A first look at
Google Android

Tomas Katysovas
tkatysovas@unibz.it

Free University of Bolzano, Internet Technologies 2. 2007-2008

Abstract

This paper examines an open source mobile phone platform Android. It explains its' advantages and disadvantages, the basic features and the market strategy. I review the adaptation of this new technology from the view of prospective mobile software developers, customers and manufacturers.

The purpose of this report is to present to the user the new and promising mobile platform based on the Linux operating system and provided by Google. Additionally, I introduce Open Source infrastructure which not only supports development, but also has the potential to become a main business activity in the future.

Contents

Abstract 2
Introduction 4
The Birth of Android 4
Android Features 5
System Architecture 6
Developing Applications 8
Application Building Blocks 8
Code example 10
Application Lifecycle 14
Security Issues 18
Android and Java ME 19
Similarities 19
"Hello World" example 20
Market Research 22
A prospective customer 22
Speculations with cellular carriers 23
Manufacturers' war 24
Mobile Future 25
Mobile Ads 25
Mobile Services 26
Conclusion 27
References 28

Introduction

The Birth of Android

Firstly, I would like to mention a small company Android Inc. based on the software development for mobile phones, which was acquired by Google for unknown amount of money on July 2005. As the experienced team started to work hardly in Google Campus, it was a first serious sign about Google entering mobile phone market. In 2007, the Open Handset Alliance (OHA) was created to develop open standards for mobile devices. It consisted of 34 grand members, such as Google itself, NVIDIA, Intel, Motorola, T-Mobile and other mobile operators, handset manufacturers, software and other companies. As OHA stands for open mobile platform, a great race has started between OHA and main competitors Apple, Microsoft, Symbian and others. Microsoft launched Windows Mobile 6.0 version with full updated Office Mobile and other features. Symbian with over 110 million smartphones released OSv.9.5, and Apple stroked market with iPhone. The entire world was waiting for the response from Google with visionary Gphone, a single mobile device which could compete with iPhone and other mobile phones. OHA came with better solution – *Google Android* – first truly open mobile phone platform based on Linux, with clear and simple user interface and applications, created in Java. This strategy, which is about to declare not a single Gphone, but to put Android into existing and new mobiles devices and to make thousands of Gphones, gives mobile operators and device manufacturers significant freedom and flexibility to design products. As Google Android will be truly released in 2008 with its source code, at this moment Google announced Android SDK together with competition, which provides 10 million dollars in awards for Android developers. This idea seems to be quit clear and logical, in order to speed up and boost Java developers, but actually the contest slowed down the knowledge integration. The contest has effectively caused developers not to share their code to others. Therefore, I found a lack of answering to questions and other support on Android groups over the internet. Nevertheless, I believe there will be plenty of code available to help inexperienced developers make ideas come to life.

Android Features

Application Framework is used to write applications for Android. Unlike other embedded mobile environments, Android applications are all equal, for instance, an applications which come with the phone are no different than those that any developer writes. The framework is supported by numerous open source libraries such as openssl, sqlite and libc. It is also supported by the Android core libraries. From the point of security, the framework is based on UNIX file system permissions that assure applications have only those abilities that mobile phone owner gave them at install time.

Dalvik virtual machine – it is extremely low-memory based virtual machine, which was designed specially for Android to run on embedded systems and work well in low power situations. It is also tuned to the CPU attributes. The Dalvik VM creates a special file format (.DEX) that is created through build time post processing. Conversion between Java classes and .DEX format is done by included "dx" tool.

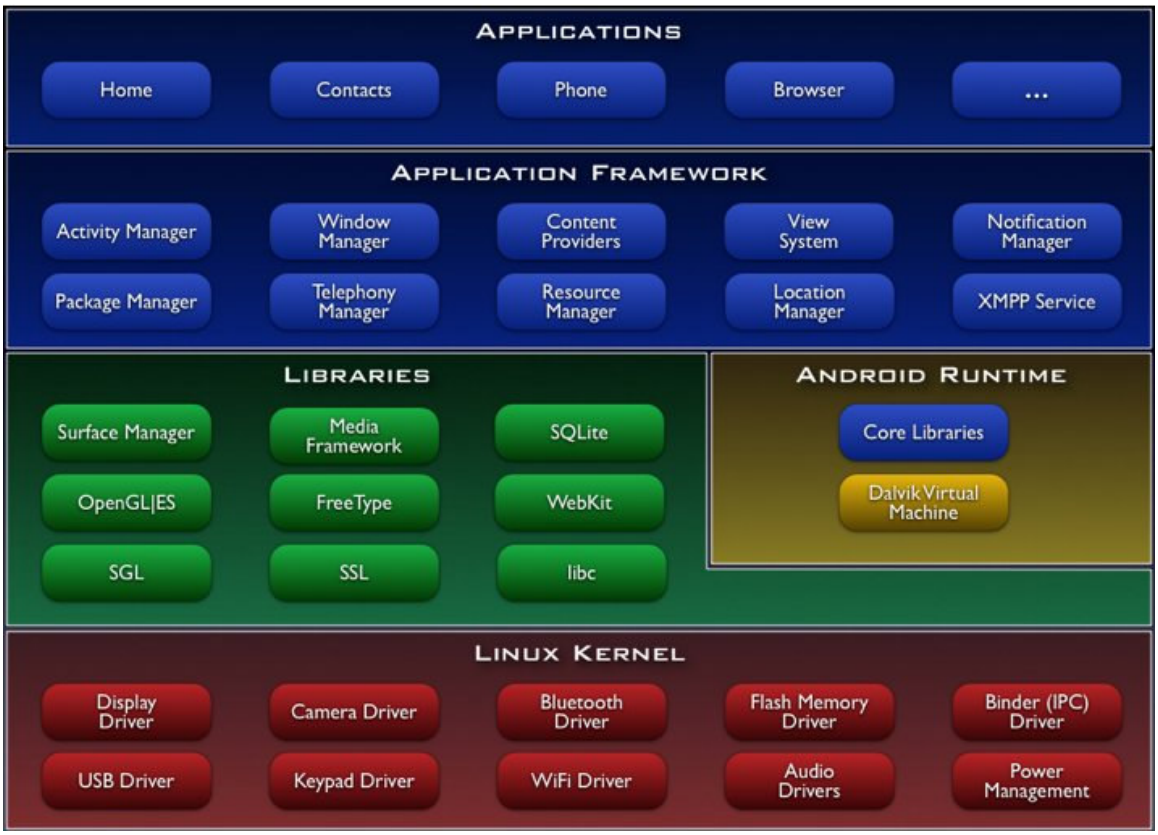
Integrated browser – in my opinion, Google made a right choice on choosing WebKit as open source web browser. They added a two pass layout and frame flattening. Two pass layout loads a page without waiting for blocking elements, such as external CSS or external JavaScript and after a while renders again with all resources downloaded to the device. Frame flattening converts founded frames into single one and loads into the browser. These features increase speed and usability browsing the internet via mobile phone.

