User Interface Adaption in Android Device for Orientation Change

Qian Xu, Sihan Cheng, Mao Zheng
Department of Computer Science
University of Wisconsin-La Crosse
La Crosse WI, 54601
mzheng@uwlax.edu

Abstract

Mobile devices can respond as the orientation changes. The user can hold the device in random orientations, and if the device supports this feature, the user may see some additional options displayed by the device when in different orientation modes. For example, the screen only displays the list of items in the portrait mode, but shows two panels in the landscape mode: one panel is the list of items, the second panel is the details/contents of the selected item. On the other hand, some apps may prefer a particular orientation over the other. For example, playing a video in the landscape mode and making a phone call in the portrait mode.

Since the device will be used in different orientations, it is important to design and implement mobile application supporting the layout in both portrait and landscape orientations. We are interested in a mobile user interface automatic adaption, for the device, as the orientation changes.

An arithmetic game application is selected to illustrate our design and implementation of the user interface adaption as the device orientation changes. This application is developed for users of different ages, with different arithmetic skills. The mobile user interface will adapt to contextual information automatically. This paper discusses some of the basic concepts that will be used in the decision of implementing a layout for both portrait and landscape orientations. Some of the challenges will be presented along with our solutions.

1 Introduction

Traditionally, people will use their mobile phone in its default mode: portrait mode. However, along with the increasing usage of mobile devices and advances in technology, the user may rotate the mobile phone to landscape mode in order to get a larger input space when typing an address. Another example is when a user is playing a game, he/she may rotate the device to landscape mode in order to be more comfortable using both hands to hold and operate the device. When a user is watching video or browsing pictures, he/she may prefer the landscape mode in order to have a larger screen size. There are also situations that the user adds context information in his/her list of preferences: a user may hold the mobile device in portrait mode because he/she is using an app to read the news while having breakfast. Since there is only one hand available to hold the device, portrait mode could be the user's first choice. In addition, if a user wants to follow a recipe via a mobile device when cooking the food in the kitchen, the user may need to lean the device against the wall in landscape mode. Vertical stability has a positive correlation with height of gravity center, thus leaning the device in landscape mode is more stable.

In above situations, the mobile device can and should respond as the device orientation changes. The user can hold the device in random orientations, or according to his/her preferences.

However, sometimes the device also supports certain advanced features. The user may see some different or additional options displayed by the device when in different orientation modes. For example, the screen only displays the list of items in the portrait mode, but shows two panels in the landscape mode: one panel is the list of items, the second panel is the details/contents of the selected item.

Since the device will be used in different orientations, it is important to design and implement mobile applications supporting layouts in both portrait and landscape orientations. We are interested in a mobile user interface automatic adaption for the device as the orientation changes.

2 Different Android Technologies in Dealing with Orientation Change

There are two major platforms in the mobile device community: iOS and Android. This project chose Android development [2] mainly for the reason of its openness. In addition, all the tools in the Android development are free and no special hardware is required.

In Android platform, there are two ways of applying different layouts to the application when the device orientation changes. One allows the activity to be recreated, the other does not.

1. To allow the activity be recreated, there are also two different techniques:

a. The simplest way is to create both “layout-land” and “layout-port” directories under “res” directory in the Android Studio. The “layout-land” directory will keep the layout xml file for the landscape mode, and the “layout-port” directory will keep the layout xml file for the portrait mode. The xml file name should stay the same in both directories. When the device orientation changes at run time, the activity is shut down and restarted by default, thus the application will need to use the onCreate() function and then it will load the correct layout file based on the device orientation. The code segment below illustrates this technique in red:

```
public class MainActivity extends Activity {
   ...
  @Override
  protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);

     /* activity_main has two versions with the same name for both portrait
        and landscape modes. The correct version will be automatically loaded
        based on the device’s orientation.
      */
        setContentView(R.layout.activity_main);
   ...
  }
   ...
}
```

b. We can also write java code using the onCreate() function, to get the device’s current orientation and then load the corresponding layout file. That means we need to explicitly write two layout files, one for portrait mode and the other is for landscape mode. When the device orientation changes, the onCreate() function will be recalled and at the same time the correct layout will be loaded. The difference between this technique and the one mentioned above is the corresponding layout file is loaded automatically, based on the device’s orientation in a. Here, we wrote our own code to check the device’s orientation and then loaded the corresponding layout file explicitly. The code segment is listed below.

```
@Override

protected void onCreate(Bundle icicle) {

 super.onCreate(icicle);

 int mCurrentOrientation = getResources().getConfiguration().orientation;

 if ( mCurrentOrientation == Configuration.ORIENTATION_PORTRAIT ) {

     // If current screen is portrait

     setContentView(R.layout.mainP); //load the portrait layout

 } else if ( mCurrentOrientation == Configuration.ORIENTATION_LANDSCAPE ) {
     //If current screen is landscape
     setContentView(R.layout.mainL);  // load the landscape layout
 }
```

