NOVI PRISTUPI I MOGUĆNOSTI U KITKAT ANDROID 4.4 API-JU

NEW APPROACHES AND POSSIBILITIES IN ANDROID 4.4 KITKAT API

Zoran Kos, Zlatko Stapić

SAŽETAK

Android OS je danas jedan od najrasprostranjenijih OS-a na novoisporučenim mobilnim i tablet uređajima. Razlog zbog kojeg postoji interes za pisanjem o ovoj temi nije samo u tome što sve više ljudi zamjenjuje rad na računalima i laptopima za rad na mobilnim uređajima i tabletima, već što su u najnovijoj verziji Android OS-a implementirana inovativna i svježa rješenja koja korisniku omogućuju jednostavniji i brži rad, uštedu energije, uštedu vremena i manje uloženog novca. Rad prikazuje nove pristupe i mogućnosti trenutno najnovije verzije Androida 4.4. KitKat API-ja, koji još nije toliko rasprostranjen na tržištu zbog ograničene dostupnosti na određenim mobilnim uređajima i tabletima. Veći naglasak je stavljen na primjenu tehnika i tehnologija koje su usko vezane za Cloud Computing, odnosno računalstvo u oblaku. Neke od novosti su ugrađena mogućnost za ispis sadržaja prikazanog na ekranu pomoću pisača, ušteda energije primjenom senzora pokreta, novi pristup u organizaciji memorijskog i spremničkog prostora izvršavanjem aktivnosti u pozadi OS-a, smanjena mogućnost rušenja aplikacija itd.

ABSTRACT

Android OS is today one of the most widespread used OS on mobile and tablet devices. The reason why there is an interest in writing about this topic is not only because more and more people replace work on computers and laptops to mobile devices and tablets, it is because in the latest version of the Android OS are implemented innovative and fresh solutions that allow the user simpler and faster work, better energy, time savings and less invested money. This work presents new approaches and opportunities currently on the latest version of Android 4.4. KitKat API, which has not been so widespread on the market due limited availability on certain mobile devices and tablets. Bigger emphasis is placed on the application of techniques and technologies that are closely related to cloud computing. Some of these innovations in KitKat include integrated option to print the content on the screen using printer devices, saving energy with using motion sensors, a new approach in the organization of memory and storage tank, performing those activities in the background of the OS, the minimum possibility of demolition applications and so on.

1. INTRODUCTION

The first appearance of the Android OS on a mobile device took place on the commercial market in September 2008. It is developed by the Google Corporation along with the Open Handset Alliance (OHA). Since the first version until today, Google has released 19 different versions of Application Programming Interface (API). This number is impressing considering that this was a period of less than six years. According to available statistical data on worldwide basis the number of new delivered units of the Android OS in 2013 is greater than 500 million which is equivalent to 78.60% of the worlds' market, followed by iOS with 26.90% and WP8 with 1.60% (MobiThinking, 2014). If we are looking only on Android OS statistics (Android Developers, 2014a), currently the most common and most popular version is Android Jelly Bean (API 16, 17, 18) with a 64.40% of contributions. Second one is Gingerbread (API 10) with 17.80%, and third is Ice Cream Sandwich (API 15) with 15.00%. On fourth place is the latest version of Android 4.4. (API 19), also known as KitKat, with a 5.30% share, having its percentage doubled if compared to the second month of the year. Demand for KitKat is steadily growing as it brings new solutions, capabilities, enhanced performance, minimized fragmentation, reduced OS; but does not dramatically change graphical design in comparison with predecessor Android Jelly Bean. There are a few new features and improvements but not all of them are perfect yet.

The purpose of this paper is to present new approaches and possibilities for developers in using Android 4.4 KitKat API. The paper is divided into 5 chapters. Upon the introduction presented in this chapter, the second chapter presents general facts related to this version of API and it also enumerates important new functionalities which represent the building blocks for the third, fourth and fifth chapter. The third chapter presents KitKat's new capabilities that are related to Memory management, file management, system-wide wireless printing and integration with cloud storage, and at last Android Transition Framework. The purpose of this chapter is to familiarize developers with new opportunities of applications development in KitKat. This is the most important chapter of the paper, while the forth chapter discusses differences between KitKat and his predecessor Jelly Bean observed by the user side (closely associated with the appearance and functions of a graphical interface) and the last chapter concludes the paper.

2. ANDROID 4.4. KITKAT API

"*Smart, simple, and truly yours*" - this is the slogan of the latest version of Android 4.4., popularly known as KitKat. "*Nestle",* the KitKat brand owner, promotes the Android logo on the chocolate wrappers and thus promotes the new OS.