Optimized graphics – as Android has 2D graphics library and 3D graphics based on OpenGL ES 1.0, possibly we will see great applications like Google Earth and spectacular games like Second Life, which come on Linux version. At this moment, the shooting legendary 3D game Doom was presented using Android on the mobile phone.

SQLite – extremely small (~500kb) relational database management system, which is integrated in Android. It is based on function calls and single file, where all definitions, tables and data are stored. This simple design is more than suitable for a platform such as Android.

There are a number of hardware dependent features, for instance, a huge media and connections support, GPS, improved support for Camera and simply GSM telephony. A great work was done for the developers to start work with Android using device emulator, tools for debugging and plugin for Eclipse IDE. Finally, as Android just started its' journey to the mobile market, we are going to see much more features through the developed applications.

System Architecture



google

Android Architecture is based on Linux 2.6 kernel. It helps to manage security, memory management, process management, network stack and other important issues. Therefore, the user should bring Linux in his mobile device as the main operating system and install all the drivers required in order to run it. Android provides the support for the Qualcomm MSM7K chipset family. For instance, the current kernel tree supports Qualcomm MSM 7200A chipsets, but in the second half of 2008 we should see mobile devices with stable version Qualcomm MSM 7200, which includes major features:

- WCDMA/HSUPA and EGPRS network support
- Bluetooth 1.2 and Wi-Fi support
- Digital audio support for mp3 and other formats
- Support for Linux and other third-party operating systems
- Java hardware acceleration and support for Java applications
- Qcamera up to 6.0 megapixels
- gpsOne – solution for GPS
- and lots of other.

In the next level we can see a set of native libraries written in C/C++, which are responsible for stable performance of various components. For example, Surface Manager is responsible for composing different drawing surfaces on the mobile screen. It manages the access for different processes to compose 2D and 3D graphic layers. OpenGL ES and SGL make a core of graphic libraries and are used accordingly for 3D and 2D hardware acceleration. Moreover, it is possible to use 2D and 3D graphics in the same application in Android. The media framework was provided by PacketVideo, one of the members of OHA. It gives libraries for a playback and recording support for all the major media and static image files. FreeType libraries are used to render all the bitmap and vector fonts. For data storage, Android uses SQLite. As I mentioned before, it is extra light rational management system, which locates a single file for all operations related to database. WebKit, the same browser used by Apples' Safari, was modified by Android in order to fit better in a small size screens.

At the same level there is Android Runtime, where the main component Dalvik Virtual Machine is located. It was designed specifically for Android running in limited environment, where the limited battery, CPU, memory and data storage are the main issues. Android gives an integrated tool "dx", which converts generated byte code from .jar to .dex file, after this byte code becomes much more efficient to run on the small processors. As the result, it is possible to have multiple instances of Dalvik virtual machine running on the single device at the same time.
The Core libraries are written in Java language and contains of the collection classes, the utilities, IO and other tools.

After that, we have Application Framework, written in Java language. It is a toolkit that all applications use, ones which come with mobile device like Contacts or SMS box, or applications written by Google and any Android developer. It has several components which I will discuss. The Activity Manager manages the life circle of the applications and provides a common navigation backstack for applications, which are running in different processes. The Package Manager keeps track of the applications, which are installed in the device. The Windows Manager is Java programming language abstraction on the top of lower level services that are provided by the Surface Manager. The Telephony Manager contains of a set of API necessary for calling applications. Content Providers was built for Android to share a data with other applications, for instance, the contacts of people in the address book can be used in other applications too. The Resource Manager is used to store localized strings, bitmaps, layout file descriptions and other external parts of the application. The View System generates a set of buttons and lists used in UI. Other components like Notification manager is used to customize display alerts and other functions.

At the top of Android Architecture we have all the applications, which are used by the final user. By installing different applications, the user can turn his mobile phone into the unique, optimized and smart mobile phone.

Developing Applications

Application Building Blocks

Google provides three versions of SDK: for Windows, for Mac OSX and one for Linux. The developer can use Android plugin for Eclipse IDE or other IDEs such as intelliJ.

First step for Android developer is to decompose the prospective application into the components, which are supported by the platform. The major building blocks are these:

- Activity
- Intent Receiver
- Service
- Content Provider

Activity – user interface component, which corresponds to one screen at time. It means that for the simple application like Address Book, the developer should have one activity for displaying contacts, another activity component for displaying more detailed information of chosen name and etc.
Intent Receiver – wakes up a predefined action through the external event. For example, for the application like Email Inbox, the developer should have intent receiver and register his code through XML to wake up an alarm notification, when the user receives email.
Service – a task, which is done in the background. It means that the user can start an application from the activity window and keep the service work, while browsing other applications. For instance, he can browse Google Maps application while holding a call or listening music while browsing other applications.
Content Provider – a component, which allows sharing some of the data with other processes and applications. It is the best way to communicate the applications between each other.

Secondly, a developer should predefine and list all components, which he wants to use in the specific AndroidManifest.xml file. It is a required file for all the applications and is located in the root folder. It is possible to specify all global values for the package, all the components and its classes used, intent filters, which describe where and when the certain activity should start, permissions and instrumentation like security control and testing. Here is an example of AndroidManifest.xml file:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.     package="dk.mdev.android.hello">
4.     <application android:icon="@drawable/icon">
```

```
5.        <activity class=".HelloAndroid" android:label="@string/app_name">
6.            <intent-filter>
7.                <action android:value="android.intent.action.MAIN" />
8.                        <category
    android:value="android.intent.category.LAUNCHER"/>
9.            </intent-filter>
10.       </activity>
11.   </application>
12. </manifest>
```

The line 2 is a namespace declaration, which makes a standard Android attributes available for that application. In the line 4 there is a single <application> element, where the developer specifies all application level components and its properties used by the package. Activity class in the line 5 represents the initial screen the user sees and it may have one or more <intent-filter> elements to describe the actions that activity supports.

There is the activityCreator script, which generates the following files and folders in your Eclipse workplace:
- AndroidManifest.xml file discussed before;
- Build.xml – an ant file which is used to package an application;
- src/ - source directory
- bin/ - the output directory

The special file R.java is generated by Eclipse in the source code folder. It is an index into all the resources defined in the file and is useful for locating the specific reference.

Android lets developers to use debugging and testing tools like DDMS (Dalvik Debug Monitor Server), logcat and others.

Code example

It is possible to send XMPP messages in Android. XMPP – open Extensible Messaging and Presence Protocol for near-real-time instant messaging (IM) and presence information. As Google Talk provides XMPP gateways to its service, some modifications of Gtalk are already created for Android platform. Here is a source code of unofficial Gtalk application, written by Davanum Srinivas (http://davanum.wordpress.com):

GTalkClient.java