The disadvantage of recreating the activity when the device's orientation changes is that the current state of the activity will be lost. For example: what choice did the user select from a ListView before the orientation changed? To solve this issue, we needed to store the current state of the activity before recreating a new activity, and then retrieving the stored data after recreating. Android calls onSaveInstanceState() before it destroys the activity so that we can save data about the application current state, and then restore the state during onCreate() or onRestoreInstanceState(). Sample code segments are shown below.

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
        super.onSaveInstanceState(savedInstanceState);
        // Save UI state changes to the savedInstanceState.
        // This bundle will be passed to onCreate if the process is killed and restarted.
        savedInstanceState.putString("selected", "domestic news");
}

@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);
        // Restore UI state from the savedInstanceState.
         // This bundle has also been passed to onCreate.
        String selected = savedInstanceState.getString("selected");
}
```

2. If we do not want the activity recreated, we can let the onConfigurationChanged() function deal with the orientation change. This can be done using the following steps. Firstly, in the AndroidManifest file set the tag android:configChanges="keyboardHidden|orientation" so that each time the orientation changes onCreate() will not be invoked. Instead, onConfigurationChanged will be used. Secondly, using the onConfigurationChanged(Configuararion newConfig) method, get the current orientation and decide which layout to use. The sample code segment is shown below.

```
@Override
protected void onConfigurationChanged(Configuration newConfig) {
        super.onConfigurationChanged(newConfig);
        // Checks the orientation of the screen
          if (newConfig.orientation ==Configuration.ORIENTATION_LANDSCAPE){
        .....
         } else if (newConfig.orientation == Configuration.ORIENTATION_PORTRAIT){
        .....
         }
}
```

The disadvantage of the second technique is that we cannot get the location, the size of the layout and any component in this function. If we need to redraw or move some components based on the current measurement, asynchronous messaging and a delayed method will need to be used. Below is an example of using a delayed method to change/update the location of a pop-up window in an application.

