International Journal of Computer and Information Technology (ISSN: 2279 – 0764)
Volume 04 – Issue 03, May 2015

Dissecting the User Interface of a Set of Highly Diffused Android Apps

Victor Matos
Department of Computer and Information Science
Cleveland State University, U.S.A.
Email: v.matos [AT] csuohio.edu

Abstract— **In this study we have dissected the User Interface (UI) of a set of highly diffused Android apps. In particular, we are interested on identifying (a) what type of stylistic components are used to craft the UIs top décor, and (b) what type of navigation support do they provide to expose the general architecture and additional functionality of their associated applications. We have found that all selected apps use some form of the ActionBar or ToolBar control on top of their UIs. The majority of apps in our sample show a preference for vertical and tab-base navigation. Among the observed navigation patterns, we found the following: clickable tabs, drop-down lists, and drawer buttons. Surprisingly, various successful apps continue to use deprecated design and navigation strategies. An appendix is added showing the skeleton of a simple app based on the ActionBar top décor.**

Keywords-Android app development, UI design patterns, highly-difussed Android apps, Actionbar, Toolbar, top décor, vertical and tab-based navigation.

I. Introduction

Software developers target today a world-wide audience made of a heterogeneous population representing all social, cultural, and economical corners of the planet. Regardless of the user's identity and background, the software designer is commanded to deliver software solutions that not only accomplish their functional goals but also provide a simple, effective and positive customer experience. Ideally, a positive user experience is one in which all wanted results are efficiently obtained while the users enjoys operating with their apps. Therefore, it is reasonable for new apps to seek some balance between innovation and imitation of the accepted visual and operational patterns defined by already established applications.

The act of creating new applications could be simplified by re-using design patterns already tested by successful and popular apps. In addition, those model apps present styles and protocols that have already been learned by the user's community. Traditional UI computing leans on windows, icons, menus, and pointing mechanisms to provide a rich user-machine interaction. Those patterns are inadequate and inappropriate for mobile applications on which a variety of new forms of interactions (swiping, tapping, pinching, shaking, etc) and sensorial hardware (location, light-detection, accelerometer, etc) are present to enhance the realm of user-app interfacing [9, 16].

Using UI Design Patterns positively impacts the creative effort by reducing the set of design possibilities to a sub-set of well understood and documented paradigms [13, 14]. Among those efficient and well established Android patterns we have: vertical and horizontal navigation, modular tabs, pagination, hierarchical actions, and so on. Our study aims at identifying what components and strategies are used in well known and successful Android apps.

II. Android Eco-System

Sales of smartphone devices have long surpassed the acquisition of any other consumer electronic artifact. According to a Gartner report [8], by the end of the year 2015, approximately 1.2 billion people will own a smartphone. That roughly indicates that one person out of every eight people in the planet will operate a smartphone and consequently will be fully aware of the notion of mobile apps.

Android OS has quickly become the leading mobile operating system. The estimated world-wide market share of devices powered by Android OS is 77% [7]. This position is followed by Apple's iOS with a 20% of the global market, and finally the remaining 3% is shared by Windows, Blackberry, and others.

The top mobile applications by the end of 2014 consisted of an assortment of categories, among them: social-media, searching, messaging, entertainment, mapping, news, shopping [5]. Facebook ranked as the top smartphone app, reaching 69.7% of the global audience, followed by YouTube (54.5%), Google Play (51.8 %) and Google Search (51.5%). Mobile subscribers are avid software users; the KBPG group [11] reports that the average mobile users check their phones approximately 150 times per day.

Android is an open platform that allows for hardware and software diversity. Android powered devices are made by a number of different manufactures, many of whom add their own variations to the base hardware and software specifications. Consequently there is a wide range of physical Android devices targeting different market segments.

A direct consequence of Android's device fragmentation is the simultaneous existence of various versions of the operating system. Due to the different features and capabilities supported by each OS version, as well as the individual contributions made by the carriers, there is not a single and uniform manifestation of the Android experience. Instead, there are various forms of user-app interactions -which although are not entirely different- are nonetheless not the same for each user. Clearly, the application software designer must carefully consider this range of possibilities and take advantage of the best alternatives for the app in mind.

III. Problem Statement

Our objective is to find a formula for crafting successful Android apps.

In order to discover *how-to* continuously design high quality applications, we turn our attention to the exploration of a set of already well established Android apps. Table1 lists the observed applications. In particular we want to know how their User Interfaces are made, and how do they allow the user to visit the main logical sub-components of each selected app. The applications considered in this study have been chosen based on their high rate of diffusion as well as their index of user-satisfaction. We have chosen to narrow our investigation to a pair of inter-related design subjects (a) ascertain what stylistic elements are placed on the app's top portion of the screen (top décor), and (b) identify what exploration mechanisms are facilitated by those top décor elements.

