Graphical User Interfaces

Victor Matos
Cleveland State University

The Model-View-Control (MVC) Pattern

The Model-View-Control (MVC) is an important software design pattern whose main goal is to separate the (1) user interface, (2) business, and (3) input logic.

How is this seen by the Android developer?

- **Model.** Consists of the Java code and objects used to manage the behavior and data of the application.
- **View.** Set of screens the user sees and interacts with.
- **Controller.** Implemented through the Android OS, responsible for interpretation of the user and system inputs. Input may come from a variety of sources such as the trackball, keyboard, touchscreen, GPS chip, background services, etc, and tells the Model and/or the View (usually through callbacks and registered listeners) to change as appropriate.

The Model-View-Control (MVC) Pattern

Getting ready to create MVC conforming solutions
The Android developer should be aware of...

- **Inputs** could be sent to the application from various physical/logical components. Reacting to those signals is typically handled by **callback methods**. Usually there are many of them, you want to learn how to choose the appropriate one.

- Moving to states in the **lifecycle** is tied to logic in the model. For instance, if forced to **Pause** you may want to save uncommitted data.

- A **notification** mechanism is used to inform the user of important events happening outside the application (such as arrival of a text message or email, phone calls, etc) and consequently choose how to proceed.

- **Views** are unlimited in terms of aesthetic and functionality. However physical constraints such as size, and hardware acceleration (or lack of) may affect how graphical components are managed.

Android & the MVC Pattern

The View - User Interfaces (Uis)
Android **graphical interfaces** are usually implemented as XML files (although they could also be dynamically created from code).

An Android UI is conceptually similar to a common HTML page

- **In a manner similar to a web page interaction**, when the Android user touches the screen, the controller interprets the input and determines what specific portion of the screen and gestures were involved. Based on this information it tells the model about the interaction in such a way that the appropriate “callback listener” or lifecycle state could be called into action.

- **Unlike a web application** (which refreshes its pages after explicit requests from the user) an asynchronous Android background service could quietly notify the controller about some change of state (such as reaching a given coordinate on a map) and in turn a change of the view’s state could be triggered; all of these without user intervention.
UI Design Patterns

For a detailed discussion on Android UI Design Patterns see video: http://www.youtube.com/watch?v=M1ZBjiCRfz0&feature=player_embedded

The View Class

- The View class is the Android’s most basic component from which users interfaces can be created. This element is similar to the Swing JComponent class for Java apps.

- A View occupies a rectangular area on the screen and is responsible for drawing and event handling.

- Widgets are subclasses of View. They are used to create interactive UI components such as buttons, checkboxes, labels, text fields, etc.

- Layouts are invisible containers used for holding other Views and nested layouts.
Graphical UI ↔ XML Layout

Actual UI displayed by the app

Text version: activity_main.xml file

Using Views

• An Android's XML view file consists of a layout holding a hierarchical arrangement of its contained elements.
• The inner elements could be simple widgets or nested layouts holding some complex viewgroups.
• An Activity uses the setContentView(R.layout.xmlfilename) method to render a view on the device’s screen.

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    
    Widgets and other nested layouts
</LinearLayout>
Using Views

Dealing with widgets & layouts typically involves the following operations

1. **Set properties:** For example setting the background color, text, font and size of a `TextView`.

2. **Set up listeners:** For example, an image could be programmed to respond to various events such as: click, long-tap, mouse-over, etc.

3. **Set focus:** To set focus on a specific view, you call the method `requestFocus()` or use XML tag `<requestFocus/>`

4. **Set visibility:** You can hide or show views using `setVisibility(…)`.

A brief sample of UI components

**Layouts**

- **LinearLayout**: Places its inner views either in horizontal or vertical disposition.
- **RelativeLayout**: A ViewGroup that allows you to position elements relative to each other.
- **TableLayout**: A ViewGroup that places elements using a row & column disposition.

A brief sample of UI components

**Widgets**

<table>
<thead>
<tr>
<th>TimePicker</th>
<th>AnalogClock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form Controls includes a variety of typical form widgets, like: image buttons, text fields, checkboxes and radio buttons.</td>
<td></td>
</tr>
</tbody>
</table>

- **DatePicker**
  - A DatePicker is a widget that allows the user to select a month, day and year.


