Android project

Android – My delicious bookmark

Get familiar with Android SDK and develop an innovative application to demonstrate the strength points of this new mobile platform.

Pham Tien Thanh (736554)

5/27/2009

Table of contents

I. Overview

1. What is android

Android is a software stack for mobile device that includes an operating system, middleware and applications. Android is powered by Linux kernel, initially developed by Google and later the Open Handset Alliance. It allows developers to write managed code in java language, controlling the device via Google developed java library.

Not like other famous rivals such as Microsoft window mobile or Symbian OS, android use developed java library because java is not just a programming language; it's a complete dynamic platform offers powerful support for embedded devices that must maintain some form of dynamic behavior. Moreover, java runtime environment can be integrated into almost any embedded device while java virtual machine includes interfaces that allow it to be readily integrated with RTOS and other native library. The RTOS supports multi-thread (scheduling), memory management, net working, and peripheral management for java VM.

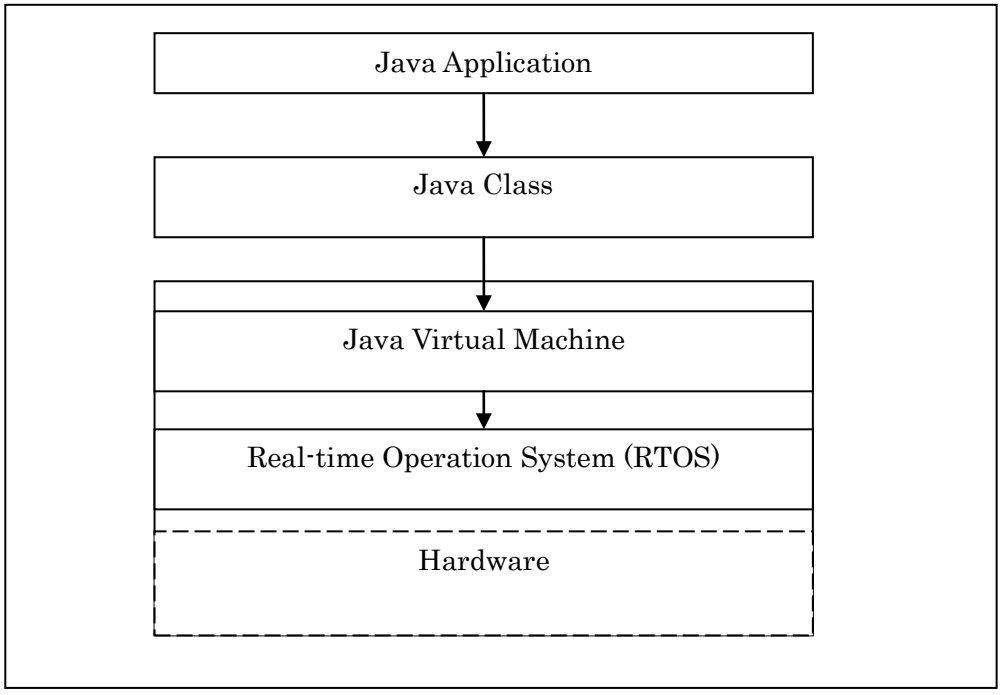


Figure 1: General architecture of java platform on embedded device

2. Android architecture:

The figure below shows the Android's general architecture:

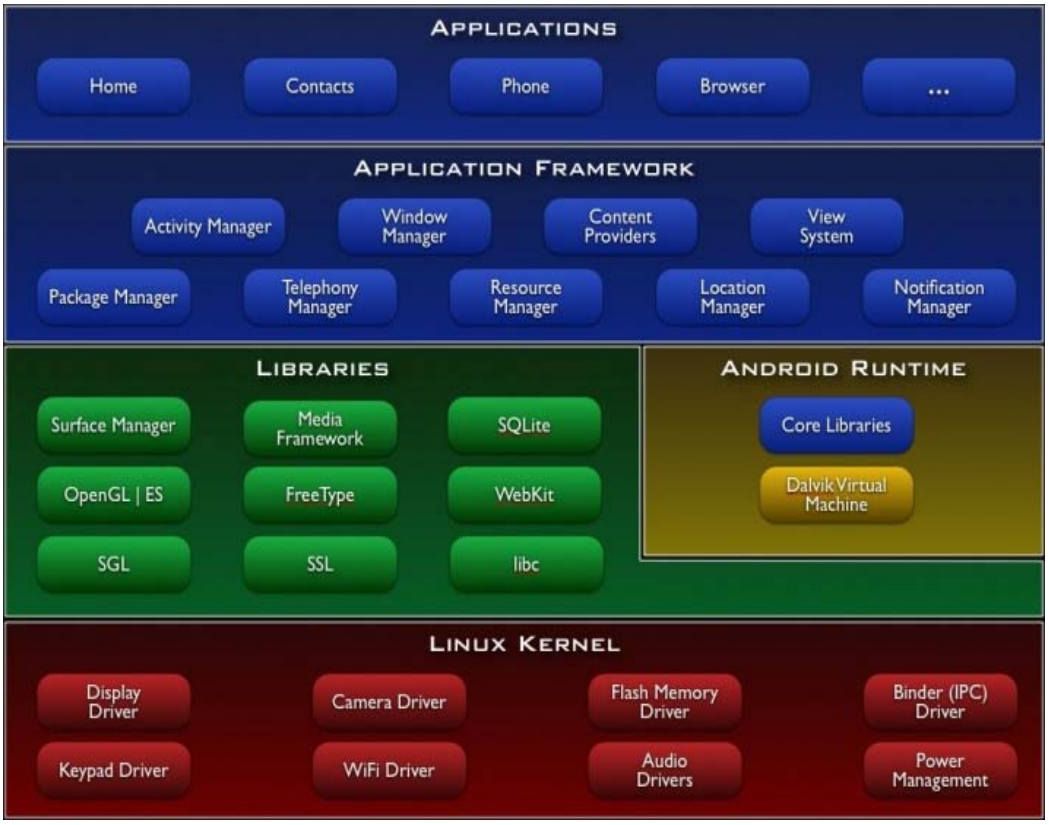


Figure 2: Android architecture

- **Application layer:** Android provides a lot of applications which come with its release including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are written using the Java programming language. Moreover, the number of developers who interested in developing Android's application is increasing and provides a huge market of application to chose.
- **An application framework layer** enabling reuse and replacement of components. Not like Window mobile which restricts developer from system API, developers have full access to the same framework APIs used by the core's applications. The strength points of Android is that the application architecture is designed for reusing of components which means any application can

publish its capabilities and any other application may the make use of those capabilities base on the definition of Intent which will be described later. Android provide a set of services and system, including:

- Views: is a flexible definition. It can be a list, a grid, text box, button or even an embedded web browser. View in Android is very different from the definition of "view" in Symbian OS or Window mobile which often means the container in which graphic components are organized and displayed to user.

- Content provider: store and retrieve data and make it accessible to all applications. They're the only way to share data across applications; there's no common storage area that all Android packages can access.

- Resource manager: Resources are external files (that is, non-code files such as image, icon, and string for internationalization) that are used by developer's code and compiled into their application at build time. Android supports a number of different kinds of resource files, including XML, PNG, and JPEG files. The XML files have very different formats depending on what they describe. Resources are externalized from source code, and XML files are compiled into a binary, fast loading format for efficiency reasons. Strings likewise are compressed into a more efficient storage form. All resource has its own id and will be added to special interface file R.java automatically:

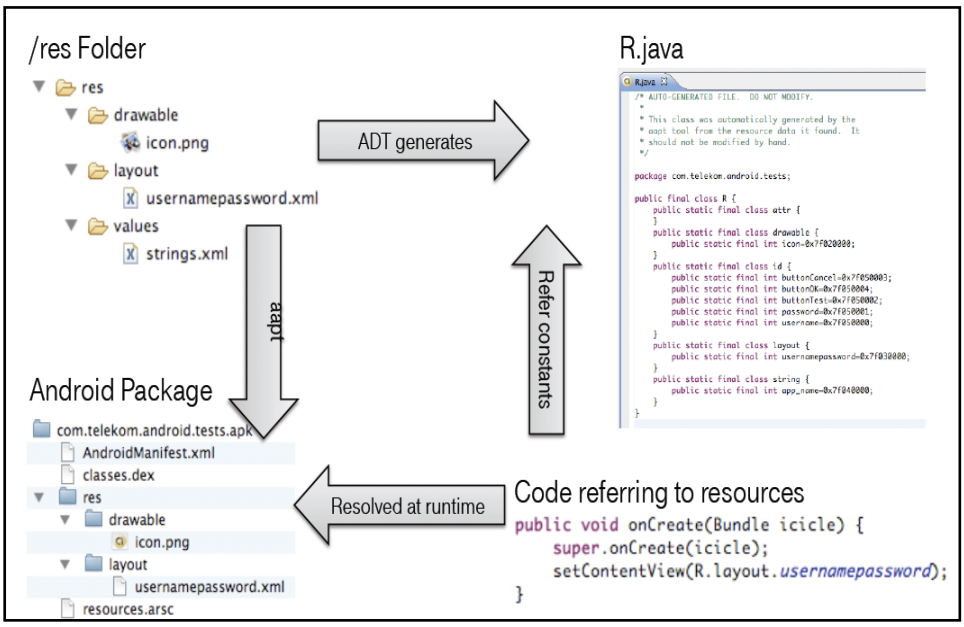


Figure 3: Resource management

- Notification manager that enables all application to display custom alerts in the status bar notify user of what happen in the back ground.
- Activity manager: manages the lifecycle of application and provide common navigation back stack. An activity focuses on what user can do by interact with user. Activity will for example create a window and place UI component to be displayed to user. There are two important methods which are implemented by most subclass is
    - ✧ OnCreate(Bundle) : where we initialize our activity, setup layout or get handle of each UI defined components.
    - ✧ OnPause(): where we deal with the event when user leaving our activity