```
// firstly, all imported packages are listed at the beginning of the code

package org.apache.gtalk;

import android.app.Activity;        //Activity class takes care of creating a window for UI
import android.app.NotificationManager;      //Class for event notifications
import android.content.ComponentName;     //Identifier for one of application component
import android.content.Context;                 //Abstract class for global information about
                                                //an application environment
import android.content.Intent;         //Abstract description of an operation to be performed
import  android.content.ServiceConnection;     //Interface for monitoring the state of an
                                                 //application service
import android.database.Cursor;   //Interface for providing random read-write access to
                                  //the result set returned by a database query
import android.os.Bundle;            //Mapping from String values to various types
import android.os.DeadObjectException;  //Exception for an object, which does not exists
import android.os.IBinder;              //Interface for a remotable Binder object
import android.provider.Im;
import android.text.TextUtils;          //Monitor or modify keypad input
import android.util.Log;                //API for sending log output
import android.view.View;               //Used to create interactive graphical user interfaces
import android.widget.*;                //Visual UI elements
import com.google.android.xmppService.IXmppService;  //IXmppService interface
                                                     //definition file for XMPP service
import com.google.android.xmppService.IXmppSession;  //IXmppSession interface
                                                     //definition file for XMPP session
import com.google.android.xmppService.Presence;  //Abstract presentation of the user's
//presence information

public class GTalkClient extends Activity implements View.OnClickListener {
   private static final String LOG_TAG = "GTalkClient";

   IXmppSession mXmppSession = null;
   EditText mSendText;
   ListView mListMessages;
```

```
    EditText mRecipient;
    Button mSend;
    Button mSetup;

// Called with the activity is first created.

    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.main);

        // gather the troops
        mSendText = (EditText) findViewById(R.id.sendText);
        mListMessages = (ListView) findViewById(R.id.listMessages);
        mRecipient = (EditText) findViewById(R.id.recipient);
        mSend = (Button) findViewById(R.id.send);
        mSetup = (Button) findViewById(R.id.setup);

        // set up handler for on click
        mSetup.setOnClickListener(this);
        mSend.setOnClickListener(this);

        bindService((new Intent()).setComponent(

com.google.android.xmppService.XmppConstants.XMPP_SERVICE_COMPONENT),
                null, mConnection, 0);
    }

// Let the user know there was an issue

    private void logMessage(CharSequence msg) {
        NotificationManager nm = (NotificationManager) getSystemService(
                Context.NOTIFICATION_SERVICE);

        nm.notifyWithText(123, msg, NotificationManager.LENGTH_LONG, null);
    }

// Here's the code that gets a XMPP session using a service connection

    private ServiceConnection mConnection = new ServiceConnection() {
        public void onServiceConnected(ComponentName className, IBinder service) {
            // This is called when the connection with the XmppService has been
            // established, giving us the service object we can use to
            // interact with the service.  We are communicating with our
            // service through an IDL interface, so get a client-side
            // representation of that from the raw service object.
```

```
            IXmppService xmppService = IXmppService.Stub.asInterface(service);

            try {
                mXmppSession = xmppService.getDefaultSession();
                if (mXmppSession == null) {
                    // this should not happen.
                    logMessage(getText(R.string.xmpp_session_not_found));
                    return;
                }
                mXmppSession.setPresence(new
Presence(Im.PresenceColumns.AVAILABLE, "Am here now!"));
            } catch (DeadObjectException ex) {
                Log.e(LOG_TAG, "caught " + ex);
                logMessage(getText(R.string.found_stale_xmpp_service));
            }

            mSendText.setEnabled(true);
        }

        public void onServiceDisconnected(ComponentName componentName) {
            // This is called when the connection with the service has been
            // unexpectedly disconnected -- that is, its process crashed.
            mXmppSession = null;
            mSendText.setEnabled(false);
        }
    };

    // Handle clicks on the 2 buttions

    public void onClick(View view) {
        if (view == mSetup) {
            Log.i(LOG_TAG, "onClick - Setup");
            // Run a query against CONTENT_URI = "content://im/messages"
            Cursor cursor = managedQuery(Im.Messages.CONTENT_URI, null,
                    "contact=\'" + mRecipient.getText().toString() + "\'", null, null);

            // Display the cursor results in a simple list
            // Note that the adapter is dynamic (picks up new entries automatically)
            ListAdapter adapter = new SimpleCursorAdapter(this,
                    android.R.layout.simple_list_item_1,
                    cursor, // Give the cursor to the list adapter
                    new String[]{Im.MessagesColumns.BODY},
                    new int[]{android.R.id.text1});

            this.mListMessages.setAdapter(adapter);
        } else if (view == mSend) {
```

```
            // use XmppService to send data message to someone
            String username = mRecipient.getText().toString();
            if (!isValidUsername(username)) {
                logMessage(getText(R.string.invalid_username));
                return;
            }

            if (mXmppSession == null) {
                logMessage(getText(R.string.xmpp_service_not_connected));
                return;
            }

            try {
                mXmppSession.sendTextMessage(username,                                    0,
mSendText.getText().toString());
            } catch (DeadObjectException ex) {
                Log.e(LOG_TAG, "caught " + ex);
                logMessage(getText(R.string.found_stale_xmpp_service));
                mXmppSession = null;
            }
        }
    }

