UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
CORSO DI LAUREA TRIENNALE IN INGEGNERIA INFORMATICA

TESI DI LAUREA

# Una applicazione basata su WebKit per la fruizione di libri elettronici

RELATORI: CARLO FANTOZZI, ALBERTO PETTARIN

LAUREANDO: MARCO GAUDINO

ANNO ACCADEMICO 2012/2013

PADOVA, 28 MARZO 2013

# SOMMARIO

Questa tesi descrive l'estensione dell'app di lettura Orion Viewer, per la piattaforma Android, con il supporto al formato EPUB in modo che questo venga visualizzato con la formattazione e gli stili assegnatigli dall'autore o dall'editore.

In dettaglio, essa dapprima analizza il mondo dell'editoria digitale prestando particolare attenzione alle possibilità di lettura di eBook nei dispositivi Android sfruttando varie applicazioni; in seguito analizza il formato EPUB, l'applicazione Orion Viewer (studiando l'architettura del programma) e la libreria Java Epublib (utilizzata per la gestione della manipolazione degli eBook EPUB). Infine, dopo aver verificato le potenzialità della classe *WebView* nella lettura di file XHTML (di cui è composto un EPUB), è stata modificata Orion Viewer, scrivendo due classi, *EpubDocument* (astrazione dell'eBook) e *SpecializedWebView* (specializzazione della WebView), e modificandone altrettante, ossia *Controller* (che gestisce la manipolazione del documento) e *OrionViewerActivity* (activity principale dell'app, che visualizza l'eBook).

In questo modo si è ottenuto che Orion Viewer supporti documenti in formato liquido (EPUB) parallelamente a quelli in formato paginato (PDF e DjVu) tramite l'uso di una *WebView*, e che la gestione dell'interazione con l'utente sia unica per i vari formati. Inoltre l'EPUB viene visualizzato rispettando fedelmente la "tipografia" assegnatagli dall'autore/editore, modalità mancante in tutti gli eReader per Android presi in considerazione.

# INDICE

# CAPITOLO 1

## INTRODUZIONE

Dalla seconda metà del XX secolo i dispositivi elettronici hanno invaso le nostre case e hanno cambiato il nostro vivere quotidiano. Questa rivoluzione ha naturalmente coinvolto anche una delle attività principali del genere umano: la lettura.
Il concetto di libro elettronico ("electronic book" o "eBook"), consultabile da qualsiasi dispositivo dotato di schermo, come pc, smartphone o tablet, è nato negli anni Settanta, quando grazie all'opera di Michael S. Hart furono convertite un centinaio di opere di pubblico dominio, che diedero vita al noto "progetto Gutenberg" (1971). Hart sosteneva la diffusione del computer e degli eBook al pubblico per combattere l'analfabetismo.
A partire dalla metà degli anni '90, prima in Giappone e poi negli Stati Uniti, sono comparsi i primi dispositivi (detti "eReader") dedicati esclusivamente alla lettura digitale e rivolti ad un mercato di massa.
L'editoria digitale ha trovato in Amazon il suo grande promotore: per merito dell'azienda statunitense i libri elettronici hanno preso sempre più piede (grazie anche alla vendita, a partire dal 2007, dell'eReader "Kindle"), tanto che nel 2011, negli USA le vendite di titoli elettronici ha superato, in volume, la vendita di libri cartacei, per tutti i maggiori retailer online (secondo i dati diffusi dal colosso fondato da Jeff Brazos). Seppur in maniera più graduale, anche in Europa negli ultimi 3-4 anni si è assistito allo sviluppo di questo nuovo mercato digitale. In Italia, in particolare, nonostante la presenza di operatori già a partire da metà anni 2000, il mercato dei titoli elettronici ha raggiunto l'1% del volume complessivo dell'editoria nel 2011, quando Amazon ha aperto ufficialmente lo store italiano.
Il grande vantaggio di un libro digitale è quello di "tagliare" il peso della lettura, potendo racchiudere in un file centinaia di pagine e, quindi, in un singolo dispositivo di pochi grammi migliaia di titoli. I libri digitali sono poi acquistabili a qualsiasi ora, in qualsiasi luogo, e possono essere fruiti immediatamente. Infine, il mezzo digitale consente operazioni testuali prima difficoltose o impossibili: collegamenti intra- e inter- testuali, ricerca di parole in dizionari e traduttori integrati, condivisione di frammenti del testo o di note e osservazioni su social network, e via elencando.
Come naturalmente capita per ogni mercato ai suoi primordi, sulla scena degli eBook si sono affacciati vari formati, molti dei quali proprietari, come MOBI/KF8 (usato da Amazon per il Kindle), PDB (per Palm Os) e FictionBook. Di converso, vari produttori di hardware, di software e di contenuti si sono riuniti in un consorzio internazionale, l'International Digital Publishing Forum (IDPF), che ha elaborato quello che oggi è accettato come lo standard di pubblicazione di eBook destinati al mercato commerciale, ossia EPUB.
La specifica EPUB permette di creare libri personalizzabili, contenenti immagini, collegamenti ipertestuali, stili configurabili dall'utente, font particolari, ed altro ancora. Essa si basa su tecnologie di rappresentazione delle informazioni già mature e collaudate, come XHTML, XML, DAISY, NCX.

Al giorno d'oggi risulta naturale usare il proprio smartphone o tablet per leggere un eBook, ma per farlo bisogna avere un'applicazione che lo visualizzi e che ne permetta la manipolazione per garantire un'esperienza d'uso almeno pari alla controparte cartacea. Prendendo in esame la piattaforma Android, la più diffusa attualmente nell'ambito dei dispositivi elettronici personali, il panorama di applicativi è ampio:

- *Google Play Libri*, fornito da Google stessa, con il suo store integrato e un'interfaccia user friendly;
- *Moon+ Reader*, il quale supporta svariati formati e biblioteche online, ed è pienamente personalizzabile;
- *Aldiko*, il quale richiede l'importazione dei libri al suo interno, e gestisce anche le protezioni DRM Adobe;
- *Go Book*, che importa al suo interno automaticamente i libri presenti in memoria, ha uno store limitato ma supporta tutti i formati, dall'EPUB, al FB2, al PDF;
- *Fbreader*, open source, molto diffuso, è adattabile per qualsiasi esperienza d'uso desiderata, potendo cambiare font e colori del testo, animazioni di cambio pagina, ed inoltre è possibile accedere a vari cataloghi di libri scaricabili liberamente o a pagamento. Non legge gli EPUB protetti da DRM;
- *CoolReader*, programma open source, in grado di leggere tutti i principali tipi di eBook, disponibile per tutte le piattaforme (Windows, Linux, Symbian, Android) con varie caratteristiche, come l'animazione per il cambio pagina personalizzabile, la possibilità di scelta dello sfondo, luminosità in base al giorno o alla notte, supporto a dizionari, e riproduzione vocale del testo;
- *Orion Viewer*, app di lettura per Android e per alcuni eReader, molto intuitiva, completa possedendo anche il supporto a dizionari esterni e consentendo di modificare i bordi del libro, l'orientamento della lettura e la riprogrammazione dei comandi touch. Tuttavia è sprovvista del supporto allo standard open source EPUB.

Tutti gli applicativi fin qui citati però hanno un difetto: prendono l'EPUB, estraggono da esso tutti i testi e le altre risorse (immagini, font, fogli di stile) e ricompongono il tutto visualizzandolo a schermo senza rispettare la formattazione originariamente concepita dall'editore.
Risulta quindi mancante, nel mondo Android, un'applicazione che legga un documento EPUB e lo visualizzi a schermo senza modificarlo, lasciando inalterata la sua formattazione, così com'era stato concepita e creata. Per questo motivo è stato deciso di modificare l'app open source *Orion Viewer*, aggiungendovi il supporto a EPUB tramite l'utilizzo di WebKit, il motore di rendering web presente su Android, grazie al quale è possibile visualizzare il libro conservando gli stili e la formattazione. Tutto ciò è realizzabile grazie alla struttura degli EPUB, il cui testo risiede in uno o più file XHTML, correttamente visualizzati, assieme alle eventuali risorse esterne (immagini, font, fogli di stile) da WebKit.

Nel resto della tesi si andrà ad analizzare dapprima il formato EPUB, per il quale sarà fornito supporto in Orion Viewer, poi gli strumenti usati per gestirlo, ossia l'app stessa, Epublib (libreria Java per gestire gli EPUB), e WebKit (tramite il quale, come anticipato sopra, viene visualizzato

l’eBook). Infine saranno descritte le modifiche apportate al programma di partenza, allegando parti del codice scritto per facilitarne la comprensione.

# CAPITOLO 2

## IL FORMATO EPUB

### *2.1 EPUB*

L'EPUB, abbreviazione per *electronic publication*, è un formato standard aperto per testi e pubblicazioni digitali, che consiste in un file zip, con estensione ".epub", contenente tutti i testi, immagini e dati che servono alla visualizzazione del libro su schermo. È stato proposto nel settembre 2007 come standard ufficiale per gli eBook da parte dell'IDPF, International Digital Publishing Forum, organizzazione no-profit internazionale per la regolamentazione dell'editoria digitale e per il suo sviluppo, composta da università, associazioni, produttori di dispositivi ed editori. EPUB ha preso il posto del formato *OEB*, *Open EBook*, precedente standard elaborato dall'Open eBook Forum, aggiornandolo e migliorandolo. Nel 2009 l'IDPF istituì gruppi di lavoro per l'aggiornamento e il mantenimento di EPUB, portando nell'aprile 2010 al rilascio della versione 2.0.1 dello standard, che di fatto costituisce il principale formato per eBook sul mercato. Nell'ottobre 2011, IDPF ha rilasciato l'ultima versione dello standard, EPUB 3.0, con significative novità rispetto alla versione 2.0.1. In particolare, EPUB 3 permette:

- l'inserimento di file multimediali, come video e audio, all'interno dell'eBook;
- l'introduzione dei linguaggi HTML5 e CSS3;
- l'uso di Javascript;
- l'inserimento di formule matematiche grazie a MathML;
- l'utilizzo di tecnologie assistive (SMIL, CSS Speech, ARIA).

In un EPUB il testo è contenuto in una o più pagine XHTML (è possibile in sostituzione usare il Digital Talking Book o DTBook, linguaggio sviluppato dal consorzio DAISY), i layout e la formattazione sono descritti da fogli di stile CSS, e infine il manifest, la "Table of Contents" (l'indice) e i metadati sono contenuti in file XML. È grazie all'uso di questi linguaggi che , come poi sarà approfondito nel corso della trattazione, è possibile usare il motore di rendering WebKit per sviluppare un eReader in Android.
La caratteristica principale per la quale è noto questo standard è il "reflowing", ossia la capacità del testo di adattarsi a qualsiasi dispositivo in uso in base alle dimensioni e all'orientamento dello schermo e la personalizzazione della resa visiva secondo le impostazioni fornite dall'utente.
Vista la varietà di piattaforme software e hardware sulle quali è possibile leggere l'eBook, è evidente come questa caratteristica lo renda molto appetibile per l'editoria digitale. Altre importanti caratteristiche dell'EPUB sono:

- standard aperto;
- possibilità d'incorporamento e scelta del font per il testo;
- supporto di stili CSS per caratterizzare il layout;
- inclusione dei metadati per fornire informazioni aggiuntive;
- grafica "raster" e vettoriale;

- supporto per i DRM, ossia le protezioni che possono essere inserite dall'editore; queste non hanno schemi obbligatori fissati dallo standard;
- supporto del linguaggio XML e relative funzionalità;
- possibilità di usare software (validatori) per verificare l'integrità e la completezza del libro creato;
- supporto a layout prefissati.

Naturalmente lo standard in esame non è esente da critiche: esso ad esempio è inadatto per la visualizzazione di grafica avanzata, come possono essere fumetti e libri ricchi d'immagini, grafici e tabelle (per cui in questo campo l'uso del formato PDF è ancora imperante).
Riprendendo ad analizzare la struttura dello standard, l'EPUB è uno ZIP, formato da tre strutture fondamentali: l'OPS, l'OPF, e l'OCF.

*OPS, Open Publication Structure*
L'Open Publication Structure è un insieme di file XHTML e CSS che descrivono i contenuti del libro. La sintassi per gli stili e il layout deve essere supportata dai sistemi di lettura (che quindi di conseguenza devono poter leggere i CSS).
I file devono essere codificati in Unicode (standard per la codifica dei caratteri), come UTF-8 o UTF-16, per poter supportare le varie lingue.

