by Peter Peerdeman & Timen Olthof
http://www.niftysystems.nl

Abstract

In this project we will look into the development options that the Android platform has to offer. Using the Google SDK, it is possible to develop small example widgets and applications for the mobile Android platform, allowing us to explore the technological connectivity possibilities of today's high end smartphones. In this document we describe our experience with the Android platform, its possibilities and limitations and discuss the example programs and widgets we created.

1 Android Platform

On the 21st of October 2008 Google launched their own operating system for mobile devices (both mobile phones and slightly larger devices such as internet tablets), called Android[1]. It promises to change the mobile experience by offering the user multiple homescreens that can be filled with widgets and shortcuts to applications, as well as a stable operating system based on the linux kernel. In addition to this, Google supplies their own Applications for easy GMail reading and Google Maps integration.

During this project, we used Android version 1.5. In the meantime, both Android 1.6 and Android 2.0 have been released, each of these updating the feature set and improving the functionality of the operating system. More information about the new features of these versions can be found in section 6 "Future of Android".

Because the Google Android platform is Java based, Applications and Widgets for Android can be built using any compiler or IDE, although Google recommends the open source Eclipse IDE [2], which supports syntax highlighting and autocompletion as well as easy integration with the specialized debugger and emulator that are supplied by Google in the Google Android SDK package [2]. Finally, Google also set up a web-based developer support center, that provides tutorials, api documentation and architecture overviews.

Providing a clear overview of the Android architecture is very necessary, because the architecture radically differs from conventional architectures to provide better performance on mobile devices. Once deployed, every application will run in its own Java Virtual Machine and in its own linux process, which provides better security and also better schedulability.

The most influential architectural feature of Google Android is that, in principe, you cannot start an application or task by yourself. Instead, it is only possible to create user interfaces that are called Activities, and to make known to the Android system scheduler that you want a certain Activity to be started. The system scheduler then dynamically decides when to execute the Intent, and thereby activate the Activity. Other types of components include Services (which are like Activities without a user interface, providing functions to other classes), BroadcastReceivers (which handle events) and ContentProviders, which expose datastructures.

The biggest advantage of this architecture is that it is very clear in theory, and helps writing consistent and stable applications, while providing schedulability options specifically targeted at mobile environments. The downside however is that every Application or Widget has to be programmed in this Activity/Intent based paradigm, which restricts the programmer in his options.

1.1 Widgets

When thinking of possible applications to develop for the platform we instantly thought about creating widgets that enhance the homescreen of the Android device. App widget are "... miniature application views that can be embedded in other applications (such as the Home screen) and receive periodic updates."[3]. These widgets provide very easy access to information, for the homescreen is immediately displayed when the device is activated. Because of the advanced and reliable internet connection that is available on the majority of currently used devices we can not only convey locally stored information but also show information from the web. This opens up a lot of possibilities to stay connected with certain online services.

Several very useful widgets are already packaged with the Android operating system. These widgets range from local phone driven data widgets such as clocks, calendars, music players and picture frames to internet connected widgets such as the Google search bar. HTC, builder of the first three Android devices, has developed several more widgets that are supplied with their last phones. Most of these widgets are internet data driven widgets such as a stocks display, a twitter widget and an internet bookmarks widget.

Because of the intensive daily use of the Android phones homescreen and the limited number of publicly available widgets we have decided to focus a big part of our Android development on widgets. We think if widgets were to evolve into truly compact and specifically goaled mini applications there would eventually be no need to intensively use full applications as we do now. If every user could just simply pick out the six tasks he or she uses on his or her phone there would be no need for menus, dialogs or launchers whatsoever. This would give a much better feature overview and improve / speed up the access to commonly used tasks and information, following the statement written in the renowned computer science literature: "make the common case fast"[4].

In practice, implementing widgets is like folding origami with your elbows. Though there is a large set of perfectly fine controls available (your hands) you are restricted to only the boldest of controls such as images and labels (elbows). This makes widgets perfect for displaying up to date information from some data source on the phone or a data source from the web, but creating interesting interactions with a widget like swipe-scrolling or accepting user input text is nearly impossible. The best thing we can do is generate a new update according to a users action and present another static screen to the user.

Updates in widgets are realized through *remoteViews*. A remote view's layout is generated from a static XML layout after which the text and image fields can be assigned with available data for that specific update. This makes it very hard to introduce complex interactions such as swipe-scrolling, though the phone company HTC seems to have been able to achieve this in their phone-bundled widgets (the twitter widget and the contacts widget). Next to this, we found out that implementing a user input box like the "google search widget" is also not available for developers. This resulted in a great deception which led us as programmers to believe it was easy to achieve such interactions, moreover because a perfectly functional, scrollable list view and user input box are available in full size application development which offers these interactions as standard controls.

