Location-Based Services Design and Implementation Using Android Platforms

Wen-Chen Hu
Department of Computer Science
University of North Dakota
Grand Forks, ND 58202
wenchen@cs.und.edu

Naima Kaabouch
Department of Electrical Engineering
University of North Dakota
Grand Forks, ND 58202
naimakaabouch@mail.und.edu

Hung-Jen Yang
Department of Industrial Technology Education
National Kaohsiung Normal University
Kaohsiung City, Taiwan
hjyang@nknucc.nknu.edu.tw

Ather Sharif
Department of Computer Science
University of North Dakota
Grand Forks, ND 58202
ather.sharif@gmail.com

Abstract

Mobile computing research evolves constantly and quickly. New mobile devices, technologies, methods, or applications are introduced every day. One of the mobile applications, location-based service (LBS), has attracted great attention recently. A location-based service is a service based on the geographical position of a mobile handheld device. Two of the LBS examples are finding a nearby ethnic restaurant and comparing prices of a product from different stores. Though location-based services are popular, most developers are not familiar with its design and implementation. This paper studies the LBS structure by dividing an LBS system into five components: (i) mobile handheld devices, (ii) positioning system, (iii) mobile and wireless networks, (iv) service providers, and (vi) geographical data providers. Subjects related to LBS implementation including (i) LBS system structure, (ii) handheld computing, (iii) Android application development, and (iv) embedded database SQLite are detailed in this paper. The final section gives a summary of this article and suggests future directions and possible projects of location-based services.

1 Introduction

The number of smartphones shipped worldwide has passed the number of PCs and servers shipped in 2011 and the gap between them is expected to keep bigger. The emerging smartphones have created many kinds of applications that are not possible or inconvenient for PCs and servers, even notebooks. One of the best-seller applications is location-based services (LBS) according to the following market research:

- Fleishman Hillard (2012, February 6) reports 80% of smartphone owners have location-based services and half of them use services that provide offers, promotions, and sales based on their current locations.
- The most convenient mobile shopping experience is price comparison and product research according to JiWire (2011, October 14).
- The number of location-based services users was increased from 12.3 million in 2009 to 33.2 million in 2010 (170% increase) in the US based on SNL Kagan (Cohen, 2011, January 20).

Location-based services are popular, but the details of their design and implementation are still not known for most people. This paper tries to relieve the problem by explaining the LBS design and implementation from a developer perspective. It divides the LBS system into five components: (i) mobile handheld devices, (ii) positioning system, (iii) mobile and wireless networks, (iv) service providers, and (v) geographical data providers. Subjects closely related to LBS including LBS architecture, handheld computing, Android application development, and embedded database SQLite will be detailed in this paper too.

The rest of this paper is organized as follows. Section 2 gives a generic LBS architecture and an LBS example showing how an LBS works. Handheld computing including a generic system structure of mobile handheld devices and client-side handheld computing is introduced in Section 3. Android application development is explained via an example in Section 4. The example includes several XML and Java files and the Eclipse IDE (Integrated Development Environment) with Android plugin and Android DDMS (Dalvik Debug Monitor Server) are introduced in this section too. An LBS requires the support from geographical data providers, which are normally a third party and charge users for their services. This study uses an embedded database SQLite for providing geographical data to small LBS projects. Section 5 introduces the embeddable database SQLite. The final section summarizes this study and gives possible LBS projects.

2 Location-Based Services (LBS)

This section introduces location-based services including a generic LBS architecture and an LBS example.

2.1 An LBS Architecture

A location-based service is a service based on the geographical position of a mobile handheld device (Kupper, 2005; Kolodziej & Hjelm, 2006). Two of the LBS examples

are (i) finding a nearby ethnic restaurants and (ii) locating a nearby store with the best price of a product. A system structure of location-based services, shown in Figure 1, includes five major components (Steiniger, Neun, & Edwardes, 2006):

(a) Mobile handheld devices, which are small computers that can be held in one hand. For most cases, they are smartphones.
(b) Positioning system, which is a navigation satellite system that provides location and time information to anyone with a receiver.
(c) Mobile and wireless networks, which relay the query and location information from devices to service providers and send the results from the providers to devices.
(d) Service providers, which provide the location-based services.
(e) Geographical data providers, which are databases storing a huge amount of geographical data such as information about restaurants and gas stations.