    private boolean isValidUsername(String username) {
        return !TextUtils.isEmpty(username) && username.indexOf('@') != -1;
    }
}
```

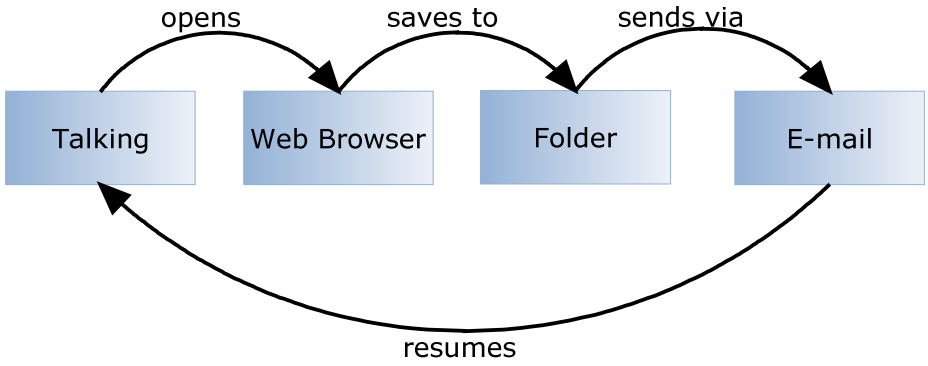
AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.apache.gtalk">
    <application>
        <activity class=".GTalkClient" android:label="GTalk Client">
            <intent-filter>
                <action android:value="android.intent.action.MAIN" />
                <category android:value="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Application Lifecycle

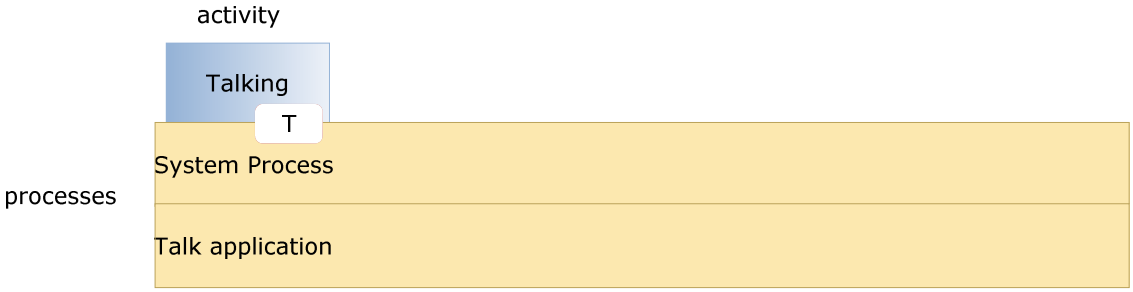
In Android, every application runs in its own process, which gives better performance in security, protected memory and other benefits. Therefore, Android is responsible to run and shut down correctly these processes when it is needed.

In the following example I will display a process flow from the Android System point of view to get a clear idea how the applications behave. Let assume the possible scenario: A user talks to his friend via mobile phone and he is asked to browse the internet (a talk is hold for a moment), find a picture of him in his Picasa Album, send it via Email back to his friend and resume a talk.

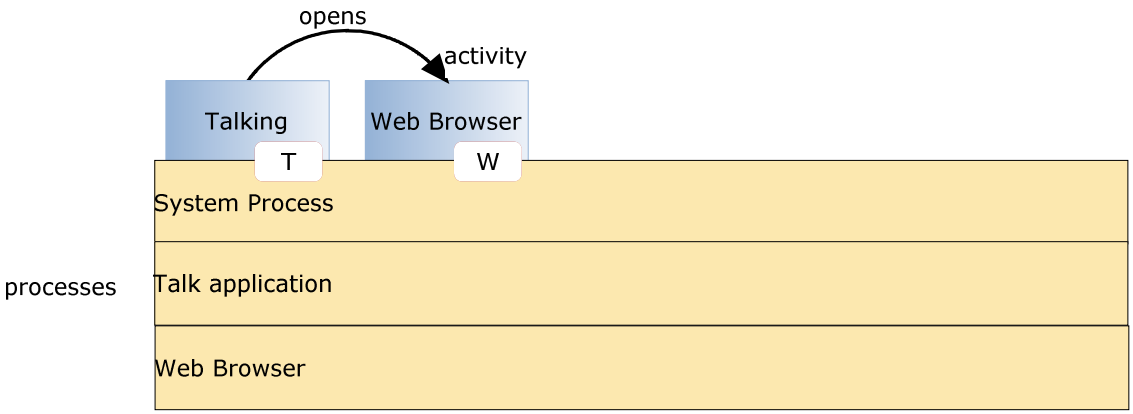


In this situation, there are 4 different applications and 4 different processes running, but from the user point of view none of them are important, as Android manages CPU work and memory usage by itself. It means the user can travel through the applications forward and back without thinking about how much memory is left or which processes are run at the time.

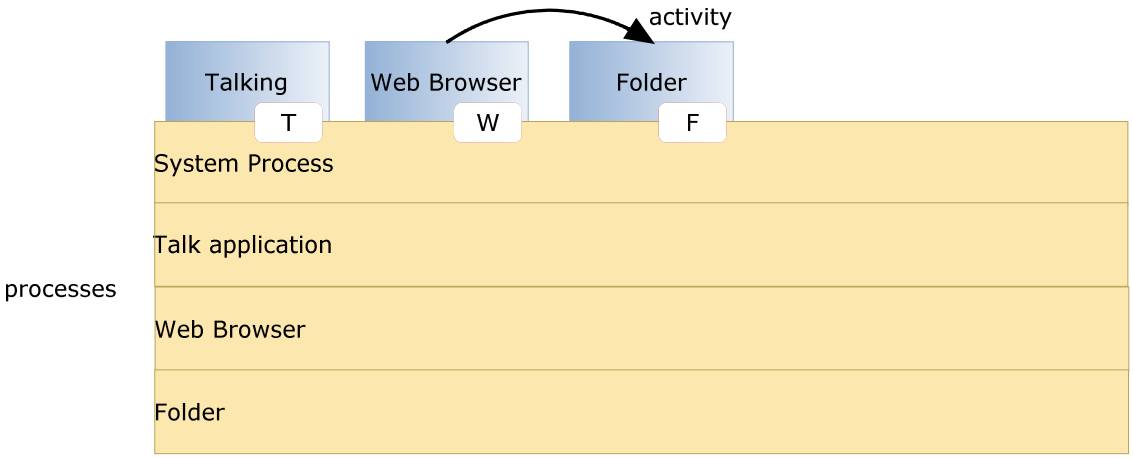
Firstly, as the user is talking to his friend, a specific Talk application is opened, which contains the activity manager. In the following stack we can see two processes running, the main system process and Talk application process. Moreover, before going to Web Browser application, the system saves a Talk state T in order to remember that process:



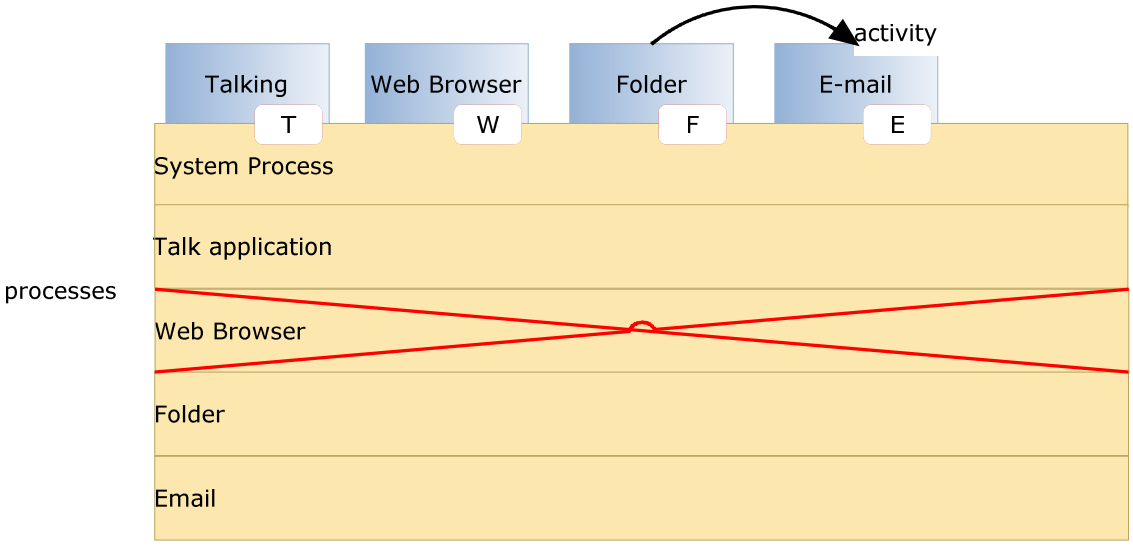
At this point, as a user holds a talk and opens a web browser, the system creates a new process and new web browser activity is launched in it. Again, the state of the last activity is saved (W):



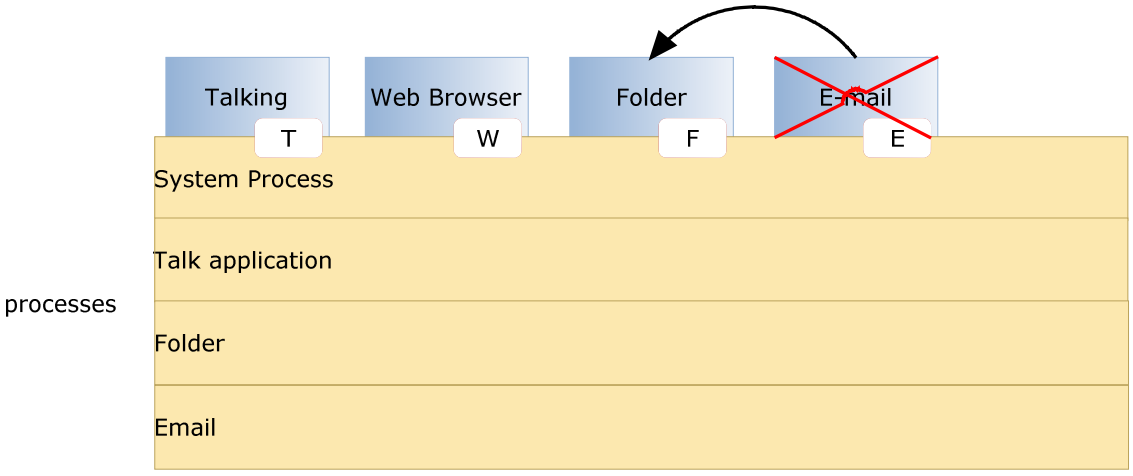
After that, the user browses the internet, finds his picture in Picasa album and saves it to particular folder. He does not close a web browser, instead he opens a folder to find saved picture. The folder activity is launched in particular process:



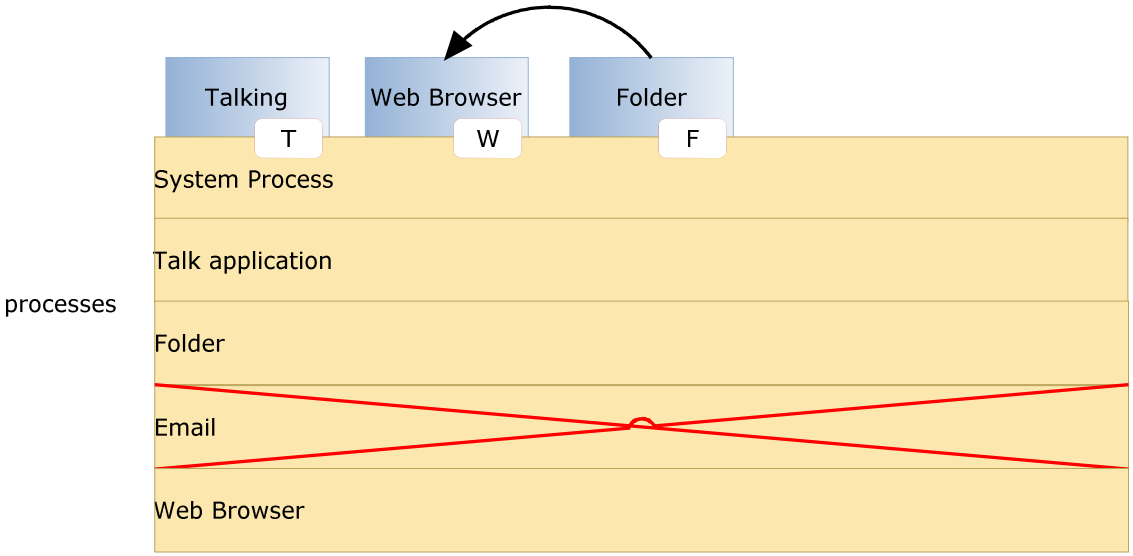
At this point, the user finds his saved picture in the folder and he creates a request to open an Email application. The last state F is saved. Now assume that the mobile phone is out of the memory and there is no room to create a new process for Email application. Therefore, Android looks to kill a process. It can not destroy Folder process, as it was used previously and could be reused again, so it kills Web Browser process as it is not useful anymore and locates a new Email process instead:



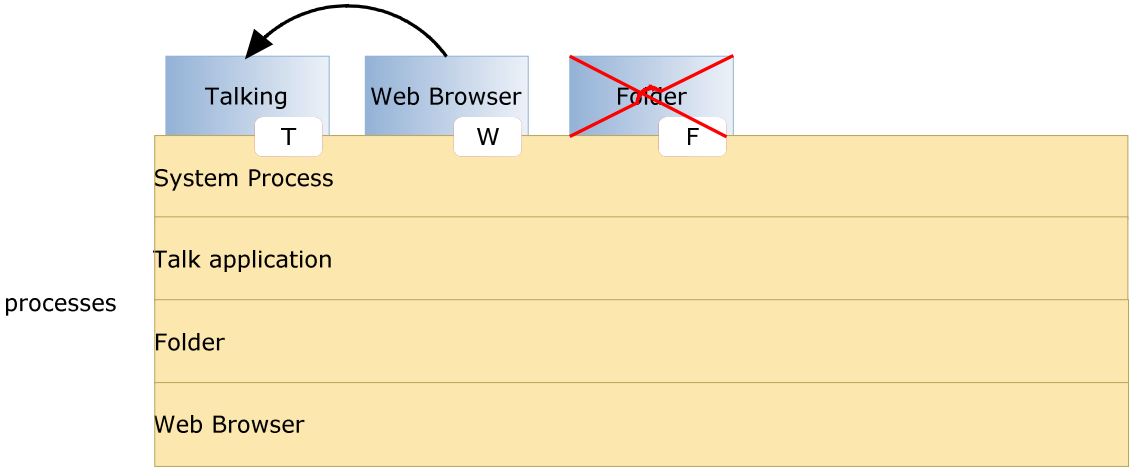
The user opens Email application and sends a picture to his friend via email. Now he wants to go back to the Talk application and to resume a talk to his friend. Because of the previously saved states, this work is done fast and easily. In this example, Email application is popped out and the user sees a previous Folder application:



Next, the user goes back to Web Browser application. Unfortunately, web browser process was killed previously so the system has to kill another process (in our case it is Email application process, which is not used anymore) in order to locate Web Browser process and manage the stack memory:



and finally:



Now the user comes back to the Talk application and resumes his talk with his friend. Because of the saved states, going back procedure is fast and useful, because it remembers previous activities and its views.
This example shows, that it does not matter how many applications and processes are active or how much available memory is left, Android it manages fast and without a user interaction.

Security Issues

It is quite difficult to discuss all the security issues, as no Android phone is build yet. By the prediction, Android mobile phone platform is going to be more secure than Apple's iPhone or any other device in the long run. There are several solutions nowadays to protect Google phone from various attacks. One of them is security vendor McAfee, a member of Linux Mobile (LiMo) Foundation. This foundation joins particular companies to develop an open mobile-device software platform. Many of the companies listed in the LiMo Foundation have also become members of the Open Handset Alliance (OHA). As a result, Linux secure coding practice should successfully be built into the Android development process. However, open platform has its own disadvantages, such as source code vulnerability for black-hat hackers. In parallel with great opportunities for mobile application developers, there is an expectation for exploitation and harm. Stealthy Trojans hidden in animated images, particular viruses passed from friend to friend, used for spying and identity theft, all these threats will be active for a long run.