These pesky limitations make it very unlikely for programmers to get enthusiastic about developing widgets for android, even though the idea behind home screen widgets is so refreshing and useful. We guess that limitied processor power and security issues are the main drivers behind this design decision.

1.2 Applications

Because Widgets are always running, they constantly occupy scarce system memory. Applications differ from Widgets in the sense that they have to be launched before usage, and as such need some time to initialize. This makes applications less ideal for real time data driven tasks such as communication services (Twitter, Facebook etc.), but better suited for ‘standalone’ functionality that uses much memory, such as games and productivity applications.

While an Android application shows up as a single entity in the Android OS (including launcher icon etc.), it internally often consists of multiple separate parts called Activities. These Activities are scheduled separately by the system scheduler, and as such provide the system with more stability and responsiveness than a ‘normal’ single-class application would have provided.

Applications are made available through the Android Marketplace, which is the Android equivalent of a linux package management system. In this application, other applications can be browsed, downloaded and installed. This is one of the key strategies to create a versatile and broadly useful device, offering easy access to thousands of applications developed by developers all over the world. Most of these applications are offered for free, though some corporate developed apps have small fees of approximately 1 to 10 euros.

Developing applications for Android and testing them on the emulators and real devices can be done without any charge, though publishing your applications to the Marketplace requires you to register yourself as an Android Developer. This membership costs 25 dollars, but will allow you to publish your apps to the Marketplace, allow you to receive feedback about your application and allow you to post updates of your application that make it easy for end-users to keep your deployed application up to date.

2 Workflow

A big plus of developing for Android is that the workflow is very easy to set up. The preparations needed to get started with programming is downloading the SDK package from the android development website[1], installing the Eclipse IDE[2] and installing the Android ADT plugin in Eclipse[3]. After these installation procedures, one can start a new Android project from the eclipse environment, copy and paste the example code and immediately run the program on the supplied emulator.

In addition to the Eclipse environment we found it very useful to synchronize our source files with the versioning system Subversion (SVN). The plugin for Eclipse named Subclipse[4] makes versioning and sharing of the Android amongst project members very easy. Not only does this backup your files to a safe server with version control, it also allows other fellow team members to view and adjust the code.

Whenever an Android application is built and launched in the emulator, an *.apk* file is created and pushed to the virtual Android device. The *.apk* file is essentially a zipped binary package containing all the resources and executables needed for an Android device to run the application. This makes it very easy to publish your program and test it on real life devices. We simply uploaded the *.apk* files to a website and downloaded the files to our real Android devices to test the programs in a real life environment. This was mainly used to show off the project to other people when we can't use the emulator, because the functionality of the emulator is truly excellent.

2.1 Project structure

Android Applications (and Widgets) consist of three important parts: program code, resources and the Android manifest file.

Code Program code consists of all Java classes created by the developer (stored in the *src/* folder) as well as a resources indexing class named R.java that is stored in the*gen/* folder. In case of an application, the main class usually extends the Activity class, while a few support classes like widgets and for example extra threads are used to abstract program functionality.

[1]Android Development Website:`http://developer.android.com/sdk/`
[2]Eclipse: `http://www.eclipse.org/downloads/`
[3]Installing Android ADT plugin`http://developer.android.com/sdk/1.6_r1/installing.html`
[4]Subclipse plugin `http://subclipse.tigris.org/`

In case of a widget, the main class usually extends the AppWidgetProvider class, and additional Services classes are used to keep the widget up to date.

Resources A resource is any non-code part of the application. This includes not only images (PNG or JPEG) but also layouts specified in Android's special XML based layout format, regular XML files such as settings files, animations (image sequences or tweening animations) and values. These are stored in *res/drawable/*, *res/layout/*, *res/xml/*, *res/anim/* and *res/values/* respectively. Values are data structures like arrays, colors, dimensions, strings and styles. Finally, any other type of non-code resource can be added to the *res/raw/* folder. Resources in this folder are added to the application in raw format, in contrast to the other resources, which are compiled to conserve space and may be encoded to optimized formats.