KitKat was first introduced in September 2013 and was initially available only on Google's Nexus 5 mobile devices. One of the important features of this version is

that it tends to use smaller amounts of RAM, which is not in common for most modern mobile devices. Google has devoted special attention to this by organizing "Project Svelte" (various memory-saving changes to Android framework, improved memory diagnostics, tuning knobs for Android etc.) whose results have recorded positive success that was used on KitKat API. The optimal amount of RAM is 512 MB and 340 MB is the minimum, this is reasonable because most of today's mobile devices have this size of RAM.

Table 1 - New capabilities in Android 4.4.

New capabilites of Android 4.4.
New Memory Management
New File Management (Google cloud support)
System-wide Wireless Printing
Step counting built-in Sensors
FullScrean Read (Immersive) Mode
Smarter caller ID
Faster Multitasking (improved touchscreen)
Quick Office – working with documents
Widgets in lockscreen
New Alarm System Notification
Chrome web view
Infrared blasting (Tv remote control)
Application sandboxes
Low-power location monitoring
Bluetooth Message Acces Profile (MAP) Support

KitKat also brings new memory and file management, native cloud support, improved multitasking, widgets in lock screen, application sandboxing, low-power application monitoring and other under-hood improvements in many OS elements. The complete list of improvements is presented in Table 1, and some of them are described in next chapters.

Generally speaking, KitKat is smoothly polished and it offers a huge number of applications that are built into the OS and give great benefits to users and developers.

3. NEW POSSIBILITIES IN KITKAT

3.1 Memory Management

Android devices need memory for OS, background services, video buffers and apps. Generally, the Android OS for work with memory uses paging or memory-mapping (mmaping). When change happens in memory, whether it is about new object or the opening mmpaped site it will be recorded in the memory and the object will not be released until the release reference is set and placed in Garbage Collection. Activity Manager constantly sorts out the applications by per-process measure called *oom_adj*, and as memory pressure increases, the low memory killer starts killing processes. As it was mentioned earlier, KitKat is designed to work with RAM memory of 512MB, and this was achieved with behind-the-scenes improvements. For this purpose they had made revisions to remove the unnecessary backgrounds services and accessories/features that had plagued the multiple memory processing. Such optimization with new APIs effects on all installed applications for entry-level and upper-class devices and allows better multi-tasking. System memory is reduced with following (Android Developers, 2014b):

- ✓ Trimmed *system server* and *SystemUI* processes (saved several *MBs*).
- ✓ Preload *dex* caches in *Dalvik* (saved several MBs in process virtual machine).
- ✓ Validated *Dalvic JIT-off* option (just-in-time compiler option saves up to 1.5MB per process).
- ✓ Reduced per-process font cache overhead.
- ✓ Introduced ArrayMap, ArraySet and used extensively in framework as a lighter-footprint replacement for HashMap, HashSet.

New API called *ActivityManager.isLowRamDevice()* will determine if apps run on low memory device and will act if apps have specific memory-intensive features that work poorly, so they can be turned off (some Google apps are reportedly coded to make this check). To be able to work, this API must be set true by the system property in the device *makefile*:

```
PRODUCT_PROPERTY_OVERRIDES +=
       ro.config.low_ram=true
```

Google also gave better support to developers and Original Equipment Manufacturers (OEM) who can easily take advantage of large set of tools and accessories available to them. Thus, developers can better monitor the memory usage and work timeline with some of the following tools and activities:

- ✓ new tool *procstats GUI* inside device settings Settings→Developer options→Process Stats
- ✓ new adb shell dumpsys procstats command-line - details about memory use over time, with run times and memory footprint
- ✓ new adb shell dumpsys meminfo (similar to tool-set *procrank* but with more clear info)
- ✓ toll *bugreports* over adb - the services now include *batterystats, procstats, usagestats* and *netstats*.


```
* com.test.procstats / u0a51:
   * com.test.procstats / u0a51:
           TOTAL: 100% (4.4MB-5.0MB-6.1MB/3.0MB-3.1MB-3.1MB over 3)
             Top: 1.7% (6.1MB-6.1MB-6.1MB/3.1MB-3.1MB-3.1MB over 1)
         Service: 90% (4.4MB-4.4MB-4.4MB/3.0MB-3.0MB-3.0MB over 2)
      Service Rs: 8.1%
   * com.test.procstats.ExpectedSlowService:
     Process: com.test.procstats
        Running count 2 / time 0.23%
        Started count 1 / time 0.23%
      Executing count 2 / time 0.01%
   * com.test.procstats.ExpectedQuickService:
     Process: com.test.procstats
        Running count 1 / time 92%
        Started count 1 / time 92%
      Executing count 1 / time 0.01%
```