Another solution for such attacks is SMobile Systems mobile package. SecurityShield – an integrated application that includes anti-virus, anti-spam, firewall and other mobile protection is up and ready to run on the Android operating system. Currently, the main problem is availability for viruses to pose as an application and do things like dial phone numbers, send text messages or multi-media messages or make connections to the Internet during normal device use. It is possible for somebody to use the GPS feature to track a person's location without their knowledge. Hence SMobile Systems is ready to notify and block these secure alerts. But the truth is that it is not possible to secure your mobile device or personal computer completely, as it connects to the internet. And neither the Android phone nor other devices will prove to be the exception.

Android and Java ME

Similarities

Java Platform, Micro Edition or Java ME (previously known as Java 2 Platform, Micro Edition or J2ME) is a specification of a subset of the Java platform aimed at providing a certified collection of Java APIs for the development of software for small, resource-constrained devices. Though, do not confuse it with Google Android, even there are some similarities:

- Eclipse plugins for J2ME and Android look very similar and interface very well with their respective SDKs;
- Both J2ME and Android seem to share the same core Java APIs, such as java.util and java.net. But their APIs for graphics, UIs, etc. are very dissimilar and philosophies for developing applications are very different;
- Android seems to be more tightly integrated (up to even the OS services provided and how they interact with the APIs), while J2ME is far more liberal in its specifications for the developer and mobile device manufacturer.

A slower application development and performance – these are the main disadvantages Java's J2ME have for today. J2ME apps are second-rate citizens in the phones. They do not have an access to most of the low-level features, like call API, external connectivity (USB) and other. There is no way to replace or extend built-in phone apps like contacts, calendar and calls. For instance, J2ME applications in Nokia devices with S60 work great for standard tasks. But more advanced users find difficulties handling Wi-Fi access points with S60, because APIs simply do not seem to be exposed to J2ME. A user may find difficulties synchronizing Google Calendar with his device - nobody seems to have been able to figure out how to make the J2ME calendar interfaces work correctly on S60. There are lots of problems with Java applications on S60, even though S60 probably has one of the best Java implementations. Android fills a void in Java Mobile applications by providing API to build richer applications - more useful for Smart Phones which contain the ability to provide these types of functionalities. If J2ME filled every void, Android as an API wouldn't be needed (though Android as an OS could still fill a void). Google has written its own virtual machine for Android most likely as a way to get around licensing issues with Sun. However, Android does not include a complete and compliant Java stack (neither JME nor JSE); only a subset and therefore it is technically not the Java platform, it just looks a lot like it.

"Hello World" example

Here is a simple "Hello World" application, written both in J2ME and Google Android. It is possible to use NetBeans v6.0 for J2ME UI easier implementation. Despite the fact, that NetBeans generated source code is quit big, here is a simpler J2ME code version (not using NetBeans):

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HelloWorld extends MIDlet implements CommandListener {
    private Command exitCommand;
    private TextBox tbox;

    public HelloWorld() {
        exitCommand = new Command("Exit", Command.EXIT, 1);
        tbox = new TextBox("Hello world MIDlet", "Hello World!", 25, 0);
        tbox.addCommand(exitCommand);
        tbox.setCommandListener(this);
    }

    protected void startApp() {
        Display.getDisplay(this).setCurrent(tbox);
    }

    protected void pauseApp() { }
    protected void destroyApp(boolean bool) { }

    public void commandAction(Command cmd, Displayable disp) {
        if (cmd == exitCommand) {
            destroyApp(false);
            notifyDestroyed();
        }
    }
}
```