➢ **Libraries:** They are all written in C/C++ internally, but you'll be calling them through Java interfaces. These capabilities are exposed to developers through the Android application framework. Some of core libraries are:

- ✧ **System C library** - a BSD-derived implementation of the standard C system library (libc), tuned for embedded Linux-based devices

- ✧ **Media Libraries** - based on PacketVideo's OpenCORE; the libraries support playback and recording of many popular audio and video formats, as well as static image files, including MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG
- ✧ **Surface Manager** - manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications
- ✧ **LibWebCore** - a modern web browser engine which powers both the Android browser and an embeddable web view
- ✧ **SGL** - the underlying 2D graphics engine
- ✧ **3D libraries** - an implementation based on OpenGL ES 1.0 APIs; the libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software rasterizer
- ✧ **FreeType** - bitmap and vector font rendering
- ✧ **SQLite** - a powerful and lightweight relational database engine available to all applications

- **Android Runtime:** A set of core libraries provides most the functionality available in the core library of java programming language. Android runtime includes the Dalvik Virtual Machine. Dalvik runs dex files, which are coverted at compile time from standard class and jar files. Dex files are more compact and efficient than class files, an important consideration for the limited memory and battery powered devices that Android targets. The core Java libraries are also part of the Android runtime. They are written in Java, as is everything above this layer. Here, Android provides a substantial set of the Java 5 Standard Edition packages, including Collections, I/O, and so forth.

- **Dalvik Virtual Machine** which is optimized for mobile devices. Dalvik is a major piece of Google's Android, runs Java platform applications which have been converted into a compact Dalvik Excutable format suitable for systems that are constrained in

terms of memory and processor speed. Unlike most virtual machines an true java virtual machine which are stack machines, Dalvik VM is a register based architecture. Like the CISC vs. RISC debate, the relative merits of these two approaches is a subject of continuous argument but the underlying technology sometimes blurs the ideological boundaries. Moreover, the relative advantages of the two approaches depend on the interpretation/compilation strategy chosen. Generally, however, stack based machines must use instructions to load data on the stack and manipulate that data and thus require more instructions than register machines to implement the same high level code. However, the instructions in a register machine must encode the source and destination registers and therefore tend to be larger. This difference is primarily of importance to VM interpreters for whom opcode dispatch tends to be expensive and other factors are relevant for JIT compilation. Being optimized for low memory requirements, Dalvik VM use less space, has no JIT compiler and uses its own byte code, not java byte code.

- **Linux Kernel:** Starting at the bottom is the Linux kernel. Android uses it for its device drivers, memory management, process management and networking. How ever we will never be programming to this layer directly. Android relies on Linux kernel version 2.6 for core system services. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.

One of the unique and powerful qualities of Android is that all applications have a level playing field which means that the applications Google writes have to go through the same public API that we use. We can even tell Android to make our application replace the standard applications.

II. Application structure

An android application is developed using Java language. After being compiled it will be packaged into Android Package (.apk) format which then can be deployed on to mobile device. Each Android application has its own process and Java virtual machine that mean application code runs in isolation from the

code of all other applications. Each application is assigned a unique Linux user ID. Permissions are set so that application's files are visible only to that user, to the application itself and there are other ways to export them to other application (through Content Provider).

1. Application Components

An Android application can make use of elements come from other application if it is permitted to do so. Because of this, system must be able to start an application process when any part of it is needed, and instantiate the java object for that part. There for an application here does not have a single entry point rather, they have essential components that system can instantiate and run as needed.

a. Activity: An activity work like "view" definition in window mobile which presents a visual interface to user. Each activity is independent, developer create his/her own activity by subclassing Activity base class. An application consists at least one activity which is presented to the user when application is launched. We can move from one activity to another by calling method startActivity() or startActivityForResult() from current activity. Activity work like a container and can make use of window which fill the screen or might be smaller than the screen and float on top of other windows like pop-up dialog. Developer often start with the call to method

```
@Override
public void onCreate(Bundle savedInstanceState){}
```

Where the visual content of views – objects are provided. View objects are derived from View class and each controls a particular retangular space within window. Developer can create View object from code or in much easier way using xml file which describes the hierachical organization of window. Parente views contain and organize the layout of their children. Views at leaf draw in their retangle and response to user actions directed at that space. A view hierachy is placed with in an activity's window by the Activity.setContentView() method. The code below show normal work done by Activity when startup:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  //Content from XML file
  setContentView(R.layout.login);
  //Get references to view object
  mEditUsername = (EditText) findViewById(R.id.edit_username);
  mEditPassword = (EditText) findViewById(R.id.edit_password);
  …
```

b. Services: A service does not have a visual user interface and runs in the background for indefinite period of time. A service play in role such are fetching data over the network or calculate something and provide the result the result to activities that need it. Each service extends the Service base class. Service run in the main thread of the application process, to avoid blocking other components or the user interface they often spawn another thread for time-consuming task. Android provide Handler class to handle message exchange between thread and not interfear other GUI thread.

c. Broadcast receiver: is a component that listening and react to broadcast announcements such as timezone has changed, battery is low, a coming call.etc..Application can also broadcasts an event to let other applications. This component do not display to user but they can start another event or use NotificationManager to alert the user. Developers can implement broadcast receiver by subclassing BroadcastReceiver base class.

d. Content provider: the onlyway to communicate between application by make a specific set of application's data available to others. These data is store in persistence sense by provider side (the one subclass ContentProvider base class). ContentProvider provide an interface with methods to store and retrieve data which then can be used by consumer. Consumers are subclasses of ContentResolver base class which takk to any content provider. ContentResolver cooperates with the provider to manage any interprocess communication that's involved.

Whenever there's a request that should be handled by a particular component, Android makes sure that the application process of the component is running, starting it if necessary, and that an appropriate instance of the component is available, creating the instance if necessary. A content provider is active only while it's responding to a request from a ContentResolver. And a broadcast receiver is active only while it's responding to a broadcast message. So there's no need to explicitly shut down these components. Activities, on the other hand, provide the user interface. They're in a long-running conversation with the user and may remain active, even when idle, as long as the conversation continues. Similarly, services may also remain running for a long time. So Android has methods to shut down activities and services in an orderly way:

+ An activity can be shut down by calling its finish() method. One activity can shut down another activity (one it started with startActivityForResult()) by calling finishActivity().

+ A service can be stopped by calling its stopSelf() method, or by calling Context.stopService().

```
mButtonLogin.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
                String username =
                        DeliciousActivityUtil.getText(mEditUsername);
                String password =
                        DeliciousActivityUtil.getText(mEditPassword);
                try {
                // Test connect by calling getLastUpdate
                MyDelicious myDelicious = new MyDelicious(username,
                                                        password);
                myDelicious.getLastUpdate();
                Intent data = new Intent();
                data.putExtra(DeliciousConstants.EXTRA_USERNAME,
                                username);
                data.putExtra(DeliciousConstants.EXTRA_PASSWORD,
                                password);
                setResult(RESULT_OK, data);
                //Terminate this activity
                finish();
                } catch (DeliciousExceptions exceptions) {}
```

Components might also be shut down by the system when they are no longer being used or when Android must reclaim memory for more active components.

2. Intent definition

Activities, services and broadcast receiver are activated by asynchronous messages which Android call “Intent”s. Intent is an object that holds the content of the message. There are some pre-defined Intent for system message like Intent.ACTION_INSERT, Intent.ACTION_SEND…For activities and services, it names the action being requested and specifies the URI of the data to act on, among other things. For broadcast receivers, the Intent object names the action being announced. There are separate methods for activiating each type of component:

- To activate an activity we by pass an Intent object to method startActivity() or startActivityForResult(). Intent can contain a lot of type of data: Boolean, String, Serializable Object..etc..in Extra part Activated activity can get back Intent message by using getIntent() method and extract sent data by using coresponding getExtra() method . Snip code below show some example which is used in project.
-

```
//New message
Intent intent = new Intent(DeliciousSearchView.this,
                                    DeliciousListView.class);
//Put message's data
intent.putExtra(DeliciousConstants.EXTRA_TIMEOUT_POST,
                timeOutCriteria);
intent.putExtra(DeliciousConstants.EXTRA_COUNT_POST,
                postCountCriteria);
intent.putExtra(DeliciousConstants.EXTRA_GETRECENT_POST,
                getRecentPost);
//Start new Activity by this intent
startActivity(intent);