Figure 1 - New Tool procstats GUI in KitKat (Wallat, 2013)

shows an app run in Dalvik Debug Monitor Server (DDMS) with two services that ran once at startup. Service *ExpectedQuickService* is running 92% of the apps run time, while service *ExpectedSlowService* 0.23% of the apps run time. Just by looking at this report we can identify where the problem is.

```
* com.test.procstats / u0a51:
    * com.test.procstats / u0a51:
            TOTAL: 100% (4.4MB-5.0MB-6.1MB/3.0MB-3.1MB-3.1MB over 3)
              Top: 1.7% (6.1MB-6.1MB-6.1MB/3.1MB-3.1MB-3.1MB over 1)
          Service: 90% (4.4MB-4.4MB-4.4MB/3.0MB-3.0MB-3.0MB over 2)
        Service Rs: 8.1%
    * com.test.procstats.ExpectedSlowService:
      Process: com.test.procstats
          Running count 2 / time 0.23%
          Started count 1 / time 0.23%
        Executing count 2 / time 0.01%
    * com.test.procstats.ExpectedQuickService:
      Process: com.test.procstats
          Running count 1 / time 92%
          Started count 1 / time 92%
        Executing count 1 / time 0.01%
```

Figure 2 - ADB shell procstats – details (Smith, 2013)

New memory management offers safer and neater work due to the use of Application sandboxes with Security-Enhanced Linux. Shortly, each app runs in a restricted environment, similar as in iOS, which has prevented the apps that perform illegal actions. Now they cannot read or change data of other Apps. On the other hand that has made a big problem in case when user or OS wants to make changes on the SD memory card and other external media with content that was created by previous API versions.

3.2 New File management

Android 4.4 offers a new, so-called, Storage Access Framework (SAF) in which users can interact with stored content as images, videos and documents. For example if user wants to send a message, UI lets user to browse files and access to recent content in consistent way across app files and providers instead of showing user a dialog to choose an application to handle content. The UI is standardized and easy to use because it combines a variety of applications as Google Drive, Dropbox, download folder with the content and gallery in one central location, together with corresponding data (see Figure 3). Developers can use and operate with **centralized file management** location where they can add their own storage service without making it on the vendor by vendor basis. Using SAF, i.e. cloud storage is useful on mobile devices for two reasons:

a) Internet connection becomes faster, efficient and provides an inexpensive transmission of large amounts of data.
b) Mobile devices have not yet overgrown the size of the PCs or laptops storage space.

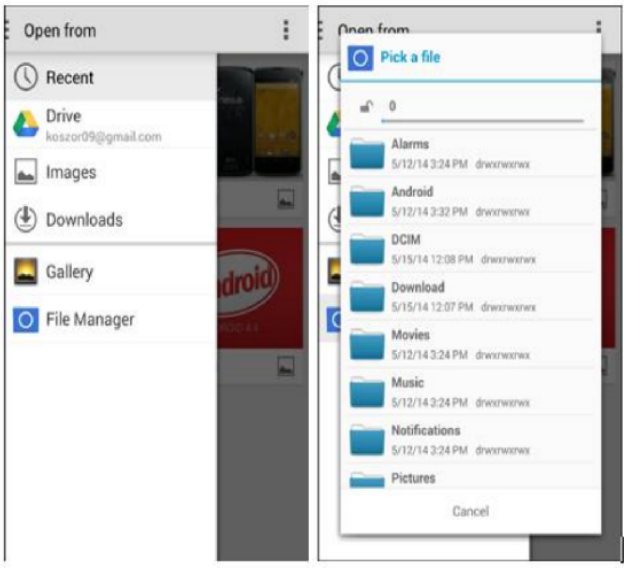

Figure 3 – New SAF view

Client apps that need access to a provider's documents can integrate with the SAF in a few lines of code by implementing a *document provider* (which offers read and write access to durable files, such as files stored on a local disk, or files in cloud storage). The SAF includes the following:

- ✓ Document provider
- ✓ Client app
- ✓ Picker - system UI that lets user access documents from document providers that satisfy the client app' search criteria.

Cloud providers or local storage services (offered by individual Android manufacturers) can use the new system by implementing a new document provider class within Android for their service, the document provider class within Android has the APIs necessary to manage, browse, read or write documents within the app from a variety of sources (Rowinski, 2013). In this way providers and clients don't interact directly because client request permission to interact with files. Some of the features offered by the SAF are following (Android Developers, 2014c):

- ✓ It lets users browse content from all document providers, not just a single app.
- ✓ It makes it possible for app to have long term, persistent access to documents owned by a document provider. With this access users can add, edit, save, and delete files on the provider.
- ✓ It supports multiple user accounts and transient roots such as USB storage providers, which only appear if the drive is plugged in.