Manifest The manifest file (AndroidManifest.xml) is what makes the application accessible to the device it is run on, so it is recognized as a valid application. The manifest file contains the most important properties of the application (or widget) as a whole, such as its packaging, available components and Android SDK API version. The manifest file is also used to list which system privileges are needed for the application to run properly. This is done using a list of boolean permission constants. When such a permission constant (for example *android.permission.INTERNET*) is included in the manifest file, the user is prompted to allow the application this permission during the installation.

3 How to get started

Because there are already very good tutorials available online, we chose to give an overview of useful tutorial resources instead of repeating that what has already been said. Writing yet another basic tutorial seemed pointless, as it would become very much like the Google introduction tutorials.

Google Android Developer tutorials The best place to start is the Google Android Developers website. It contains a number of very clear basic and intermediate tutorials.

`http://developer.android.com/guide/tutorials/hello-world.html`

AndDev.org AndDev.org is a good website/forum to search for solutions to specific problems. The forum is not that well organized, but contains very much information.

`http://www.anddev.org/`

Introducing home screen widgets A short tutorial that covers all of the widget specific programming aspects in Android resulting in a nice and small webservice display widget.

`http://android-developers.blogspot.com/2009/04/introducing-home-screen-widgets-and.html`

Blogoscoped A very well-written tutorial on using gps and the MapView.

`http://blogoscoped.com/archive/2007-11-19-n27.html`

IBM A more theoretical tutorial by IBM.

`http://www.ibm.com/developerworks/opensource/library/os-android-devel/`

4 Mashup Applications and Widgets

4.1 Hello World application

The Hello World application was the first application we "developed" to get the workflow started. It is based on the first Google Android tutorial. While the application itself is not that functional, the tutorial code that is used to produce the application covers all of the basics of Android programming, including the separated layout files in the resources folder, creating an activity, using the layout files to create a new view and editing the manifest file. These are very valuable first steps to start developing for Android.

4.2 Public Timeline Widget

The first widget we built is the PublicTimelineWidget, which essentially displays the latest “tweet” (a short message posted by a twitter user) from Twitter. The application is based on the Wiktionary Homescreen Widget example by Jeffrey Sharkey[5]. In this widget we set up a service that makes a service request at a predetermined interval to the twitter api. This service request asks the API for the latest “tweets” in the public timeline and returns a JSON document containing this data. After the request the JSON file is parsed using the JSON libraries of the Android framework. The parsed data is then formatted into the layout and displayed to the user. The user manually makes a new request by pressing the widget which will create a new API request.

4.3 Contact Carousel Widget

The contact carousel widget was one of the first ideas we had for development. The plan was to create a smooth sliding gallery of contact photos which would call the contact when clicked. In retrospect this was by far the worst widget to develop in the beginning of the project because it simply involves too much different aspects of application development.

Firstly we were faced with the retrieval of data from the devices datastore. In the tutorials this seemed easy, but actually retrieving a specific dataset turned out to be hard to learn, especially in the start of the project. Next to this we wanted to use the fancy sliding gallery control which wasn’t available for us to use in the widget itself. This resulted in adjusting the design into a static threefold of the previous contactbutton, the current contacts picture and the next contact button.

Finally we had to implement the next and previous control buttons which turned out to be a lot more of a hassle than we imagined. The specific code example will be discussed in section 5.1, but it basically drills down to an immensely cluttered architecture to perform a very simple task such as handling a button press. In retrospect we found out that we could have solved this using broadcasts and broadcast control action handlers, but we decided not to rewrite the application for it would not change the user experience but just cost us more time to create a better implementation.

4.4 9292ov Widget

Because we are interested in location based services and applications, we tried to create a widget that would access the 9292ov API to give an end-user easy access to real time public transport information. However we stumbled upon some problems that impeded us from finishing this mashup.

The first problem was that the 9292ov API is not publicly exposed like the twitter API, but that it is restricted to registered developers only. Because we knew that there was a programming contest a while back that used the API, we decided to contact 9292ov to see if we could get the credentials needed to create an application that could use their services. After calling 9292ov about our request and sending an initial email to the public email adress we were asked to redirect our request to someone within the company that should know more about the API. After emailing this person we explained our situation but before we could get access we would have to arrange a meeting to discuss our request.

Several emails later we got an appointment with 9292ov in their main building, which was cancelled the night before the actual meeting. At this point we decided to divert our attention to the other ideas we had for development which would not be restricted by formal requests to third parties. This is really a shame because in this way formalities get in the way of the creative process, bringing development to a grinding halt.

For the same reason, we were unable to create an application/widget which shows the next bus or train on a preselected line. This application is useless without up to date public transport information.