HelloWorld.java class written in Google Android using Eclipse:

```
import android.widget.TextView;

public class HelloWorld extends Activity {
  public void onCreate(Bundle icicle) {
     super.onCreate(icicle);
     TextView tv = new TextView(this);
     tv.setText("Hello World");
     setContentView(tv);
  }
}
```

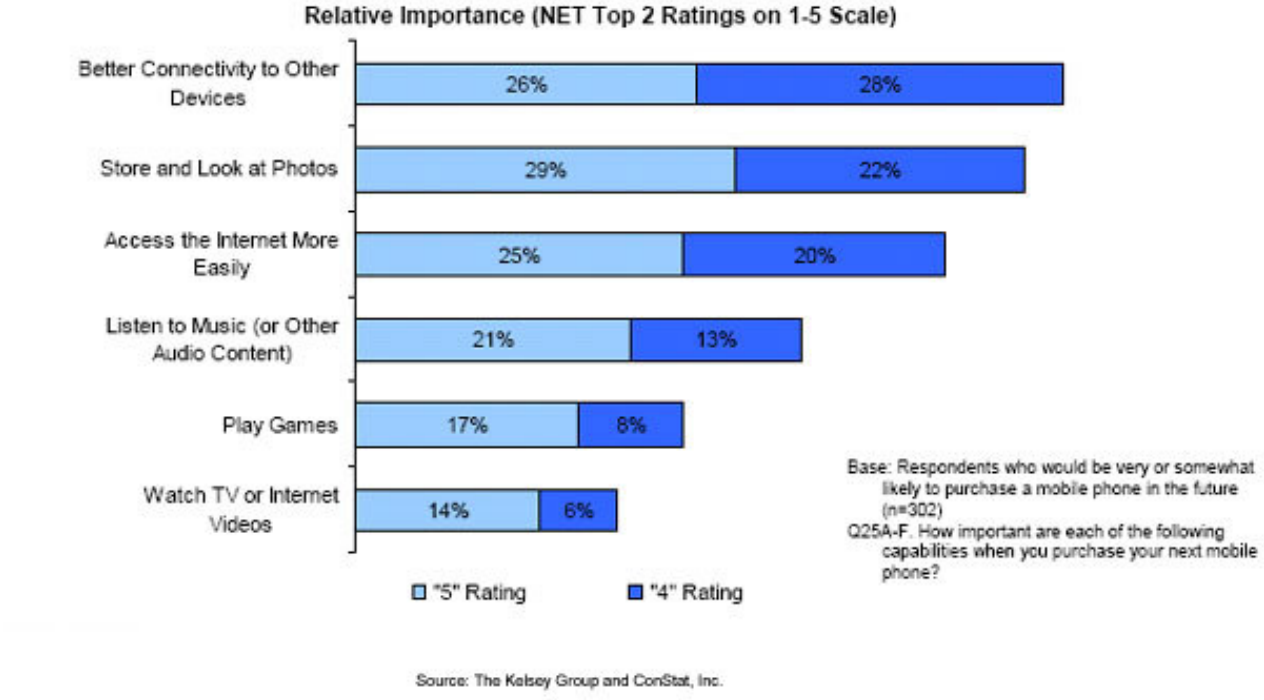
Firstly, a text label object is imported through TextView class. After that TextView constructor is created and we tell what to display (tv.setText("Hello World")). Finally, we connect constructed TextView with the on-screen display.
In conclusion, Android code is simple to understand while J2ME code requires better understanding in Java mobile development and NetBeans graphical tool.

