coinvolte nell'implementazione dell'applicazione firmata e permettere la non modificabilità dei pacchetti.

Altro tema emerso che merita un approfondimento è al gestione di dati sensibili in applicazioni Android, purtroppo lasciato ai possibili sviluppi.

Capitolo 8
Possibili sviluppi

1. Personalizzazione di applicazioni in ambiente Windows Phone
2. Personalizzazione di applicazioni in ambiente IOS
3. Personalizzazione di Applicazioni in ambienti Multipiattaforma
4. Analisi delle soluzioni per la gestione di dati sensibili in Android
5. Implementazione di una delle scelte progettuali di firma non convenzionali
6. Analisi del diritto d'autore nei Markets Mobile
7. Approfondimento sui Contratti dei Markets Mobile
8. Analisi su contratti d'uso di soluzioni di tipo "Application Builder"
9. Analisi approfondita di App Inventor
10. Analisi approfondita di prestazioni e della procedura di Build di Applicazioni Android
11. Sviluppo di un caso elaborato di applicazione con rilascio Lite e Pro
12. Sviluppo di un videogioco mobile progettato per il riuso del codice in altre tematiche

Riferimenti e Link

Sorgenti del progetto:

1. **https://github.com/peppo1616/Mywebview**

Pagine informative:

2. **http://developer.android.com/guide/developing/building/index.html.**
3. **http://developer.android.com/tools/**
4. **http://tools.android.com/recent**
5. **http://developer.android.com/tools/projects/index.html**
6. **http://www.whymca.org/intervento/android-tools-e-la-gestione-di-progetti-complessi**
7. **http://stackoverflow.com**
8. **http://beta.appinventor.mit.edu**
9. **http://www.apps-builder.com**
10. **http://www.appbrain.com**

Ringraziamenti

Voglio ringraziare doverosamente Andrea Catalini e Matteo Romanelli con i quali ho discusso e avuto riscontri sui temi trattati in questa tesi. Queste discussioni sono state utili, stimolanti e fondamentali per avviare discorsi a riguardo dello sviluppo di applicazioni Android, di scelte progettuali in vari campi dell'informatica tra cui anche quello in questo testo, non facendo mai mancare la curiosità.

Voglio ringraziare Stefano Manzi per la lunga serata dedicata ad AppInventor che ha stimolato la scrittura di questa tesi.

Voglio ringraziare tutte le persone amiche e anche un po' di più, che mi sono state accanto durante il mio percorso universitario e che mi hanno aiutato a tenere duro per confermare le passioni che ho consolidato durante questa esperienza.

Ringrazio i gruppi di studio, fondamentali per la preparazione degli esami e i gruppi di svago per i periodi fuori esame!

Ringrazio la mia famiglia che mi ha permesso di intraprendere le mie scelte liberamente.

E come non ringraziare il fato ricordando che:

«La fortuna è quel momento

in cui la preparazione incontra l'opportunità»

cit: Randy Pausch