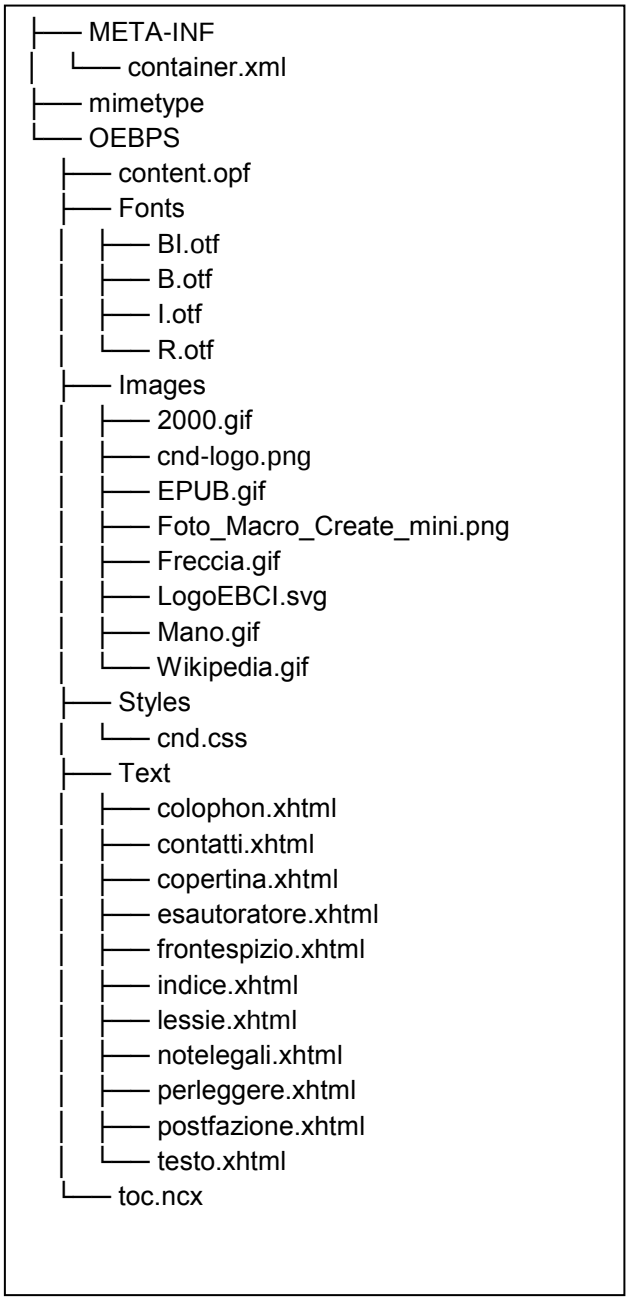
*OPF, Open Packaging Format*
L'Open Packaging Format definisce il meccanismo tramite il quale le varie parti dell'OPS sono legate tra di loro , fornendo inoltre strutture addizionali e semantiche all'eBook. Nella maggior parte dei casi, è composto da due file scritti in XML, con estensione *".opf"* e *".ncx"*.

- OPF
  Generalmente chiamato "content.opf", contiene i metadati, il file manifest e l'ordine di lettura dei testi. È composto da un elemento radice e da quattro figli: *metadata*, *manifest*, *spine* e *guide*, tutti obbligatori a parte *guide*. Il *metadata* elenca i metadati del libro (titolo, autore, lingua, editore, eccetera); il *manifest* contiene una lista di tutti i file interni al pacchetto, assegnando a ognuno *id*, *href e mediatype*; gli unici file che non vengono elencati qui sono l'OPF, il container.xml e il mimetype. Lo *spine* elenca tutti i testi XHTML, nel loro ordine di lettura, e al suo interno si trovano inoltre i contenuti raggiungibili dai collegamenti ipertestuali del libro e la "Table of Content" (TOC), contenente l'id del file .NCX . La *guide* infine identifica le componenti strutturali fondamentali dell'eBook (scelte rapide o *landmarks*).
- NCX
  Usualmente chiamato "toc.ncx", il Navigation Control File contiene l'indice dei contenuti del libro in una struttura XML organizzata ad albero.

*OCF, Open Container Format*
L'Open Container Format specifica come sono organizzati i file all'interno dell'archivio zip e definisce due file addizionali che vanno inclusi nel pacchetto finale: *mimetype* e *META-INF/container.xml.*

Il file mimetype è un documento di testo codificato in ASCII ed è il primo file dell'EPUB. Esso permette a un'applicazione di riconoscere il tipo di file tramite un classico meccanismo di numero magico. Il file *container.xml* invece contiene la descrizione della posizione del content.opf rispetto alla radice dell'archivio ZIP.

Un esempio dell'organizzazione interna di un EPUB è  rappresentato nella figura che segue.

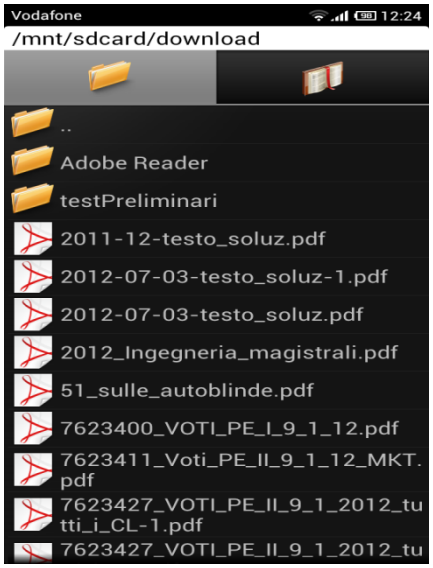


### *2.2 ORION VIEWER*

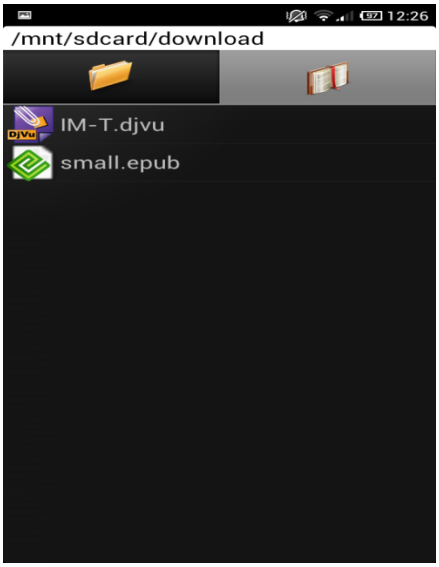
L'applicazione che è stata l'oggetto dello studio di questa tesi è Orion Viewer. Sviluppata da Mikhael Bogdanov, giunta alla versione 0.41.2 e rilasciata con licenza GNU GPL v3, questa applicazione ha come obiettivo la lettura di eBook in formati PDF, DjVu, XPS e CBZ cercando di garantire all'utente la massima confortevolezza e la piena possibilità di personalizzare l'esperienza di lettura. Facilmente reperibile sul Play Store e in Internet, è disponibile per i dispositivi Android dalla versione "Cupcake" (versione 1.5) in poi. Il supporto dei formati PDF e DjVu viene assicurato tramite le librerie MuPDF e DjVuLibre. Le caratteristiche principali dell'applicazione sono le seguenti:

- possibilità di sottolineare il testo scegliendo il colore;
- inserimento di segnalibri all'interno del libro elettronico;
- selezione del testo per poterlo copiare;
- zoom personalizzabile per rendere migliore la leggibilità del testo;
- possibilità di ritagliare i bordi del libro per eliminare parti bianche inutili e visualizzare il testo su tutto lo schermo;
- visualizzazione del testo in portrait o landscape, per permettere di leggere al meglio anche eBook ricchi di figure o schemi;
- navigazione nel testo tramite scorrimento o inserendo il numero di pagina prescelto;
- personalizzazione e riconfigurazione tutti i tocchi sullo schermo, potendo cambiare anche le aree sensibili al tocco;
- possibilità di scegliere lo schema di lettura all'interno della pagina, da sinistra a destra, o da destra a sinistra (per esempio se il testo contiene colonne, è possibile cambiare il pattern di navigazione al suo interno tra A B C D a A C B D);
- supporto a dizionari esterni per ricercare il significato di una parola immediatamente e facilmente;
- file manager integrato che elenca inoltre gli eBook recentemente aperti.

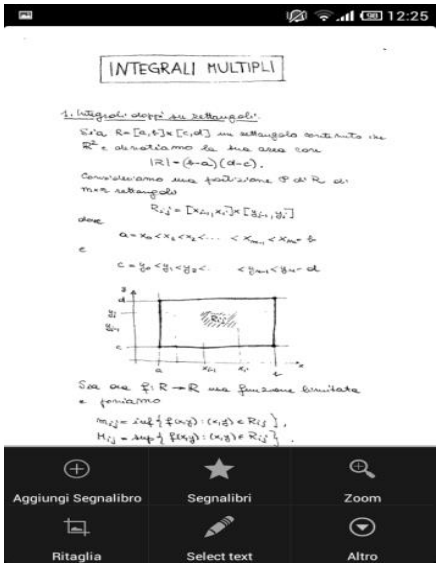
Le possibili azioni per usare Orion Viewer sono due. La prima consiste nel selezionare direttamente un eBook usando un file manager e scegliendo di aprirlo con l'applicazione sopracitata. In questo caso sarà visualizzato direttamente il libro con zoom preimpostato a zero e la possibilità di scegliere l'orientamento e di tagliare i bordi. Un secondo modo è cliccare direttamente sull'icona dell'app nel menù di Android, e in questa maniera sarà aperto il file manager integrato di Orion Viewer, tramite il quale si andrà a scegliere l'eBook all'interno della memoria del dispositivo.



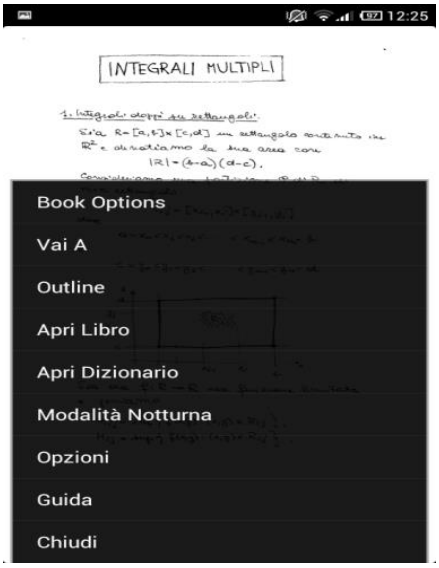

Apertura di Orion Viewer: file manager interno all'app




Documenti aperti di recente




Esempio di un libro con il menù contestuale aperto




Menù aperto in maniera estesa

Sul sito del programma creato dallo sviluppatore russo, è possibile segnalare bug, o suggerire funzionalità mancanti e necessarie per la piena fruibilità degli eBook, ed è inoltre possibile contribuire al progetto con donazioni, traduzioni e codice.

È stato deciso di intervenire sull'app, estendendo Orion Viewer con il supporto agli EPUB, e trattando l'eBook in maniera diversa rispetto agli altri formati.

I file PDF e DjVu sono gestiti usando due librerie, MuPDF e DjVuLibre, e visualizzati tramite oggetti *Canvas*, ossia oggetti per mostrare e manipolare immagini in Android. In questo caso l'eBook viene preso e incapsulato in un nuovo oggetto, rendendolo accessibile per mezzo di una chiamata alla libreria corrispondente al formato all'interno del costruttore dell'oggetto stesso. Quest'oggetto, che viene definito dalla classe *PdfDocument* nel caso riguardi un PDF o dalla

*DjVuDocument* nel caso riguardi un DjVu, estende l'interfaccia generica *DocumentWrapper*, che definisce le primitive con le quali manipolare il documento. È fornito poi come parametro, insieme all'oggetto *OrionView* (estensione della classe *View* che viene usato per visualizzare il libro a schermo, supportando il rendering di bitmap e quindi degli oggetti *Canvas* citati in precedenza) e all'oggetto di *LayoutStrategy* (che serve appunto per gestire il layout del libro), in ingresso al costruttore della classe *Controller*. In questa saranno eseguite tutte le chiamate conseguenti alle azioni dell'utente, come la scelta della pagina da "disegnare" a schermo, lo zoom prescelto, l'orientazione, il cambio pagina, le informazioni su pagina corrente, zoom corrente, contrasto e selezione del testo. É proprio questo il compito del *Controller*: gestire e compiere le chiamate ai metodi sugli oggetti, necessarie al funzionamento di Orion Viewer. L'activity che coordina tutto il processo di apertura dell'eBook e si occupa di visualizzarlo su schermo, ospitando la *OrionView*, si chiama *OrionViewerActivity*. Questa è la classe principale che inoltre richiama il *Controller* nel caso l'utente interagisca con il documento. Per fornire il supporto all'EPUB, come vedremo nel prossimo capitolo, sono state sfruttate le classi preesistenti e lo schema d'azione usato da Bogdanov per gli altri formati, in modo da rendere uniforme l'integrazione fatta con il codice esistente, ma usando una *WebView* al posto della *OrionView* e non oggetti *Canvas* per contenere e visualizzare il testo.

### *2.3 EPUBLIB*

Per gestire un file XHTML all'interno di un'applicazione Android non è necessario avvalersi di alcuna libreria esterna, poiché WebKit, nativa in Android, soddisfa ogni bisogno che si possa incontrare nel tentativo di visualizzare la pagina, come vedremo nel seguito della trattazione. Un EPUB tuttavia, com'è già stato analizzato in precedenza, è una collezione di XHTML e altri file compressi in un archivio, e quindi risulta necessario adottare uno strumento per la loro corretta gestione.