The motivation behind this project is simple; it is reasonable to believe we may learn from these examples and become better developers. Emphasizing the importance of the UI's architectural top element is consistent with the natural lexicographic process of inspecting documents written in Romance-based languages. On those documents (and therefore visual UIs) the human eye moves in a top-to-bottom, and left-to-right order (we recognize the fact that in some scripts such as Chinese, Hebrew and Arabic, the lexicographic order is different). Consequently the first graphical object to be naturally seen on an arbitrary screen tends to be its top-décor.

IV. Defining Navigation

Android apps are made by integrating a variety of building blocks among them: activities, fragments, background services, broadcast receivers, and content providers [1, 10, 12]. Some of them (services, receivers, and content providers) do not have a visual representation. However; each UI or screen, is typically bound to an individual activity or fragment.

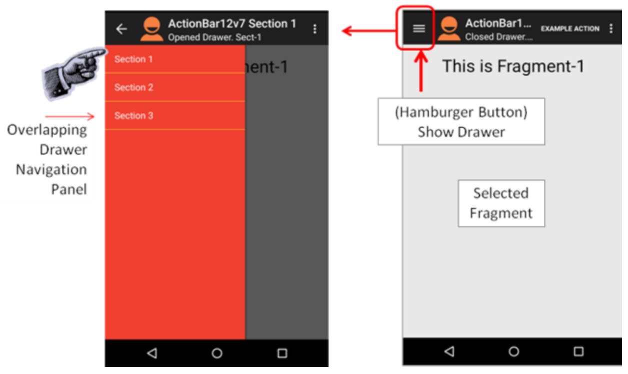


Figure1. An Android App displaying a simple Drawer-Navigation style.

User controlled Navigation is the process of reaching sections of an app as they are presented by their screens. Consequently, navigation and execution of activities and fragments are expressions of the same experience.

Figure1 illustrates a form of 'Vertical Navigation' in which a DrawerView control is activated by pressing the optional 'Hamburger' button (≡) found on the top-left corner of the app's ActionBar. The overlapping curtain presents a (vertical) list of possible options or sections the user may want to explore. Backtracking from a section to the main screen is accomplished by either pressing the ActionBar's 'UP Navigation' button (⟵ ActionBar's top-left corner) or the device's Back-button (↲ or ◀ at the bottom of the device).

Navigation has two phases: exploration and backtracking. In the first case the user moves from one screen to any of its direct logical children. On backtracking the user returns to a previously exposed window which can be either its immediate ancestor, or a distant node on the view hierarchy.

Android's Back key historically reverses the visitation sequence, in a way similar to the *previous-window* transition provided by the *backward* button of a web browser. Tapping the Up-key allows jumping from a distant child view to a non-immediate ancestor (say 'MainActivity'). For this scheme to work, the app's manifest must include for the child activity the clause `android:parentActivityName="MainActivity"` which identifies the target destination to be reached.

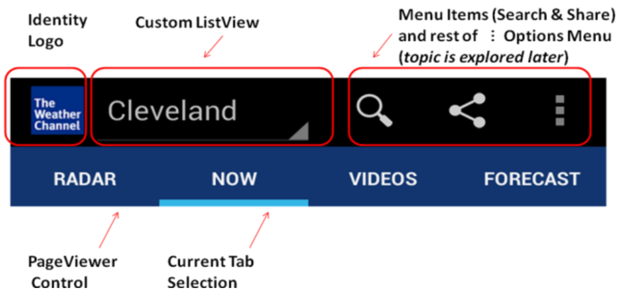


Figure2. ActionBar shows a custom dropdown list-view, menu items, and navigation tabs.Figure 2.

Figure 2 shows the Weather-Channel app's top décor. This app offers two exploration strategies: first there is a vertical custom-list embedded in the ActionBar facilitating the selection of City/Location as well as a Tab-Based navigation bar to expose individual sections of the app (such as Radar, Video, Forecast). Under this model, whenever a tab-button is clicked, a new UI is presented to the user.

In general, Navigation Tabs can be either directly embedded into the ActionBar or -as in this example- held into a horizontal scroll layout underneath the ActionBar. The selected tab is usually marked using a different color or a highlight.

Apps older than SDK5.0 used the ActionBar to anchor their navigation tabs; however this technique is now deprecated. Currently tab-based exploration still remains a recommended navigation pattern, but tabs should not be part of the ActionBar, and instead they should be held into a horizontal-scroll, tab-strip, or custom control [12].

V. DATA SOURCES

We have included in our study a group of twenty one highly *diffused* and *successful* Android applications. We will refer to this group as the HDSAPP collection (Highly Diffused Successful-Apps).

Our summary data shown in Table1 was obtained from information gathered from the *Google Play* electronic store (http://play.google.com, visited on April 1, 2015). Google Play is the site where users go to buy and download music, videos, books, apps, etc.