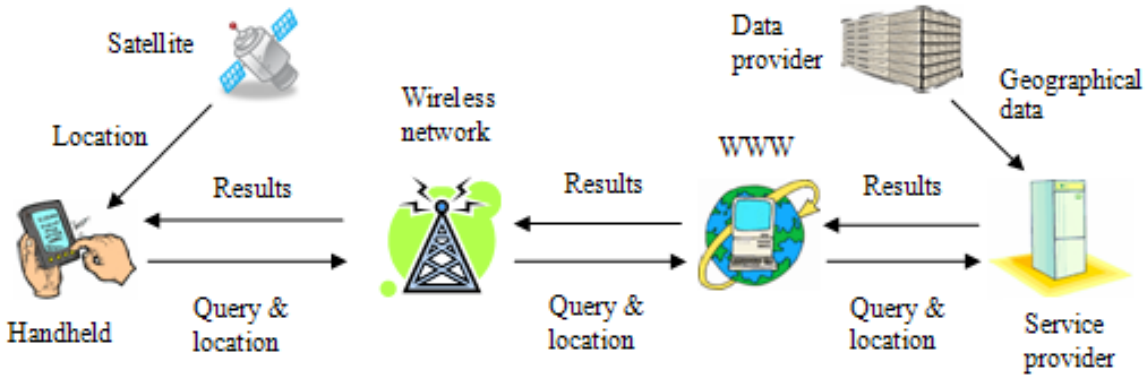


Figure 1: A System Structure of Generic Location-Based Services.

2.2 An LBS Example

An LBS example of finding route anomalies is given step by step as follows:

1. The smartphone (a) includes an application of finding route anomalies.
2. A mobile user submits a query of finding route anomalies along with the location information from a positioning system (b) to the application program, which runs in background.
3. The application program calls the server-side programs (d) located at the Aerospace School of the University of North Dakota along with the location information via mobile or wireless networks (c).
4. The programs at the servers perform the route anomaly detection using an Oracle database (e) which stores route data. Appropriate actions such as sending an alert are taken when an anomaly occurs.
5. The results such as an acknowledgement of sending an alert are sent back to the smartphone.

A nice introduction of LBS technologies and standards is given by Wang, Min, & Yi (2008, May 19-23).

3 Handheld Computing

Handheld computing is the use of handheld devices like smart cellular phones to perform wireless, mobile, handheld operations such as browsing the mobile Web and finding the nearest gas stations. It is an essential part of location-based services. This section discusses two handheld computing subjects: mobile handheld devices and client-side handheld computing.

3.1 Mobile Handheld Devices

Mobile users interact with mobile commerce applications by using small wireless Internet-enabled devices, which come with several aliases such as handhelds, palms, PDAs, pocket PCs, and smart phones. To avoid any ambiguity, a general term, mobile handheld devices, is used in this article. Mobile handheld devices are small general-purpose, programmable, battery-powered computers, but they are different from desktop PCs or notebooks due to the following special features:

- limited network bandwidth,
- small screen/body size, and
- Mobility.

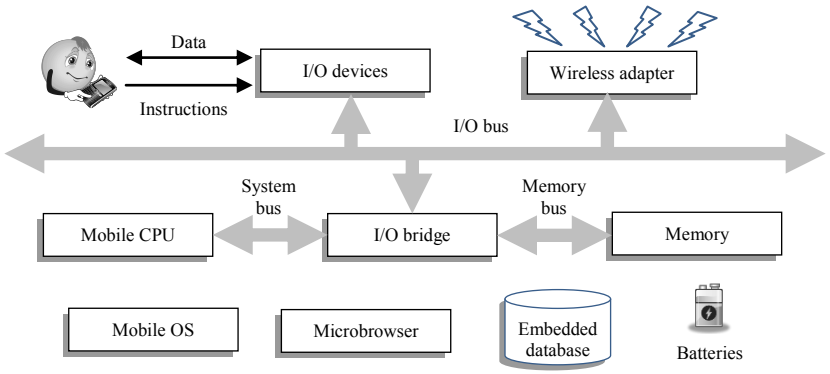


Figure 2: System Structure of Mobile Handheld Devices.