Market Research

A prospective customer

A new generation of mobile device users is coming in the next decade. These users are going to explore the mobile internet afresh with its new features, compatible mobile phones, new services and applications. One of the Google mobile software engineers Dave Burke predicts, that by 2010, the number of global mobile phone subscribers is expected to be 4 billion, comparing to the number of desktop internet connections, which stand at 1.2 billion today. This is a huge leap for mobile advertisement business, where revenue could rise 8 times more by 2012. As other mobile platforms failed to do so for seven years, Google Android is going to present new solutions through the fast search engine, open source applications and other services. The Kelsey Group, which works with public opinion polls and statistics, published the results released October 11 2007, which say, that one hundred out of 500, or 20 percent of people would be interested in purchasing a Google phone. Despite the fact, that Google Android is in alpha version and it is unknown for the customers and mobile market, the results look promising.

The diagram below shows the study, which was conducted in September 2007 via an online 30-question survey of 500 U.S. mobile phone users aged 18 and older.



People do not find a good Internet experience in their phones today, so they are more interested in gravitating toward an Internet or technology company telephone because they think connectivity between devices and to the Internet is going to be much better on those phones. They use Google search, GMail, Google Maps, Picasa albums and other popular services on their computers, and this is what they expect to have in their mobile devices in the close future.

Speculations with cellular carriers

Google Android enters a tangled mess of cellular carrier world. As a new player in the mobile market, Android brings an open platform with the new rules. On the one hand there is OHA with major companies and carries, such as T-Mobile and Sprint. On the other hand, there are two largest cellular carries AT&T and Verizon Wireless in United States, which have a vested interest in operating systems of their own. It is predictable, that Sprint or T-Mobile will be first carriers providing devices with Google Android. This ensures equal development time for the networks, GSM/HSDPA side and CDMA/EV-DO. But the main problem, which faces all the cellular carriers around the world, is the availability to download and use free applications that could block almost every communications product they sell. A user does not need to pay for GPS mapping service anymore. He can simply download a free one that taps into Google Maps. Why pay for text messages to his friends when he can download an instant messaging client? In fact, why pay for cellular minutes at all when a user can download Skype, Gtalk or other client and just use his data plan? OS's such as Android threaten carriers with a loss of control over the applications on the phones on their network and they may find themselves becoming nothing more than wireless Internet service providers, forced to compete on price and bandwidth. Another aspect is hardware cost: Google Android owns 10 percent of the total cost of a phone, which combined with falling hardware prices could eventually result a fertile unlocked handset market. In conclusion, Google has a better start in this race than any company had before to bring new rules to the mobile market with all carriers, mobile devices and its customers.