In Orion Viewer è stato deciso di usare la libreria esterna Epublib per manipolare gli eBook. Questa libreria scritta in Java permette di leggere (e scrivere) in maniera programmatica gli EPUB; si può sfruttare sia all'interno di un'applicazione, sia tramite linea di comando. Nel nostro caso ovviamente sarà integrata all'interno dell'applicativo, giacché è compatibile anche con il sistema di Google. È distribuita dallo sviluppatore Paul Siegmann sotto licenza GNU Lesser General Public License, in modo tale da poter far parte sia di applicazioni open source/free software che di app proprietarie.

Per sfruttare tali libreria all'interno di un app bisogna importare all'interno del progetto dell'applicativo sia il core di Epublib, ossia "epublib-core-latest.jar", e sia "slf4j-android". Il *Simple Logging Facade for Java* è una libreria che serve come semplice astrazione per interfacciare vari frameworks di log con il sistema.

Studiamo ora in particolare il funzionamento della libreria, andando ad analizzare come viene aperto e gestito un EPUB. La prima operazione da svolgere è istanziare un nuovo oggetto "*EpubReader*"; in seguito l'eBook sarà incapsulato all'interno di un nuovo oggetto "*Book*", tramite la chiamata:

*Book book = epub.readEpub (new FileInputStream ("indirizzo dell'EPUB da leggere"));*

dove "*epub*" corrisponde all'oggetto EpubReader istanziato inizialmente. Una volta ottenuto l'oggetto *Book*, su di esso saranno effettuate tutte le chiamate per gestire e visualizzare le varie parti dell'eBook.

Un primo metodo da utilizzare è *getContents()* : questa chiamata effettuata su *Book* restituisce una lista di oggetti del tipo *List <Resource>*, contenente tutte le risorse del libro raggiungibili tramite Spine, Table of Contents e Guide secondo l'ordine di visualizzazione prescelto alla creazione. Sul singolo oggetto *Resource* poi è possibile ottenere il path che si riferisce al file in archivio, usando *getHref()*, potendolo così concatenare al path assoluto dell'EPUB contenuto in memoria e ottenere l'indirizzo da dare in ingresso alla *WebView* che visualizzerà la pagina.

Altre chiamate più specifiche che possono essere eseguite sull'oggetto *Book* sono:

- *getTableOfContents()* , che restituisce un oggetto *TableOfContent*, ossia permette di ottenere l'accesso alla tabella dei contenuti del libro;

- *getSpine()* , che restituisce un oggetto *Spine*, grazie al quale si può ottenere l'accesso a ogni file puntato dallo spine dell'EPUB nell'ordine dato dall'autore dell'eBook;
- *getGuide()* , che restituisce un oggetto *Guide*, agendo sul quale si accede alla guide del libro;
- *getMetadata()* , che, dando in risposta un oggetto *Metadata*, consente di lavorare con i metadati dell'EPUB;
- *getTitles()* , che, com'è facilmente intuibile, restituisce il titolo del libro sotto forma di *String*;
- *getCoverImage()* , il quale in uscita avrà un oggetto *Resource* che punta alla copertina dell'eBook;
- *getOpf()* , grazie al quale si ottiene , come oggetto *Resource* , il file OPF dell'EPUB;
- *getNcx()* , tramite cui si ricava, sotto forma di *Resource*, il file NCX del libro.

Per manipolare l'EPUB e inserire i vari URI dei file in ingresso alla *WebView*, è stato deciso di preventivamente scompattare l'eBook in una directory provvisoria, la quale, una volta chiuso il programma (e quindi l'eBook), verrà eliminata.

# CAPITOLO 3

## LA CLASSE WEBVIEW

Lo strumento usato per garantire il supporto dell'EPUB è l'oggetto *WebView*, che estende la classe "*AbsoluteLayout*", o più precisamente:

```
java.lang.Object
  ↳ android.view.View
      ↳ android.view.ViewGroup
          ↳ android.widget.AbsoluteLayout
              ↳ android.webkit.WebView
```

Come si evince dallo schema, la *WebView* è una sottoclasse della classe *View*, che rappresenta il blocco fondamentale per la costruzione delle interfacce. Una *View* è un oggetto rettangolare sul quale visualizzare ogni cosa desiderata e con il quale si può interagire. La *WebView* perciò è una particolare *View* utilizzata per mostrare pagine HTML, sfruttando il motore di rendering WebKit; è il fondamento quindi per la creazione di un proprio Web browser. Nel nostro caso sarà usata per visualizzare il codice XHTML dell'eBook all'interno dell'app. Va ovviamente inclusa nel layout della nostra activity:

```
<WebView
  android:id="@+id/myWebView"
  android:layout_width="match_parent"
  android:layout_height="match_parent" />
```

Successivamente è necessario istanziarla all'interno del metodo "*onCreate*()" sempre dell'activity all'interno della quale vogliamo usare la *WebView*:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    WebView wbView = (WebView) findViewById(R.id.myWebView);
}
```

Nel caso si voglia usarla per navigare in Internet, inoltre, vanno aggiunti all'interno del manifest anche i permessi per l'accesso alla rete:

```
<uses-permission android:name="android.permission.INTERNET" />
```

La *WebView* ha dei metodi propri per la navigazione all'interno delle pagine e per interagire con esse, per vedere la cronologia , cambiare lo zoom, effettuare una ricerca e molto altro. Inoltre di default non gestisce eventuali errori all'interno del codice HTML ma lo visualizza senza manipolarlo. Esistono due metodi principali per caricare la pagina desiderata nella *View*:

- "*loadUrl("indirizzo della pagina da visualizzare")*": carica la pagina corrispondente all'indirizzo inserito come parametro. Questo è il metodo che è stato usato nelle modifiche apportate a Orion Viewer;
- *loadData("testo da visualizzare come String", "mimeType, ossia text/html", "codifica")*: carica il codice HTML inserito all'interno di un oggetto di tipo *String* e dato in ingresso al metodo come primo parametro; il secondo elemento indica il mimeType del codice; il terzo indica la codifica da usare;

In aggiunta è possibile utilizzare altre due chiamate, varianti di quelle precedenti: è possibile ad esempio aggiungere una serie di headers HTTP addizionali alla pagina tramite l'inserimento di un secondo parametro *Map<String, String>* al metodo *loadUrl()*. In alternativa si può usare il metodo *loadDataWithBaseURL (String baseUrl, String data, String mimeType, String encoding, String historyUrl)* : in questo caso è aggiunto come primo parametro un url usato per risolvere i path relativi delle pagine da caricare, e come ultimo parametro una stringa dove salvare la cronologia.

Se la pagina caricata nella *WebView* contenesse link e l'utente ci cliccasse sopra, di default verrebbe aperto il browser di Android (o più in generale verrebbe data la possibilità di scegliere con che browser aprire il link tra quelli installati). Per far sì che il collegamento ipertestuale sia aperto nella stessa *View* bisogna abilitare una *WebViewClient* che serve per intercettare l'eventuale caricamento di un url e per gestire tutto ciò che va ad impattare sul rendering del contenuto, come può fare un errore ad esempio. Nel dettaglio, il codice che deve essere utilizzato è:

```
wbView.setWebViewClient(new WebViewClient(){
        @Override
        public void onReceivedError(WebView view, int errorCode, String description, String failingUrl){
                //gestione dell'eventuale errore          }
        @Override
        public boolean shouldOverrideUrlLoading(WebView view, String url_new){
                view.loadUrl(new_url); //carico la pagina cliccata nella WebView
                return true;      }
}};
```

In aggiunta , sempre di default, non è attiva la gestione di Javascript. Per abilitarli è necessaria una modifica dei *WebSettings*. Per gestirli bisogna creare una *WebChromeClient*, che andrà a ricevere e gestire gli alert Javascript e per di più servirà per manipolare l'interfaccia utente (UI) della *View*. Ad esempio è possibile aggiungere una barra di caricamento per mostrare il progresso del loading della pagina, o una di scorrimento laterale, o ancora una barra per inserire gli indirizzi da visualizzare.