.....
// In activated activity get back sent data
Intent data = this.getIntent();
post = (DeliciousPost)
data.getSerializableExtra(DeliciousConstants.EXTRA_POST_INFO);
```

One activity often starts the next one. If it expects a result back from the activity it's starting, it calls startActivityForResult() instead of startActivity(). The result is returned in an Intent object that's passed to the calling activity's onActivityResult() method.

- A service is started (or new instructions are given to an ongoing service) by passing an Intent object to Context.startService(). Android calls the service's onStart() method and passes it the Intent object. Similarly, an intent can be passed to Context.bindService() to establish

an ongoing connection between the calling component and a target service. The service receives the Intent object in an onBind() call. (If the service is not already running, bindService() can optionally start it.)

- An application can initiate a broadcast by passing an Intent object to methods like sendBroadCast(), sendOderedBroadCast(), and sendSticktBroadCast of Context class in any of their variations. Android delivers the intent to all interested broadcast receivers by calling their onReceive() methods.

3. The AndroidManifest file:

AndroidManifest file is an XML file declare application's components, naming any libraries the application needs to be linked and identifying any permission the application expected to be granted. Its name is fixed and is the same for every application. This can be consider the configuration file of application. The snip code below show part of XML file which is used by My Delicious project and we can see the declaration of activities with corresponding class, Intent filter which dedicate the component is going to receive the intent message..etc.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="del.icio.us" android:versionCode="1" android:versionName="1.0.0">
        <application android:icon="@drawable/icon" android:label="@string/app_name">
                <activity android:name=".delicious" android:label="@string/app_name">
                   <intent-filter>
                        <action android:name="android.intent.action.MAIN" />
                        <category android:name="android.intent.category.LAUNCHER" />
                    </intent-filter>
                </activity>
                <activity android:name="del.icio.us.activities.DeliciousLogin"
                        android:label="@string/title_login"
                        android:theme="@android:style/Theme.Dialog">
                </activity>
                <activity android:name="del.icio.us.activities.DeliciousListView"
                        android:label="@string/title_list_view">
                </activity>
                <activity android:name="del.icio.us.activities.DeliciousSearchView"
                        android:label="@string/title_search_view">
                </activity>
                <activity android:name="del.icio.us.activities.DeliciousPostDetails"
                        android:label="@string/title_post_details">
                </activity>
                ...
                </activity>
        </application>
        <uses-permission android:name="android.permission.INTERNET"></uses-permission>
</manifest>
```

4. Process and thead in Android

Android support multi-thread tasking allow developer to spawn additional threads for any process.

a. **Process:** The process where a component runs is controlled by the manifest file. The component elements — <activity>, <service>, <receiver>, and <provider> — each have a process attribute that can specify a process where that component should run. These attributes can be set so that each component runs in its own process, or so that some components share a process while others do not. They can also be set so that components of different applications run in the same process — provided that the applications share the same Linux user ID and are signed by the same authorities. The <application> element also has a process attribute, for setting a default value that applies to all components.

   All components are instantiated in the main thread of the specified process, and system calls to the component are dispatched from that thread. Separate threads are not created for each instance. Consequently, methods that respond to those calls — methods that report user actions and the lifecycle notifications always run in the main thread of the process. This means that no component should perform long or blocking operations (such as networking operations or computation loops) when called by the system, since this will block any other components also in the process. You can spawn separate threads for long operations.

   Android may decide to shut down a process at some point, when memory is low and required by other processes that are more immediately serving the user. Application components running in the process are consequently destroyed. A process is restarted for those components when there's again work for them to do. When deciding which processes to terminate, Android weighs their relative importance to the user. For example, it more readily shuts down a process with activities that are no longer visible on screen than a process with visible activities. The decision whether to terminate a process, therefore, depends on the state of the components running in that process.

b. **Thread:** There will be somtime developers need to spawn a thread to do some background work. Thread are created in code using standard Java Thread object. Android provides a number of convenience classes for managing threads like Looper for running a message loop within a thread, Handler for processing messages, and HandlerThread for setting up a

thread with a message loop. For example in My Delicious project, many time we need to perform network operation that takes long time to receive data. Solution is to perform that kind of work in a different thread while displaying a progress bar to user. Progress bar will be dimissed when sub-thread complete:

```
//Define a handler for sub-thread
private Handler handler = new Handler() {
        @Override
          public void handleMessage(Message msg) {
                progressDialog.dismiss();
          }
};

...

progressDialog = ProgressDialog.show(this, "Connecting..",
                Getting Information", true, false);
//Start sub-thread work
Thread thread = new Thread(this);
thread.start();

....
@Override
public void run() {
        //Heavy workload
        list = myDelicious.getRecentPosts(username, password, count,
                                        tagCriteria, timeout);
        //Notify sub-thread has complete its job
        handler.sendEmptyMessage(0);
}
```

5. Activity lifecycle

Application components have a lifecycle — a beginning when Android instantiates them to respond to intents through to an end when the instances are

destroyed. In between, they may sometimes be active or inactive,or, in the case of activities, visible to the user or invisible. Because Activity will be used most oftenly here we only discusses about the lifecycle of activities including the states that they can be in during their lifetimes, the methods that notify you of transitions between states, and the effect of those states on the possibility that the process hosting them might be terminated and the instances destroyed.

An activity has essentially three states:

- It is *active* or *running* when it is in the foreground of the screen (at the top of the activity stack for the current task). This is the activity that is the focus for the user's actions.
- It is *paused* if it has lost focus but is still visible to the user. A paused activity is completely alive (it maintains all state and member information and remains attached to the window manager), but can be killed by the system in extreme low memory situations.
- It is *stopped* if it is completely obscured by another activity. It still retains all state and member information. However, it is no longer visible to the user so its window is hidden and it will often be killed by the system when memory is needed elsewhere.

If an activity is paused or stopped, the system can drop it from memory either by asking it to finish , or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state. As an activity transitions from state to state, it is notified of the change by calls to the following protected methods:

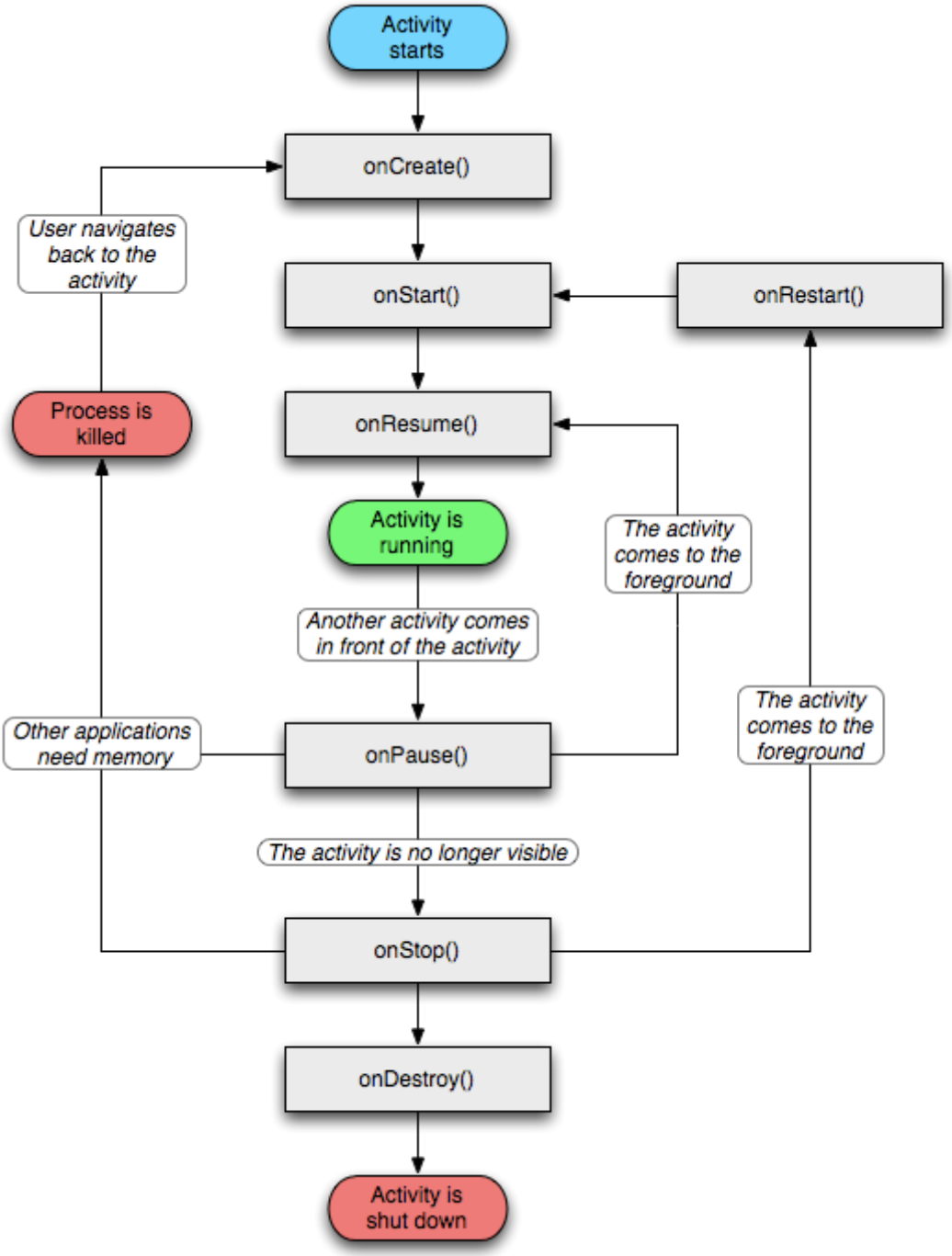
```
+void onCreate(Bundle savedInstanceState)
+void onStart()
+void onRestart()
+void onResume()
+void onPause()
+void onStop()
+void onDestroy()
```

All activities must implement onCreate() to do the initial setup when the object is first instantiated. Many will also implement onPause() to commit data changes and otherwise prepare to stop interacting with the user.