[5]wiktionary-android, a Wiktionary home screen widget for Android by Jeffrey Sharkey. http://code.google.com/p/wiktionary-android/

We considered creating an application with built in public transport data, but that would require the user to input the schedules or even require the application to be recompiled every time the data changes, which is useless and bothersome in practice.

4.5 XboxLive Stats Widget

The widget that best fits our vision on the power Android widget development is the xbox live stats widget. This widget queries a web service for a certain Xbox Live Gamertag[6], that is supplied by the user the first time the widget is loaded. The web service then responds with an XML document containing all the associated information with this Gamertag such as the amount of Gamerpoints, information about the last game that the user played, the current online status and an url to the users avatar picture. This data is parsed by the widget using the SAXParser[7] and displayed to the user.

When the widget is added to the homescreen and the Gamertag is assigned, the widget reloads its information each 5 minutes. Because the widgets are reloaded seperately, it is possible for the user to add multiple instances of the widgets to the homescreen, each assigned with a different Gamertag. In this way, a gamer can keep track of his own progress and the progress of his or her online friends by simply looking at their phones homescreen.

Unfortunately, during the final weeks of our project the webservice, which is located in the U.S.A. started to collapse, taking very long to respond to our requests or even ignoring them completely, causing the widget to crash. This shows that developers using webservice API's to construct their applications are very dependant on the stability of the specific webservice.

4.6 Wikipedia Search Widget

When browsing for available API's to display interesting data in widgets we stumbled upon the Wikipedia API which is a stable and reliable, publicly exposed service to get information about any possible topic. The first idea for this widget was to create a user input box in the homescreen in which a query for Wikipedia information could be made in a similar fashion as the google search widget that is already available.

Unfortunately we found out that the user input box control is not available in widget development and the Wikipedia search widget part of our project was put aside and flagged as "failed due to framework restrictions". However, after testing some other widgets that were available in the marketplace we found out that it is possible for a widget to launch a configuration screen, which is an implementation of an Activity instead of an AppWidgetManager like the widget itself. The advantage of an Activity is that all of the Android controls are available, which provides us a with an extra user input vector for out widgets.

This reopened the possibility of creating an implementation of the Wikipedia widget, which we embraced and tried out. The result is a widget displaying its information quite alike the public timeline widget, but opens a configuration window when clicked which offers the user an input box in which the search query can be inserted, along with two language radiobuttons to allow the user to search for either Dutch or English results. This could be extended in for example a dropdown menu containing all the possible Wikipedia languages but we chose for simplicity as this is the main goal of widgets in general.

The downside of this user experience approach is that the user has to click the widget before we can accept the user input which is an extra barrier for the user to start an interaction. However, because we don't clutter the homescreen with another input box we can focus our widget space on the information that we want to display, which is the found Wikipedia information itself. By leaving the searched Wikipedia information on the screen we can even help the user remember or learn a certain term because the last search query remains active in the screen.

While implementing the API service request we found out that it is possible to query the service for different file formats, including XML, JSON and plaintext. We decided to choose the JSON standard

[6]A Gamertag is an online persona over the popular Xbox LIVE feature on the Xbox 360 game console.

[7]SAXParser http://developer.android.com/reference/javax/xml/parsers/SAXParser.html

because the Android JSON parser is very easy to use and as JSON uses less bandwith than the other standards it should be the perfect candidate. However, we found out that the JSON formatted result doesn't return the same results as the XML feed, which we needed in our application. In the end we have used the XML feed and parsed the data using the DocumentBuilderFactory, which used less code than the SAXParser we used in the xbox live stats widget but is a little harder to use.

4.7 Game application

Next, we experimented with gaming on Android. To get started we based our Game on an adapted version of the LunarLander tutorial application (that can be found in the Google Android tutorials). We changed this application to contain ball sprites (abstracted into a GameObject class, so more balls or other game objects can easily be added later on). Afterwards, we added a basic physics system, which we also improved to have the balls not only collide with the screen boundaries, but also with each other. However, this physics system is not perfect in the sense that sometimes multiple collisions take place in the same physics iteration, leading to complex collisions and unpredictable behaviour. This problem may be due to the fact that the CPU speed isn't able to handle the physics updates fast enough, so maybe a specialized optimized physics algorithm is needed.