```
wbView.setWebChromeClient(new WebChromeClient(){
        @Override
        public void onProgressChanged(WebView view, int progress){
                //qui  andrà inserito il codice per mostrare una barra di caricamento}
}};
```

Se cambiano le configurazioni del dispositivo sul quale stiamo visualizzando la *WebView*, come l'orientazione, esso ricaricherà la pagina. Per evitare ciò bisogna gestire i cambi di configurazioni per l'activity stessa che ospita la nostra *View*, usando il parametro "*configChanges*" all'interno del manifest dell'applicazione. I principali metodi invocabili per una *WebView*, alcuni dei quali usati anche in Orion Viewer, sono i seguenti:

- *boolean pageDown(boolean bottom)* : scrolla il contenuto della *WebView* di mezza pagina verso il basso; se il parametro in ingresso è "true" allora il contenuto verrà scrollato fino alla fine;
- *boolean pageUp(boolean top)* : scrolla il contenuto della *View* verso l'alto di mezza pagina; in questo caso se il parametro d'ingresso è "true" la pagina tornerà all'inizio;
- *int getContentHeight()* : ha come parametro d'uscita l'altezza del contenuto della *WebView*;
- *String getUrl():* restituisce sotto forma di una stringa l'indirizzo della pagina caricata;
- *boolean zoomIn(), zoomOut():* eseguono rispettivamente l'ingrandimento e il rimpicciolimento della pagina visualizzata; restituiscono un parametro *boolean* in base alla buona riuscita dell'operazione;
- *void findAllASync(String word)* : trova tutte le istanze della stringa passata come parametro d'ingresso al metodo nella pagina, e le evidenzia;
- *void clearMatches()* : elimina le sottolineature sul testo che si erano ottenute come risultato dell'invocazione di *findAllASync*;
- *boolean onTouchEvent(MotionEvent event)* : gestisce gli eventi derivanti dall'esecuzione di tocchi sulla WebView da parte dell'utente;
- *boolean onKeyDown(int keycode, KeyEvent event), onKeyUp(int keycode, KeyEvent event)* : questa invocazione invece gestisce gli eventi generati dall'uso dei tasti fisici per la regolazione del volume; ad esempio può essere utile impostare lo scrolling delle pagine tramite il loro click;
- *boolean onTrackballEvents()* : gestisce gli eventi derivanti dall'azione dell'utente sulla trackball (se presente nel device);
- *void destroy()* :distrugge lo stato interno della *View*;
- *String getTitle():* ha come parametro d'uscita il titolo della pagina caricata , sotto forma di stringa;
- *void goBack(), goForward()* : questi metodi consentono la navigazione tra la cronologia della *WebView* indietro o in avanti, utili nel caso si utilizzi quest'oggetto per la creazione di un browser;
- *void goBackOrForward(int step)* : usato per lo stesso scopo dei due metodi precedenti, ma stavolta è possibile, fornendo un numero in ingresso, decidere di quanto andare avanti o indietro;

- *boolean setOnLongClickListener()* : gestisce un click prolungato sulla *View* assegnandogli un'azione;
- *void reload()* : ricarica la pagina nella *View*;
- *void setBackgroundColor( int color)* : assegna un colore di background alla *WebView*, in base al codice colore passato in ingresso all'invocazione mediante un numero in *int*;
- *void setInitialScale(int scaleInPerc)* : questo metodo imposta un livello di zoom iniziale per la *View*; di default l'impostazione è a zero;
- *int getHeight(), getWidth()* : restituiscono rispettivamente l'altezza e la larghezza della *View*;
- *public ViewGroup.LayoutParams getLayoutParams ()* : restituisce i parametri di layout della *View*, che poi possono essere modificati (ad esempio assegnando nuovi valori a larghezza e altezza);
- *int getScrollX(), getScrollY()* : utilizzati per conoscere di quanto la pagina è stata scrollata rispettivamente in base alla coordinata orizzontale e verticale;
- *void scrollBy(int x, int y), scrollTo(int x, int y)* : la prima chiamata scrolla la pagina orizzontalmente e verticalmente secondo l'ammontare di pixel inseriti in ingresso, e la seconda scrolla la *View* fino alle coordinate inserite in ingresso.

Dopo quest'analisi si può perciò capire quanto la *WebView* sia adatta alla lettura di eBook, potendo avanzare con le pagine del libro, ingrandire o ridurre il testo e compiere ogni azione necessaria per rendere completa la fruizione del libro elettronico da parte del lettore.

Sorge spontaneo domandarsi per qual motivo si è scelto di estendere, con il supporto all'EPUB, Orion Viewer usando la *WebView* invece di aprire una delle tante app preesistenti sul mercato per la lettura di libri elettronici nello standard analizzato. Gli altri eReader scompattano il libro, estraggono il testo, e lo visualizzano impostando i loro stili, i loro formati del carattere e la formattazione in generale; molto spesso presentano all'utente una versione del libro semplificata. In questo modo vengono tralasciate le scelte dell'autore, poiché viene ignorato tutto il codice scritto da esso che sia diverso dal testo vero e proprio. Sfruttando invece la *WebView* è possibile visualizzare il libro interamente, in ogni sua parte. Il creatore dell'EPUB utilizza codice CSS per impostare i propri stili e le proprie personalizzazioni all'eBook, e la *View*, come già visto, supporta tale linguaggio. Non sono necessarie altre manipolazioni del testo, ma si può visualizzare istantaneamente il libro elettronico con ogni personalizzazione creata dall'editore. In questo modo, perciò, la presentazione dell'eBook è fedele ai propositi editoriali. Un successivo vantaggio di quest'approccio risulta essere la semplicità di gestione del file, in quanto basta scompattare l'archivio e, usando una libreria, passare alla *WebView* il path delle parti da mostrare a schermo.

Ora verranno spiegate le modifiche e le estensioni ad Orion Viewer che abbiamo sviluppato per poter visualizzare un eBook in formato EPUB.

# CAPITOLO 4

## IMPLEMENTAZIONE DEL SUPPORTO EPUB IN ORION VIEWER

Per supportare EPUB all'interno di Orion Viewer si rende necessario implementare le seguenti aggiunte e modifiche:

- definire una nuova classe, *EpubDocument*, che sia analoga alle già esistenti *PdfDocument* e *DjVuDocument*, e rappresenti l'astrazione dell'eBook in formato EPUB;
- definire un nuovo oggetto, *SpecializedWebView*, che sia analogo alla già esistente *OrionView*, ma che si occupi del rendering delle pagine XHTML contenute nell'eBook in formato EPUB;
- modificare opportunamente il *Controller*, classe che si occupa di "interpretare" le azioni dell'utente e le "comunica" alle classi che sono incaricate del rendering del documento;
- modificare l'activity principale, *OrionViewerActivity*, soprattutto per quanto riguarda la creazione della *View* (*OrionView* o *SpecializedWebView*) appropriata al documento aperto dall'utente.

Le successive quattro sezioni illustrano nel dettaglio il codice sviluppato per ciascuno dei punti precedenti.

### *4.1 EPUBDOCUMENT*

*L'EpubDocument* è l'oggetto che rappresenta un'astrazione dell'eBook in formato EPUB. Esso implementa l'interfaccia *DocumentWrapper*, la quale definisce le principali primitive con le quali è possibile manipolare il documento, valide sia per i formati paginati (PDF, DjVu) sia per quelli liquidi (EPUB), come ad esempio: "apri file", "ottieni il titolo del documento", "passa alla sezione successiva o precedente", ecc... *DocumentWrapper* contiene anche un'enumerazione che consente di dichiarare la natura paginata o liquida del documento che effettivamente implementa l'interfaccia, consentendo al resto dell'applicazione di gestirlo opportunamente. Un oggetto di tipo *EpubDocument* contiene diverse variabili d'istanza:

- *pageCount*, che attualmente mantiene in memoria il numero di elementi contenuti nello Spine dell'EPUB, ma che in futuro verrà modificato per salvare il numero di schermate del documento calcolate in base alla dimensione logica della *View* su cui è riprodotto;
- *numberSpineElements* e *currentSpineElementIndex*, i quali rappresentano il numero di file all'interno dello Spine, e l'indice del file dello Spine correntemente visualizzato;
- location, che memorizza l'indirizzo dove verrà creata una directory temporanea atta a contenere i libri scompattati e che verrà rimossa alla chiusura del programma;
- *spineList* e *pathSpineElements* che mantengono, rispettivamente, una lista di riferimenti agli elementi dello Spine e un array di URI degli elementi dello Spine estratti nella memoria, per poter essere visualizzati direttamente dalla *WebView*;

- *nameEpub*, che è un identificativo numerico dell'EPUB aperto durante la sessione d'uso del programma.

All'interno del costruttore di questa classe, che riceve il path del file scelto come parametro in ingresso, sono definiti innanzitutto il tipo di documento, ossia EPUB, e il nome della directory temporanea che lo conterrà una volta estratto dal contenitore ZIP. Per leggere l'eBook, e costruire la struttura logica *dell'EpubDocument*, viene usata la libreria Epublib. Viene prima istanziato l'oggetto *EpubReader* per avere il riferimento all'EPUB, e su di esso viene effettuata la chiamata *getSpine*() per poter ottenere accesso allo Spine del libro. Viene quindi istanziata la variabile *spineList* citata precedentemente invocando il metodo *getSpineReferences*(). Dopo aver fatto ciò, è possibile ottenere il numero di file interni allo Spine. Poiché l'EPUB è un archivio, bisogna scompattare il libro in una directory temporanea creata appositamente: per fare ciò si invoca la chiamata *unzip*. Il metodo *unzip(String inputZip, String destinationDirectory)* sfrutta al suo interno *java.util.zip;* riceve in ingresso due stringhe, una contenente l'indirizzo del file da scompattare e una contenente l'indirizzo della directory all'interno della quale il file va estratto.

Successivamente, prima di inizializzare le celle dell'array *pathSpineElements* con gli URI dei file contenuti nello Spine dell'eBook e terminare così il costruttore, si compie una chiamata al metodo *getPathOPF*, mostrato di seguito:

```
private static String getPathOPF (String unzipDir) {
    String pathOPF = "";
    // get the OPF path, directly from container.xml
    try {
       BufferedReader br = new BufferedReader(new FileReader(unzipDir + "/META-INF/container.xml"));
       String line;
       while ((line = br.readLine()) != null) {
          if (line.indexOf("full-path") > -1) {
                    int start = line.indexOf("full-path");
                   int start2 = line.indexOf("\"", start);
                   int stop2 = line.indexOf("\"", start2 + 1);
                   if (start2 > -1 && stop2 > start2) {
                             pathOPF = line.substring(start2 + 1, stop2).trim();
                             break;
                   }
          }
       }
       br.close();
    } catch(IOException e) {
         // do nothing
    }
    // in case the OPF file is in the root directory
    pathOPF = "./" + pathOPF;
    // remove the OPF file name and the preceding '/'
    int last = pathOPF.lastIndexOf('/');
    if (last > -1) {
         pathOPF = pathOPF.substring(0, last);
    }
    return pathOPF;    }
```

Questo metodo restituisce una stringa contenente l'URI dell'OPF contenuto all'interno del file *META-INF/container.xml* dell'EPUB. Come descritto in precedenza, l'OPF contiene l'elenco delle risorse (pagine XHTML, immagini, fogli di stile, ecc.), i cui URI sono relativi alla directory contente l'OPF.

A differenza di *DjVuDocument* e *PdfDocument*, la classe *EpubDocument* contiene il metodo *deleteDir (File f)*, che sarà richiamato nel *Controller* tramite il metodo *destroy* (come vedremo più avanti nella trattazione) per cancellare, alla chiusura del programma, la directory temporanea contenente gli EPUB scompattati e tutte le sue sottodirectory.

Infine, si trovano tre metodi che consentono di navigare tra le risorse dell'Ebook: *goToPage*, *nextChapter* e *prevChapter*. Il metodo *goToPage (int page)* restituisce in uscita una stringa contenente l'URI dell'elemento dello Spine da visualizzare, in base al numero passato in ingresso. I metodi *nextChapter* e *prevChapter* richiamano al loro interno il metodo *goToPage* per restituire rispettivamente l'indirizzo del capitolo successivo e precedente (ossia l'elemento successivo e precedente dello Spine) del libro.

### *4.2 SPECIALIZEDWEBVIEW*

Come già accennato in precedenza, per supportare l'EPUB, non è più possibile utilizzare la classe *OrionView* per visualizzare il libro su schermo, poiché essa è derivata da *View* e supporta solo il rendering di una bitmap, mentre l'EPUB contiene pagine XHTML. Per questo motivo è stata creata una nuova classe, *SpecializedWebView*, la quale estende *WebView* mantenendo le modalità di gestione dell'eBook preesistenti.
L'estensione è stata scritta in modo da essere ulteriormente sviluppabile da terzi in futuro, ad esempio per aggiungere facilmente ulteriori features.
In maniera analoga alla classe *OrionView, SpecializedWebView* aggiunge a *View* una variabile di tipo *Controller*, potendo accoppiare così tramite il metodo *setController (Controller controller)* la *WebView* e il *Controller* che gestirà l'EPUB visualizzato in essa.
Inoltre è stata prevista una variabile boolean *isNightMode*, la quale segnala se è abilitata o meno la modalità notturna (attualmente non implementata), e due metodi getter/setter, *isNightMode()* e *setNightMode(boolean nightMode)*. Infine è presente un metodo, *onSizeChanged (int w, int h, int oldw, int oldh)*, il quale viene invocato nel caso in cui le dimensioni della *View* vengano modificate. In questo caso sarà notificato al *Controller* il cambiamento, in modo che esso possa effettuare le opportune correzioni sulla *WebView* controllata.

### *4.3 CONTROLLER*

La classe *Controller* si occupa di gestire tutte le azioni dell'utente, interpretandole e manipolando di conseguenza la visualizzazione del documento. Come si può osservare dal suo costruttore, *Controller(OrionViewerActivity activity, DocumentWrapper doc, LayoutStrategy layout, View view)* sono accoppiati all'oggetto *Controller* l'activity principale (*OrionViewerActivity*), l'oggetto *DocumentWrapper* che incapsula il documento, il *LayoutStrategy* necessario alla gestione del layout della pagina visualizzata e la *View* stessa. Per quanto concerne la gestione di PDF e DjVu si trova inoltre l'inizializzazione, sempre all'interno del costruttore della classe, di un thread (*RenderThread*) e il suo avvio. Il thread si incarica del rendering della bitmap della pagina e la gestione della cache. È stato dunque necessario aggiungere uno switch, basato sul risultato di *doc.getDocType()*, per gestire diversamente i documenti PDF/DjVu (formati paginati) dall'EPUB (formato liquido).
È presente un metodo *drawPage(int page)* che si occupa di visualizzare la pagina in base al suo numero, passato come parametro. Questo metodo inizialmente compie una chiamata al metodo *reset()* sull'oggetto *LayoutStrategy*, che aggiornerà i contatori e i dati relativi al layout di pagina. Nel caso di PDF e DjVu, poi, esso richiama il metodo *drawPage()*, dove si ha la gestione concreta della visualizzazione della pagina con il *RenderThread* e le informazioni aggiornate del layout. Per la gestione degli EPUB è stata inserita l'invocazione del metodo *goToPage* sull'oggetto *DocumentWrapper* per ottenere l'URI che poi sarà caricato nella *WebView*.

```
public void drawPage(int page) {
        layout.reset(layoutInfo, page);
        switch (doc.getDocType()) {
                case PDF:
                  break;
                case EPUB:
                   try {
                           // page ranges from 1 to N but index ranges from 0 to N-1
                           String url = ((EpubDocument) doc).goToPage(page - 1);
                          if(!url.equals(((SpecializedWebView)view).getUrl())) {
                                  ((SpecializedWebView)view).loadUrl(url); }
                   } catch(Exception e) {}
                  break;}
        drawPage();}
  public void drawPage() {
        switch (doc.getDocType()) {
                case PDF:
                   sendPageChangedNotification();
                   renderer.render(layoutInfo);
                   break;
                 case EPUB:
                    sendPageChangedNotification();
                    break;
    }
  }
```

Si trovano quindi i metodi *drawNext()* e *drawPrev()* che scrollano rispettivamente in avanti e all'indietro l'eBook; nel caso di PDF e DjVu si osserva la stessa gestione precedente, con il richiamo del metodo *drawPage* dopo aver aggiornato il layout. Con gli EPUB invece si utilizzano i metodi *pageDown* e *pageUp* forniti dalla classe *WebView*, invocati sulla *View* stessa.
Con questa tecnica, si pone il problema della transizione tra capitoli, ossia bisogna gestire la transizione da un capitolo all'altro dello Spine una volta che sul viewport sia stata visualizzata la "fine" o l'"inizio" della pagina XHTML e l'utente voglia proseguire nella lettura o tornare indietro. Per gestire questa transizione si salva in una variabile booleana il parametro d'uscita dei metodi *pageDown* e *pageUp*; se la variabile ha valore false significa che la *WebView* non è stata scrollata ulteriormente e quindi bisogna cambiare capitolo. Nel *drawNext* viene usato il metodo *nextChapter* invocato sul *DocumentWrapper* per ottenere il path del capitolo successivo da caricare nella *View*. Nel drawPrev invece viene prima richiamato il metodo *prevChapter* sul documento per visualizzare il capitolo precedente caricando l'URI ottenuto, poi vengono utilizzati oggetti *Handler* e *Runnable* per gestire lo scrolling del capitolo ora mostrato fino a fine pagina, come nel codice qui di seguito:

```
try {
                url = ((EpubDocument) doc).prevChapter();
                if (!url.equals(((SpecializedWebView)view).getUrl())) {
                    ((SpecializedWebView)view).loadUrl(url);
                    final Handler mWebViewScrollHandler = new Handler();

                   final Runnable mScrollWebViewTask = new Runnable() {
                            public void run() {
                                    ((SpecializedWebView)view).pageDown(true);
                            }
                    };
                   mWebViewScrollHandler.removeCallbacks(mScrollWebViewTask);
                   mWebViewScrollHandler.postDelayed(mScrollWebViewTask, 100);
                }
 } catch (Exception e) {
                e.printStackTrace();
}
```

In questo modo si attende che la pagina sia caricata del tutto per essere scrollata fino al termine.

Un altro metodo che ha richiesto una gestione differente rispetto gli altri formati supportati è stato *changeZoom(int zoom)*. Per eseguire lo zoom di un EPUB, oltre alla chiamata a un oggetto *LayoutStrategy* per aggiornarne i valori, si agisce direttamente sulla *SpecializedWebView* con il codice:

```
((SpecializedWebView)view).getSettings().setTextZoom(zoom / 100);
```

Si sfrutta così il metodo *setTextZoom* della classe *WebView*, che richiede come parametro d'ingresso lo zoom percentuale (viene effettuata la divisione per cento rispetto al valore percentuale scelto dall'utente) ed agisce sui caratteri del testo ingrandendoli.
Nei metodi *onPause()* e *destroy()*, dedicati rispettivamente a mettere in pausa il thread per il rendering e a "distruggere" i documenti, sono stati aggiunti degli switch in quanto il codice relativo al *RenderThread* non è necessario nel caso degli EPUB, come scritto in precedenza. La stessa

osservazione vale per il metodo *screenSizeChanged (int newWidth, int newHeight),* poiché non è ancora stata implementata questa funzione per l'EPUB.

### *4.4 ORIONVIEWERACTIVITY*

Questa classe rappresenta l'activity principale dell'applicazione, occupandosi di visualizzare un documento sullo schermo tramite un oggetto di tipo *View* e di chiamare il *Controller* quando è necessario gestire l'interazione con l'utente. La prima modifica che si è resa necessaria è stata l'aggiunta di una *SpecializedWebView*, sia al codice dell'activity, sia al corrispondente file XML. Come l'istanza di tipo *OrionView*, questa *WebView* è inizializzata dal metodo *onCreate*. Nel caso in cui il documento aperto sia un EPUB, le dimensioni *dell'OrionView* esistente vengono impostate a zero, mentre la *SpecializedWebView* viene impostata a tutto schermo per poter visualizzare l'eBook. Al contrario, se il documento è un PDF o DjVu, la *SpecializedWebView* viene impostata con dimensioni nulle e le dimensioni *dell'OrionView* vengono impostate secondo le dimensioni dello schermo, come mostrato nel frammento di codice seguente:

```
try {
        String filePAthLowCase = filePath.toLowerCase();
        if (filePAthLowCase.endsWith("pdf") || filePAthLowCase.endsWith("xps") ||
            filePAthLowCase.endsWith("cbz")) {
                view.getLayoutParams().height = LayoutParams.WRAP_CONTENT;
                view.getLayoutParams().width = LayoutParams.WRAP_CONTENT;
                viewW.getLayoutParams().height = 0;
                viewW.getLayoutParams().width = 0;
                doc = new PdfDocument(filePath);
        } else if (filePAthLowCase.endsWith("DjVu")) {
                view.getLayoutParams().height = LayoutParams.WRAP_CONTENT;
                view.getLayoutParams().width = LayoutParams.WRAP_CONTENT;
                viewW.getLayoutParams().height = 0;
                viewW.getLayoutParams().width = 0;
                doc = new DjVuDocument(filePath);
        } else {
                view.getLayoutParams().height = 0;
                view.getLayoutParams().width = 0;
                viewW.getLayoutParams().height = LayoutParams.WRAP_CONTENT;
                viewW.getLayoutParams().width = LayoutParams.WRAP_CONTENT;
                viewW.getSettings().setJavaScriptEnabled(true);
                doc = new EpubDocument(filePath);
         }
}
```

Dopo aver fatto ciò, in base al formato del documento aperto, viene inizializzato l'oggetto *Controller* con i parametri adeguati. Viene quindi "disegnata" la pagina da visualizzare, richiamando il metodo *drawPage* sul *Controller* in base al tipo dell'eBook: nel caso si abbia un EPUB viene inserita una chiamata *drawPage(int n)*, dove n=1 sarà il numero della pagina da visualizzare all'apertura, ossia la prima.

Al restante codice della classe è stato aggiunto un metodo *getViewW ()* per manipolare la *SpecializedWebView* successivamente; inoltre è stato impostato un *TouchListener* alla *WebView*, analogamente a quanto impostato per *l'OrionView*, che andrà a gestire gli eventi touch, comunicando le opportune azioni al *Controller*.

# CAPITOLO 5

## CONCLUSIONI E SVILUPPI FUTURI

Questo studio ha avuto inizio dall'osservazione che non esiste sul mercato un'applicazione matura per Android che sia in grado di visualizzare un eBook in formato EPUB fedelmente, ossia rispettando le impostazioni tipografiche dell'autore o dell'editore.

Inizialmente è stata verificata la possibilità di visualizzare una pagina XHTML (poiché il testo all'interno di un eBook EPUB si presenta in tale formato) tramite una *WebView* osservandone il rispetto della struttura e della resa tipografica, come definiti dal relativo foglio di stile CSS. La verifica è stata positiva.

Basandosi sui risultati della verifica, si è successivamente introdotto il supporto EPUB3 a livello prototipale nel lettore di eBook open source Orion Viewer. Il supporto al formato EPUB è stato implementato in parallelo ai preesistenti PDF e DjVu. Per conseguire questo obiettivo:

- si è studiato il formato EPUB e la libreria Epublib, che consente di gestire un eBook in tale formato;
- si sono studiate le funzionalità e le modalità di gestione dei documenti da parte di Orion Viewer, per poter estendere l'app rispettando l'architettura originale;
- si è sviluppata la classe *EpubDocument* per poter rappresentare e manipolare un documento in formato EPUB usando le primitive definite dall'interfaccia *DocumentWrapper*, già presente in Orion Viewer;
- si è sviluppata la classe *SpecializedWebView*, estendendo l'oggetto *WebView*, per visualizzare i file XHTML di cui è composto un EPUB, in quanto la classe originale *OrionView* (usata per PDF e DjVu) non supporta il rendering di codice HTML, ma solo di pagine bitmap;
- si è estesa la classe *Controller*, inserendo il codice per manipolare il documento EPUB secondo le azioni compiute dall'utente;
- si è modificata e estesa l'activity principale dell'app, *OrionViewerActivity*, la quale inizializza tutte le componenti necessarie al funzionamento del programma, processa le interazioni dell'utente e si occupa della visualizzazione dell'eBook.

Dalla discussione nei capitoli precedenti appare evidente come si sia deciso di aggiungere il supporto per gli eBook in formato EPUB in Orion Viewer rispettando il più possibile la struttura originaria dell'applicazione. Con questa scelta è naturale che il codice sorgente presenti delle ridondanze e necessiti di un'ampia azione di *refactoring*. Tuttavia, essendo lo scopo di questa tesi puramente esplorativo, si è ottenuto rapidamente un triplice risultato:

- l'applicazione finale supporta documenti sia in formato paginato (PDF, DjVu), sia in formato liquido (EPUB);

- la gestione dell'interazione con l'utente è unica per i vari formati supportati, e offre un ampio grado di personalizzazione dell'esperienza di lettura;
- gli eBook EPUB sono visualizzati tramite *WebView* e quindi, a differenza delle altre app Android per leggere EPUB, sono visualizzati nel modo più fedele possibile, rispettando le impostazioni "tipografiche" che il loro autore/editore ha scelto.

Oltre al già citato consolidamento dell'applicazione in termini di pulizia del codice sorgente, questo progetto lascia aperte molte direzioni per interessanti sviluppi futuri. Alcuni esempi includono: l'implementazione di alcune funzioni minori menzionate in precedenza (night mode, conteggio schermate), la gestione della TOC (menù rapido di navigazione) degli EPUB, la possibilità di effettuare l'overriding dei fogli di stile dell'eBook con un CSS definito dall'utente, e la gestione di dizionari e annotazioni.

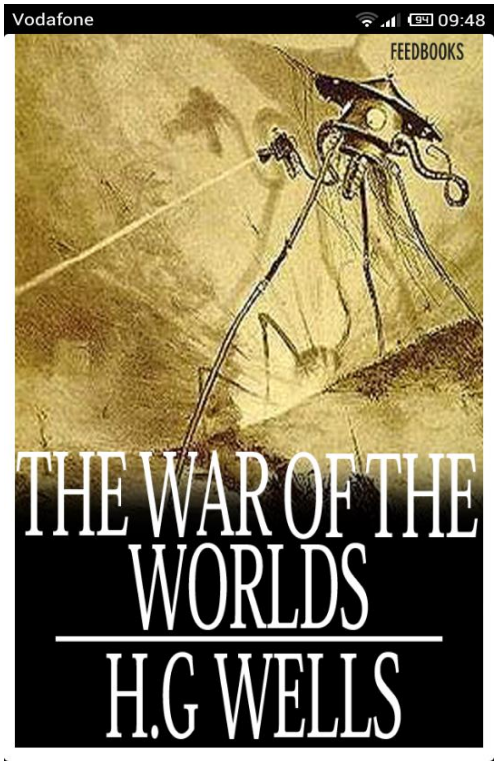
# CAPITOLO 6

## SCREENSHOTS DI ORION VIEWER

Per dimostrare come Orion Viewer, in seguito al lavoro descritto in questa tesi, gestisce un eBook in formato EPUB, di seguito sono proposti alcuni screenshots dell'app in azione.

### 6.1 COPERTINA DI UN LIBRO

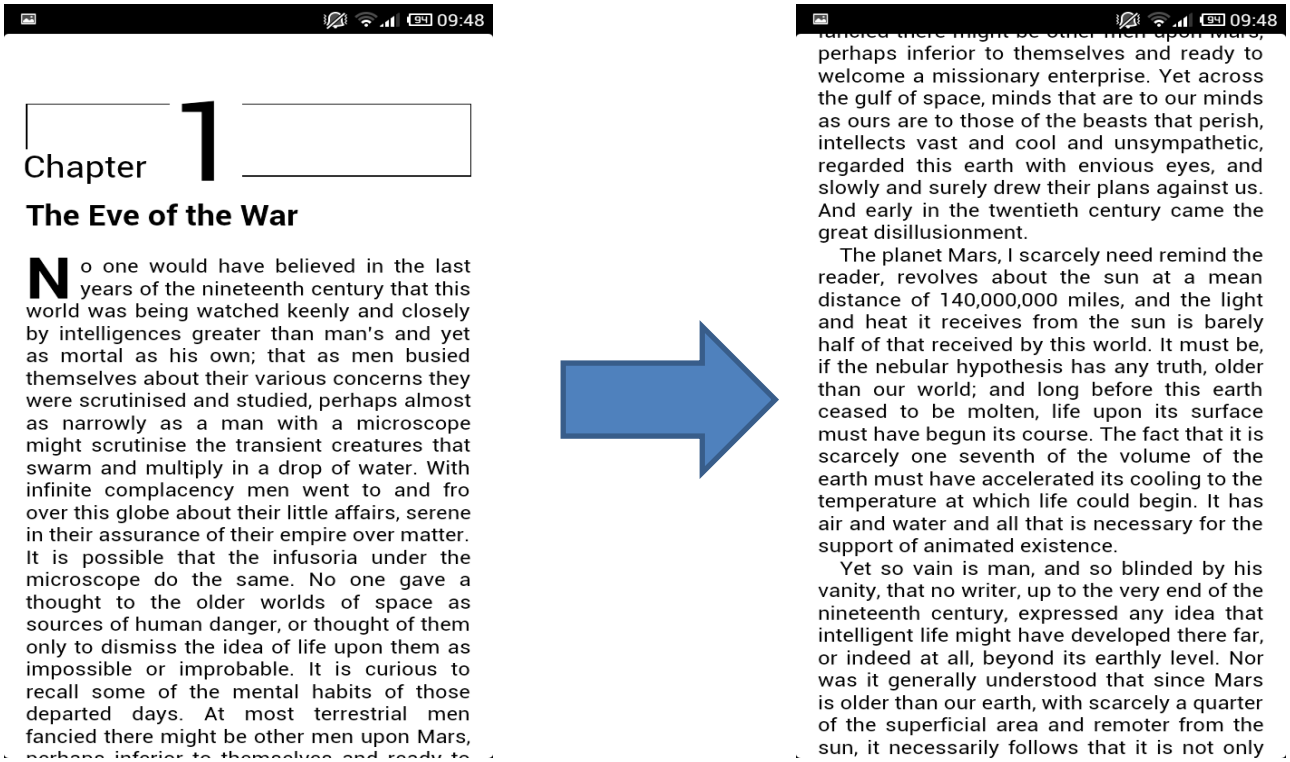
Nelle immagini seguenti si osservano la copertina e la prima pagina di un eBook, in questo caso "The War of the Worlds", liberamente scaricabile dal sito FeedBooks.



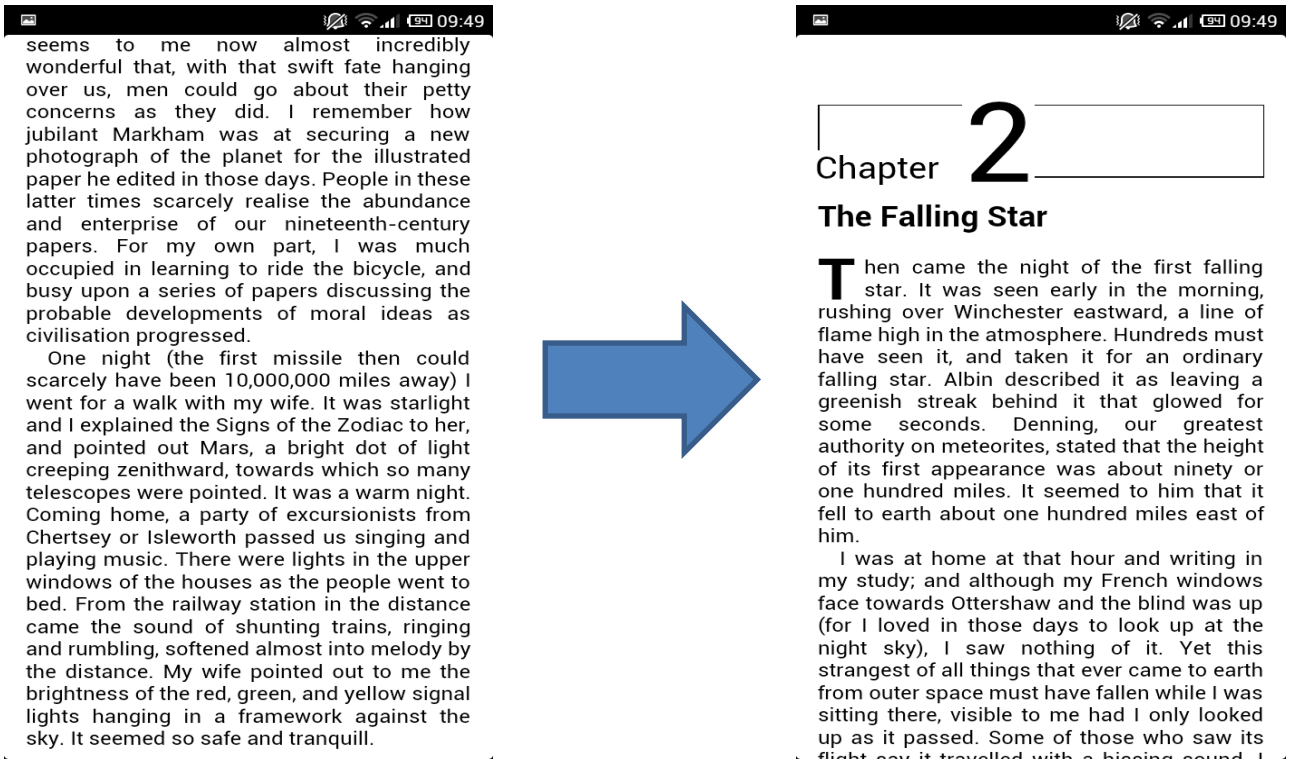

## 6.2 SCROLLING DELL'EBOOK

Ora si può vedere lo scrolling dell'eBook, che si realizza toccando il margine destro (avanti) o sinistro (indietro) dello schermo; questa mappatura delle zone di tocco è personalizzabile dall'utente.

- Inizio del primo capitolo;

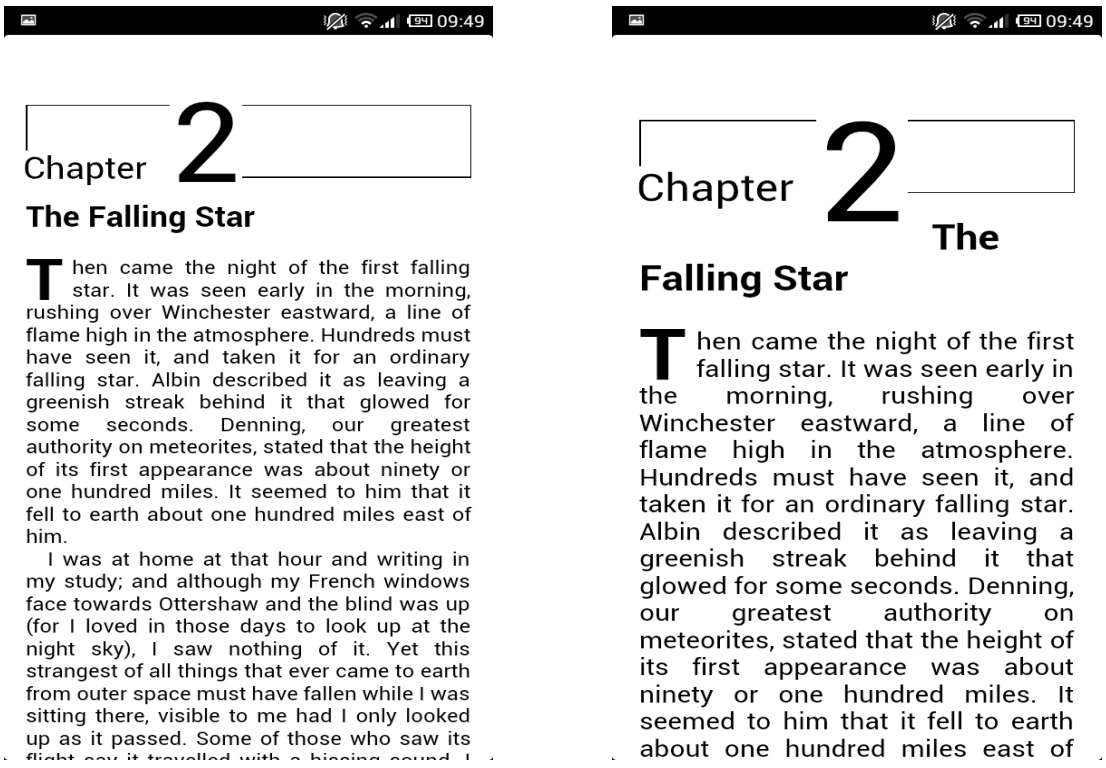


- Fine del primo capitolo: toccando nuovamente a destra, ci si sposta all'inizio del secondo capitolo.



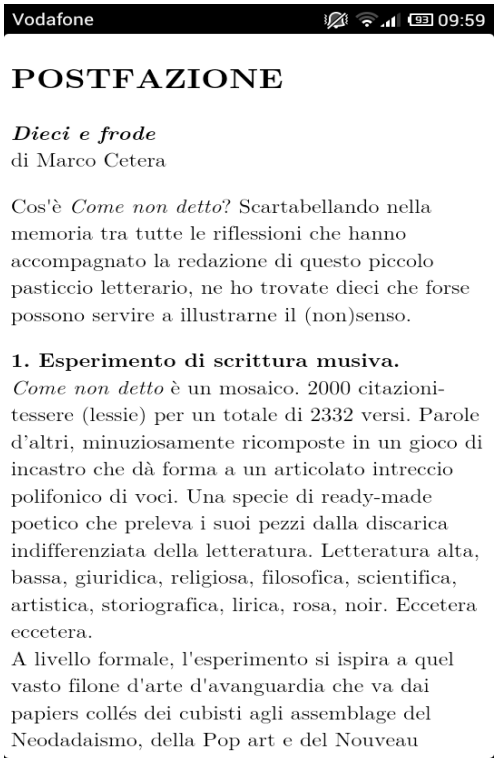

## 6.3 ZOOM DI UNA PAGINA

È mostrato qui di seguito un aumento dello zoom della pagina: è evidente come vengano ingranditi i caratteri del testo.



## 6.4 VISUALIZZAZIONE DI UN EBOOK: IMMAGINI E CARATTERI

Nelle immagini successive viene mostrato come la *WebView* visualizzi un eBook con testo, immagini, e caratteri embedded. (L'eBook è “Come non detto”, di Marco Cetera, liberamente scaricabile da *www.comenondetto.net*)

# CAPITOLO 7

## SITOGRAFIA


- Play Store Google, sezione Orion Viewer
  *[https://play.google.com/store/apps/details?id=universe.constellation.orion.viewer&hl=it&referrer=utm_source%3Dgoogle%26utm_medium%3Dorganic%26utm_term%3Dorion+viewer]*

- Play Store Google
  *[http://play.google.com/store?hl=it]*

- Pagina del progetto "Orion Viewer"
  *[https://code.google.com/p/orion-viewer/]*

- International Digital Publishing Forum
  *[http://idpf.org]*

- Tutorial sull'EPUB
  *[http://www.ibm.com/developerworks/xml/tutorials/x-epubtut/index.html]*

- Pagina Framework "Epublib"
  *[http://www.siegmann.nl/epublib]*

- Android Developers, sezione WebView
  *[http://developer.android.com/reference/android/webkit/WebView.html]*

- Android StackOverflow
  *[http://stackoverflow.com]*

- Giornale "La Rapubblica"
  *[http://www.repubblica.it]*

# APPENDICE

## 1 LA CLASSE EPUBDOCUMENT