Taken together, these seven methods define the entire lifecycle of an activity. There are three nested loops that you can monitor by implementing them:

- The **entire lifetime** : An activity does all its initial setup of "global" state in onCreate(), and releases all remaining resources in onDestroy().
- The **visible lifetime** : During this time, the user can see the activity on-screen, though it may not be in the foreground and interacting with the user. Between these two methods, you can maintain resources that are needed to show the activity to the user. The onStart() and onStop() methods can be called multiple times, as the activity alternates between being visible and hidden to the user.
- The **foreground lifetime** During this time, the activity is in front of all other activities on screen and is interacting with the user. An activity can frequently transition between the resumed and paused states.

The following diagram illustrates these loops and the paths an activity may take between states. The colored ovals are major states the activity can be in. The square rectangles represent the callback methods you can implement to perform operations when the activity transitions between states.



When the system, rather than the user, shuts down an activity to conserve memory, the user may expect to return to the activity and find it in its previous state. To capture that state before the activity is killed, you can implement an onSaveInstanceState() method for the activity. Android calls this method before making the activity vulnerable to being destroyed — that is, before onPause() is

called. It passes the method a Bundle object where you can record the dynamic state of the activity as name-value pairs. When the activity is again started, the Bundle is passed both to onCreate() and to a method that's called after onStart(), onRestoreInstanceState(), so that either or both of them can recreate the captured state.

III. "My Delicious" project

1. Delicious Bookmark web service

Delicious or formerly call del.icio.us is a social bookmarking web service for store, sharing and discovery web bookmarks. Delicious allows user to tag each of their book marks with freely chosen index terms. All bookmarks posted to Delicious are publicly viewable by default although users can mark specific bookmarks as private and imported bookmarks are private by default.

Delicious web service allows client to read/write to their bookmarks and tags via HTTP-based interface. All delicious APIs are done over https and require basic authentication. These APIs is still under development and below is display the list of APIs which are currently used :

Method	Meaning
Post/update	Check to see when a user last posted an item.
Posts/add	Add a new bookmark
Posts/get	Get bookmark for a single date, or fetch specific items
Posts/dates	List dates on which bookmarks were posted
Posts/recent	Fetch recent bookmarks
Posts/all	Fetch all bookmarks by date or index range.
Posts/all?hashes	Fetch a change detection manifest of all items
Posts/suggest	Fetch popular, recommended and net work tags for a specific url
Tags/get	Fetch all tags
Tags/delete	Delete a tag from all posts
Tags/rename	Rename a tag on all posts

Tags/bundle/all	Fetch tag bundles
Tags/bundles/set	Assign a set of tags to a bundle
Tags/bundles/delete	Delete a tag bundle

Each API's method structure details can be reference at official site of Delicious https://api.del.icio.us/

2. Project requirement

Project goal is to develop an innovative application on Android SDK, I decided to work on HttpClient library default goes with Android SDK which make use of Del.icio.us web service's interface allow user to work on his data:

- ✓ Bookmark a website
- ✓ Search bookmark base on some criteria (url, date, tag) or recently bookmarked site.
- ✓ Get all bookmarked site
- ✓ Edit specific bookmark

"My Delicious" application will provide a friendly and easy way to Android's users to control their Delicious bookmarks without using a web browser. By this project I tried to demonstrate as much as possible strength points which are provided by Android.

3. Project implementation

a. Apache HttpClient library

Http protocol seem to be the most significant protocol used on internet today. Web services, network-enabled appiances and the growth of computing continue to expand the role of HTTP protocol beyond user-driven web browsers while increasing the number of applications that require HTT supports. My Delicious will be such a kind of application be cause it needs to work with webservice provided by Del.icio.us. Apache HttpClient library is the best choice because by default it goes with Android SDK. Although java.net package provides basic funtionalities for accessing resources via HTTP, it does not provide the full flexibility or functionality needed by many Application

HttpClient is not another browser but primary responsibility is the HTTP protocol excuted directly or through an HTTP proxy to provide interface and default implementation for cookie and password management but not for persisting such data. HttpClient do not provide user interface like browser but work like a core; rest functinalities of browser is our program responsibility

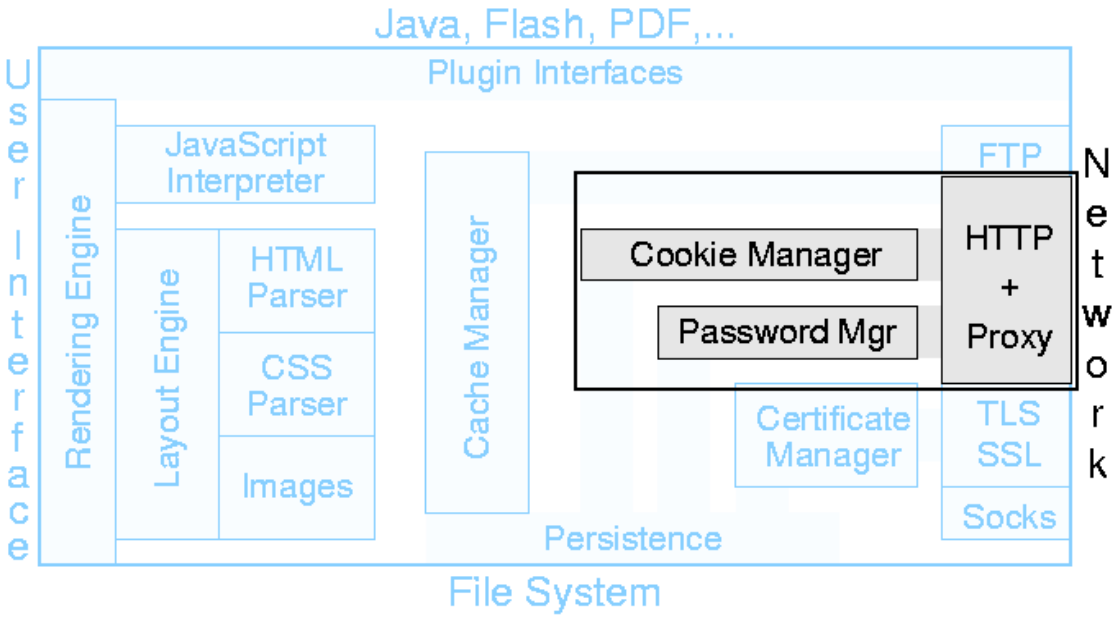


Figure 4: HttpClient scope in relation with other browser's component

To connect to a web service through a secured Http connection, HttpClient instance needs to be configured and provided necessary connection parameters (such as authentications's username and password...). The snipcode below show step by step to configure an HttpClient instance to connect to Delicious web service :

```
SchemeRegistry supportedSchemes = new SchemeRegistry();
// Register the "http" and "https" protocol schemes, they are
// required by the default operator to look up socket factories.
    supportedSchemes.register(new Scheme("http", PlainSocketFactory
                                    .getSocketFactory(), 80));
    supportedSchemes.register(new Scheme("https", SSLSocketFactory
                                    .getSocketFactory(), 443));
// Prepare parameters
    HttpParams params = new BasicHttpParams();
    HttpProtocolParams.setVersion(params, HttpVersion.HTTP_1_1);
                HttpProtocolParams.setContentCharset(params, "UTF-8");
                HttpProtocolParams.setUseExpectContinue(params, true);
                HttpProtocolParams.setUserAgent(params,
                                    DeliciousConstants.USER_AGENT_VALUE);
    ClientConnectionManager ccm = new ThreadSafeClientConnManager(params,
                                    supportedSchemes);
    httpClient = new DefaultHttpClient(ccm, params);
//Auto Proxy setting by JRE
    ProxySelectorRoutePlanner routePlanner = new ProxySelectorRoutePlanner(
    httpClient.getConnectionManager().getSchemeRegistry(),
    ProxySelector.getDefault());
    httpClient.setRoutePlanner(routePlanner);
// Retry handler
    HttpRequestRetryHandler defaultHttpMethodRetryHandler = new
    DefaultHttpRequestRetryHandler(0, false);
    httpClient.setHttpRequestRetryHandler(defaultHttpMethodRetryHandler);
// Delicious username and password
    httpClient.getCredentialsProvider().setCredentials(SCOPE,
                new UsernamePasswordCredentials(username, password));
```

b. My Delicious core

As described above, del.icio.us webservice expose a set of APIs which allow user to work on their own bookmark data. These APIs are invoked simple through secured HTTP protocol with basic authentication. For example, by entering the url below to web browser we invoke an API to

retreive all posted bookmark of a contact :

```
https://api.del.icio.us/v1/posts/all
```