Short battery life and limited memory, processing power, and functionality are additional features, but these problems are gradually being solved as the technologies improve and new methods are constantly being introduced. The limited network bandwidth prevents the display of most multimedia on a microbrowser. Though the Wi-Fi and 3G networks go some way toward addressing this problem, the wireless bandwidth is always far below the bandwidth of wired networks. The small screen/body size restricts most handheld devices to using a stylus for input. Figure 2 shows a typical system structure for handheld devices, which includes the following six major components (Hu, Wiggen, & Yang, 2005):

1. *Mobile operating systems*: Simply adapting desktop operating systems for handheld devices has proved to be futile. A mobile operating system needs a

completely new architecture and different features to provide adequate services for handheld devices. A generalized mobile operating system structure can be visualized as a six-layer stack: (i) applications, (ii) GUI, (iii) API framework, (iv) multimedia, communication infrastructure, and security, (v) computer kernel, power management, and real-time kernel, and (vi) hardware controller.

2. *Mobile central processing units*: Handheld devices are becoming more sophisticated and efficient every day and mobile users are demanding more functionality from their devices. To achieve this advanced functionality, in addition to the obvious feature, low cost, today's mobile processors must have the following features: (i) high performance, (ii) low power consumption, (iii) multimedia capability, and (iv) real-time capability. The cores and architectures designed by Cambridge-based ARM Holdings Ltd. have begun to dominate the mobile CPU market.
3. *Microbrowsers*: Microbrowsers are miniaturized versions of desktop browsers such as Netscape Navigator and Microsoft Internet Explorer. They provide graphical user interfaces that allow mobile users to interact with mobile commerce applications. Microbrowsers usually use one of the following four approaches to return results to the mobile user: (i) wireless language direct access, (ii) HTML direct access, (iii) HTML to wireless language conversion, and (iv) error.
4. *Input/output components*: Various I/O devices have been adopted by mobile handheld devices. The only major output device is the screen, but there are several popular input devices, among them: (i) keyboards and (ii) touch screens/writing areas that need a stylus.*Memory*: Three types of memory are usually employed by handheld devices: (i) RAM, (ii) ROM, and (iii) flash memory. Hard disks, which provide much more storage capacity, are likely to be adopted by handheld devices in the near future.
5. *Batteries*: At present, rechargeable Lithium Ion batteries are the most common batteries used by handheld devices. However, the life of this kind of battery is short and the technology will not significantly improve unless and until manufacturers begin to switch to fuel cells, which may not happen for at least several years.

Synchronization connects handheld devices to desktop computers, notebooks, or peripherals to transfer or synchronize data. Without needing serial cables, many handheld devices now use either an infrared (IR) port or Bluetooth technology to send information to other devices.

3.2 Client-Side Handheld Computing

Client-side handheld computing is the programming for handheld devices and it does not need the supports from server-side programs. Typical applications created by it include (i) address books, (ii) video games, (iii) note pads, and (iv) to-do list. Various environments/operating systems/languages are available for client-side handheld programming. Table 1 lists the top-five mobile operating systems.

Mobile OS	Android	BlackBerry OS	iOS	Symbian	Windows Phone
Creator	Open Handheld Alliance	RIM	Apple	Nokia	Microsoft
2011 Market Share (472.7 million)	1st (46.4%)	4th (12.1%)	2nd (18.9%)	3rd (18.7%)	6th (2.1%) (5th: Samsung's Bada)
Development Languages	Java	Java	Objective C/C++	C++, Java, others	Visual C++
Kernel Type	Linux	Unix	Hybrid	Microkernel	Windows CE 6/7
IDEs, Libraries, Frameworks	Android SDK; ADT plug-in for Eclipse	BlackBerry JDE	iPhone SDK		Windows Phone SDK (works with Visual Studio)
Source Model	Open	Closed	Closed (open for the core)	Closed	Closed
Initial Release	2008	1999	2007	2001	2010
Latest Version as of August 2011	3	7	4	Anna	7
Mobile Application Store	Android Market	App World	App Store	Ovi Store	Windows Marketplace for Mobile

Table 1: Top-5 Mobile Operating Systems and Their Features.