	App Name	Down loads	Rating	Reviewers	Category
1	Google Search	1B	4.4	1.5 M	Tools
2	Gmail	1B	4.3	1.8 M	Communi-cation
3	Google Maps	1B	4.3	4.7 M	Travel& Local
4	YouTube	1B	4.1	6.1 M	Media& Video
5	Facebook	1B	4.0	27.2 M	Social
6	WhatsApp	1B	4.4	24.9 M	Communi-cations
7	Instagram	500 M	4.5	20.6 M	Social
8	Skype	500 M	4.1	6.5 M	Communi_ cations
9	Pandora	100 M	4.4	1.9 M	Music& Audio
10	Netflix	100 M	4.4	1.7 M	Entertainment
11	Adobe Reader	100 M	4.3	1.9 M	Productivity
12	Twitter	100 M	4.1	5.1 M	Social
13	eBay	100 M	4.3	1.3 M	Shopping
14	Kindle	100 M	4.1	0.4 M	Books& References
15	Weather Channel	50 M	4.3	1.1 M	Weather
16	Wikipedia	10 M	4.4	0.3 M	Books& References
17	Zillow	10 M	4.4	0.2 M	Lifestyle
18	ESPN SportCenter	10 M	4.2	0.4 M	Sports
19	BBC News	10 M	4.2	0.1 M	News& Magazines
20	Amazon (Tablets)	10 M	4.2	0.1 M	Shopping
21	Expedia	10 M	4.0	0.05 M	Travel_& Local
22	Angry Birds	100 M	4.4	4 M	Arcade

Table 1. Set of Highly Diffused and Successful Android Applications

All apps listed in our HDSAPP collection are available for download at no cost to the user. Paid versions are not considered in this study. (As of April 1, 2015)[1]

For each app in the selected group we have included: (1) its name, (2) a lower-bound estimation of the number of total downloads for that app, (3) a rating value representing the users' perception of quality (values are chosen from a scale of 1…5 stars, were 5 stars represents the highest quality), (3) the number of people who have provided a score for the app, and (5) the main category to which the application belongs. Table1 shows the selected HDSAPP apps and summarizes the data items enumerated above.

VI. DIFFUSION RATE

The *Download* attribute shown in Table1, is used to divide the HDSAPP set into five categories (remember that this field estimates the number of times an app has been installed in a device). The first group includes applications that have been downloaded at least 1billion times each. According to *Google Play*, the actual numbers are rather between 1 and 5billion; but for conservative reasons, only the lower bound index is shown in Table1. The next download categories are broken down (using their lower index indicators) as follows: 500, 100, 50, and finally 10 million deployments.

Altogether, the apps in the HDSAPP set have been installed over 6 billion times. Considering there are close to eight billion people in the planet, the diffusion rate of the selected apps is significantly high (in average 1 out of each 8 people in the planet is a Gmail user, a Facebook user, and so on). Consequently, we may argue that very large pools of people who are disperse all over the planet, have already learned how to use those apps. In addition, the stylistic and operational protocols shown by HDSAPP apps define a familiar pattern that is expected and welcomed on new apps.

VII. MEANING OF 'SUCCESSFUL APP'

The issue of *quality* regarding the HDSAPP sample has been addressed by deferring to the Google Play's *Five-Star ranking system*. In this schema, users submit a score for an app, analogue to the 'Five Stars' approach used for grading hotels and restaurants. The lowest possible rate given to an app is one 'star', while apps producing the highest degree of user satisfaction are granted five stars. In our study all selected apps have been rated as low as 4.0 and as high as 4.5 (with an average of 4.25 points). It is important to acknowledge the large number of end-users submitting their evaluations.

The column 'Reviewers' in Table 1 shows the count of votes received by each app. Please observe that some apps such as Facebook, Instagram, Whatsapp, and Skype have been rated by tens of millions of users (between 2% and 6% of total users) and others such as Gmail, Google Maps, Amazon Shopping have been rated by a much lower percentage of users (between 0.01% and 2%).

[1] All apps in Table1 are copyrighted © products belonging to their authors and corresponding companies.

VIII. Background - Anatomy of a Typical Android ActionBar/ToolBar

In general, the UI of any Android app could be visually divided into three parts: *header*, *body*, and *footer*. Normally the header and footer are optional components, whereas the body is ordinarily the main visual manifestation of the app. All the apps in this study include a header component which is implemented either as a primitive ActionBar or ToolBar widget or a custom designed décor element imitating the structure of a ToolBar [3, 10, 12].