The result will be in XML format like below :

```
<posts user="thanhpt_25" update="2009-05-26T08:15:33Z" tag="" total="4">
<post href=http://www.youtube.com/
    hash="dfac2ac2f8102bbfbd4ef18a247d74cb" description="Youtube"
    tag="video" time="2009-05-26T08:15:33Z" extended="My favourite"
    meta="077bb569a39b303b94344117ffa3cdbf"/>
</posts>
-
<!--
 fe06.api.del.ac4.yahoo.net uncompressed/chunked Sat May 30 08:55:04 PDT
2009
-->
```

My Delicious's main engine is to compose correctly and send a HttpRequest to webservice, receive the response from server and parsing this XML data to display human friendly data to user. Formally, base on request from user, application will compose an URI like above to invoke corresponding API and also pass necessary parameters on this URI. By executing HttpClient.get() method, application connect to web service based on configured parameters, invoke expected API and receive data resonse from server. Status of request will be checked to see if request has been successfully performed or not. Structured XML data then will be parsed by the XML parser DocumentBuilder. The snipcode below show corresponding process of above description :

```
//New a Get request base on URI
HttpGet get = new HttpGet(String.valueOf(uri));
try {
//Get connection and retrieve data
  httpResponse = httpClient.execute(get);
  if (httpResponse != null) {
//process returned status from server
  StatusLine status = httpResponse.getStatusLine();
  int statusCode = status.getStatusCode();
  checkNotAuthorized(statusCode);
     if (statusCode != HttpStatus.SC_OK) {
        throw new DeliciousExceptions("Http response with status code: " +
                                        statusCode);
     }
  }
//process returned status from server
  HttpEntity entity = httpResponse.getEntity();
  InputStream inputStream = entity.getContent();
  if (inputStream != null) {
//Process XML structured data
     BufferedReader bufferedReader = new BufferedReader(
     new InputStreamReader(inputStream, DeliciousUtils.UTF_8));
     String input;
     while ((input = bufferedReader.readLine()) != null) {
        result.append(input).append(DeliciousUtils.LINE_SEPARATOR);
     }
     entity.consumeContent();
     Document document = documentBuilder.parse(new InputSource(
     new StringReader(result.toString())));
     NodeList postsTag = document
        .getElementsByTagName(DeliciousConstants.POSTS_TAG);
     if (postsTag != null && postsTag.getLength() > 0) {
     Node postsItem = postsTag.item(0);
     String updateTime = postsItem.getAttributes().getNamedItem(
        DeliciousConstants.UPDATE_ATTRIBUTE).getNodeValue();
     resultMetaInformation = DeliciousUtils.getDateFromUTCString(updateTime);
     }
```

All others method of My Delicious's core are implemented in the same maner. For more information please refer to the code.

c. Graphical user interface

- **Main screen :** Main screen allows user to choose application's functionality. This screen is implemented as an Activity. If refer to AndroidManifest.xml file, we can find the definition which point out that class delicious will be the entrance point of this application:

```
<activity android:name=".delicious"
android:label="@string/app_name">
        <intent-filter>
           <action android:name="android.intent.action.MAIN" />
           <category
            android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
</activity>
```

In the implementation, when user click on button , a Intent message will be send to activate corresponding Activity associate with it. For example, activity which display login dialog will be invoke when user click on buton "Login":

```
/* When they click on the login button show the login screen*/
 Button button = (Button) findViewById(R.id.button_login);
 button.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
        Intent intent = new Intent(delicious.this,
                                   DeliciousLogin.class);
        startActivityForResult(intent, VIEW_LOGIN_ID);
        }
   });
```

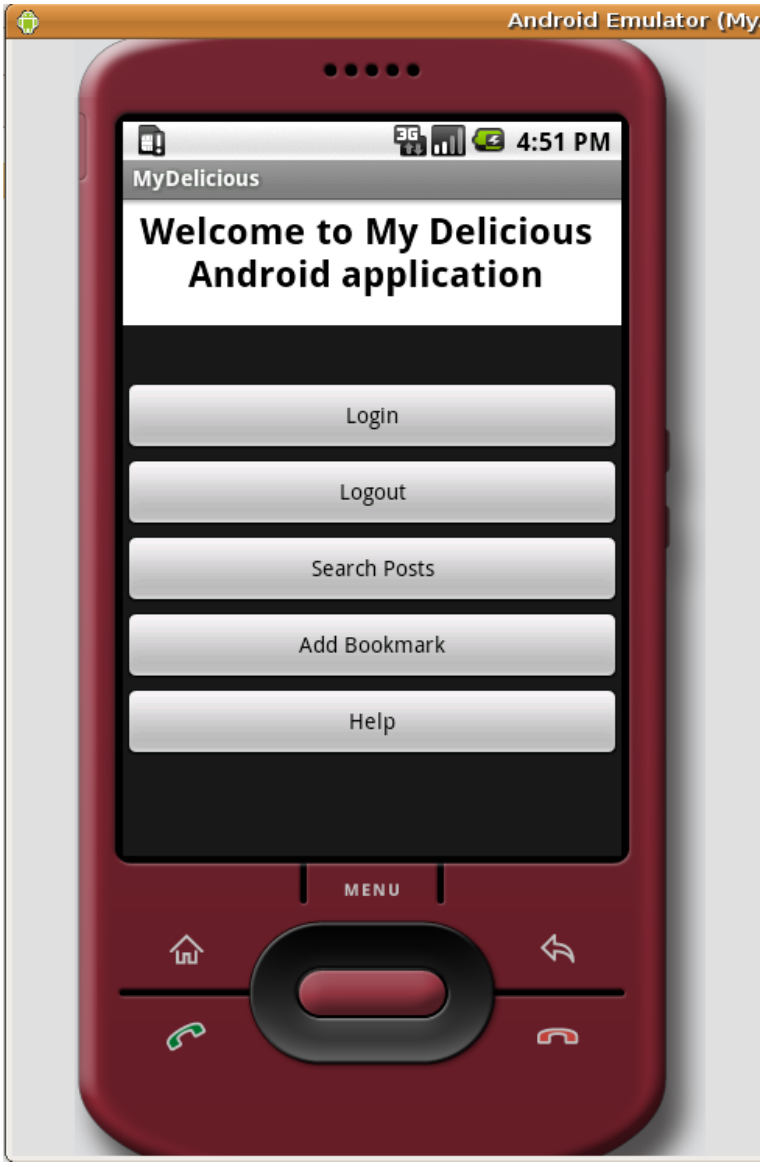


Figure 5: My Delicious's main screen

- **Login screen:**

  Login screen allow user to enter credential data (username, password) to use del.icio.us web service. These user name and password must be registered their website because there is no API to do this kind of stuff. Here we demonstrate the ability of Android when it comes to display GUI components. A Dialog will be defined very much alike to other activity but with different android predefined "**Theme**"

```
<activity android:name="del.icio.us.activities.DeliciousLogin"
        android:label="@string/title_login"
        android:theme="@android:style/Theme.Dialog">
</activity>
```