```
package universe.constellation.orion.viewer.epub;

import universe.constellation.orion.viewer.Common;
import universe.constellation.orion.viewer.DocumentWrapper;
import universe.constellation.orion.viewer.outline.OutlineItem;
import universe.constellation.orion.viewer.PageInfo;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.Reader;
import java.util.ArrayList;
import java.util.Date;
import java.util.Enumeration;
import java.util.Iterator;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipFile;
import java.util.zip.ZipInputStream;

import nl.siegmann.epublib.domain.Book;
import nl.siegmann.epublib.domain.Resource;
import nl.siegmann.epublib.domain.ResourceReference;
import nl.siegmann.epublib.domain.Spine;
import nl.siegmann.epublib.domain.SpineReference;
import nl.siegmann.epublib.domain.TOCReference;
import nl.siegmann.epublib.domain.TableOfContents;
import nl.siegmann.epublib.epub.EpubReader;

import android.util.Log;
import android.webkit.WebView;

public class EpubDocument implements DocumentWrapper {

    private int pageCount;

    private int currentSpineElementIndex;
    private int numberSpineElements;
```

```
private docType dtype;

private Book book;

private static String nameEpub;
private static int nE = 1;

List<SpineReference> spineList;
String[] pathSpineElements;

String location = "/mnt/sdcard/epubtemp/";

public EpubDocument(String fileName) throws IOException {

	nameEpub = "" + ++nE;

	dtype = docType.EPUB;

	try {
		book = (new EpubReader()).readEpub(new FileInputStream(fileName));
	} catch (IOException e) {
		Log.e("epublib", e.getMessage());
	}

	Spine spine = book.getSpine();
	spineList = spine.getSpineReferences();
	numberSpineElements = spineList.size();
	pageCount = numberSpineElements;
	currentSpineElementIndex = 0;

	pathSpineElements = new String[pageCount];
	Log.v("TAG", fileName);
	Log.v("TAG", location + nameEpub + "/");

	unzip(fileName, location + nameEpub);

	String pathOPF = getPathOPF(location + nameEpub);
	Log.v("TAG", pathOPF);

	// store paths of spine elements
	for (int i = 0; i < spineList.size(); ++i) {
		pathSpineElements[i] = ("file://" + location + nameEpub + "/" + pathOPF + "/" +
		((spineList.get(i)).getResource()).getHref());
	}
}

public boolean openDocument(String fileName) {
	return true;
}
```

```
private static String getPathOPF(String unzipDir) {
	String pathOPF = "";

	// get the OPF path, directly from container.xml
	try {
		BufferedReader br = new BufferedReader(new FileReader(unzipDir + "/META-
		INF/container.xml"));
		String line;
		while ((line = br.readLine()) != null) {
			if (line.indexOf("full-path") > -1) {
				int start = line.indexOf("full-path");
				int start2 = line.indexOf("\"", start);
				int stop2 = line.indexOf("\"", start2 + 1);
				if (start2 > -1 && stop2 > start2) {
					pathOPF = line.substring(start2 + 1, stop2).trim();
					break;
				}
			}
		}
		br.close();
	} catch(IOException e) {
		// do nothing
	}

	 // in case the OPF file is in the root directory
	pathOPF = "./" + pathOPF;
	// remove the OPF file name and the preceding '/'
	int last = pathOPF.lastIndexOf('/');
	if (last > -1) {
		pathOPF = pathOPF.substring(0, last);
	}

	return pathOPF;
}

public void unzip(String inputZip, String destinationDirectory)
throws IOException {
	int BUFFER = 2048;
	List zipFiles = new ArrayList();
	File sourceZipFile = new File(inputZip);
	File unzipDestinationDirectory = new File(destinationDirectory);
	unzipDestinationDirectory.mkdir();

