**Introduction**

**Project Overview**

The objective of City Event Management (CEM) is to guide the user of this application to get the details of the RUNNING EVENTS in the current city he is residing in.

City Event Management is an application which could be used on any Android Mobile phone. It is to keep a complete list of the EVENTS, RUNNING EVENTS like College Fest’s, Movie Release, Audio Launching Functions, etc. and all sorts of things.

This application not only provides the details of the events but also helps to register to those events anywhere and anytime by connecting to a WI-FI or network on your Android Phone. Registration fees can be paid online through a special gateway which is secure and reliable.

Using this application the user can even get the route map of the Event Place using the features of GPS (default on Android) and Google Maps, when connected to a network. Finding the location of the event is very easy using this Android application.

**Scope**

The scope of the project includes the development of a mobile app for the Android operating system and the installation of this application on the phone. The functionality of the application includes the following features

1. **Date based search**: This helps in searching for an Event with respect to DATES.
2. **Event based exploration**: This is a specific search which is useful when the user is completely aware of the ailment.
3. **Register to an Event**: This feature enables user to register to the event he wishes to.
4. **Route Maps:** This feature helps the user to get the directions of the respective event on his/her Android mobile phone.
5. **Registration Details:** This feature is provided to the EVENT ORGANISER where we are providing a list of registrations to a particular event. This will help the organizer to make a list of the people attending the event.

**Purpose**

The purpose of the project is to showcase the use of the Android operating system to build useful applications for the users so that they can access information at any time on their mobile phone. We also intend to showcase the power of the android application in displaying information on the phone and the user to be able to access the information at any point of time anywhere in the city.

**Software requirements analysis**

**Existing System**

The existing system occurs as an online web application. The drawback with the existing system is that in case of an emergency we may not be able to access a computer and search for the relevant data. There is no application available on the mobile phone to show this same information to the user.

**Feasibility Analysis**

The technical feasibility of the project was conducted after the requirements were collected. The development environment is well established and easy to use. The development environment and the Android Operating System are all in free source and can be setup on any desktop hardware with minimal configuration.

**Hardware and Software Requirements**

**Software requirements:**

1. JDK 5 or 6
2. Android SDK
3. Eclipse IDE
4. ADT plugin for Eclipse
5. Operating Systems
   1. Windows XP (32-bit), Vista (32- or 64-bit), or Windows 7 (32- or 64-bit)

**Hardware requirements**

1. Processor Core 2 duo
2. RAM 2 GB
3. Hard disk 50 GB

|  |  |  |
| --- | --- | --- |
| Component type | Approximate size | Comments |
| SDK Tools | 35 MB | Required. |
| SDK Platform-tools | 6 MB | Required. |
| Android platform (each) | 150 MB | At least one platform is required. |
| SDK Add-on (each) | 100 MB | Optional. |
| USB Driver for Windows | 10 MB | Optional |
| Samples (per platform) | 10M | Optional. |
| Offline documentation | 250 MB | Optional. |

**About Android Operating System**

**Introduction**

Android is a software stack for mobile devices that includes an operating system, middleware and key applications. Android is a software platform and operating system for mobile devices based on the Linux operating system and developed by Google and the Open Handset alliance. It allows developers to write managed code in a Java-like language that utilizes Google developed Java libraries, but does not support programs developed in native code.

The unveiling of the Android platform on 5 November 2007 was announced with the founding of the Open Handset Alliance a consortium of 34 hardware, software and telecom companies devoted to advancing open standards for mobile devices. When released in 2008, most of the Android platform be made available under the Apache free-softy;-arc and open-source license.

**Features**

**Application Framework**

It 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, SQLLite 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 law-memory based virtual machine, which was designed especially 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 past processing. Conversion between Java classes and .DEX format is done by included "dx" tool.

**Integrated Browser**

Google made a right choice on choosing WebKit as an 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.

**Optirnized 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 Limn version. At this moment, the shooting legendary 3D game Doom was presented using Android on the mobile phone.