They apply different approaches to accomplishing the development of mobile applications. Figure 2 shows a generic development cycle applied by them. Handheld emulators instead of the handhelds themselves are used for the development because of the convenience reason.

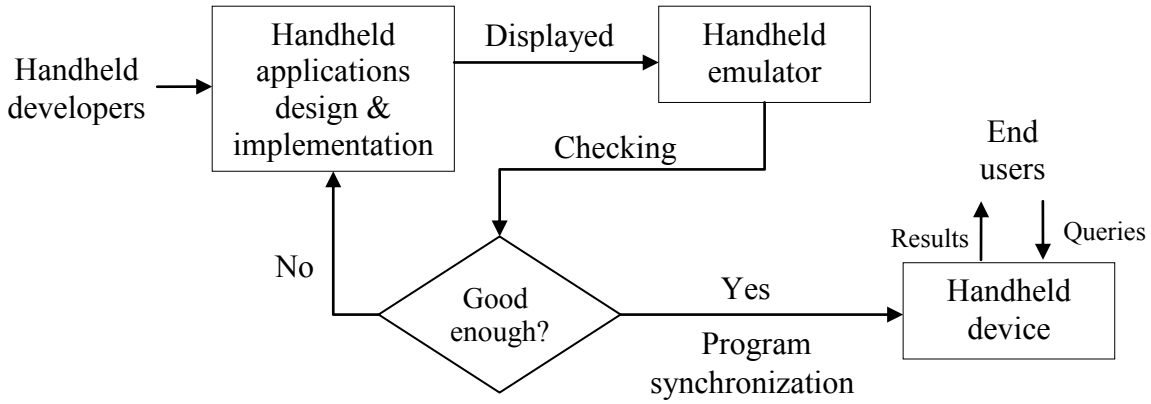


Figure 2: A Generalized Client-Side Handheld Computing Development Cycle.

4 Android Application Development

Android (N.D.) is a software stack for mobile devices such as smartphones and tablet PCs. It was developed by the Open Handset Alliance, a consortium of 80 hardware, software, and telecommunication companies led by Google devoted to advancing open standards for mobile devices. Google purchased the initial developer of the software, Android Inc., in 2005. Android includes an operating system, middleware, and key applications. The Android SDK (Software Development Kit) provides the tools and APIs necessary for developing applications on the Android platform using the Java programming language. Google released most of the Android code under the Apache License, a free software license. Android is tightly integrated with Google Maps, which provides a powerful tool for location-based services. Figure 3 shows a screenshot of a Google map with a location marker.

Figure 3: A Screenshot of a Google Map with a Location Marker.

4.1 A Simple Android Example

This application is based on the default template when you create an Android project. It has two different screens and is able to move one screen to another screen. Figure 4 shows the result screenshots. Two activities/pages are included in this application:

- `HelloWorldActivity.java`, which is the startup page and
- `NextPageActivity.java`, which is the other page.

Other than the text, each of the pages contains a button, which is used to display the other page. The following five components of this app will be detailed next:

- Android manifest file (`AndroidManifest.xml`),
- Layout XML code (`main.xml` and `next.xml`),
- Strings (`strings.xml`),
- The R file (`R.java`), and
- Java source code (`HelloWorldActivity.java` and `NextPageActivity.java`).

Figure 4: Two Screenshots of the Android Application.

4.2 Eclipse IDE with Android Plugin

Eclipse (N.D.) is a software development environment for multiple programming languages including Ada, C, C++, COBOL, Java, Perl, PHP, Python, R, Ruby, etc. It comprises an integrated development environment (IDE) and an extensible plug-in system. Android Development Tools (ADT) is a plugin for the Eclipse IDE, which gives developers an integrated environment for building Android applications. Figure 5 shows a screenshot of the Eclipse IDE with Android plugin, which provides the functionality of a well-established IDE, along with Android-specific features that are bundled with ADT. It allows developers to perform various tasks such as (i) setting up Android projects, (ii) creating an application user interface, (iii) adding components based on the Android Framework API (Application Programming Interface), (iv) debugging applications using the Android SDK (Software Development Kit) tools, and (v) exporting `.apk` files in order to post your application.