The Android ActionBar control was introduced in SDK 3.0 and plays a special role on the crafting of contemporary apps. It is depicted as a graphical tool-bar at the top of each screen and it is usually persistent across the app. Its dedicated placement and repeated style provides a sense of consistent identity and facilitates -in a predictable way- access to important actions and navigation. As illustrated by Figure3, an ActionBar may contain the following pieces: Navigation

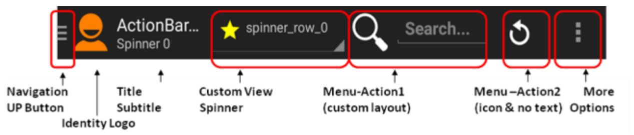


Figure 3. Example on an ActionBar displayed on a Tablet

Button (Hamburger or UP Arrow icon), an identity logo, title and subtitle, an optional custom view, action tiles (clickable buttons showing icons, text, or custom layouts), and finally an overflow Options-Menu (⁝) button placed on the rightmost corner of the bar. Legacy apps may also render ActionBar décors holding embedded Navigation-Tab (bear in mind this practice was deprecated after SDK4.4).

The ActionBar can be used to support various navigation patterns, for instance (1) its “Hamburger” button (≡ top-left corner, see Figure1) could be tapped to display an overlapping DrawerView. The DrawerView is a sliding curtain (see Figure1) holding a list of sections or entry-points deemed important to the app. This navigation pattern is called **Vertical Navigation**, (2) scrollable horizontal tabs (either embedded or neighboring) could be tapped to expose a selected view (perhaps a page from a ViewPager control, an inflated resident fragment, or a window shown by another activity invoked via Intents. This style is called **Tab Navigation**. Finally, the *UP* (*Arrow*) button (⟵ top-left corner, see Figure 1) could be used to jump back to the previously visited screen or any higher place in the app’s view hierarchy.

The clickable action buttons of an ActionBar are defined in an XML menu file. Figure4 shows the menu specification used for the ActionBar depicted in Figure3. This resource file is later programmatically inflated and presented as part of the app’s global Options-Menu. In addition, menu items may also appear on the drop-down list known as the overflow OptionsMenu. An OptionsMenu generally persists for the lifetime of the app, however it could be dynamically enabled, disable, and changed.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android=
  "http://schemas.android.com/apk/res/android" >