**SQLite**

Extremely small (~500 kb) 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.

**Handset Layouts**

The platform is adaptable to both larger, VGA, 2D graphics library, 3D graphics library based on OpcnGL ES 1.0 specifications, traditional smart phone layouts. An underlying 2D graphics engine is also included. Surface Manager manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications.

**Data Storage**

SQLite is used for structured data storage. SQLite is a powerful and lightweight relational database engine available to all applications.

**Connect**

Android supports a wide variety of connectivity technologies including GSM, Bluetooth, EDGE, EVDO, 3G and Wi-Fi

**Messaging**

SMS, MMS and XMPP are available forms of messaging including threaded text messaging.

**Web Browser**

The web browser available in Android is based on the open-source WebKit application framework. It includes LibWebCore which is a modern web browser engine which powers both the Android browser and an embeddable web view.

**Java Virtual Machine**

Software written in Java can be compiled into Dalvik bytecodes and executed in the Dalvik virtual machine, which is a specialized VM implementation designed for mobile device use, although not technically a standard Java Virtual Machine.

**Media Support**

Android will support advanced audio video and still media formats such as MPEG-4, H.264, MP3, and AAC, AMR and JPEG, PNG, GIF.

**Additional Hardware Support**

Android is fully capable of utilizing video/still cameras, touchscreens, GPS, compasses, accelerometers, and accelerated 3D graphics.

**Operation**

**Android Runtime**

Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language. Every Android application runs in its own process with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently.

The Dakik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included "dx" tool. The Dalvik relies on the Linux kernel for underlying functionality such as threading and low-level memory management.

**Linux Kernel**

Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.

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.

Developers have full access to the same framework APIs used by the core applications. The application architecture is designed to simplify the reuse of components; any application can publish its capabilities and any other application may then make use of those capabilities (subject to security constraints enforced by the framework). This same mechanism allows components to be replaced by the user. Underlying all applications is a set of services and systems.

**Architecture**

The following diagram shows the major components of the Android operating system. Each section is described in more detail below.



Architecture of Android

**Linux Kernel**

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
* Camera up to 6.0 megapixels
* gpsOne — solution for GPS
* and lots of other

**Libraries**

In the next level there are 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 arc 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 PocketVideo, 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 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.

**Android Runtime**

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 which converts generated byte code from .jar to .dex file, after this byte code becomes much more efficient to run on the small processors.



Conversion from .jar to .dex file

As the result, it is possible to have multiple instances of Dalvik virtual machines 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.

**Application Framework**

After that, there is 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.

The Activity Manager manages the life cycle of the applications and provides a common navigation back stack 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.

**Application Layer**

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. All applications are written using the Java programming language.

**Developing applications**

**Application Building Blocks**

We can think of an Android application as a collection of components, of various kinds. These components are for the most part quite loosely coupled, to the degree where you can accurately describe them as a federation of components rather than a single cohesive application.

Generally, these components all run in the same system process. Its possible (and quite common) to create multiple threads within that process, and its also possible to create completely separate child processes if you need to. Such cases are uncommon, because Android tries very hard to make processes transparent to your code.

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 application into the components, 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***

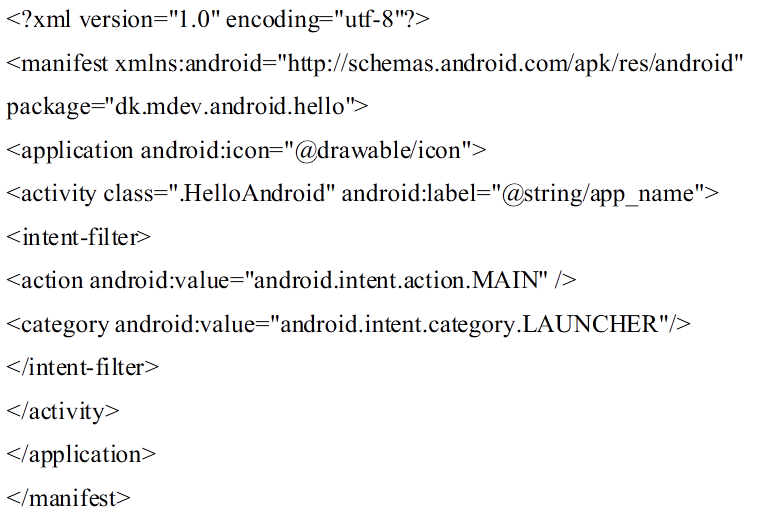
A component, which allows sharing some of the data other processes and applications. It is the best way to communicate the applications between each other. Android will ship with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications arc written using the Java programming language.