```
@Override
protected void onConfigurationChanged(Configuration newConfig) {
   super.onConfigurationChanged(newConfig);
   postDelayed(new Runnable() {
      @Override
      public void run () {
         updatePopup();
      }
   }, 500);
}
```

With a situational layout, you will encounter redundancy, since you may need to write a similar layout in different cases. All the techniques mentioned above, may also require additional work, such as overriding the onRetainNonConfiurationInstance() method or using a delayed method. The designer has to choose and balance the trade-offs among these techniques.

3 A Case Study: Arithmetic Game

We are interested in developing a mobile user interface automatic adaption for a device as the orientation changes. An Android app, arithmetic game is chosen to illustrate our design.

The arithmetic game is developed for users of different arithmetic skills, with a context-based GUI adaptation. Users are required to register and obtain an account to login. During the user's registration process, the user's age is stored as one of the logic context information that will be used in the beginning of the app to assign the appropriate question level to the user.

There are two modes in the application: standard mode and review mode. The standard mode allows the user to practice or test his/her arithmetic skills through questions at different levels and units. The review mode is to allow the user to redo the questions he/she made mistakes on.

In the standard mode, there are three levels corresponding to three age domains. Each level is divided into 10 units and each unit contains 10 problems. When the user logs in for the first time, he/she is automatically assigned to a level based on his/her age. All the problems are generated randomly based on the rules shown in the Table 1 found below. Generating questions randomly, instead of retrieving questions from a database, can prevent the user from remembering the order of the answers when he/she plays the game again and again.

Age	Level	Operands for +	Operands for -	Operands for *	Operands for /
[0, 5]	1	Both [0, 10]	Both [0, 10] minuend > subtrahend	Not available	Not available
[6, 11]	2	Both [0, 50]	Both [0, 50] minuend > subtrahend	Both[0, 10]	Both [0, 10] Quotient is integer
>= 13	3	Both [-100, 100]	Both [-100, 100]	Both[-10, 10]	Both [-10, 10] Quotient is integer

Table 1 Classification of Question Levels

When the users answer questions, there are 10 seconds for each question. A graphic countdown timer will work as a reminder.

4 Context-based Mobile User Interface Adaption

We categorized the context into logical and physical categories. Logical context is the information related to the user's profile or preference. Physical context is information about the environment, such as the location and time, which can be obtained through the device's built in sensors. When the user logs in for the first time, the user's age is the only information the device has. The interface will show the appropriate question level based on the user's age. As the user plays more and more with the app, the mobile user interface will be automatically changed based on real-time context: the user's performance in the last unit he/she practiced.

We divided the accuracy of the last unit into three groups: [0%, 60%], (%60, 90%), [90%, 100%]. The arithmetic game presents three themes for the three different accuracy groups, respectively. For the first group, the accuracy of the last unit is between 0% and 60% both inclusive, the application will simply present a default white and black theme, shown in Figure 1. For the second group, the accuracy of the last unit is between 60% and 90% both exclusive, the user is able to have a customized theme for the mobile user interface. In the application's setting, the user can choose preferred colors for different components, shown in Figure 2. For the third group, the accuracy of the last unit is between 90% and 100% both inclusive. The user can have the options of choosing weather and time related themes. In this theme, the weather icon will follow the local weather information and the background picture will also change at different times of day. Figure 3 shows a snowy day and Figure 4 shows a rainy night, respectively. These pictures were generated based on the current weather information and time of day at the time the device was used.

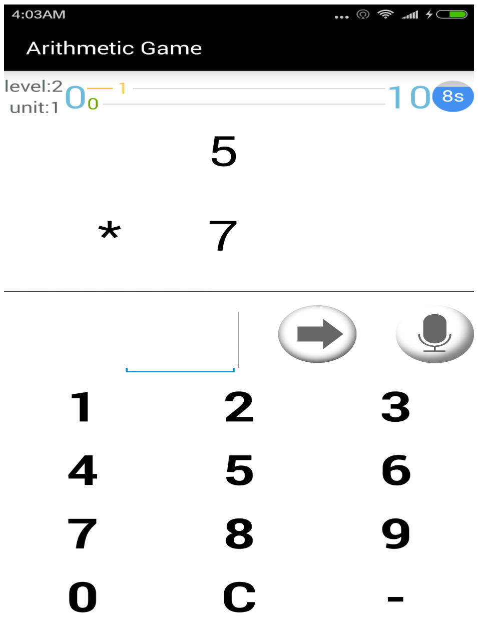
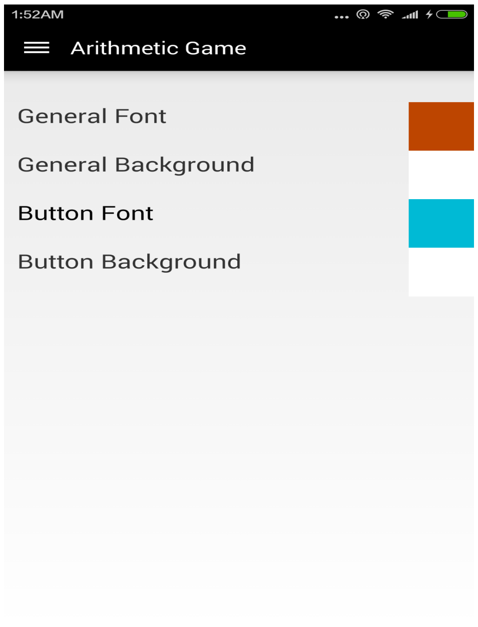
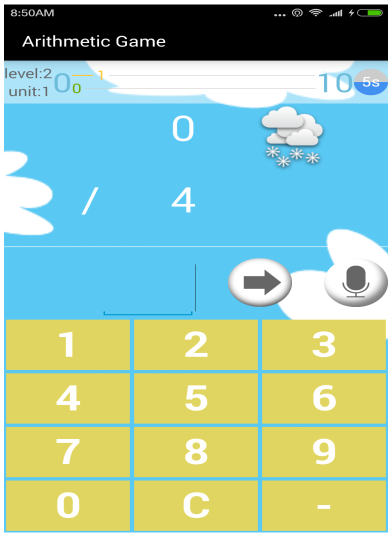


Figure 1 Black and White Theme



General Font 🟧
General Background ⬜
Button Font 🟦
Button Background ⬜

Figure 2 User's Preferred Colors



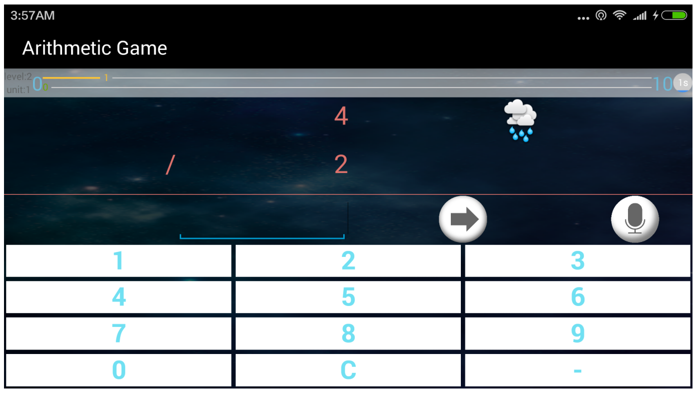


Figure 3 Snowy Day

Figure 4 Rainy Night

4.1 Orientation Change

In the arithmetic game application, we applied the same layout for both portrait and landscape orientations. The reason is that the design for portrait mode can fully satisfy the need in landscape mode. There are no additional or different options available in landscape mode. In addition, because the count down timer and random number generator existed in the activity, if a different layout is re-loaded, restoring the same interface components, extra developing effort would be required and the application's performance would be decreased.

We tried to handle the configuration change by overriding onConfigurationChanged() without recreating the activity.

If we use accurate size components, the layout will be difficult to design as the orientation changes. Instead, we set most components' width and height values to "match parent" or "0dp" and used `android:layout_weight`. The weight assigns an "importance" value to a view in terms of how much space it should occupy on the screen. A basic idea would be that, if we want the same layout adaptable in both orientations, we should adjust the components' width based on current screen width, and adjust components' height based on current screen height. This is the reason we used the weight to specify a size ratio between multiple views.

However, this technique is not appropriate to use to adjust the size of the text. We will need additional time using java code to adjust text size based on the current size of the screen. In Android, auto scale text to fit within bounds is not supported in a xml file. A good and simple solution is to get the size of the current screen using onConfigurationChanged(), and calculate the text size and set the new value. The code segment is shown below.