	ZipFile zipFile;
	zipFile = new ZipFile(sourceZipFile, ZipFile.OPEN_READ);
	Enumeration zipFileEntries = zipFile.entries();

	// Process each entry
	while (zipFileEntries.hasMoreElements()) {
		ZipEntry entry = (ZipEntry) zipFileEntries.nextElement();
		String currentEntry = entry.getName();
```

```
                File destFile = new File(unzipDestinationDirectory, currentEntry);

                if (currentEntry.endsWith(".zip")) {
                        zipFiles.add(destFile.getAbsolutePath());
                }
                File destinationParent = destFile.getParentFile();
                destinationParent.mkdirs();
                try {
                        if (!entry.isDirectory()) {
                                BufferedInputStream is = new
                                 BufferedInputStream(zipFile.getInputStream(entry));
                                int currentByte;
                                // buffer for writing file
                                byte data[] = new byte[BUFFER];

                                 FileOutputStream fos = new FileOutputStream(destFile);
                                 BufferedOutputStream dest =new BufferedOutputStream(fos, BUFFER);

                                while ((currentByte = is.read(data, 0, BUFFER)) != -1) {
                                        dest.write(data, 0, currentByte);
                                }
                                dest.flush();
                                dest.close();
                                is.close();
                }
                } catch (IOException ioe) {
                        ioe.printStackTrace();
                }
        }
        zipFile.close();

        for (Iterator iter = zipFiles.iterator(); iter.hasNext();) {
                String zipName = (String)iter.next();
                unzip(zipName,  destinationDirectory + File.separatorChar +
                zipName.substring(0,zipName.lastIndexOf(".zip")));
        }
}

public int getPageCount() {
        return pageCount;
}

public PageInfo getPageInfo(int pageNum) {
        PageInfo info = new PageInfo();
        return info;
}

public int[] renderPage(int pageNumber, double zoom, int w, int h, int left, int top, int right, int bottom)
{
        return null;
}
```

```
    public String getText(int pageNumber, int absoluteX, int absoluteY, int width, int height) {
        return null;
    }

    public void destroy() {
        File c = new File(location);
        deleteDir(c);
    }
    public void deleteDir(File f) {
         if (f.isDirectory())
                for (File child : f.listFiles())
                        deleteDir(child);
         f.delete();
    }

    public String getTitle() {
        return this.book.getTitle();
    }

    public void setContrast(int contrast) {
    }

    public void setThreshold(int threshold) {
    }

    public OutlineItem[] getOutline() {
        return null;
    }

    public docType getDocType() {
        return dtype;
    }

    public String goToPage(int page) throws Exception {
        if (page < 0)
                page = 0;
        if (page >= numberSpineElements)
                page = numberSpineElements - 1;
        currentSpineElementIndex = page;
        return pathSpineElements[currentSpineElementIndex];
    }

    public String nextChapter() throws Exception {
        return goToPage(currentSpineElementIndex + 1);
    }

    public String prevChapter() throws Exception {
        return goToPage(currentSpineElementIndex - 1);
    }
}
```

## 2 LA CLASSE SPECIALIZEDWEBVIEW

```
package universe.constellation.orion.viewer;

import android.content.Context;
import android.graphics.*;
import android.util.AttributeSet;
import android.view.View;
import android.webkit.WebView;
import universe.constellation.orion.viewer.device.Nook2Util;
import universe.constellation.orion.viewer.prefs.GlobalOptions;

import java.util.Date;
import java.util.concurrent.CountDownLatch;

public class SpecializedWebView extends WebView {

    private Controller controller;

    private int counter = 0;

    private boolean isNightMode = true;

    public SpecializedWebView(Context context) {
        super(context);
    }

    public SpecializedWebView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    public SpecializedWebView(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
    }

    public void setController(Controller controller) {
        this.controller = controller;
    }

    @Override
    protected void onSizeChanged(int w, int h, int oldw, int oldh) {
        Common.d("SpecializedWebView: onSizeChanged " + w + "x" + h);
        super.onSizeChanged(w, h, oldw, oldh);
        if (w != 0 && h != 0 && (w != oldw || h != oldh)) {
            if (controller != null ) {
                controller.screenSizeChanged(w, h);
            }
        }
    }
```

```
    public boolean isNightMode() {
        return isNightMode;
    }

    public void setNightMode(boolean nightMode) {
        this.isNightMode = nightMode;
    }
}
```

## 3 LA CLASSE CONTROLLER

```
package universe.constellation.orion.viewer;

import android.graphics.Point;
import android.os.Handler;
import android.util.Log;
import android.view.View;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.GridLayout.Spec;
import android.widget.Toast;
import universe.constellation.orion.viewer.epub.EpubDocument;
import universe.constellation.orion.viewer.outline.OutlineItem;
import universe.constellation.orion.viewer.prefs.GlobalOptions;

public class Controller {

    private LayoutPosition layoutInfo;

    private DocumentWrapper doc;

    private LayoutStrategy layout;

    private View view;

    private OrionViewerActivity activity;

    private RenderThread renderer;

    private int lastPage;

    private DocumentViewAdapter listener;

    private String screenOrientation;

    private Point lastScreenSize;

    private int contrast;

    private int threshold;

    private boolean hasPendingEvents = false;

    public Controller(OrionViewerActivity activity, DocumentWrapper doc, LayoutStrategy layout, View view)
    {
        this.activity = activity;
        this.doc = doc;
        this.layout = layout;
        this.view = view;
```

```
        this.lastPage = -1;
        switch (doc.getDocType()) {
                case PDF:
                        renderer = new RenderThread(activity, (OrionView)view, layout, doc);
                        renderer.start();
                        listener = new  DocumentViewAdapter() {
                                public void viewParametersChanged() {
                                        if (Controller.this.activity.isResumed) {
                                                ((RenderThread) renderer).invalidateCache();
                                                drawPage();
                                                hasPendingEvents = false;
                                        } else {
                                                hasPendingEvents = true;
                                        }
                                }
                        };
                        activity.getSubscriptionManager().addDocListeners(listener);
                        break;
                case EPUB:
                        Log.v("controller", "costruttore controller");
                        listener = new  DocumentViewAdapter();
                        activity.getSubscriptionManager().addDocListeners(listener);
                        break;
        }
}

public void drawPage(int page) {
        layout.reset(layoutInfo, page);
        switch (doc.getDocType()) {
                case PDF:
                        break;
                case EPUB:
                        try {
                                // page ranges from 1 to N
                                // but index ranges from 0 to N-1
                                String url = ((EpubDocument) doc).goToPage(page - 1);
                                if(!url.equals(((SpecializedWebView)view).getUrl())) {
                                        ((SpecializedWebView)view).loadUrl(url);
                                 }
                        } catch(Exception e) {
                        }
                        break;
        }
        drawPage();
}

public void drawPage() {
        switch (doc.getDocType()) {
                case PDF:
                        sendPageChangedNotification();
                        renderer.render(layoutInfo);
```

```
                    break;
                case EPUB:
                    sendPageChangedNotification();
                    break;
        }
    }

    public void processPendingEvents() {
    }

    public void screenSizeChanged(int newWidth, int newHeight) {
        switch (doc.getDocType()) {
                case PDF:
                    Common.d("New screen size " + newWidth + "x" + newHeight);
                    layout.setDimension(newWidth, newHeight);
                    GlobalOptions options = getActivity().getGlobalOptions();
                     layout.changeOverlapping(options.getHorizontalOverlapping(),
                     options.getVerticalOverlapping());
                    int offsetX = layoutInfo.cellX;
                    int offsetY = layoutInfo.cellY;
                    layout.reset(layoutInfo, layoutInfo.pageNumber);
                    if (lastScreenSize != null) {
                        if (newWidth == lastScreenSize.x && newHeight == lastScreenSize.y) {
                            layoutInfo.cellX = offsetX;
                            layoutInfo.cellY = offsetY;
                        }
                        lastScreenSize = null;
                    }
                    sendViewChangeNotification();
                    renderer.onResume();
                    break;
                case EPUB:
                    break;
       }
    }

    public void drawNext() {
        layout.nextPage(layoutInfo);
        switch (doc.getDocType()) {
                case PDF:
                    break;
                case EPUB:
                    boolean isOk = ((SpecializedWebView) view).pageDown(false);
                    if (!isOk) {
                        try {
                            String url= ((EpubDocument) doc).nextChapter();
                            if(!url.equals(((SpecializedWebView)view).getUrl())){
                                ((SpecializedWebView)view).loadUrl(url);
                            }
                        } catch(Exception e) {
                        }
```

```
                        }
                        break;
        }
        drawPage();
}

public void drawPrev() {
        layout.prevPage(layoutInfo);
        switch (doc.getDocType()) {
                case PDF:
                        break;
                case EPUB:
                        boolean isOk = ((SpecializedWebView) view).pageUp(false);
                        if (!isOk) {
                                String url;
                                try {
                                        url = ((EpubDocument) doc).prevChapter();
                                        if (!url.equals(((SpecializedWebView)view).getUrl())) {
                                        ((SpecializedWebView)view).loadUrl(url);
                                        final Handler mWebViewScrollHandler = new Handler();
                                        final Runnable mScrollWebViewTask = new Runnable() {
                                                public void run() {
                                                        ((SpecializedWebView)view).pageDown(true);
                                                } };
                                        mWebViewScrollHandler.removeCallbacks(mScrollWebViewTask);
                                        mWebViewScrollHandler.postDelayed(mScrollWebViewTask, 100);
                                        }
                                } catch (Exception e) {
                                        e.printStackTrace();
                                }
                        }
                        break;
        }
        drawPage();
}

public void changeZoom(int zoom) {
        if (layout.changeZoom(zoom)) {
                layout.reset(layoutInfo, layoutInfo.pageNumber);}
        switch (doc.getDocType()) {
                case PDF:
                        break;
                case EPUB:
                        Log.v("ZOOM", "zoomming");
                          ((SpecializedWebView)view).getSettings().setTextZoom(zoom / 100);
                          break;
        }
        sendViewChangeNotification();
}

public int getZoom10000Factor() {
```

```
}

 public double getCurrentPageZoom() {
 }

//left, top, right, bottom
public void changeMargins(int [] margins) {
}

public void changeMargins(int leftMargin, int topMargin, int rightMargin, int bottomMargin, boolean
isEven, int leftEvenMargin, int rightEvenMargin) {
}

public void getMargins(int [] cropMargins) {
}

public boolean isEvenCropEnabled() {
}

public void destroy() {
	Common.d("Destroying controller...");
	activity.getSubscriptionManager().unSubscribe(listener);
	switch (doc.getDocType()) {
		case PDF:
			if (renderer != null ) {
				renderer.stopRenderer();
				renderer = null;
			}
			 if (doc != null) {
				doc.destroy();
				doc = null;
			}
			break;
		case EPUB:
			if (doc != null){
				doc.destroy();
				doc = null;
			}
			break;
	}
	System.gc();
}

public void onPause() {
	switch (doc.getDocType()) {
		case PDF:
			renderer.onPause();
			break;
		case EPUB:
			break;
	  }
```

```
}

public void setRotation(int rotation) {
}

public void changeOverlap(int horizontal, int vertical) {
 }

public int getRotation() {
}

public int getCurrentPage() {
}

public int getPageCount() {
}

public void init(LastPageInfo info) {
}

public void serialize(LastPageInfo info) {
}

public void sendViewChangeNotification() {
}

public void sendPageChangedNotification() {
}

public String getDirection() {
}

public int getLayout() {
}

public void setDirectionAndLayout(String walkOrder, int pageLayout) {
}

public void changetWalkOrder(String walkOrder) {
}

public void changetPageLayout(int pageLayout) {
}

public void changeContrast(int contrast) {
}

public void changeThreshhold(int threshold) {
}
```

```
    public OrionViewerActivity getActivity() {
    }

    public boolean isEvenPage() {
            //zero based
            return (getCurrentPage() + 1) % 2 == 0;
    }

    public void changeOrinatation(String orientationId) {
    }

    public OutlineItem[] getOutline() {
    }

    public String getScreenOrientation() {
    }

    public String selectText(int startX, int startY, int widht, int height) {
    }
}
```

## 4 LA CLASSE ORIONVIEWERACTIVITY