4.8 Other unused ideas

Other application and widget ideas we considered, but not realized, are the usage of the Flickr API (there are already a lot of Flickr Applications) and the Facebook API (there is an official Facebook application, but the Facebook Widget is not that great). In addition to this we thought of using other API's to show Bungie (the creator of the Halo game series) user statistics, show the track&trace information of UPC/FedEx/TNT parcels, and for example use some flight schedule API to display flight status updates. We also wanted to create an application that uses 3D graphics, audio or 2D generated graphics/art. Android supports OpenGL to provide 3D rendering, but we eventually decided that pursuing this option would be outside the scope of our project. Unfortunately Android does not (yet) have an audio API, so generating or manipulating sound is not possible. Samples can be loaded and played, but mixing different samples would most likely lead to timing issues. In the Android Market, we could find only one application that really makes optimal use of the limited audio features that Android offers at this time [5]. Generative 2D graphics or generative art should be possible in Android, Finally we tried to think of ways to connect Android to our previous projects, PowerRail [6] and tOMS [7], but decided to focus on the Android core itself.

5 Problems and Opportunities

Like any development platform, the Android environment has both strong and weak points. In this section we sum up our development experience to shed some light on these points.

5.1 Development Experience

While Android is a typically embedded platform (focussed on speed, stability and correctness) the entry requirements for developing an Android application or widget are fairly low, because Java is a well established programming language with a clear syntax. The Android system architecture takes some time to get used to, but it is not a problem when creating basic applications like the ones described by the Android Developer Documentation Tutorials. This results in your first applications being created before you know it, because Google does not charge any money to just run a custom build application on your own phone, unlike Apple does. Without becoming an Apple developer you can only use the emulator. The environment is also unrestrictive in this sense. The different features of Applications and Widgets work very well within their own context. Custom developed applications fit seamlessly into the Android OS, that is itself very stable as well. But here is the main problem: the Android architecture 'works' well as long as you build applications that are similar to the example applications, or at least fit into the same architectural design. If, however, you want to create applications that go beyond this standard architecture, like we did, things get unnecessarily complicated. For example, because we think that the use of Widgets for repetitive small tasks is a very good concept, we wanted to create very small widgets suited for specific tasks. This didn't work out as well as we hoped it would.

For example, certain simple tasks such as adding an eventlistener for an onclick event on a certain control would in an Android application typically be handled in the following piece of code:

```
leftClickButton = (Button) findViewById(R.id.button_left);
leftClickButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        currentId--;
    }
});
```

In this snippet we retrieve a reference to the leftclick button through the resources id (R.id) and then use the setOnClickListener method to add an inline defined function to decrease the counter that is used to display the current contact in the contact carousel. However, when programming Android widgets we can't dynamically retrieve references to buttons, because these have to be predetermined in the RemoteView that we create for a certain widget refresh. The following code shows the amount of code needed to achieve the same result in a widget as we did in an application before.

```
Intent leftClickIntent = new Intent(context, previousContact.class);
PendingIntent pendingleftClickIntent = PendingIntent.getBroadcast(context,
    0 /* no requestCode */, leftClickIntent, 0 /* no flags */);
updateViews.setOnClickPendingIntent(R.id.button_left, pendingleftClickIntent);

static public class previousContact extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        currentId--;
        AppWidgetManager gm = AppWidgetManager.getInstance(context);
        int[] appWidgetIds = gm.getAppWidgetIds(new ComponentName(context,
            ContactCarouselWidget.class));

        final int N = appWidgetIds.length;
        for (int i=0; i<N; i++) {
            ContactCarouselWidget.updateAppWidget(context, gm,
                                                     appWidgetIds[i]);
        }
    }
}
```

In this piece of code we see that we can't just add an eventlistener to a button, we have to create a `leftClickIntent`, which is based on the class `previousContact`. This intent is transformed into a "Pending Intent" which can in turn be attached to the button that we retrieve using the same resources id as before.

In the `previousContact` class we override the method `onReceive` which is called whenever the button is pushed. In this method we decrease the same counter as before, but because we have to manually prepare the next widget update, we have to retrieve the `AppWidgetManager` and the `AppwidgetIds` in order to manually call the `updateAppWidget` method for every widget that we have created. Having cluttered architectures such as these make it very hard to create fast mashups because a lot of architecture overhead is needed to perform very simple tasks.

A second limitation of the Android environment is the fact that it is a relatively new platform. Therefore a lot of functionality has not been implemented yet. For example Bluetooth file transfer has only recently been implemented in Android version 2.0, for which the SDK is now available. However, there are no devices available that use Android 2.0, and most devices haven't even made the transition from 1.5 to 1.6 yet. An audio generation/manipulation API is also not available, and has not been announced. Another limitation is the fact that most Android devices that are currently available have a limited CPU speed. In most cases the CPU is just fast enough to run the OS and basic applications smoothly. Whether these CPU's will be able to cope with more advanced multimedia applications remains to be seen.