  <item
    android:id="@+id/search"
    android:actionLayout="@layout/custom_search_on_actionbar"
    android:showAsAction="ifRoom"
   android:title="Search"/>
  <item
    android:id="@+id/reset"
    android:icon="@drawable/ic_action_reset"
    android:showAsAction="ifRoom"
    android:title="Reset"/>
  <item
    android:id="@+id/about"
    android:icon="@drawable/ic_about"
    android:showAsAction="never"
    android:title="About"/>
</menu>
```

Figure4. Resource menu file used to populate the ActionBar shown on Figure2.

Two methods do most of the work related to interacting with ActionBar’s buttons:

(1) **onCreateOptionsMenu**(…) Inflates the XML menu-file executing the statement **actionbar.inflateMenu (R.menu.menu)**

(2) **onMenuItemClick(MenuItem item)** responsible for capturing the click-event on a tile (menu item) and channeling the proper response to the user’s request.

Each menu action item shown in Figure3 has an ID (such as **android:id="@+id/search"**). This ID field is used by the **onMenuItemClick** listener to determine what button has been tapped. In addition items may include an icon or a layout, a title, and an entry indicating how to render the item on the menu bar. For instance, the clause **showAsAction="ifRoom"** displays the action tile only if the device has enough room for it, whereas the clause **showAsAction="never"** places the item directly into the Overflow-Menu. A long ActionBar may split and be shown as top and bottom décors; this division often occurs on devices with a small screen. If defined, an optional custom view is placed between the title and the first menu item.

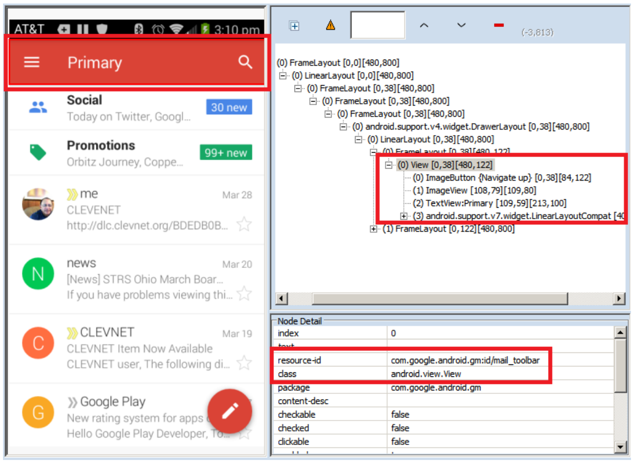
The ActionBar shown in Figure3 includes a custom Spinner or DropDown control. Each row held in the spinner consists of an icon (a *star* in our example) and text (“*spinner-row-n*”). The pattern **Drop-down list navigation** occurs when the user picks a row from the list and the app adjusts to the requested choice. For instance, the drop-down list of Figure2 (Weather Channel app), allows the user to select a city. The app responds by presenting a weather forecast for the chosen locality. Adding a custom layout to an action bar is programmatically accomplished by the following logic **actionBar.setCustomView(R.layout.*custom_view_on_actionbar*).** This layout specification indicates the structure of the custom view that is to be embedded in the action bar. The custom Spinner control was used in early Android apps as an alternative navigation pattern. However; this practice is now considered obsolete and has been deprecated.

IX. Research Methodology

We used the *SDK UI-Automation Tool* [2] to explore the main screen of each app listed in Table1. The UI-Automation Tool (or Hierarchy Tool) provides a visual representation of the app's view hierarchy together with performance information for each node in the layout. For example, Figure5 shows the results obtained after sampling a Gmail session running on a small handset device. Each app listed in Table1 was inspected on two physical devices: a small handset (running SDK 4.4) and a large tablet (under SDK 5.1). In most cases the layouts for handsets and tablets are the same; however we have made notes of any exceptions found.

The image on the left panel of Figure5 is an attempt to replicate a pixel-by-pixel rendition of the window shown by the app on the hardware device. On the right hand side of the screen there are two panels. The top view shows the **Tree-Hierarchy** representation of the UI, the lower panel provides specific **Properties** of a selected UI's component. To some extend the Hierarchy tool facilitates a reverse engineering operation in which one can take an existing app and dissect most elements of its UI. Figure5 captures the moment in which we clicked on the Gmail app's top décor. The Tree-Hierarchy panel synchronizes with the visual selection by highlighting the corresponding tree node. In this case, the top décor is identified as a 'view'. The 'view' label is followed by the location (in pixels) of its corresponding *top-left* and *bottom-right* corners. The descendants of the chosen 'view' node include the following: Hamburger (Navigation Button), an ImageView surrounding a TextView displaying the caption "Primary" and finally a compact linear layout on which a magnifier icon (search action) is allocated.

We detected a limitation on the way the SDK Hierarchy tool describes nodes of the UI under examination. For instance the root node of a sub-tree could be just labeled "view [*pixel top-left cornet*] [*pixel bottom-right corner*]" without providing specific declaration of its true nature. There is no way to determine if the selected *view* is exactly a *Toolbar* or an *ActionBar*. In our case this is not an issue because both components are treated as equivalent elements.


AT&T
3:10 pm
Primary
Social
30 new
Today on Twitter, Googl...
Promotions
99+ new
Orbitz Journey, Coppe...
me
Mar 28
CLEVENET
http://dlc.clevnet.org/BDEDB0B...
news
Mar 20
[News] STRS Ohio March Boar...
If you have problems viewing thi...
CLEVNET
Mar 19
CLEVNET Item Now Available
CLEVNET user, The following di...
Google Play
New rating system for apps
Hello Google Play Developer, To...
(-3,813)
Node Detail