***Android Manifest.xml***

The Android Manifest.xml file is the control file that tells the system what to do with all the top-level components (specifically activities, service, intent receivers, and content providers described below) you've created. For instance, this is the ‘glue’ that actually specifies which Intents your Activities receive.

A developer should predefine and list all components which he wants to use in the specific Android Manifest.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 a sample of Android Manifest.xml file:



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 specifics 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.

**Application Lifecycle**

In Android, every application runs on 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.

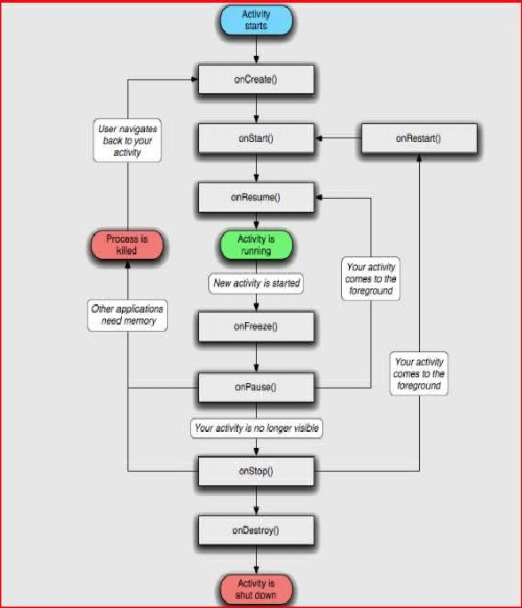
It is important that application developers understand how different application components (in particular Activity Service, and Broadcast Receiver) impact the lifetime of the application's process. Not using these components correctly can result in the system killing the application's process while it is doing important work.

To determine which processes should be killed when low on memory, Android places each process into an "importance hierarchy" based on the components running in them and the state of those components. These process types are (in order of importance).

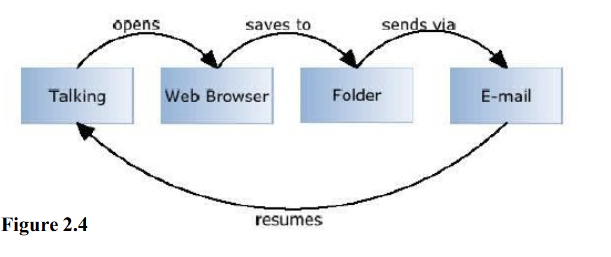
1. A foreground process is one that is required for what the user is currently doing. Various application components can cause its containing process to be considered foreground in different ways. A process is considered to be in the foreground if any of the following conditions hold:
   1. It is running an Activity at the top of the screen that the user is interacting with (its onResume() method has been called).
   2. It has a BroadcastReceiver that is currently running (its BroadcastReceiver.onReceive() method is executing)
   3. It has a Service that is currently executing code in one of its callbacks (Service.onCreate(), Service.onStart(), or Service.onDestroy()).

There will only ever be a few such processes in the system, and these will only be killed as a last resort if memory is so low that not even these processes can continue to run. Generally, at this point, the device has reached a memory paging state, so this action is required in order to keep the user interface responsive.