For example, Figure 4 shows how a photo app might use the SAF to access stored data:

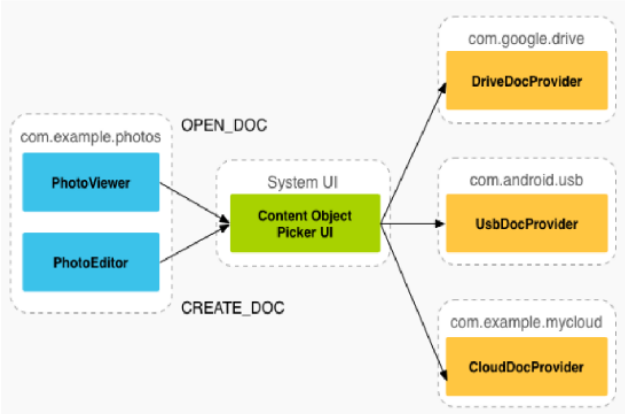

Figure 4 - Storage Access Framework Flow (Android Developers, 2014c)

The following example shows how to load and save content with SAF on client side (request Content from a Provider). Local providers, cloud services, and external storage use `DocumentsProvider` class with abstract `ContentProviders` interface and are treated in the same way providing the developer with one place to interact with the user's content. Developers need to call `ACTION_OPEN_DOCUMENT` Intent to pick content, which signifies that we want to connect to all content providers available to the device. Calling `StartActivityForResult` will launch SAF UI (see Figure 3).

It is possible to implement filtering to this Intent based on `MimeType` by specifying `CategoryOpenable`, which means only content that can be opened will be returned. The code below shows filters for image results by specifying the image `MimeType`:

```
public void searchFile(){
 Intent intent = new Intent
 (Intent.ACTION_OPEN_DOCUMENT);
 intent.AddCategory
 (Intent.CATEGORY_OPENABLE);
 intent.SetType ("image/*");
 StartActivityForResult (intent,
 SAVE_REQUEST_CODE);
}
```

Code 1 – Method for filtering the documents by Mimetype

If image is selected, `OnActivityResult` returns the `Android.Net.Uri` of the chosen file. The code sample below displays the user's image selection:

```
protected override void
OnActivityResult(int requestCode, Result
resultCode, Intent data) {
 base.OnActivityResult(requestCode,
 resultCode, data);

   if (resultCode == RESULT.Ok &&
   data != null && requestCode ==
   SAVE_REQUEST_CODE) {
        ImageView imageView =
    FindViewById<ImageView>
    (Resource.Id.imageView);
    imageView.SetImageURI
    (data.Data);
  }
}
```

Code 2 – User's image selection

With document URI, document can be open; here is an example of how to open a Bitmap:

```
private Bitmap getBitmapFromUri(Uri uri)
throws IOException {
  ParcelFileDescriptor
  par = gettContentResolver().
  openFileDescriptor(uri, "r");
  FileDescriptor file =
  par.getFileDescriptor();
  Bitmap image = BitmapFactory.
  decodeFileDescriptor(file);
  par.close();
  return image;
}
```

Code 3 – Method for opening the selected bitmap file

In case when new file is created, user must press Save button, `OnActivityResult` gets passed the `URI` which can be accessed with `data.Data`. The Uri can be used to stream data into the new file:

```
private void createFile(String type,
String fileName) {
  Intent intent = new Intent
  (Intent.ACTION_CREATE_DOCUMENT);
  // Filter to only show results that
  can be "opened"
  intent.addCategory
  (Intent.CATEGORY_OPENABLE);

  // Create a file
  intent.setType(„text/plain");
  intent.putExtra
  (Intent.EXTRA_TITLE, fileName);
  startActivityForResult(intent,
  WHRITE_REQUEST_CODE);
}
```

Code 4 – Creating new document in a document provider

Persistent access to a specific file can be obtained by "taking" the necessary permissions for the Uri. The following code read and writes permissions for referenced file by the `fileUri` instance (Android Developers, 2014c):

```
final int takeFlags = intent.getFlags()
&(Intent. FLAG_GRANT_READ_URI_PERMISSION
|Intent.
FLAG_GRANT_WRITE_URI_PERMISSION);
// Check for the freshest data
getContentResolver().takePersistableUriPe
rmission(fileUri, takeFlags);
```

Code 5 – Permissions

New changes have been made in the approach to External storage, divided into **two types**:

- ✓ Storage unique to developer app,
- ✓ Data shared by multiple applications

The external storage API was split out to include multiple volumes, one *primary* and one or more *secondary* (modify *write* permissions). Apps can't read or write shared files on the external storage while they are running, unless they have `READ_EXTERNAL_STORAGE` or `WRITE_EXTERNAL_ STORAGE` permission. Write permission implies the read permission. Therefore it is necessary to set one of these two permissions inside the application.