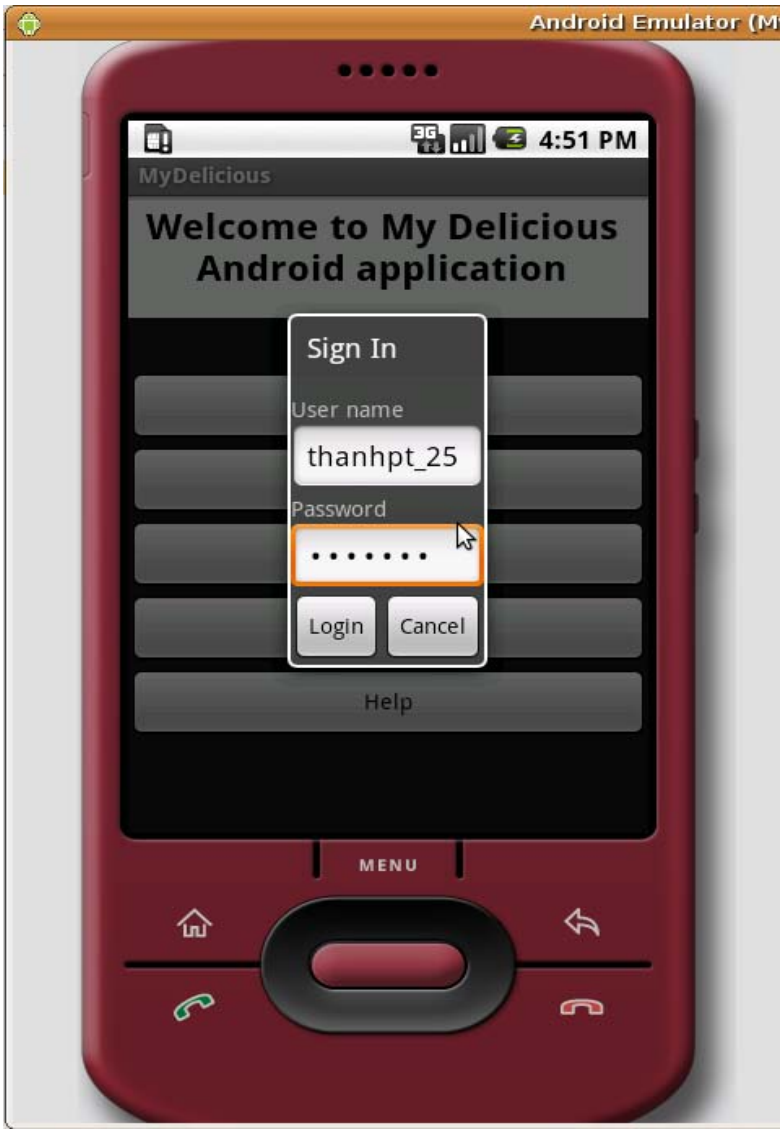


Figure 6: My delicious's Login screen

Del.iciou.us also do not have an API to do authentication but instead authentication information is sent with each invocation of API using Http's basic authentication rather than session maintainence. To confirm that the input information is correct, this application will perform a trick, it will perform a light load task such as getting last update time. If user name and password is correct then it will be stored

and used in next transaction otherwise user will be notify about unauthorized access.

```
mButtonLogin.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
          String username =
                DeliciousActivityUtil.getText(mEditUsername);
          String password =
                DeliciousActivityUtil.getText(mEditPassword);
          try {
            // Test connect by calling getLastUpdate
            MyDelicious myDelicious = new MyDelicious(username,
                                                      password);
            myDelicious.getLastUpdate();
             Intent data = new Intent();
             data.putExtra(DeliciousConstants.EXTRA_USERNAME,
                           username);
             data.putExtra(DeliciousConstants.EXTRA_PASSWORD,
                           password);
             setResult(RESULT_OK, data);
             finish();
          } catch (DeliciousExceptions exceptions) {
             Intent data = new Intent();
             data.putExtra(DeliciousConstants.EXTRA_VIEW_DATA,
                                                   exceptions);
             setResult(DeliciousConstants.RESULT_LOGIN_FAILED,
                       data);
            finish();
          }
        }
});
```

- **Search screen:**

Search screen allow user to search on variety of criteria:

- ✓ By tag
- ✓ By URL
- ✓ By Date
- ✓ Get all posted bookmark or get recently posted bookmark.

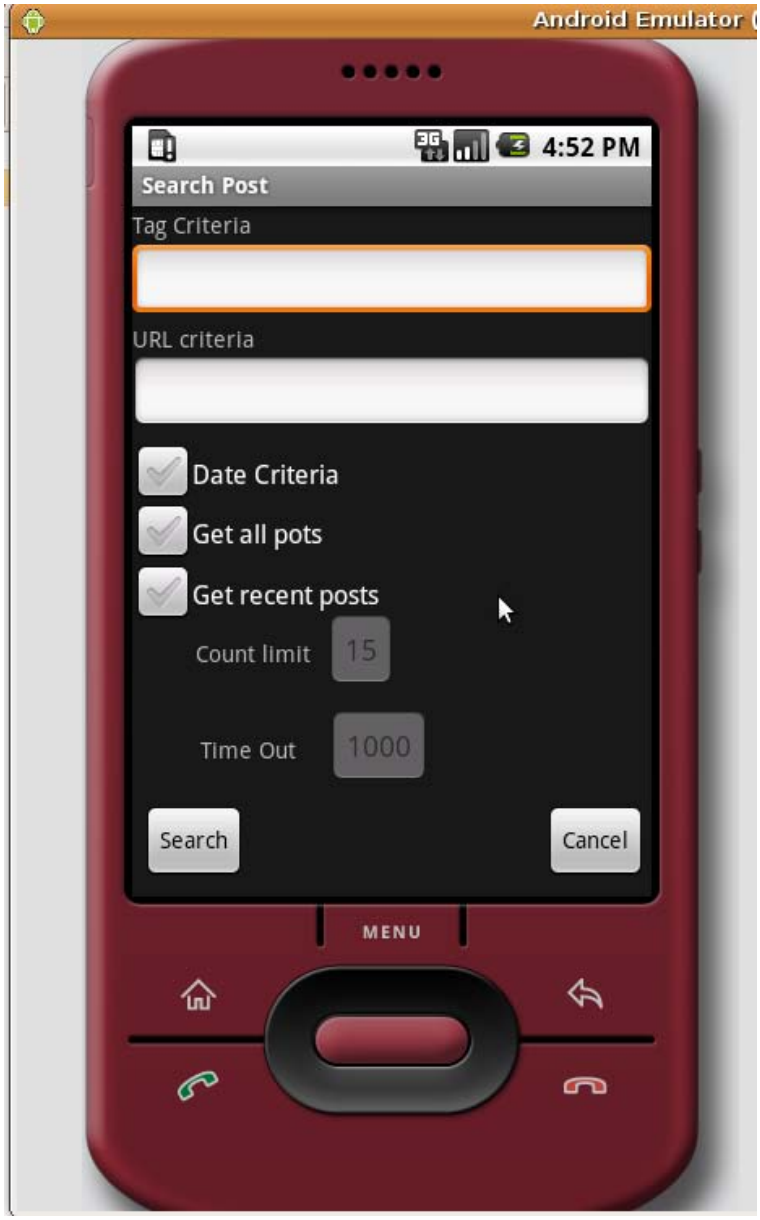


Figure 7 : My Delicious's search screen

By this activity, we demonstrated the use of user interface component, and how to use Layout to organize them to display to user.

We also show example of how to use defined dialog designed by Androids. The code below show how to display a DatePickerDialog:

```
private DatePickerDialog.OnDateSetListener
    dateSetListener = new
          DatePickerDialog.OnDateSetListener() {
              public void onDateSet(DatePicker view,
                            int year, int monthOfYear,
                     int dayOfMonth) {
                  mYear = year;
                  mMonth = monthOfYear;
                  mDay = dayOfMonth;
                  GregorianCalendar date = new
                                GregorianCalendar();
                  date.set(mYear, mMonth, mDay);
                  dateSelect.setText(date.getTime().
                                        toString());
              }
       };
@Override
protected Dialog onCreateDialog(int id) {
       switch (id) {
           case DATE_DIALOG_ID:
           return new DatePickerDialog(this,
                  dateSetListener, mYear, mMonth,
                                  mDay);
       }
       return null;
}
...
//Call show dialog in our program
showDialog(DATE_DIALOG_ID);
```

Figure 8: Set Date criteria dialog

Network on mobile devices are heavy load task we should use progress bar to tell user that application is not hang but in processing state. This situation force us to use multi-thread one for performing network task and other for displaying so that none of them interfered each other but they should communicate with each other to synchronize the process.