Manufacturers' war

Presently, Google main competitors like Nokia, Microsoft and Apple do not see Google Android as a serious rival or threat to their business strategies. "It really sounds that they are getting a whole bunch of people together to build a phone and that's something we've been doing for five years," said Scott Horn, from Microsoft's Windows Mobile marketing team. John Forsyth from Symbian said: "We take it seriously but we are the ones with real phones, real phone platforms and a wealth of volume built up over years." However, the current situation is not so unsophisticated. There is a huge flurry in the companies, which are not in the list of OHA. For instance, Nokia, which is the largest handset manufacturer in the world, nowadays owning some 39% market share, was one of the companies snubbed on the invitation list to the 34-party Open Handset Alliance that is growing daily. In contrast, Nokia is buying companies and dumping cash into development, while Google is releasing an open platform hoping the applications and services will build themselves with the help of a strong developer community, development contests and large alliance of grand companies. Despite of this, Nokia is ready to combat whatever Google has to throw with Google Android in 2008. Another company Apple has already stroked the market with iPhone and its closed operating system. Accordingly, iPhone in the US remains loyal to AT&T mobile carrier for five years**.** That is plenty of time for Google to conquer the market with open Android. Obvious advantage of Android is cost: while iPhone is priced at a weighty $400, Google says it hopes to reach a more mainstream market by pricing Android-powered devices at around $200. Microsoft, selling 21 millions copies of Windows Mobile software, stays calm at this point, waiting for some particular results from Google Android.

This nice and healthy competition is just what the mobile industry needs at the moment, at least for the consumers. The wars being waged between Google and the field will only create better, cheaper handsets and more advanced applications.

Mobile Future

Mobile Ads

Jaiku - an activity stream and sharing service that works from the Web and mobile phones was bought by Google as important investment into the mobile advertisement. People wondered why Google preferred the micro-blogging service to Twitter, which is much more popular nowadays. The answer lies in Jaiku's unique ability to combine micro-blogging with user's location. An integral part of the service is a Jaiku client application for Symbian S60 platform mobile phones, which should come to Android platform as well. The client uses location APIs within device to get the handset and the users' location based on nearby cellular network towers. Though the location is not very precise, the mobile phone is able to broadcast it automatically. At that point the text can be connected to users' location and create a list of preferences for each place the user frequently visits. Using such a technology, it is simple to track down a user via phone's IP address, whenever he comes into McDonald or is sitting in the airport.
Google is not a million miles away from being able to push advanced advertising to individuals based on their profile, their location and their availability. They already offer regional and local targeting for ads for desktop users, but this could be much more useful for a mobile phone. And if the ads are truly relevant, interesting and unobtrusive, people might actually start to like them.

Mobile Services

Adding to its fast growing suite of mobile applications and services, Google has applied for a patent for a mobile payments service that would allow users to make payments at retail shops using their mobile phones. The Text Message Payment patent describes a system where Google offers mobile focused payments called GPay. This describes a system where a SMS message would be sent containing a payment amount and other information. That payment amount would then be validated, debited from the user's account, and communicated from server to server. Payment confirmation that had been received would also simultaneously be sent to the relevant party, as illustrated in the diagram below:

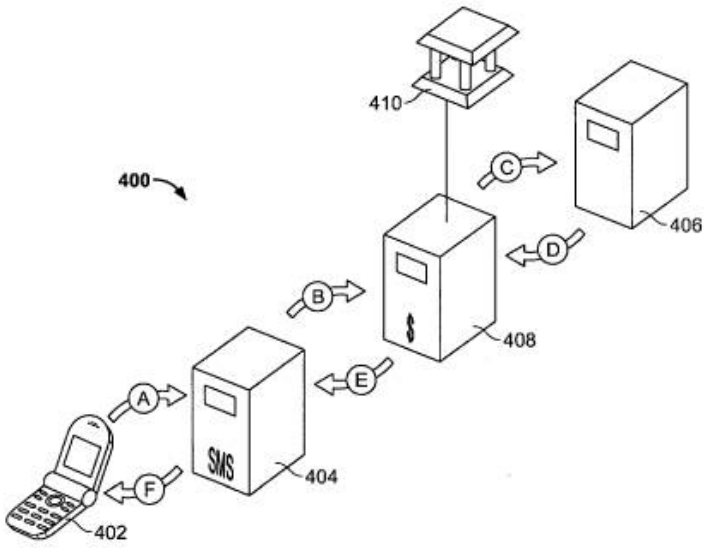


FIG. 4

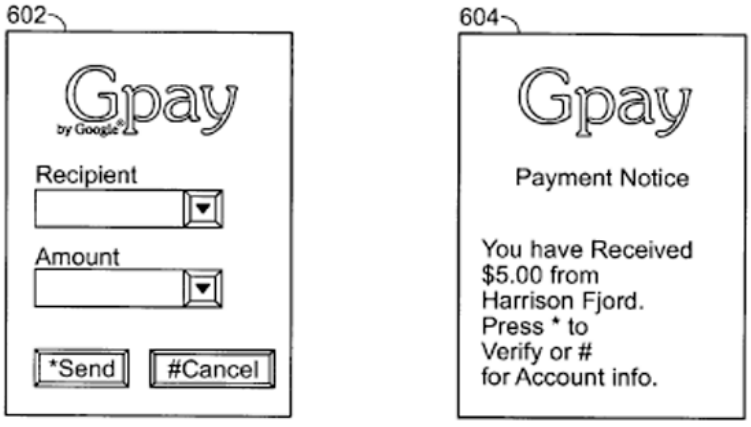


FIG. 6

Patent Application Publication

Described as "a computer-implemented method of effectuating an electronic on-line payment," the system mentioned in the patent application is similar to existing mobile-payment services. These services like mobile version of PayPal have been available for some time but have had little success bursting with merchants and with customers. The main difference between existing mobile payment systems and GPay is, of course, that GPay is created by Google and will be easily adopted by Android Platform.

Conclusion

In summary, with all upcoming applications and mobile services Google Android is stepping into the next level of Mobile Internet. Android participates in many of the successful open source projects. That is, architect the solution for participation and the developers will not only come but will play well together. This is notable contrast with Apple and other companies, where such architecture of participation is clearly belated. The first Android based official devices may well be launched sometime in the second half of 2008. Obviously, that's an age away when it comes to handset design, and Android may well find itself competing against the forthcoming Nokia touch screen phones and maybe even the iPhone 2. Who knows?

References

As the subject is quite new and there are no books and papers published yet, I wrote this report based on the information I found on these web pages:

http://code.google.com/android/ - Google Android official webpage
http://www.openhandsetalliance.com/ - Open Handset Alliance webpage
http://en.wikipedia.org/wiki/Android_(mobile_phone_platform) – Wikipedia information
http://googleblog.blogspot.com/ - Official Google Blog
http://davanum.wordpress.com – Gtalk code example written by Davanum Srinivas

Moreover, I found the interesting topics on Google Android browsing Google Groups, and, of course, using Google search by itself.