```
(0) FrameLayout [0,0][480,800]
 (0) LinearLayout [0,0][480,800]
  (0) FrameLayout [0,38][480,800]
   (0) FrameLayout [0,38][480,800]
    (0) FrameLayout [0,38][480,800]
     (0) android.support.v4.widget.DrawerLayout [0,38][480,800]
      (0) LinearLayout [0,38][480,800]
       (0) FrameLayout [0,38][480,122]
        (0) View [0,38][480,122]
         (0) ImageButton {Navigate up} [0,38][84,122]
         (1) ImageView [108,79][109,80]
         (2) TextView:Primary [109,59][213,100]
         (3) android.support.v7.widget.LinearLayoutCompat [4
       (1) FrameLayout [0,122][480,800]
```

index	0
text	
resource-id	com.google.android.gm:id/mail_toolbar
class	android.view.View
package	com.google.android.gm
content-desc	
checkable	false
checked	false
clickable	false

Figure5. An example of Gmail's View Hierarchy reported by the UI-Automation Tool

X. Summary of Observations

Table2 summarizes our observations. For each entry listed in Table1 we have recorded two dimensions: (1) Nature of the design element used on the UI's header, and (2) Navigation styles provided by the top décor. Possible entries for the first category include: (a) Presence of an ActionBar/Toolbar, (b) Presence of a custom designed view, and (c) None (top décor not used). Entries for the second dimension describe navigation modes including: DrawerView, Tabs, Spinner, custom methods, and none. If applicable, there are separate entries for handset and tablet implementations. See for instance the case of Pandora, Whatsapp, and Twitter which use different strategies based on the screen size of the user's device.

The popular arcade app 'Angry Birds' is an exception to the design patterns evaluated in this paper. This application does not have a distinguishable top décor element; instead the app dedicates its entire screen to render the multiple images used in the game. In this case we have entered "none" under top décor and navigation support (the app is excluded from the statistics given below).

		Top Décor		**Navigation Strategy**			
	(Free) App Name	**Actionbar Toolbar**	**Custom Design**	**Drawer View**	**Tabs**	**Drop Down List**	**Custom Strategy**
1	Google Search		Y1	Yes			
2	Gmail	Yes		Yes			
3	Google Maps		Y3	Yes			
4	YouTube	Yes		Yes	Yes		
5	Facebook		Y5	Y5	Y5		
6	WhatsApp (Phone)	Y6					
7	Instagram		Y7		Y7		
8	Pandora (Phone)	Yes			Y8P		
8	Pandora (Tablet)	Yes					Y8T
9	Netflix	Yes		Yes			
10	Adobe Reader	Yes		Yes			
11	Skype		Y11				
12	Twitter (Tablet)	Yes			Yes		
12	Twitter (Phone)		Y12		Y12		Y112
13	eBay	Yes					Y13
14	Weather Channel	Yes			Y14	Y14	Y14
15	Kindle	Yes		Yes			
16	Wikipedia	Yes		Yes			
17	Zillow	Yes					Y17
18	ESPN SportCenter	Yes		Yes			
19	BBC News	Yes					
20	Amazon (Tablet)		Y19				Y19
21	Expedia	Yes					
22	Angry Birds	None					

TABLE2. Observed Top-Décor & Associated Navigation Features

Comments: Top Décor, Navigation

- Y1 Custom toolbar showing a 'QuickSearchPlate'. No additional navigation features.
- Y3 Custom toolbar showing a 'QuickSearchPlate'. No additional navigation features.
- Y5 Toolbar-Like (Search, Contacts). Navigation Tabs plus DrawerView-like "Contacts"
- Y6 Our speculation. View hierarchy not observed [app does not allow view inspection]
- Y7 Toolbar-Like(Identity, Messages). Navigation: Tab-based using older TabHost, TabWidget style
- Y8P Toolbar, Navigation: Tab-based using TabHost, TabWidget
- Y8T Toolbar, Navigation: Custom List (Add Station, Shuffle, RadioStations)
- Y11 Toolbar-Like (Identity, Call, IMS, Contacts, Avatar, MoreOptions)
- Y12 Custom toolbar-like (Search, Write). TabHost & TabWidget
- Y13 Toolbar. Navigation is Tab-Based using a custom layout placed away from the toolbar
- Y14 Toolbar. Navigation is Tab-Based, tabs are on a ScrollView linked to a ViewPager, additional custom view (Spinner control)
- Y17 Toolbar, Navigation provided by a floating DialogBox exposing the app's OptionsMenu
- Y19 Custom toolbar, DrawerView based navigation.

TABLE2(*continuation*). Observed Top-Décor & Associated Navigation Features

There are twenty-one different apps listed in Table1. All of those highly diffused apps make use of the ActionBar/Toolbar top décor style. Fourteen apps (or 66% of the sample) expose a view based on a native ActionBar (or analogous Toolbar) widget as provided by the Android SDK, while seven apps (or 33% of the sample) utilize a custom made top décor. One app (Twitter) uses the native ActionBar element for its handset design and a custom top décor for tablets.

A common denominator of the observed ActionBars is the consistent presence on the right corner of an Overflow Menu button (⋮). This button repeatedly facilitates among others choices, the app's 'Settings' option. The applications Google Search and Google Maps use a very simple TextView-like top décor called 'QuickSearchPlate'. The QuickSearchPlate widget is specialized on calling a search-provider using the supplied text. Nothing else is used in the header of those two apps. Other common design strategy observed in our sample is the custom crafting of a toolbar-like control that looks and works as a native ActionBar widget but is reported by the UI-Hierarchy Tool as a custom layout.

Applications such as Facebook, Instagram, Skype, Twitter, and Amazon (Tablet version) rely on look-alike ActionBars. The top décor of these apps hosts familiar toolbar components such as identity logo, icons, and Overflow menu. We speculate this was done to give the designer more control on the behavior and longevity of these organizations. Remember that the ActionBar standard have been changed often and currently its tab and drop-down list navigation modes are deprecated.

We have observed three dominant navigation styles connected to the top décor of apps in the HDSAPP collection, (1) **Tab-mode** (see Figure2), (2) **DrawerView-mode** (see Figure1), and (3) **Custom-designed** strategies. There is almost an even split between the use of the DrawerView and Tab styles (at 48% each). Only one app (Zillow) uses an exploration pattern based on a legacy *floating option menu* [10, 12]. Styling variations between apps using the DrawerView pattern are almost inexistent. The DrawerView is decorated with some artwork on top and proceeds to list options on a row-wise mode. Tapping an option closes the curtain and exposes a new window as requested. Users of those apps will encounter a familiar structural and behavioral experience in the exploration of the DrawerViews.