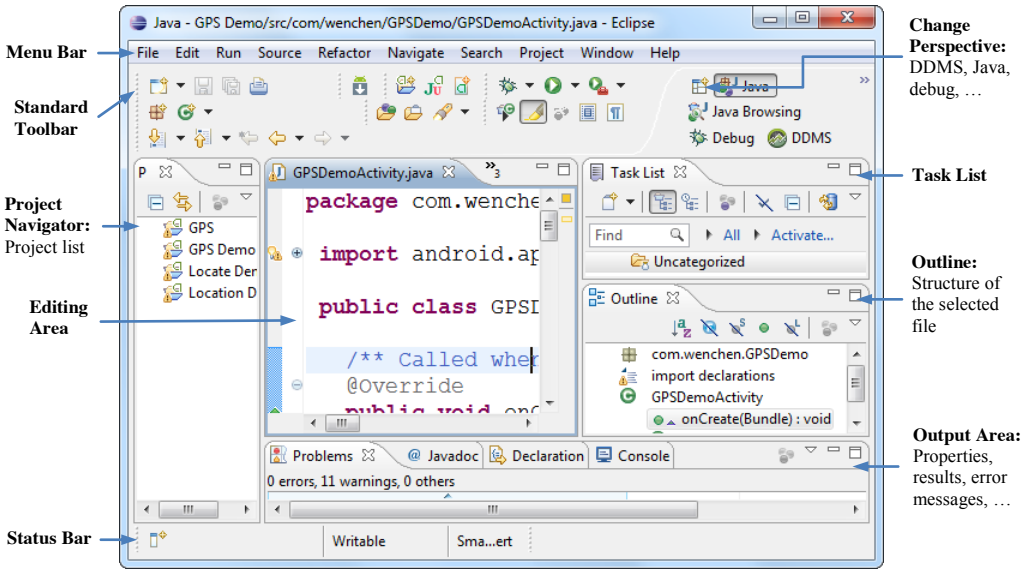


Figure 5: A Screenshot of the Eclipse IDE with Android Plugin.

4.3 Android Manifest File

The main missions of the manifest file are to explain what the application consists of, what its major building blocks are, what permissions it requires, etc. It must declare all activities, services, broadcast receivers, and content provider of the application. For example, if the application wants to access the Internet, it must define in this file that it would like to use the related permission.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.HelloWorld"
    android:versionCode="1"
    android:versionName="1.0">
  <uses-sdk android:minSdkVersion="8" />
  <application android:icon="@drawable/icon"
               android:label="@string/app_name">
    <activity android:name=".HelloWorldActivity"
              android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name=".NextPageActivity"
              android:label="Next Page" />
  </application>
</manifest>
```

4.4 Layout XML Code

The user interface for activities is defined via layouts. The layout defines the UI elements, their properties and their arrangement. A layout can be defined via Java code or via XML:

- *Java code*: You typically use Java code to generate the layout if you do not know the content until runtime; e.g., if your layout depends on content which you read from the Internet. An example of this is the next three statements whose screenshot is shown in Figure 6:

  ```
  TextView tv = new TextView( this );
  tv.setText( "Hello, Android" );
  setContentView( tv );
  ```

  which compose the string “Hello, Android” and display it.

Figure 6: A Screenshot of a layout generated by Java code.

- *XML*: XML based layouts are defined via a resource file in the folder `/res/layout`. This file specifies the view groups, views, their relationship, and their attributes for a specific layout. Defining layouts via XML is usually the preferred way as this separates the programming logic from the layout definition. For example, the following two layouts are used in this application:
    - o home page at `res/layout/main.xml` and
    - o next page at `res/layout/next.xml`

  The application loads the screen specified in `main.xml` by the following Java command in the `HelloWorldActivity.java`:

```
setContentView( R.layout.main );
```

The following code shows the contents of the file `main.xml`, one of the two screens of this application:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:gravity="center"
  android:background="@android:color/white">
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:text="@string/hello"
    android:textSize="25sp"
    android:textStyle="bold"
    android:textColor="@android:color/black"
    />
  <Button android:id="@+id/next"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="15sp"
    android:text="@string/button"
    />
</LinearLayout>
```