**WebView**

- **AutoCompleteTextView**
  - It is a version of the EditText widget that will provide auto-complete suggestions as the user types. The suggestions are extracted from a collection of strings.

- **ListView**
  - A ListView is a View that shows items in a vertically scrolling list. The items are acquired from aListAdapter.

Android considers XML-based layouts to be *resources*, consequently layout files are stored in the `res/layout` directory inside your Android project.

A reasonable UI representation of an XML file can be seen in Eclipse by clicking the [Graphical Layout] tab of the `res/layout/main.xml` resource.
Tools for Android UI/XML Design

ASIDE - CREATING ANDROID UI & APPS

You could create Layout XML files using UI tools such as:

- **Eclipse ADT UI Designer.** It is getting better, integrates code & UI design in the same platform. Not yet at the same high level of Apple’s iOS and Microsoft Visual Studio UI Tools.

- **DroidDraw** Very simple, incomplete, not integrated to the Eclipse IDE, aging!  [http://www.droiddraw.org/](http://www.droiddraw.org/)

- **App Inventor** (very promising & ambitious, ‘hides’ coding ...)
  [http://appinventor.googlelabs.com/about/index.html](http://appinventor.googlelabs.com/about/index.html)

*More on this issue later…*

---

How to create complex UIs?

- **The LinearLayout** is arguably the most common type of container.
- It offers a "box" model where inner elements could be placed side-by-side or up-and-down.
- In general, complex UI designs could be made by combining simpler *nested* boxes and stacking them in either a *horizontal* or *vertical* orientation.
Common Layouts

We will discuss the following common and useful layouts:
Frame, Linear, Relative, Table, and Absolute.

1. FrameLayout
- FrameLayout is the simplest type of layout.
- Useful as outermost container holding a window.
- Allows you to define how much of the screen (high, width) is to be used.
- All its children elements are aligned to the top left corner of the screen.

The LinearLayout

1. Linear Layout

The widgets or inner containers held in a LinearLayout are collocated one next to the other in either a column or a row.

Configuring a LinearLayout usually requires you to set the following attributes:
- orientation,
- fill model,
- weight,
- gravity,
- padding,
- margin
1. **Linear Layout Orientation**

The `android:orientation` property can be set to the values: `horizontal` for rows or `vertical` for columns.

```
android:orientation="horizontal"
```

```
android:orientation="vertical"
```

The orientation can be modified at runtime by invoking `setOrientation()`.

---

**The LinearLayout**

1.1 **Linear Layout: Orientation**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:id="@+id/myLinearLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ff0033cc"
    android:padding="4dip"

    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/labelUserName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#ffff0066"
        android:text="User Name"
        android:textSize="16sp"
        android:textStyle="bold"
        android:textColor="#ff000000"/>

    <EditText
        android:id="@+id/ediName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18sp"/>

    <Button
        android:id="@+id/btnGo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Go"
        android:textStyle="bold"/>

</LinearLayout>
```
The LinearLayout – Fill Model

1.2 Linear Layout: Fill Model

• Widgets have a "natural" size based on their included text.
• You may want to specify how tall & wide a widget should be even if no text is involved (as is the case of the empty text box shown below).