```
public void onConfigurationChanged(Configuration newConfig) {
   super.onConfigurationChanged(newConfig);
   initScreenLayout();
}

protected void initScreenLayout() {

   /*
    *  Calculate current screen height
    */
   display = getWindowManager().getDefaultDisplay();
   size = new Point();
   display.getSize(size);
   float maxHeightInPX = size.y;

   /*
    *  Convert pixel to sp
    */
   float bigSizeInSP =  pixelsToSp(getApplicationContext(), maxHeightInPX / 15);
   float smallSizeInPX =  maxHeightInPX /40;
   float smallSizeInSP =pixelsToSp(getApplicationContext(),  smallSizeInPX);
   float clockSizeInPX = maxHeightInPX/20;
   float clockSizeInSP = pixelsToSp(getApplicationContext(),  clockSizeInPX);

   // apply to all GUI widgets
   for (int i = 0; i < buttonSet.size(); i++) {
      buttonSet.get(i).setTextColor(buttonFont_color);
      buttonSet.get(i).setBackgroundColor(button_color);
      buttonSet.get(i).setTextSize(TypedValue.COMPLEX_UNIT_SP, bigSizeInSP);
      buttonSet.get(i).setGravity(Gravity.CENTER);
   }
  …

   LinearLayout.LayoutParams params = new
                    LinearLayout.LayoutParams((int)(clockSizeInPX), (int)(clockSizeInPX));

 …
}
```

The first step is to calculate the current screen size, and the second step is to use the method pixelsToSp to convert the results in pixels to sp. Sp means scale-independent pixels in Android. This can then be applied to all the GUI widgets to set the font size by using setTextSize(TypedValue.COMPLEX_UNIT_SP, bigSizeInSP). The last step is to set the layout programmatically via LayoutParams instead of using specific layout-files to have customView update its layout.

5 Conclusions

There are many more additional context adaptions in this arithmetic game application. For the scope of this paper, we only discussed the design and implementation of the mobile user interface adaption when the device's orientation changes.

References

[1] Weiser, M. "*The computer for the 21st century*", Scientific American, 1991
pp. 94-104.

[2] Android Developer's Guide. http://developer.android.com/guide/index.html

[3] Dey A. "Providing Architectural Support for Building Context-Aware Applications", Ph.D. thesis, College of Computing, Georgia Institute of Technology, Dec. 2000.

[4] Derek Riley, Using Mobile Phone Programming to teach Java and Advanced Programming to Computer Scientist, ACM Special Interest Group on Computer Science Education SIGSCE 2012, pp:541-546. Feb. 29-March 3, 2012.

[5] B. N. Schilit, N. Adams, and R. Want. Context-aware Computing Applications. In Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, pp. 85-90, Santa Cruz, CA, Dec. 1994. IEEE Computer Society Press.

[6] https://github.com/AndroidDeveloperLB/AutoFitTextView