3.3 New System-wide Wireless Printing and Integration with existing Cloud Storage

KitKat also brings a new feature to print documents and photos right from the phone. Google made it possible by implementing direct support for Google Cloud Print, a technology connecting printers to the cloud. It enables users to connect their phone to the printer over WiFi or Bluetooth and print selected documents in a very short period of time. However, this feature is limited to compatible printers having Cloud Print support and wireless technology built-in. It's important to point out that before printing; each document has to be converted to PDF which is a default format for printing. In addition, printer manufacturers (OEM) can use new APIs to develop their own printing services - pluggable components that add services (vendor-specific logic) for communicating with specific types of printers that are available on Google Play. They can use the `android.printservice` framework to provide interoperability with printers from Android devices.

One of the KitKat's important features is Google Cloud Storage, its previously existing support to use Google Drive service meant for storing and sharing documents and files (users have fast access to your app's data from any location based on global edge-caching). A user can save and open document or file directly from cloud, without previously saving it to their phone's storage. An app like QuickOffice (made for creating and editing Microsoft® Office documents, spreadsheets and presentations) already has the ability to connect with Google Drive.

By combining Cloud Storage and Cloud print technologies, Google enabled users to print files from their PC or tablet, and developers got a possibility to save files with high reliability and availability. The following example is dedicated to document printing possibilities of content that is currently shown on the screen. Base for that is usage of WebVIew class, which in its latest version supports HTML5 features, CSS3, V8 JavaScript engine and remote debugging of WebView content. The same was not supported in the previous versions, but now, developers got the new powerful way to develop web-based apps. New WebView is based on Chromium (open-source browser project that aims to build a safer, faster, and more stable way for all Internet users to experience the web) designed for Chrome Android browser version 30.

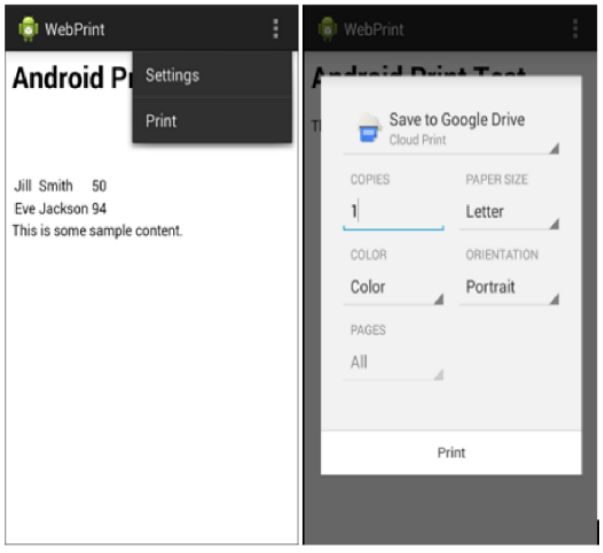

Figure 5 – System-wide Wireless Printing

For printing a web-page which consists of dynamic HTML content, Android Printing Framework has to be implemented within the source code of the application. The beginning step is to change the `manifest.xml` file where it's necessary to set `targerSdkVersion` to `19` and turn on the rule to enable internet connection.

To work with objects that are related to the WebView class the following classes must be imported:

- ✓ android.webkit.WebView;
- ✓ android.webkit.WebViewClient;

And to work with objects that are related to printing the contents, following classes should be imported:

- ✓ android.print.PrintAttributes;
- ✓ android.print.PrintDocumentAdapter;
- ✓ android.print.PrintManager;

Inside the main class which extends the *Activity class* in the method *onCreate()* the following code loads the dynamic html page and creates an instance of *WebView* class which is assigned to *WebViewClient*:

```
WebView webView = new WebView(this);
webView.setWebViewClient(new
WebViewClient() {
  public boolean
  shouldOverrideUrlLoading(WebView
  view, String url) {
       return false;
       }

 @Override
 public void onPageFinished(WebView
 view, String url) {
   webPrint(view);
   myWebView = null;
   }
});
webView.loadDataWithBaseURL(null, „html“,
"text/HTML", "UTF-8", null);
}
```

Code 6 – Loading website and creating new WebView instance

When HTML content of the web page is loaded into the WebView object, method `onPageFinished()` will be triggered that will save a reference of WebView object and invite WebPrint (view) method that opens a window for the print properties (printing panel). The document will be converted to PDF format, and this code is shown below:

```
webPrint(WebView view){
 PrintManager printManager =
 (PrintManager)this.getSystemService
 (Context.PRINT_SERVICE);

 PrintDocumentAdapter printAdapter =
 view.createPrintDocumentAdapter();

 String name = "Print Test";
 printManager.print(name,
 printAdapter, new PrintAttributes.
 Builder().build());
}
```

Code 7 – Method for calling print settings

`WebPrint` method retrieves references to `PrintManager` service, gives `WebView` instance and creates a print adapter. Metod `print()` inside the print

manager method pass through the job name, print adaptor and a set of default print attributes. Once the print job is instantiated, it is necessary to check whether the generated output is located at the selected location.