```
android:layout_width
android:layout_height
```

Values used in defining height and width can be:

1. A specific dimension such as 125dp (device independent pixels, a.k.a. dp)
2. wrap_content indicates the widget should just fill up its natural space (if it is too big other options such as word-wrap could be used to make it fit).
3. match_parent (previously called ‘fill_parent’) indicates the widget wants to be as big as the enclosing parent.
The LinearLayout – Fill Model

1.2 Linear Layout: Fill Model

The LinearLayout – Weight

1.2 Linear Layout: Weight

Indicates how much of the extra space in the LinearLayout will be allocated to the view. Use 0 if the view should not be stretched. The bigger the weight the larger the extra space given to that widget.

Example

The XML specification for the window is very similar to the previous example.

The TextView and Button controls have the additional property android:layout_weight="1"

whereas the EditText control has android:layout_weight="2"

Default value is 0
1.3 Layout_Gravity

- It is used to indicate how a control will align on the screen.
- By default, widgets are left- and top-aligned.
- You may use the XML property `android:layout_gravity="..."` to set other possible arrangements: left, center, right, top, bottom, etc.

1.3 CAUTION: gravity vs. layout_gravity

The difference between:

- **android:gravity** indicates how to place an object within a container. In the example the text is centered:
  
  ```xml
  android:gravity="center"
  ```

- **android:layout_gravity** positions the view with respect to its container:
  
  ```xml
  android:layout_gravity="center"
  ```
1.4 Linear Layout: Padding

- The padding specifies how much extra space there is between the boundaries of the widget's "cell" and the actual widget contents.

- Either use
  - `android:padding` property
  - or call method `setPadding()` at runtime.

The LinearLayout – Padding

1.3 Linear Layout: Padding and Margin

![Diagram showing boundaries and margins in a LinearLayout layout with padding and margin properties.]

- Widget’s Original Frame
- Padding Top
- Margin Top
- Padding Bottom
- Margin Bottom
- Boundaries touching other widgets
The LinearLayout – Padding

1.3 Linear Layout: Internal Margins Using Padding

Example:
The EditText box has been changed to display 30dip of padding all around

The LinearLayout – Margin

1.4 Linear Layout: (External) Margin

• Widgets –by default– are tightly packed next to each other.
• To increase space between them use the android:layout_margin attribute
The Relative Layout

2. Relative Layout

The placement of widgets in a RelativeLayout is based on their positional relationship to other widgets in the container and the parent container.

Example:
A is by the parent’s top
C is below A, to its right
B is below A, to the left of C

The Relative Layout

2. Relative Layout - Referring to the container

Below there is a list of some positioning XML boolean properties (true/false) mapping a widget according to its location respect to the parent’s place.

- android:layout_alignParentTop the widget’s top should align with the top of the container
- android:layout_alignParentBottom the widget’s bottom should align with the bottom of the container
- android:layout_alignParentLeft the widget’s left side should align with the left side of the container
- android:layout_alignParentRight the widget’s right side should align with the right side of the container
- android:layout_centerInParent the widget should be positioned both horizontally and vertically at the center of the container
- android:layout_centerHorizontal the widget should be positioned horizontally at the center of the container
- android:layout_centerVertical the widget should be positioned vertically at the center of the container
### 2. Relative Layout – Referring to other widgets

The following properties manage positioning of a widget with respect to other widgets:

- **android:layout_above** indicates that the widget should be placed above the widget referenced in the property.
- **android:layout_below** indicates that the widget should be placed below the widget referenced in the property.
- **android:layout_toLeftOf** indicates that the widget should be placed to the left of the widget referenced in the property.
- **android:layout_toRightOf** indicates that the widget should be placed to the right of the widget referenced in the property.

### 2. Relative Layout – Referring to other widgets – cont.

- **android:layout_alignTop** indicates that the widget's top should be aligned with the top of the widget referenced in the property.
- **android:layout_alignBottom** indicates that the widget's bottom should be aligned with the bottom of the widget referenced in the property.
- **android:layout_alignLeft** indicates that the widget's left should be aligned with the left of the widget referenced in the property.
- **android:layout_alignRight** indicates that the widget's right should be aligned with the right of the widget referenced in the property.
- **android:layout_alignBaseline** indicates that the baselines of the two widgets should be aligned.
2. Relative Layout – Referring to other widgets

When using relative positioning you need to:

1. Put identifiers (android:id attributes) on all elements that you will be referring to.

2. XML elements are named using: @id/... For instance an EditText box could be called: android:id="@+id/txtUserName"

3. You must refer only to widgets that have been defined. For instance a new control to be positioned below the previous EditText box could refer to it using: android:layout_below="@+id/txtUserName"

The Relative Layout

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/myRelativeLayout"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:background="#ff000099"
>
  <TextView
    android:id="@+id/lblUserName"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:background="#ffff0066"
    android:text="User Name"
    android:textColor="#ff000000"
    android:textStyle="bold"
  />
  <EditText
    android:id="@+id/txtUserName"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/lblUserName"
    android:padding="20dp"
    android:background="#ff000000"
    android:textStyle="bold"
  />
  <Button
    android:id="@+id/btnGo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/txtUserName"
    android:layout_toRightOf="@+id/lblUserName"
    android:layout_toLeftOf="@+id/btnCancel"
    android:layout_alignParentRight="true"
    android:text="Go"
    android:textStyle="bold"
  />
  <Button
    android:id="@+id/btnCancel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/txtUserName"
    android:layout_toLeftOf="@+id/btnGo"
    android:layout_toRightOf="@+id/lblUserName"
    android:layout_alignParentRight="true"
    android:text="Cancel"
    android:textStyle="bold"
  />
</RelativeLayout>
The Relative Layout

2. Relative Layout (as of Sept 2012)

Using Eclipse ADT Layout Editor for designing a RelativeLayout.

The Table Layout

3. Table Layout

1. Android’s TableLayout uses a grid to position your widgets.
2. Cells in the grid are identifiable by rows and columns.
3. Columns might shrink or stretch to accommodate their contents.
4. The element TableRow is used to define a new row in which widgets can be allocated.
5. The number of columns in a TableRow is determined by the total of side-by-side widgets placed on the row.
3. Table Layout

The number of columns in a row is determined by Android.

So if you have three rows, one with two widgets, one with three widgets, and one with four widgets, there will be at least four columns.

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
```