```
package universe.constellation.orion.viewer;

import android.app.Activity;
import android.app.Dialog;
import android.content.Context;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.os.Debug;
import android.os.SystemClock;
import android.util.Log;
import android.view.*;
import android.view.ViewGroup.LayoutParams;
import android.webkit.WebView;
import android.widget.*;


import android.widget.ImageButton;
import android.widget.RadioButton;
import universe.constellation.orion.viewer.djvu.DjvuDocument;
import universe.constellation.orion.viewer.epub.EpubDocument;
import universe.constellation.orion.viewer.pdf.PdfDocument;
import universe.constellation.orion.viewer.prefs.GlobalOptions;
import universe.constellation.orion.viewer.prefs.OrionPreferenceActivity;
import pl.polidea.customwidget.TheMissingTabHost;
import universe.constellation.orion.viewer.selection.SelectedTextActions;
import universe.constellation.orion.viewer.selection.SelectionAutomata;

import java.io.*;

public class OrionViewerActivity extends OrionBaseActivity {

  private Dialog dialog;

  public static final int OPEN_BOOKMARK_ACTIVITY_RESULT = 1;

   public static final int ROTATION_SCREEN = 0;

   public static final int MAIN_SCREEN = 0;

   public static final int PAGE_SCREEN = 1;

   public static final int ZOOM_SCREEN = 2;

   public static final int CROP_SCREEN = 3;

   public static final int PAGE_LAYOUT_SCREEN = 4;

   public static final int ADD_BOOKMARK_SCREEN = 5;
```

```
public static final int HELP_SCREEN = 100;

public static final int CROP_RESTRICTION_MIN = -10;

private static final int CROP_DELTA = 10;

public static final int CROP_RESTRICTION_MAX = 40;

private final SubscriptionManager manager = new SubscriptionManager();

private SpecializedWebView viewW;

private OrionView view;

private ViewAnimator animator;

private LastPageInfo lastPageInfo;

//left, right, top, bottom
private int [] cropBorders = new int[6];

private Controller controller;

private OperationHolder operation = new OperationHolder();

private GlobalOptions globalOptions;

private Intent myIntent;

public boolean isResumed;

private boolean selectionMode = false;

private SelectionAutomata textSelection;

private SelectedTextActions selectedTextActions;

 int numD=1;

@Override
public void onCreate(Bundle savedInstanceState) {
    loadGlobalOptions();

    getOrionContext().setViewActivity(this);
    OptionActions.FULL_SCREEN.doAction(this, !globalOptions.isFullScreen(),globalOptions.isFullScreen());

    super.onCreate(savedInstanceState);
    setContentView(device.getLayoutId());

    view = (OrionView) findViewById(R.id.view);
```

```
    String mode=getOrionContext().getOptions().getStringProperty(GlobalOptions.DAY_NIGHT_MODE,
      "DAY");
    ((OrionView) view).setNightMode("NIGHT".equals(mode));

    viewW=  (SpecializedWebView) findViewById(R.id.web_view);

    if (!device.optionViaDialog()) {
        initAnimator();
        initMainScreen();
    } else {
        initOptionDialog();
        initRotationScreen();
    }

    //page chooser
    initPagePeekerScreen();

    initZoomScreen();

    initCropScreen();

    initPageLayoutScreen();

    initAddBookmarkScreen();

    myIntent = getIntent();
}

protected void initHelpScreen() {
}

public void updateCrops() {
}

public void updatePageLayout() {
}

protected void onNewIntent(Intent intent) {
}

public void openFile(String filePath) {
    DocumentWrapper doc = null;
    Common.d("File URI  = " + filePath);
    getOrionContext().onNewBook(filePath);
    try {
        String filePAthLowCase = filePath.toLowerCase();
        if (filePAthLowCase.endsWith("pdf")|| filePAthLowCase.endsWith("xps")||
            filePAthLowCase.endsWith("cbz")) {
                view.getLayoutParams().height = LayoutParams.WRAP_CONTENT;
                view.getLayoutParams().width = LayoutParams.WRAP_CONTENT;
                viewW.getLayoutParams().height = 0;
```

```
            viewW.getLayoutParams().width = 0;
            doc = new PdfDocument(filePath);
} else if (filePAthLowCase.endsWith("djvu")) {
            view.getLayoutParams().height = LayoutParams.WRAP_CONTENT;
            view.getLayoutParams().width = LayoutParams.WRAP_CONTENT;
            viewW.getLayoutParams().height = 0;
            viewW.getLayoutParams().width = 0;
            doc = new DjvuDocument(filePath);
} else {
            view.getLayoutParams().height = 0;
            view.getLayoutParams().width = 0;
            viewW.getLayoutParams().height = LayoutParams.WRAP_CONTENT;
            viewW.getLayoutParams().width = LayoutParams.WRAP_CONTENT;
            viewW.getSettings().setJavaScriptEnabled(true);
            doc = new EpubDocument(filePath);
}
LayoutStrategy str = new SimpleLayoutStrategy(doc, device.getDeviceSize());
switch (doc.getDocType()) {
            case PDF:
                        controller = new Controller(this, doc, str,  view);
                        break;
            case EPUB:
                        controller = new Controller(this, doc, str,  viewW);
                        break;
}

int idx = filePath.lastIndexOf('/');

lastPageInfo = LastPageInfo.loadBookParameters(this, filePath);

getOrionContext().setCurrentBookParameters(lastPageInfo);

OptionActions.DEBUG.doAction(this,false, getGlobalOptions().getBooleanProperty("DEBUG",false));

controller.changeOrinatation(lastPageInfo.screenOrientation);
controller.init(lastPageInfo);
getSubscriptionManager().sendDocOpenedNotification(controller);

switch (doc.getDocType()) {
            case PDF:
                        ((OrionView) getView()).setController(controller);
                        break;
            case EPUB:
                        controller.drawPage(2);
                         break;
}
controller.drawPage();

String title = doc.getTitle();
if (title == null || title.equals("")) {
            title = filePath.substring(idx + 1);
```

```
                title = title.substring(0, title.lastIndexOf("."));
        }

        device.updateTitle(title);
        globalOptions.addRecentEntry(new GlobalOptions.RecentEntry(new
                                                                File(filePath).getAbsolutePath()));
    } catch (Exception e) {
        Common.d(e);
        if (doc != null) {
                doc.destroy();
        }
        finish();
    }
}

public void onPause() {
}

public void initPagePeekerScreen() {
}

public void updatePageSeeker() {
}

public void initZoomScreen() {
    //zoom screen
}

private int zoomInternal = 0;

public void updateZoom() {
}

public void initPageLayoutScreen() {
}

public void initAddBookmarkScreen() {
}

public void initCropScreen() {
}

public void linkCropButtonsAndText(final ImageButton minus, final
                                        ImageButton plus, final TextView text, final int cropIndex) {
}

private void initMainScreen() {
}

protected void onResume() {
}
```

```
protected void onDestroy() {
}

private void saveData() {
}

public boolean onKeyDown(int keyCode, KeyEvent event) {
}

public void changePage(int operation) {
}

public void loadGlobalOptions() {
}

public void saveGlobalOptions() {
}

public View getView() {
}
public WebView getViewW() {
   return  viewW;
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
}

public void initAnimator() {
}

public SubscriptionManager getSubscriptionManager() {
}

public void initOptionDialog() {
   dialog = new Dialog(this);
   dialog.requestWindowFeature(Window.FEATURE_NO_TITLE);
   dialog.setContentView(R.layout.android_dialog);
   animator = ((ViewAnimator)dialog.findViewById(R.id.viewanim));

   getView().setOnTouchListener(new View.OnTouchListener() {
       private int lastX = -1;
       private int lastY = -1;
       long startTime = 0;
       private static final long TIME_DELTA = 600;
       public boolean onTouch(View v, MotionEvent event) {
```

```
if (!selectionMode) {
    if (event.getAction() == MotionEvent.ACTION_DOWN) {
        startTime = SystemClock.uptimeMillis();
        lastX = (int) event.getX();
        lastY = (int) event.getY();
        return true;
    } else {
        boolean doAction = false;
        if (event.getAction() == MotionEvent.ACTION_MOVE ||event.getAction()
            ==MotionEvent.ACTION_UP) {

                if (event.getAction() == MotionEvent.ACTION_UP) {
                    Common.d("UP " + event.getAction());
                    doAction = true;
                } else {
                    if (lastX != -1 && lastY != -1) {
                        boolean isLongClick =(SystemClock.uptimeMillis() - startTime) > TIME_DELTA;
                        doAction = isLongClick;
                    }
                }
                if (doAction) {
                    Common.d("Check event action " + event.getAction());
                    boolean isLongClick = (SystemClock.uptimeMillis() –startTime) > TIME_DELTA;
                    if (lastX != -1 && lastY != -1) {
                        int width = getView().getWidth();
                        int height = getView().getHeight();
                        int i = 3 * lastY / height;
                        int j = 3 * lastX / width;
                        int code = globalOptions.getActionCode(i, j, isLongClick);
                        doAction(code);
                        startTime = 0;
                        lastX = -1;
                        lastY = -1;
                    }
                }
                  return true;
            } else if (event.getAction() == MotionEvent.ACTION_CANCEL || event.getAction() ==
                      MotionEvent.ACTION_OUTSIDE) {

                    startTime = 0;
                    lastX = -1;
                    lastY = -1;
            }
    }
    return true;
} else {
    boolean result = textSelection.onTouch(event);
    if (textSelection.isSuccessful()) {
        selectionMode = false;
        String text = controller.selectText(textSelection.getStartX(), textSelection.getStartY(),
                        textSelection.getWidth(), textSelection.getHeight());
```

```
                if (text != null) {
                    if (selectedTextActions == null) {
                        selectedTextActions = new SelectedTextActions(OrionViewerActivity.this);
                    }
                    selectedTextActions.show(text);
                } else {
                }
            }
            return result;
        }
    }
});

getViewW().setOnTouchListener(new View.OnTouchListener() {
    private int lastX = -1;
    private int lastY = -1;
    long startTime = 0;
    private static final long TIME_DELTA = 600;
    public boolean onTouch(View v, MotionEvent event) {
      if (!selectionMode) {
          if (event.getAction() == MotionEvent.ACTION_DOWN) {
              startTime = SystemClock.uptimeMillis();
              lastX = (int) event.getX();
              lastY = (int) event.getY();
              return true;
          } else {
              boolean doAction = false;
              if (event.getAction() == MotionEvent.ACTION_MOVE || event.getAction() ==
                  MotionEvent.ACTION_UP) {

                  if (event.getAction() == MotionEvent.ACTION_UP) {
                      Common.d("UP " + event.getAction());
                      doAction = true;
                  } else {
                      if (lastX != -1 && lastY != -1) {
                          boolean isLongClick = (SystemClock.uptimeMillis()–startTime) >TIME_DELTA;
                          doAction = isLongClick;
                      }
                  }
                  if (doAction) {
                      Common.d("Check event action " + event.getAction());
                      boolean isLongClick = (SystemClock.uptimeMillis() –  startTime) > TIME_DELTA;
                      if (lastX != -1 && lastY != -1) {
                          int width = getViewW().getWidth();
                          int height = getViewW().getHeight();
                          int i = 3 * lastY / height;
                          int j = 3 * lastX / width;
                          int code = globalOptions.getActionCode(i, j, isLongClick);
                          doAction(code);
                          startTime = 0;
                          lastX = -1;
```

```
                                lastY = -1;
                            }
                        }
                        return true;
                    } else if (event.getAction() == MotionEvent.ACTION_CANCEL ||event.getAction() ==
                            MotionEvent.ACTION_OUTSIDE) {
                        startTime = 0;
                        lastX = -1;
                        lastY = -1;
                    }
                }
                return true;
            } else {
                boolean result = textSelection.onTouch(event);
                if (textSelection.isSuccessful()) {
                    selectionMode = false;
                    String text = controller.selectText(textSelection.getStartX(), textSelection.getStartY(),
                            textSelection.getWidth(), textSelection.getHeight());

                    if (text != null) {
                        if (selectedTextActions == null) {
                            selectedTextActions = new  SelectedTextActions(OrionViewerActivity.this);
                        }
                        selectedTextActions.show(text);
                    } else {
                    }
                }
                return result;
                }
        }
    });
}

private void doAction(int code) {
}


public void doAction(Action action) {
}


protected View findMyViewById(int id) {
}

public void onAnimatorCancel() {
}

@Override
protected void onApplyAction() {
}
```

```
protected void onApplyAction(boolean close) {
}

public void initRotationScreen() {
}

void updateRotation() {
}

@Override
public int getViewerType() {
}

public GlobalOptions getGlobalOptions() {
}

public Controller getController() {
}

public void updateBrightness() {
}


public long insertOrGetBookId() {
}

public boolean insertBookmark(int page, String text) {
}

public long getBookId() {
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
}

public void showOrionDialog(int screenId, Action action, Object parameter) {
}

public void textSelectionMode() {
}

public void changeDayNightMode() {
}

public class MyArrayAdapter extends ArrayAdapter implements SpinnerAdapter {

   public MyArrayAdapter() {
   }
   @Override
   public View getView(int position, View convertView, ViewGroup parent) {
```

```
        }
    }

    public static class Nook2ListAdapter extends ArrayAdapter  {

        private int color;
        public Nook2ListAdapter(Context context, int textViewResourceId, Object[] objects, TextView view) {
        }
        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
        }
    }

}
```