3.4 Android Transition framework

This chapter demonstrates how to use Animating User Interfaces with the new Android Transitions Framework (transitioning from one scene to another). In short, this Framework makes animations easier to use and make. Developers have now more control over animations, they can build complex animations with minimal working hours. This is because KitKat allows them to perform a simple property animation with one line of code and by importing libraries:

- ✓ android.Transitons;
- ✓ android.transition.TransitionManager.

To animate showing and hiding a `TextView`, in case when the button is pressed, we can use the following code example:

```
LinearLayout linear = (LinearLayout)
 findViewById(R.id.myWebView);
Button button = (Button)
 findViewById(R.id.button);
final TextView text = (TextView)
 findViewById(R.id.textView);

button.setOnClickListener(new
View.OnClickListener() {
  @Override
  public void onClick(View v) {
   TransitionManager.beginDelayed
   Transition(linear);
   if(text.getVisibility() !=
    View.VISIBLE){
    text.setVisibility(View.
    VISIBLE);
        }
        else{
    text.setVisibility(View.GONE);
        }
   }
});
```

Code 8 – Text view transition

We can get more control over the transition with Scenes. Scenes create a dynamic area in the UI using *Scenes class*. Android does the rest of the work to animate the transitions between the scenes but first developers must specify a container and several versions, or "scenes", for the XML content inside the container. To transition between two scenes, developers generally need to perform the following (Android Developers, 2013):

- ✓ Specify the `ViewGroup` containing the UI components you want to change.
- ✓ Specify the layout representing the end-result of the change (the next scene).
- ✓ Specify the type of transition that should animate the layout change.
- ✓ Execute the transition.

`Scene` object must be used to accomplish steps 1 and 2, and `TransitionManager` object to accomplish steps 3 and 4. A `Scene` contains metadata describing the properties of a layout that are necessary to perform a transition, including the scene's parent view and the scene's layout. For the following code example, dynamic content inside the container requires two new Android layouts. These layouts specify only the code inside the container which in our case is `scene_activity.xml`.

For proposes of this example, the two XML layouts have been created, and are included inside the container: *`first_scene_layout.xml`* where all elements are aligned horizontally and *`second_scene_layout.xml`* where all elements are aligned vertically – see Figure 6. A `Scene` is created by calling `Scene.getSceneForLayout`, passing in the container object, the Resource ID of the Scene's layout file, and the current `Context`. Below is a sample code for MainActivity class:

```
Scene first_scene;
Scene second_scene;
Transition transition;
@Override
protected void onCreate(Bundle
 savedInstanceState) {

 super.onCreate(savedInstanceState);
 setContentView(R.layout.
 scene_activity);
 ViewGroup rootContainer = (ViewGroup)
 findViewById(R.id.rootContainer1);

 transition = TransitionInflater.
  from(this).inflateTransition
  (R.transition.transition);
 first_scene = Scene.
  getSceneForLayout(rootContainer,
  R.layout.first_scene_layout,
  this);
  second_scene = Scene.
  getSceneForLayout(rootContainer,
  R.layout.second_scene_layout,
  this);

  first_scene.enter();
 }

public void firstScene(View view){
  TransitionManager.go(second_scene,
  transition);
}

public void secondScene(View view){
      TransitionManager.go(first_scene,
   transition);
}
```

Code 9 – The Main activity class showing way to manage the scene transition

First, it is necessary to add some code to load the layout from *`second_scene_layout.xml`* file into a `Scene` instance. After that we can implement the transition between the first and second scene with two methods `firstScene(view)` and `secondScene(view)`. These methods are called when one of the buttons is pressed in layout:

```
<Button
android:id="@+id/button1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Button 1"
android:onClick="firstScene" />
```

Code 10 – Button settings

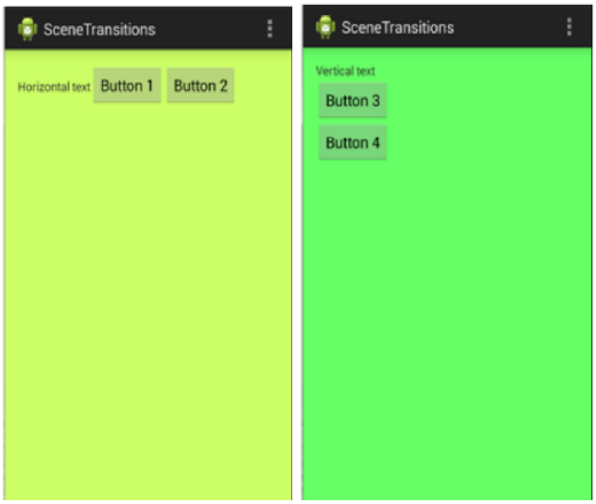

Figure 6 – Transition between two scenes

Their function is to load the transitions into a `TransitionManager` instance and reference it in the scene changes. Developers can also define sets of transitions programmatically with the `TransitionManager` APIs. All of the transition effects for this project are implemented within a transition XML resource file. Transition resource files must be placed in the *res/transition* folder of the project. With the newly created *transition.xml* file selected and loaded into the editing panel, the following XML content is added in a transition set that enables the change bounds transition animation with a duration attribute setting:

```
<transitionSet
xmlns:android="http://schemas.android.com
/apk/res/android">
  <changeBounds
    android:duration="1000">
  </changeBounds>
</transitionSet>
```