Tabs are implemented using a wide range of design options. The most common solution found for holding navigation tabs is based on the use of a TabStrip or a Horizontal-Scroll control. An interesting finding is that the apps Pandora and Twitter base their Tab-Navigation on the legacy TabHost-TabWidget controls. The Weather Channel app uses a custom-view top décor component holding among others a drop-down list from which the user may navigate from one location to another.

XI. Conclusion

Mobile UI patterns are transient and tend to change as the hardware platforms become more capable and the software more mature and stable. However, at the time of writing, our study suggests that the apps in the HDSAPP have a remarkable common structural and functional architecture. This design sameness is surprising considering they operate on very different problem domain areas including among others: entertainment, news, geo-mapping, messaging, shopping, etc.

The two most relevant characteristics of the observed HDSAPP sample are: (1) all the apps place as top décor an ActionBar-Toolbar (or look-alike) control. (2) navigation tied to the top décor is predominantly based on two stylistic options: DrawerView (Vertical Navigation) and Tab controls.

Best design practice suggests that Tab buttons should not be embedded into the ActionBar, instead, if used, they should be held in either a TabStrip or Horizontal-Scroll container adjacent to the ActionBar. The HDSAPP sample is evenly divided between tabbed and list based exploration. However, a previous research [4] suggests that when list-based exploration was used, results were found faster than under tabbed-navigation, and also users reported a perceived better ease-of-use advantage.

We may conclude that imitating the UI patterns used by the HDSAPP set is a recommended first step in creating a successful Android app.

References

[1] Android Developer. Application Fundamentals. Accessed on March30, 2015 from http://developer.android.com/guide/components/fundamentals.html

[2] Android Developer. Hierarchy Viewer – Accessed on March 22, 2015 from http://developer.android.com/tools/help/hierarchy-viewer.html

[3] Android Developer. Action Bar – Accessed on April 10, 2015 from http://developer.android.com/guide/topics/ui/actionbar.html

[4] Balagtas-Fernandez, Florence, Forrai, Jenny, Hussmann, Heinrich. Evaluation of User Interface Design and Input Methods for Applications on Mobile Touch Screen Devices. Human-Computer Interaction – INTERACT 2009 Lecture Notes in Computer Science Volume 5726, 2009, pp 243-246

[5] ComScore Inc. Top 15 Smartphone Apps, January 2015. http://www.comscore.com/Insights/Market-Rankings/comScore-Reports-January-2015-US-Smartphone-Subscriber-Market-Share (accessed Marh 17, 2015)

[6] Digital TV Research Inc. Number of connected TV sets worldwide from 2010 to 2018 (in millions). Retrieved March 17, 2015, from http://www.statista.com/statistics/247160/ forecast-of-the-number-of-connected-tv-sets-worldwide/

[7] IDC Inc. Global smartphone operating system market share held by Android from 1st quarter 2011 to 4th quarter 2014. Retrieved March 17, 2015, from http://www.statista.com/statistics/236027/global-smartphone-os-market-share-of-android/

[8] Gartner Inc. Number of smartphones sold to end users worldwide from 2007 to 2014 (in million units). Retrieved March 17, 2015, from http://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/

[9] Gavalas, Damianos & Economou, D. Development Platforms for Mobile Applications: Status and Trends, by. IEEE Software, (Volume: 28, Issue: 1) Jan 2011.

[10] Matos, V. Lecture Notes: Mobile Application Development Using Android OS. Retrieved March 22, 2015 from http://grail.cba.csuohio.edu/~matos/notes/cis-493/2014-fall/Android-Syllabus-2014-fall.pdf

[11] Meeker, Mary & Liang, Wu. KPBG - Internet Trends D11 Conference. Retrieved April 1, 2015 from http://kpcb.cc/d3144e9

[12] Murphy, Mark. The Busy Coder's Guide to Android Development Version 6.5. Published by CommonsWare, LLC, 2015. ISBN: 978-0-9816780-0-9

[13] Nudelman, Greg. Android Design Patterns. Edited by Wiley & Sons Inc. 2013, ISBN 978-1-118-39415-1

[14] Neil, Theresa. Design Pattern Gallery: UI Patterns for Smartphone Apps. O'Reilley Publications. 2nd Edition, 2014, ISBN-13: 978-1449363635

[15] Neil, Theresa & Malley, Rich. Rethinking Mobile Tutorials: Which Patterns Really Work? Accessed April 12, 2015 from http://www.smashingmagazine.com/2014/04/22/rethinking-mobile-tutorials-which-patterns-really-work/

[16] Subramanya, by S.R. ; Yi, B.K. User interfaces for mobile content. IEEE Computer (Volume:39 , Issue: 4) April 2006.

APPENDIX. CREATING A SIMPLE ACTIONBAR APP

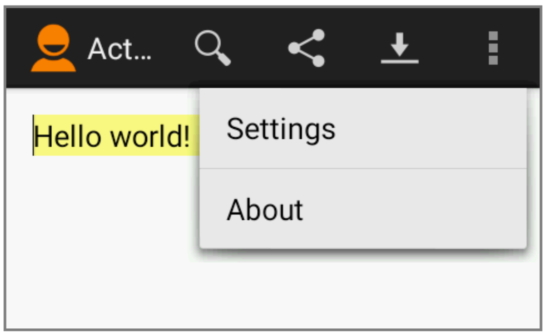


Figure 6. Layout of a simple ActionBar-based app.

A. *Menu XML-Definition*

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="csu.matos.MainActivity" >

    <item
        android:id="@+id/action_search"
        android:icon="@drawable/ic_action_search"
        android:orderInCategory="120"
        android:showAsAction="always|withText"
        android:title="Search"/>
    <item
        android:id="@+id/action_share"
        android:icon="@drawable/ic_action_share"
        android:orderInCategory="140"
        android:showAsAction="always"
        android:title="Share"/>
    <item
        android:id="@+id/action_download"
        android:icon="@drawable/ic_action_download"
        android:orderInCategory="160"
        android:showAsAction="always"
        android:title="Download"/>

<item
        android:id="@+id/action_settings"
        android:orderInCategory="180"
        android:showAsAction="never"
        android:title="Settings"/>
    <item
        android:id="@+id/action_about"
        android:orderInCategory="200"
        android:showAsAction="never"
        android:title="About"/>

</menu>
```