1. A visible process is one holding an Activity that is visible to the user on­screen but not in the foreground (its onPause() method has been called). This may occur, for example, if the foreground Activity is displayed as a dialog that allows the previous Activity to be seen behind it. Such a process is considered extremely important and will not be killed unless doing so is required to keep all foreground processes running.
2. A service process is one holding a Service that has been started with the startService() method. Though these processes are not directly visible to the user, they are generally doing things that the user cares about (such as background mp3 playback or background network data upload or download), so the system will always keep such processes running unless there is not enough memory to retain all foreground and visible process.
3. A background process is one holding an Activity that is not currently visible to the user (its onStop() method has been called). These processes have no direct impact on the user experience. Provided they implement their Activity Life-cycle correctly (see Activity for more details), the system can kill such processes at any time to reclaim memory for one of the three previous processes types. Usually there are many of these processes running, so they are kept in an LRU list to ensure the process that was most recently seen by the user is the last to be killed when running low on memory.
4. An empty process is one that doesn't hold any active application component. The only reason to keep such a process around is as a cache to improve startup time the next time a component of its application needs to run. As such, the system will often kill these processes in order to balance overall system resources between these empty cached processes and the underlying kernel caches.



In the following example we 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.

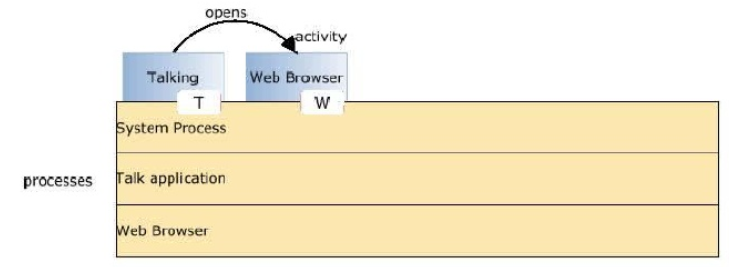


In this situation, there are 4 different applications and 4 different processes running, but from the user point of view none of them are important, as Android manages CPU work and memory usage by itself. It means the user can travel through the applications forward and back without thinking about how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

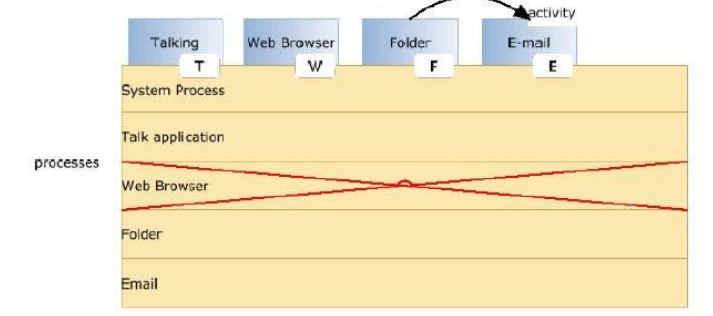


At this point, as a user holds a talk and opens a web browser, the system creates a new process and new web browser activity is launched in it. Again, the state of the last activity is saved (W):

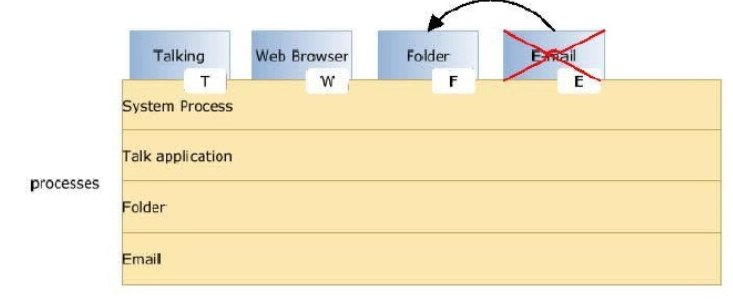
After that, the user browses the internet, finds his picture in Picasa album and saves it to particular folder. He does not close a web browser, instead he opens a folder to find saved picture. The folder activity is launched in particular process:



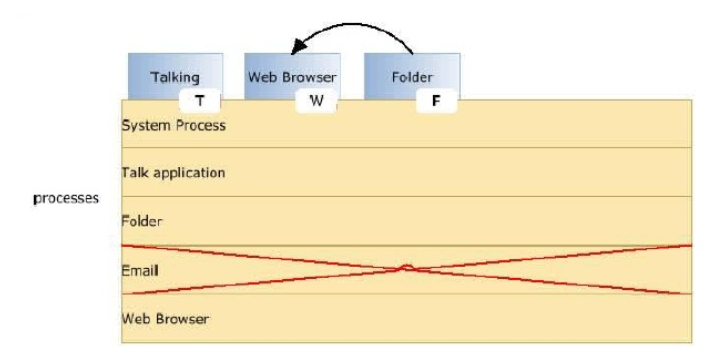
At this point, the user finds his saved picture in the folder and he creates a request to open an Email application. The last state F is saved. Now assume that the mobile phone is out of the memory and there is no room to create a new process for Email application. Therefore, Android looks to kill a process. It cannot 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:



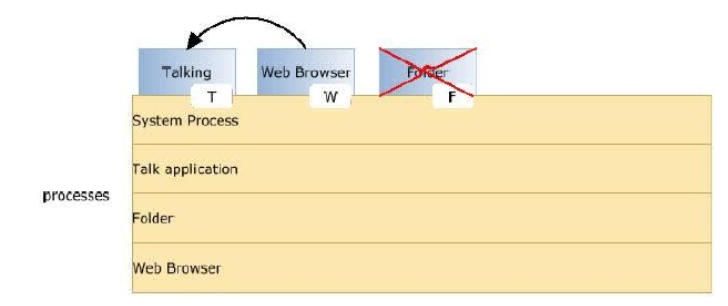
The user opens Email application and sends a picture to his friend via email. Now he wants to go back to the Talk application and to resume a talk to his friend. Because of the previously saved states, this work is done fast and easily. In this example, Email application is popped out and the user sees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

**Application Framework**

Developers have full access to the same framework APIs used by the core applications. The application architecture is designed to simplify the reuse of components; any application can publish its capabilities and any other application may then make use of those capabilities (subject to security constraints enforced by the framework). This same mechanism allows components to be replaced by the User.

Underlying all applications is a set of services and systems, including:

1. A rich and extensible set of Views that can be used to build an application, including lists, grids, text boxes, buttons, and even an embeddable web browser
2. Content Providers that enable applications to access data from other applications (such as Contacts), or to share their own data
3. A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files
4. A Notification Manager that enables all applications to display custom alerts in the status bar
5. An Activity Manager that manages the life cycle of applications and provides a common navigation back stack

**Library**

* Android includes a set of C/C++ libraries used by various components of the Android system. These capabilities are exposed to developers through the Android application framework. Some of the core libraries are listed below:
* System C library - a BSD-dcrived implementation of the standard C system library (libel, tuned for embedded Linux-based devices
* Media Libraries - based on Packet idea's OpenCORE; the libraries support playback and recording of many popular audio and video formats.
* as well as static image files, including lvlPEG,I, H.264, MP3, AAC, AMR, JPG, and PNG
* Surface Manager - manages access to the display subsystem and seamlessly composites 21) and 3D graphic layers from multiple applications.
* LibWebCore - a modern web browser engine which powers both the Android browser and an cmbcddablc web view
* SGL - the underlying 2D graphics engine
* 3D libraries - an implementation based on OpenGL LS 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

**Comparison**

**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 Goode Android. This ensures equal development time for the networks, GSM side and CDMA. 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.

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. 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 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 conies 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 he 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:

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, The more issues regarding Gpay are yet to be released.

**What makes Android special?**

There are already many mobile platforms on the market today, including Symbian, iPhone, Windows Mobile, BlackBerry, Java Mobile Edition, Linux Mobile (LiMo), and more.

While sonic of its features have appeared before, Android is the first environment that combines:

A truly open, free development platform based on Linux and open source. Handset makers like it because they can use and customize the platform without paying a royalty. Developers like it because they know that the platform has legs" and is not locked into any one vendor that may go under or be acquired.