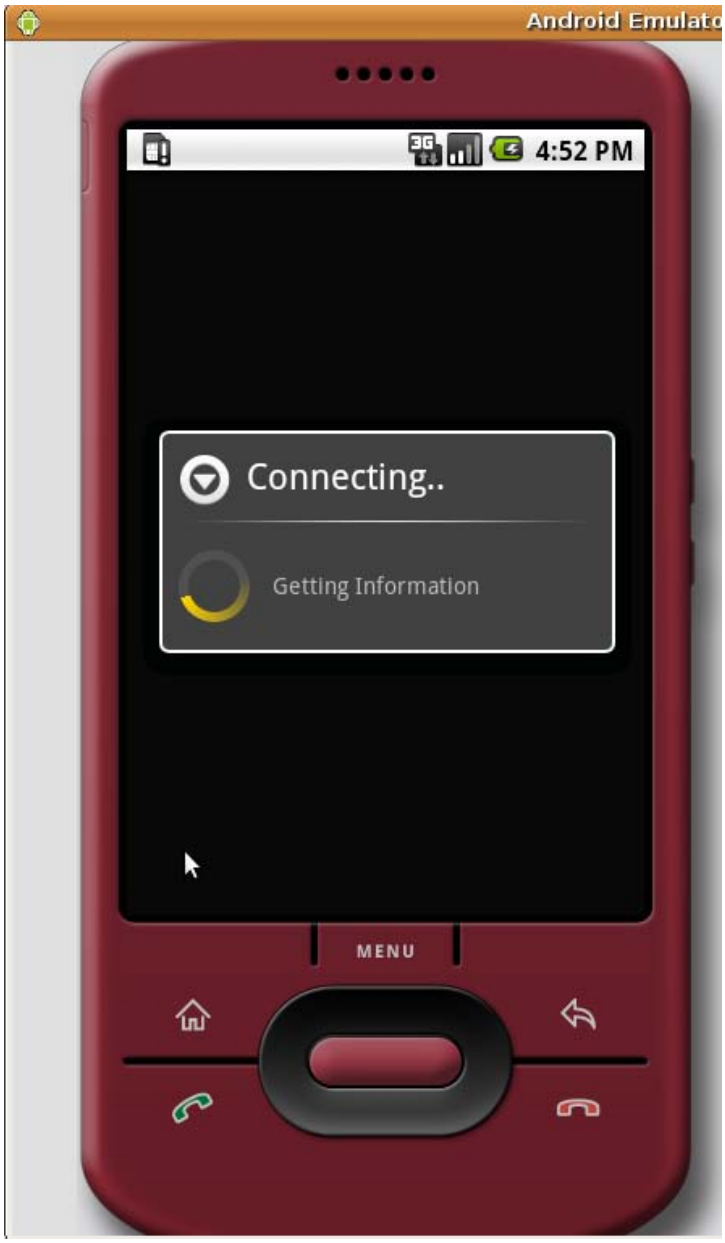


Figure 9: Progress bar is displayed while retrieving information

The code below shows how to use Handler and Thread class to do this stuff.

```
//Define a system handler instance which tell GUI thread that working thread
//Has already finished.
private Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
                progressDialog.dismiss();
                if ((list != null) && (list.size() != 0))
                setContentView(createList(DeliciousListView.this));
                else {
                   new AlertDialog.Builder(DeliciousListView.this)
                        .setTitle("Result is empty")
                        .setMessage(
                        "Search result is empty. Please try other criteria
                        or use [Search All]
                option¥n").setNeutralButton("Ok",
                        new DialogInterface.OnClickListener() {
                         public void onClick(DialogInterface dialog,
                                int whichButton) {
                        }
                }).show();
             }
        }
};
...
//Display progress bar and start working thread
progressDialog = ProgressDialog.show(this, "Connecting..",
                        "Getting Information", true, false);
Thread thread = new Thread(this);
thread.start();
...
//Heavy load work stuff
@Override
public void run() {
        list = myDelicious.getAllPosts();
//Send synchronize message
        handler.sendEmptyMessage(0);
}
```

- **Search result screen :**

In search result screen, we tried to create a custom ListView object to demonstrate the flexibility of Android in allowing developer to extend default class. Here we can create and organize GUI components dynamically through code which means do not use XML resource layout xml file:

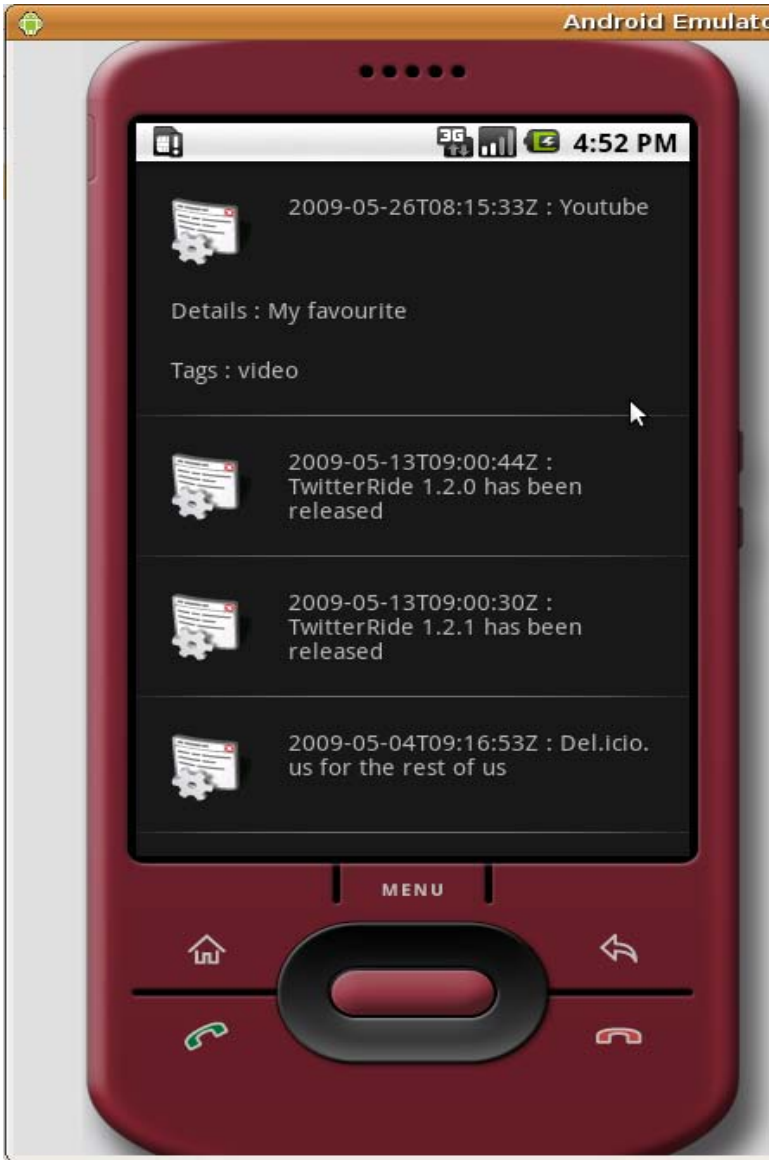


Figure 10: My Delicious search result screen

For example, we can find the following code which defines the each components and its configuration in search result screen:

```
private void createUI() {
        setColumnShrinkable(1, true);
        setColumnStretchable(1, true);
        setPadding(10, 10, 10, 10);
        TableRow row = new TableRow(_context);
        LayoutUtils.Layout.WidthFill_HeightWrap
                        .applyTableLayoutParams(row);
        /*Configure first line of row */
        lblName = new TextView(_context);
        LayoutUtils.Layout.WidthWrap_HeightWrap
                                .applyTableRowParams(lblName);
        lblName.setPadding(10, 10, 10, 10);
        lblIcon = AppUtils.createImageView(_context, -1, -1, -1);
        LayoutUtils.Layout.WidthWrap_HeightWrap
                                .applyTableRowParams(lblIcon);
        lblIcon.setPadding(10, 10, 10, 10);
        row.addView(lblIcon);
        row.addView(lblName);
        /*Configure second line of row */
        lblDescription = new TextView(_context);
        LayoutUtils.Layout.WidthFill_HeightWrap
                        .applyTableLayoutParams(lblDescription);
        lblDescription.setPadding(10, 10, 10, 10);
        lblTag = new TextView(_context);
        LayoutUtils.Layout.WidthFill_HeightWrap
                        .applyTableLayoutParams(lblDescription);
        lblTag.setPadding(10, 10, 10, 10);
        /*Add the rows to the table */
        addView(row);
        addView(lblDescription);
        addView(lblTag);
}
```

- **Edit bookmark screen:** By click on specific bookmark on result screen, user are moved to edit bookmark screen where they can change their own information or go to their bookmark URL (automatically open a new browser to display).