Code 11 – Transition settings

With the transition file integrated into the project, any number of additional transitions may be added to the file without the need to make any further changes to the source of the activity. When specifying a transition, developers can use several predefined types defined by subclasses of `Transition`, such as `Fade` and `ChangeBounds`. If transition type is not specified a transition type, the system uses `AutoTransition` by default, which automatically fades, moves, and resizes views as necessary.

4. COMPARING KITKAT AND ANDROID JELLY BEAN

Important new things in Android KitKat, which have not been mentioned before, if compared to the previous version Jelly-Bean (API 16, 17, 18) are listed in the table 2 below:

Table 2 – Differences between Android Jelly Bean and KitKat (Allumalla, 2013)

Difference between Android Jelly Bean and Kitkat
Improved battery usage
Supports 4k resolution
Multi-sensory user experience
Supports tri-core CPU
New framework for UI transitions
Improved performance and security
Disabled access to battery statistics
Edge to Edge display provides a comfort for the user in the work
Low-power location monitoring
Audio tunneling and monitoring, loudness enhancer
More camera options: faster shooting, more accurate focusing, better balance etc.
Built-in infrared blaster support and a redesigned downloads app

The differences between these two versions at first sight may not seem to be much in the graphic design (search tab, notification bar, etc.). So for example, KitKat uses a thinner font of the text and the dominant color is white, instead of the previous Jelly Bean blue. Standard icons embedded applications are redesigned, while the Apps and Widgets tabs are not changed and the Play Store icon remains on the top-right part of the app drawer. If the call was made, and the number was not found in the user's phone book, Google will recognize it, do a quick search and if it finds in info site, it will display basic information along with the address of the site.

There is installed possibility of restricting related data transmission over the network; under the option "*Data usage*". It is necessary to include a *limit set* and determine the allowed size of the volume. Google Hangouts (all-in-one solution) is a new messaging system, which locates in one place all discussions of SMS and MMS. Additionally there is support for high-definition video calls, so it is not necessary reach out to install industry-leading applications as Skype, Viber or WhatsApp's.

"*Android KitKat has a step detector and step counter so that fitness-related apps can track when the user is walking, running, and climbing stairs. The step detector uses the accelerometer input to recognize when the user has taken a step. The step counter tracks the total number of steps since the device's last restart. Google is working with hardware manufacturer's work together to allow for collection and delivery of sensor data on Android device while allowing device's processor (CPU) to stay in low-power mode.*" (Chowdhry, 2013)

KitKat takes system performance to an all-time high by optimizing memory and improving touchscreen so that it responds faster and more accurately than ever before, this means that user can listen to music while browsing the web, or race down the highway with the latest hit game, all without a hitch (Rowinski, 2013). KitKat improves user access by providing support for system-wide closed captioning settings; the user defines which sections he want to be shown and how. With device that runs KitKat API and a Chromecast (thumb-sized media streaming device that plugs on HDTV into the HDMI port), users can enjoy in online entertainment (YouTube, Hulu-Plus, Google Play). Android supports closed captioning and subtitles, and music can be now listened longer, up to 60 hours of audio playback. In previous versions major problems were caused by working with fragments. In KitKat this was resolved with memory optimization, what helps Android to expand better and decrease crash rate.

However, even though there are many changes in the new version, the glimpse view in changes made in source code between API 18 and 19 reveal that the overall difference is 2.63%.

5. CONCLUSION

Android 4.4. KitKat API is 19th API release for Android platform that offers new features for users and app developers. Google doesn't deliver a huge list of transformative design and feature changes, but provides support that allows developers to develop applications easier. The main feature of the new mobile OS is adaptation to lower hardware capabilities with less working memory and lower screen resolution. Within this article, the attention is placed on possibilities for app development using new memory management, working with documents by using the new storage access framework, setting print options and content integration with Google cloud print and a new form of work with transmissions and scenes changes on the device screens.

The developers will find code examples and guidelines in using the most important benefits of presented KitKat API.

References:

1 Allumalla, A.R., 2013. Difference between Android Jellybean 4.3 and Kitkat 4.4 / [WWW Document]. Techzost. URL http://blog.techzost.com/4185/difference-android-jellybean-43-kitkat-44/2013/10 (accessed 5.1.14).