All in all we think Android is a great platform, with a few peculiarities. It is accessible, open, very usable, fast and has a very good yet restrictive architecture.

5.2 Development Alternatives

While the Google Android SDK provides the developer with tons of options and a very clear, yet rigid, architectural framework, there are a few other options to consider when developing applications for the Android operating system. These alternatives differ in the sense that they try to be cross platform, so that one application can be developed to work on multiple mobile environments like Android, iPhone, Windows Mobile and others.

First there is PhoneGap, which aims to be "an open source development tool for building fast, easy mobile apps with JavaScript."[8]. The unique feature of PhoneGap is that it allows the developer to create his or her application once using well known programming languages like HTML/JavaScript/CSS, and then compiles this application into platform optimized applications for iPhone, Android and BlackBerry. PhoneGap provides a universal JavaScript API that exposes specific device functionality like Contacts, GPS, Vibration, Accelerometer and Sound in a platform independent way.

Another development alternative is to just use regular HTML/JavaScript/CSS and for example a PHP/MySQL web server back end to build mobile optimized web applications. Of course in this case the developer doesn't have access to device specific functionality, and additionally an internet connection will be required during the usage of the application. This connection requirement will also increase the latency and therefore make the program less responsive. However, it is probably the easiest and most cross-platform way to create mobile applications. With the impressive new capabilities of HTML5 such as the Canvas and Video tags, Web based applications are a strong rival for the Android SDK applications.

A third alternative is Flash. While Apple is rejecting any form of Flash on the iPhone, Google does allow and support the use of Flash on its mobile platform. Currently the implementation of the Flash 10 VM is largely working in Android 1.5. Basic Flash applications like banners and ads are working very well. More complex interactive applications are still a bit buggy, but Adobe is working hard to improve this. If, in the future, Flash will be completely supported and maybe even include access to device specific functionalities, Flash could become a very powerful mobile platform on top of Android.

6 Future of Android

During the 2,5 months of project time in which we explored the functionalities of Android 1.5, two new versions of the Android operating system were released, version 1.6[8] and more recently Android 2.0[9]. This shows that the Android system is evolving rapidly and is still under heavy construction. Several interesting new functionalities have been implemented in these new versions that we would like to point out, for possible future research / development on this system.

One of the new features in the 1.6 version of Android is the quick search box, which extends the google search box to search for local sources as well such as text messages, contacts and music. In addition to this, third party developed applications can now be adapted to be searchable by the search box as well, which makes third party data driven applications even more accessible.

[8]Android 1.6 http://developer.android.com/sdk/android-1.6.html

[9]Android 2.0 http://developer.android.com/sdk/android-2.0.html

Another interesting new feature is the rise of gestures and multi touch controls. In version 1.6 gestures have been added that can control an application by performing certain gestures on the screen such as swipes and drags across the screen. Full multi touch support has been added in Android 2.0 which allows a developer to get touch information from up to three pointers simultaneously for devices that support it.

One new feature that open up a lot of possibilities is the upgrade of the Bluetooth stack and API, which in Android 2.0 is now capable of turning Bluetooth on and off, discovering devices and services, connecting to these devices and transmitting data through connections. Though this functionality is not new in the handset world, the addition proves that Android is doing the best it can to become the best mobile operating system on the market.

References

[1] Android.com. Android is now available as open source. `http://source.android.com/posts/opensource`, October 2008.

[2] Android.com. Download the android sdk. `http://developer.android.com/sdk/index.html`, October 2009.

[3] Android.com. App widgets. `http://developer.android.com/guide/topics/appwidgets/index.html`, October 2009.

[4] John L. Hennessy and David A. Patterson (A. *Computer Architecture: A Quantitative Approach, Second Edition.* MacGraw-Hill, 1996.

[5] Pedro J. Estbanez. Robotic guitarist. `http://nl.androlib.com/android.application.com-pedrocorp-android-guitar-qqBD.aspx`.

[6] Niftysystems.nl. Powerrail. `http://www.niftysystems.nl/powerrail`, October 2008.

[7] Niftysystems.nl. the object management system. `http://www.niftysystems.nl/toms`, June 2009.

[8] Phonegap.com. What is phonegap? `http://phonegap.com/`.