4.5 Strings

It is an XML file containing all text, such as button names, labels, and default text, used by the application. The purpose of using several XML files in an Android application is each XML file having its specific tasks. For example, a layout XML file is for user interface and a strings XML file is for their textual content. The code below shows the file `strings.xml` used in this application:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello">Hello, World!\n</string>
  <string name="button">Next Page</string>
  <string name="app_name">Hello World</string>
</resources>
```

The attribute value `@string/hello`, where `string` is the resource type and `hello` is the resource name, represents the string "Hello, World\n!," where the symbol '`\n`' is the character newline, stored in the file:

`/res/values/strings.xml`

You can use this syntax in an XML resource any place where a value is expected that you provide in a resource.

4.6 The R File

The directory `gen` in an Android project contains generated values. `R.java` is a generated class containing references to resources of the `res` folder in the project. These resources can be values, menus, layouts, or icons/pictures/animations. It is an automatically generated file so it should not be modified. The code below shows the file `R.java` for this app:

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package com.example.HelloWorld;

public final class R {
  public static final class attr {
  }
  public static final class drawable {
    public static final int icon=0x7f020000;
  }
  public static final class id {
    public static final int home=0x7f050001;
    public static final int next=0x7f050000;
  }
```

```
  public static final class layout {
    public static final int main=0x7f030000;
    public static final int next=0x7f030001;
  }
  public static final class string {
    public static final int app_name=0x7f040002;
    public static final int button=0x7f040001;
    public static final int hello=0x7f040000;
  }
}
```

The Android system provides methods to access the resources specified in the R file. Assume that two string variables `hello` and `app_name` are defined in the file `res/values/strings.xml`. They could be accessed anywhere in the Java code by using the following syntax:

`[<package_name>.]R.<resource_type>.<resource_name>`

An example of the string accesses is `R.string.hello` where `string` is the resource type and `hello` is the resource name. The reference is "`final static int`" instead of `String`. There are many Android APIs that can access your resources when you provide a resource ID in this format. For example, the method `getString(R.string.hello)` returns a string "Hello, World!\n" with the reference ID `R.string.hello`.

4.7 Java Source Code

The Java code is what drives everything. It is this code that ultimately is converted to Dalvik executable and runs the application. This application includes two screens/activities/classes:

- `HelloWorldActivity.java`, which is the main page, and
- `NextPageActivity.java`, which is the other page.

These two screens are similar. Each of the screens displays text and a button. The button is used to activate the other activity/class, which displays the other screen. Notice that the class is based on the `Activity` class. An `Activity` is a single application entity that is used to perform actions. The `onCreate( )` method is called by the Android system when your `Activity` starts—it is where you should perform all initialization and UI setup. An activity is not required to have a user interface, but usually does. The code below shows the file `HelloWorldActivity.java`:

```
package com.example.HelloWorld;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class HelloWorldActivity extends Activity {
```

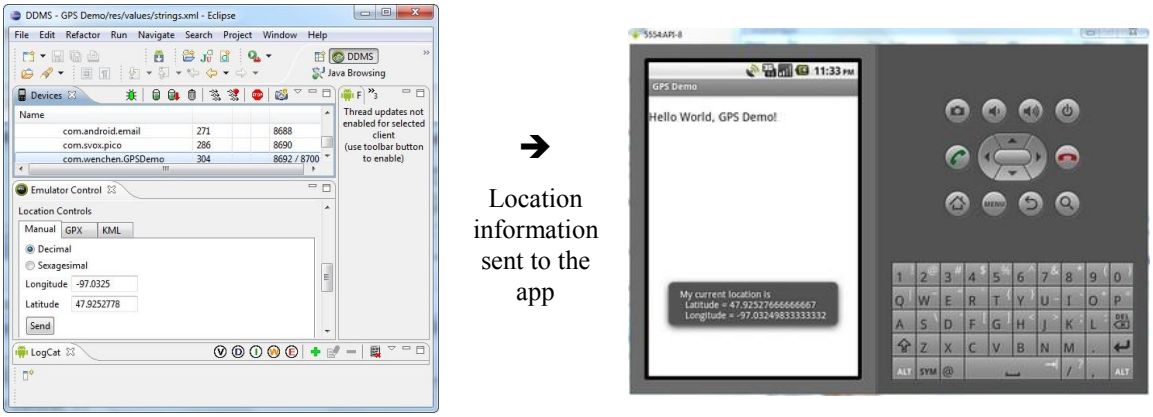
```
  /** Called when the activity is first created. */
  @Override
  public void onCreate( Bundle savedInstanceState ) {
    super.onCreate( savedInstanceState );
    setContentView( R.layout.main );

    final Button button = (Button) findViewById( R.id.next );
    button.setOnClickListener( new View.OnClickListener( ) {
      public void onClick( View v ) {
        /** Here i calls a new screen. **/
        Intent i = new Intent(
          HelloWorldActivity.this, NextPageActivity.class );
        startActivity( i );
      }
    } );
  }
}
```