2 Android Developers, 2013. Android 4.4 APIs [WWW Document]. Android Developers. URL https://developer.android.com/about/versions/android-4.4.html#SMS (accessed 5.5.14).

3 Android Developers, 2014a. Dashboards | Android Developers [WWW Document]. Android Developers. URL http://developer.android.com/about/dashboards/index.html (accessed 4.29.14).

4 Android Developers, 2014b. Running Android with low RAM [WWW Document]. Android Developers. URL https://source.android.com/devices/low-ram.html (accessed 4.30.14).

5 Android Developers, 2014c. Storage Access Framework [WWW Document]. Android Developers. URL https://developer.android.com/guide/topics/providers/document-provider.html (accessed 5.5.14).

6 Chowdhry, A., 2013. 10 Of The Best Android KitKat Features [WWW Document]. Forbes. URL http://www.forbes.com/sites/amitchowdhry/2013/11/04/10-of-the-best-android-kitkat-features/2/ (accessed 5.12.14).

7 MobiThinking, 2014. Global mobile statistics 2014 Part A: handset market share [WWW Document]. URL http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/a#smartphone-shipments (accessed 5.8.14).

8 Rowinski, D., 2013. 10 Things Developers And Users Need To Know About Android KitKat 4.4 [WWW Document]. ReadWrite. URL http://readwrite.com/2013/11/07/android-kitkat-developers-users (accessed 5.1.14).

9 Smith, C., 2013. Android KitKat Development Tools - Procstats [WWW Document]. Double Encore. URL http://www.doubleencore.com/2013/11/android-kitkat-development-tools/ (accessed 5.12.14).

10 Wallat, J., 2013. Android 4.4 KitKat: here are the "hidden" functions - AndroidPIT [WWW Document]. AndroidPIT. URL http://www.androidpit.com/android-4-4-hidden-functions (accessed 5.1.14).

Information on authors:

Zoran Kos B.Sc

e-mail: zoran.kos@foi.hr

Zoran Kos is 2nd year student of Information and Programming engineering of Faculty of Organization and Informatics in Varaždin. Main interests are programming web, desktop and mobile application, new IT trends like cloud computing and mobility, making and presenting business models for startup projects. While developing new apps he focused on front-end development. He already has few smaller but successful projects behind him.

Zoran is the main contributor to this paper. He wrote the first version of the text and did coding of the presented examples.

Zlatko Stapić, PhD.

e-mail: zlatko.stapic@foi.hr

Faculty of Organization and Informatics

Pavlinska 2, 42000 Varaždin, Tel: +385 42 390 820, Fax: +385 42 213 413

Zlatko Stapić, PhD., works at the Information Systems Development Department at Faculty of Organization and Informatics in Varaždin. He obtained his PhD in computer sciences from University of Alcalá (Spain) and in information sciences from University of Zagreb (Croatia) in cotutelled doctorate program. His scientific and research interests include software- and mobile applications development methodologies. He participated in more than 15 scientific and professional projects and published more than 30 scientific and professional papers. Currently he leads the *Laboratory for Development and Transfer of Mobile Technologies* (FOI MT-Lab). Zlatko is putting a special focus in inclusion of students in his scientific and professional activities. The published papers, projects, awards and other relevant information can be found on his personal website: http://www.foi.unizg.hr/djelatnici/zlatko.stapic.

Zlatko's contributions to this paper are only of corrective nature. He mentored student Zoran Kos in defining the body, the structure and the style of the paper. After the first version of the paper is written, he proof-read the text and made necessary changes to make the text more focused and readable.

CASE d.o.o.

RAZVOJ POSLOVNIH
I INFORMATIČKIH SUSTAVA

CASE 26

Srebrni pokrovitelji

foi

INFODOM

Brončani pokrovitelji

citus grupa
Internet Development Company

.1 ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

VELEUČILIŠTE U RIJECI
COLLEGIUM FLUMINENSIS

02.06.-03.06.2014, Zagreb

ORGANIZATOR

CASE d.o.o.

ORGANIZACIJSKI I PROGRAMSKI ODBOR

TOMISLAV BRONZIN mag. ing. el.
ANTE POLONIJO
MISLAV POLONIJO
IVAN POGARČIĆ mr.sc.
ZLATKO SIROTIĆ univ.spec.inf.
ZLATKO STAPIĆ mag.inf.

Izdavač:
CASE d.o.o., Rijeka

Urednik:
Mislav Polonijo

Priprema za tisak:
CASE d.o.o., Rijeka

Tisak:
CASE d.o.o., Rijeka

ISSN 1334-448X
UDK 007.5 : 621.39 : 681.324

Case d.o.o., Šetalište XIII divizije 28, 51000 Rijeka
tel: 051/217-875, tel/fax: 051/218-043, e-mail: case@case.hr, internet: www.case.hr

S A D R Ž A J

CASEdev

CASEmobile