A component-based architecture inspired by Internet mash-ups. Parts of one application can be used in another in ways not originally envisioned by the developer. You can even replace built-in components with your own improved versions. This will unleash a new round of creativity in the mobile space.

Tons of built-in services out of the box. Location based services use GPS or cell tower triangulation to let you customize the user experience depending on where they are. A full-powered SQL database lets you harness the power of local storage for occasionally connected computing and synchronization. Browser and Map views can be embedded directly in your applications. All these built-in capabilities help to raise the bar on functionality while lowering your development costs.

Automatic management of the application life cycle. Programs are isolated from each other by multiple layers of security, which will provide a level of system stability not seen before in smart phones. The end user will no longer have to worry about what applications are active, or close some programs so that others can run. Android is optimized for low-power, low-memory devices in a fundamental way that no previous platform has attempted.

High quality graphics and sound. Smooth, anti-aliased 2D vector graphics and animation inspired by Flash is melded with 3D accelerated Open if\_ graphics to enable new kinds of games and business applications. Codecs for the most common industry standard audio and video formats are built right in, including H.264 (AVC), MP3, and AAC.

Portability across a wide range of current and future hardware. All your programs arc written in Java and executed by Android's Dalvik virtual machine so your code will be portable across ARM,x86, and other architectures. Support for a variety of input methods is included such as keyboard, touch, tilt, camera, voice, and trackball. User interfaces can be customized for any screen resolution and orientation. Android is a fresh take on the way mobile applications interact with users, along with the technical underpinnings to make it possible. But the best part of Android is the software that you are going to write for it. This book will help you get off to a great start.

**Android and Java ME**

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 plug-ins 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, Ul’s, 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 AN 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 J SE); only a subset and therefore it is technically not the Java platform, it just looks a lot like it.

**Openness of The Platform**

The open source school of thought implies that differentiation and competitive advantage come from innovation on top of the underlying platform rather than the platform itself. The robustness and scalability of the platform is secured by the community's stewardship, and open access to a central repository of updated code. Beyond this, a strong third-party development environment and software development kit (SDK) are critical to attracting innovation.

"Open" is an invariably subjective term. Symbian and Microsoft can claim a degree of openness for their mobile platforms, for example, but ultimate control of API access and source code remains with a single entity. On the contrary, as Google has pointed out, there's nothing keeping any of the alliance members from using Android to build a Yahoo! Go phone.

Motorola has had some success delivering high-volume Linux-based devices such as the Ming and RAZR II to market. But mobile Linux initiatives have failed to scale On the basis of attractiveness to third-party developers; it's been supply-push with the development focus in Java ME or other application framework components.

Importantly, Android includes almost the entirety of the applications-related software stack, less key technical pieces such as telephony protocol stacks, which are left to silicon vendors. Android bundles critical components such as a Linux kernel from Wind River, various optimized graphics engines, codecs, notification software, a "clean room" JVM implementation, and the KI-ITML open source browser. The latter forms the basis of Apple's Safari and Nokia's S60 offerings.

**Advantages**

**Open** - Android allows you to access core mobile device functionality through standard API calls.

**All applications are equal** - Android does not differentiate between the phone's basic and third-party applications -- even the dialer or home screen can be replaced.

**Breaking down boundaries** - Combine information from the web with data on the phone -- such as contacts or geographic location -- to create new user experiences.

**Fast and easy development** - The SDK contains what you need to build and run Android applications, including a true device emulator and advanced debugging tools.

**Disadvantages**

**Security** - Making source code available to everyone inevitably invites the attention of black hat hackers.

**Open Source** - A disadvantage of open-source development is that anyone can scrutinize the source code to find vulnerabilities and write exploits.

**Login** - Platform doesn't run on an encrypted file system and has a vulnerable log-in.

**Incompetence** - Google's dependence on hardware and carrier partners puts the final product out of their control.