However, a single widget can take up more than one column by including the `android:layout_span` property, indicating the number of columns the widget spans (this is similar to the `colspan` attribute one finds in table cells in HTML).

```
<TableRow>
    <TextView android:text="URL:" />
    <EditText android:id="@+id/entry"
               android:layout_span="3" />
</TableRow>
```
Basic XML Layouts - Containers

3. Table Layout

Ordinarily, widgets are put into the first available column of each row.

In the example below, the label ("URL") would go in the first column (column 0, as columns are counted starting from 0), and the TextField would go into a spanned set of three columns (columns 1 through 3).

```
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myTableLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ffffff00"
    android:orientation="vertical">

    <TableRow>
        <TextView
            android:text="URL:"/>
        <EditText
            android:id="@+id/ediUrl"
            android:layout_span="3"/>
    </TableRow>

    <View
        android:layout_height="3dip"
        android:background="#0000FF"/>

    <TableRow>
        <Button
            android:id="@+id/cancel"
            android:layout_column="2"
            android:text="Cancel"/>
        <Button
            android:id="@+id/ok"
            android:text="OK"/>
    </TableRow>

    <View
        android:layout_height="3dip"
        android:background="#0000FF"/>
</TableLayout>
```

Note to the reader:
Experiment changing layout_span to 1, 2, 3
Basic XML Layouts - Containers

3. Table Layout

By default, each column will be sized according to the "natural" size of the widest widget in that column.

If your content is narrower than the available space, you can use the TableLayout property:

```xml
android:stretchColumns="..."
```

Its value should be a single column number (0-based) or a comma-delimited list of column numbers. Those columns will be stretched to take up any available space yet on the row.

---

Basic XML Layouts - Containers

3. Table Layout

In our running example we stretch columns 2, 3, and 4 to fill the rest of the row.

```xml
...<TableLayout
android:id="@+id/myTableLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#ff0033cc"
android:orientation="vertical"
android:stretchColumns="2,3,4"
xmns:android="http://schemas.android.com/apk/res/android">
...</xml>

TODO: try to stretch one column at the time 1, then 2, and so on.
Lesson 4

Basic XML Layouts - Containers

4. ScrollView Layout

When we have more data than what can be shown on a single screen you may use the ScrollView control.

It provides a sliding or scrolling access to the data. This way the user can only see part of your layout at one time, but the rest is available via scrolling.

This is similar to browsing a large web page that forces the user to scroll up the page to see the bottom part of the form.

4. Example ScrollView Layout

```xml
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myScrollView1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ff009999">
    <LinearLayout
        android:id="@+id/myLinearLayoutVertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical">
        <TextView
            android:id="@+id/textView2"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Line2"
            android:textSize="150dip"/>
        <View
            android:layout_width="fill_parent"
            android:layout_height="6dip"
            android:background="#ffccffcc"/>
        <TextView
            android:id="@+id/textView1"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Line1"
            android:textSize="150dip"/>
        <View
            android:layout_width="fill_parent"
            android:layout_height="6dip"
            android:background="#ffccffcc"/>
        <TextView
            android:id="@+id/textView3"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Line3"
            android:textSize="150dip"/>
    </LinearLayout>
</ScrollView>
```
Basic XML Layouts - Containers

4. Example ScrollView Layout

5. Miscellaneous. Absolute Layout

- A layout that lets you specify exact locations (x/y coordinates) of its children.
- Absolute layouts are less flexible and harder to maintain than other types of layouts without absolute positioning.