B. *Main Activity Java-Code*

```
public class MainActivity extends Activity {
    EditText txtMsg;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      txtMsg = (EditText)findViewById(R.id.txtMsg);
      // setup ActionBar
      actionBar = getActionBar();
      actionBar.setTitle("ActionBarDemo2");
      actionBar.setSubtitle("Version2.0");
      actionBar.setLogo(R.drawable.ic_action_logo);

      // choose a custom background(color, gradient)
      actionBar.setBackgroundDrawable(getResources()
      .getDrawable(R.drawable.mybackground0));
      actionBar.setDisplayShowCustomEnabled(true);
      // allow custom views to be shown
      actionBar.setDisplayHomeAsUpEnabled(true);
      // show 'UP' affordance < button
      actionBar.setDisplayShowHomeEnabled(true);
      // allow app icon – logo to be shown
      actionBar.setHomeButtonEnabled(true);
      // needed for API14 or greater
   }

   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      // Inflate the menu; add items to the action bar
      getMenuInflater().inflate(R.menu.main, menu);
      return true;
   }

   @Override
```

```
    public boolean onOptionsItemSelected(MenuItem item) {
        // user clicked a menu-item from ActionBar
        int id = item.getItemId();

        if (id == R.id.action_search) {
            txtMsg.setText("Search...");
            // perform SEARCH operations...
            return true;
        }

        else if (id == R.id.action_share) {
            txtMsg.setText("Share...");
            // perform SHARE operations...
            return true;
        }
        else if (id == R.id.action_download) {
            txtMsg.setText("Download...");
            // perform DOWNLOAD operations...
            return true;
        }
        else if (id == R.id.action_about) {
            txtMsg.setText("About...");
            // perform ABOUT operations...
            return true;
        }
        else if (id == R.id.action_settings) {
            txtMsg.setText("Settings...");
            // perform SETTING operations...
            return true;
        }

        return false;
    }
}
```

C. Comments

1. Calls to the method **findViewById**() provides plumbing operations. Here we establish access to the GUI's EditText field displaying the "Hello World" line.
2. The method **onCreateOptionsMenu()** is called to prepare the app's OptionsMenu. The xml file res/memu/main.xml containing the ActionBar item specifications is inflated using a MenuInflater object. Some action items will be shown on the ActionBar as an Icon/Text tile and the rest moved to the overflow menu window.
3. When the user clicks on a reactive portion of the ActionBar, its item's ID is supplied to the **onOptionsItemSelected**() method. There you branch to the appropriated service routine where the action is served. Finally return true to signal the event has been fully consumed.
4. A call to the **getActionBar()** method returns a handle to the app's ActionBar. Using this reference you now have programmatic control to any of its components.
5. In this example a new title & subtitle is assigned to the ActionBar. Notice that when you work with a complex app exposing many screens, changing title and/or subtitle becomes a simple, yet powerful way of guiding the user through the app.
6. The app's identifying logo could be changed with a call to .**setLogo**(drawable). Similarly, the ActionBar's background image could be changed to any drawable you chose that is stored as a .PNG image into the **res/drawable** folder.