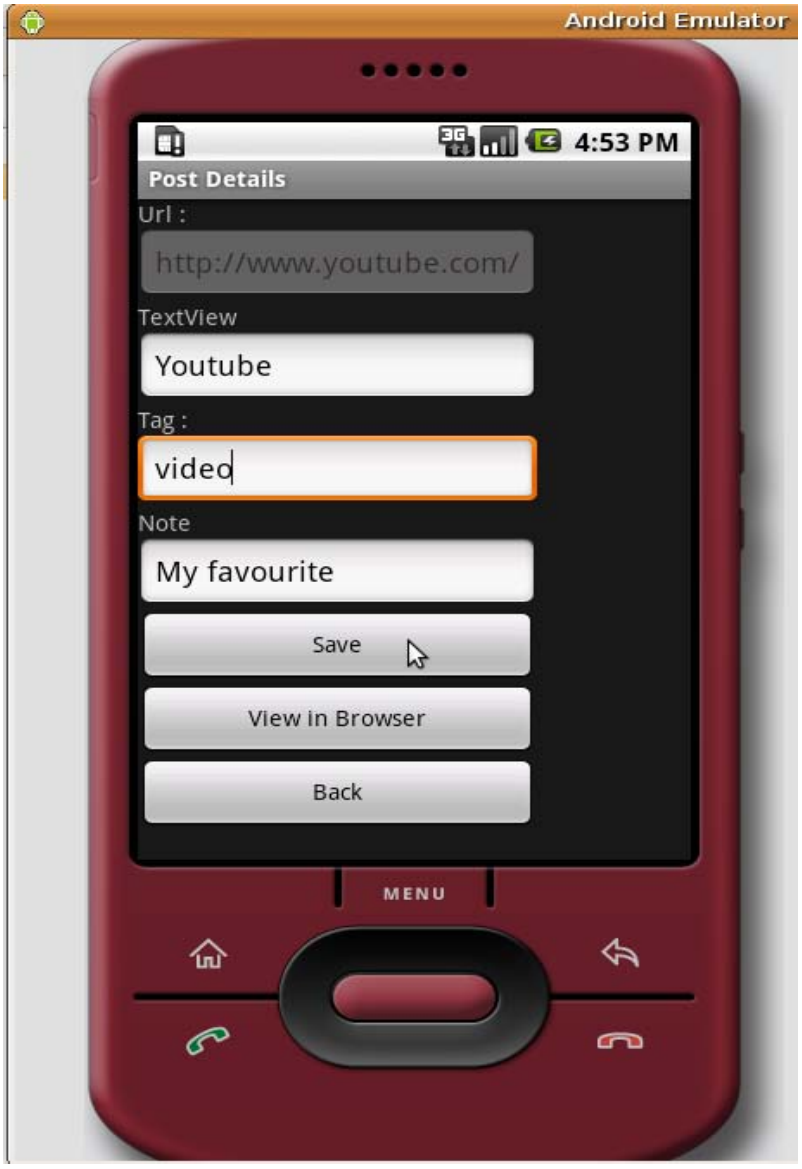


Figure 11: My Delicious's edit bookmark screen

To view bookmarked URL in default browser we send a pre-defined intent Intent.ACTION_VIEW. This cause browser to be opened and displays specific URL that we passed to it:

```
btnLink = (Button) findViewById(R.id.btnLink);
btnLink.setOnClickListener(new OnClickListener() {
       @Override
       public void onClick(View v) {
         Intent intent = new Intent(Intent.ACTION_VIEW,
                                    Uri.parse(post
                                          .getHref()));
           startActivityForResult(intent, 4);
       }
});
```

- **Add bookmark screen:** In this screen user is required to enter an URL and its information. My Delicious also bases on input parameter URL to give suggestion about tag should be use or common tags which are used by other users for this address. Here we use some most common used GUI component such as: auto complete text box with content handler, progress bar, check box …etc.

  In fact, add book mark screen also support user to bookmark their favorite site while they are surfing the net by using "Share Page" menu in "More" web browser's menu. By doing this and choose to share page with My Delicious application, an Intent with data is the URL which is currently being display will be send to My Delicious to be bookmark. But we have the "Knowing Issue" that by opening the browser will cause obstacle to HttpClient which then cause HttpClient cannot connect to server. Because of this, this functionality is in "Waiting" state for the patch from Android SDK provider. The implemented code show as below:

```
if (savedInstanceState == null) {
        Intent intent = getIntent();
        String action = intent.getAction();
        Uri data = intent.getData();
        String type = intent.getType();
        if (Intent.ACTION_INSERT.equals(action) &&
                Browser.BOOKMARKS_URI.equals(data)) {
                String url = intent.getStringExtra("url");
                 String title = intent.getStringExtra("title");
...
```

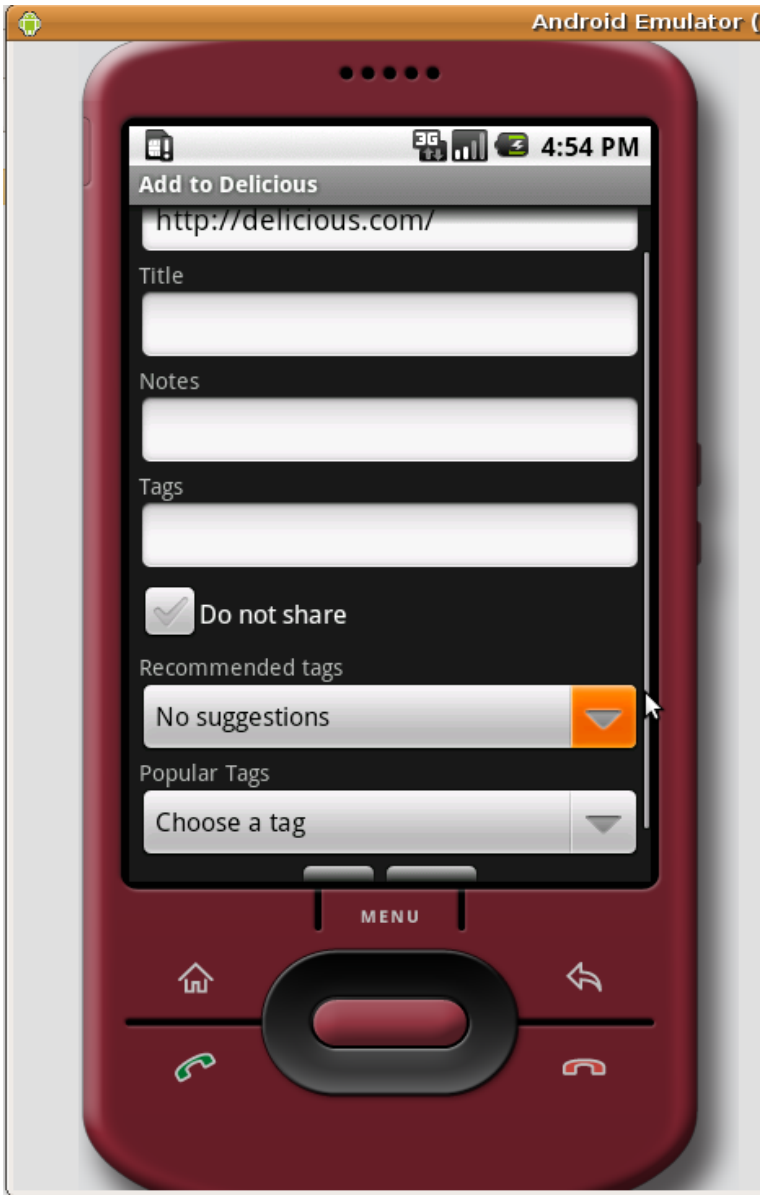


Figure 12: My delicious's add bookmark screen

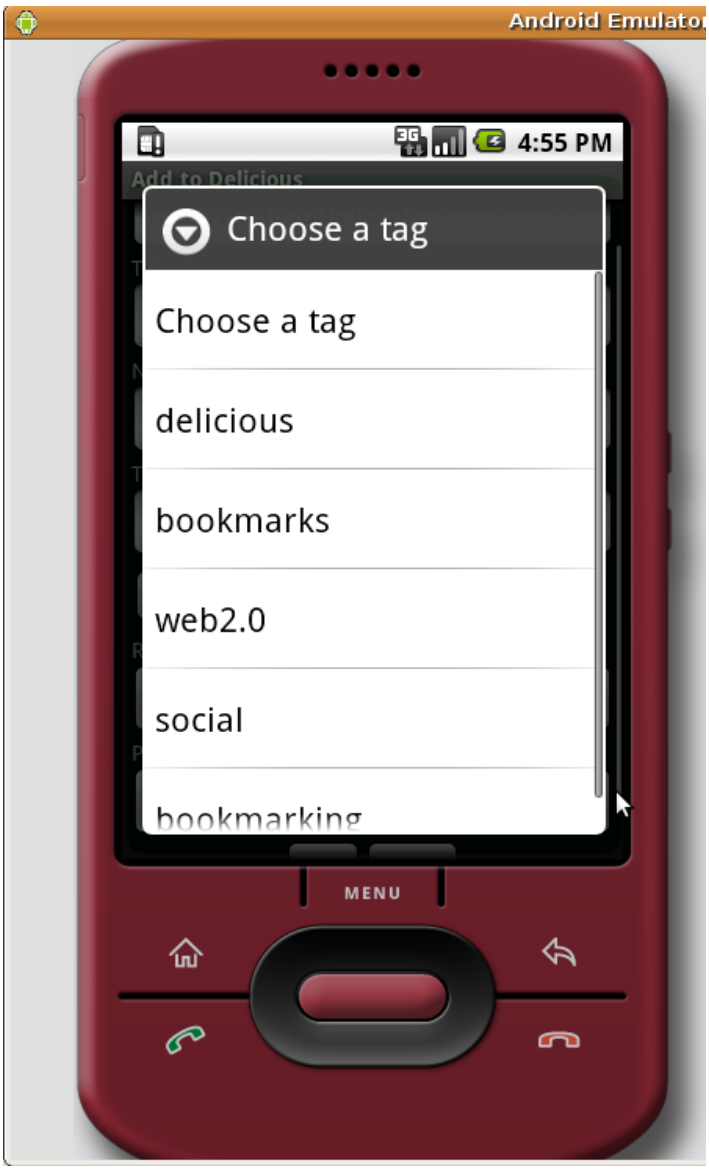


Figure 13: Tag suggestion dialog

IV. Conclusion

My Delicious is developed to be an innovative application and its main purpose is to get familiar with Android SDK. Because of this reason, here we do not consider much about optimality attitude of application. The next version of this will concentrate on remaining problem like portability between different kinds of device, adaptability with the change of API, friendlier interface, and also optimality in network usage.

V. References

- ✓ https://api.del.icio.us
- ✓ Android A Programmer Guid –McGraw Hill
- ✓ Posts on http://androidforums.com/
- ✓ http://www.androidmobileforum.com/
- ✓ http://androidcommunity.com/forums/
- ✓ Android for Java Developers - Dr. Markus Schmall, Jochen Hiller

VI. Table of figure