4.8 DDMS (Dalvik Debug Monitor Server)

The ability of receiving locations is the fundamental requirement for all location-based applications. Android allows its emulator to receive a location (including longitude and latitude) from the DDMS (Dalvik Debug Monitor Server). The DDMS can be activated from the standard toolbar of Eclipse or by selecting the following options of Eclipse:

a. Select the Window menu.
b. Select the Open Perspective option.
c. Select the DDMS option.
d. Fill in the longitude and latitude values and click the Send button:

Figure 7 shows an application receiving and displaying the location information sent from the DDMS.



(a) (b)

Figure 7: (a) A Screenshot of the DDMS and (b) the Application Receiving and Displaying the Location Information Sent from the DDMS.

5 Embedded Database SQLite

A location-based service does not require an embedded database in the handheld device. A server-side geographical database provides the necessary information for LBS. For example, GeoNames (N.D.) is a geographical database available and accessible through various Web services. However, they also charge users for their services. An embeddable database is a convenient tool for small LBS projects. This section discusses the embedded database SQLite that comes with Android packages.

4.1 An SQLite Structure

SQLite (N.D.a) is an open source embeddable relational database management system, which is embedded within an application process without the overhead associated with a client-server configuration. Embedded databases are lightweight because they require little memory during run time and are written in compact code. SQLite includes the following major features: (i) supporting most of the SQL-92 standard, (ii) running on all major operating systems, (iii) having support for the major computer languages, and (iv) including multitasked read operations and sequential writes. The SQLite structure (N.D.b) consists of four components: (i) SQL Compiler, (ii) Core, (iii) Backend, and (iv) accessories as shown in Figure 8. It can store data up to 2 TB with each database saved in a single disk using a B^+ tree data structure. The SQL statements are compiled into assembly code executed on the SQLite virtual machine, Virtual Database Engine (VDBE).

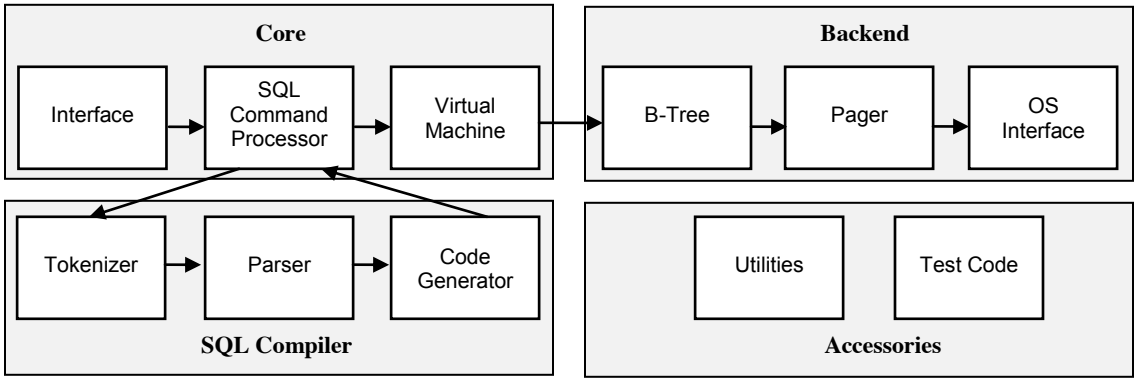


Figure 8: SQLite Internal Structure.

4.2 SQLite in Android

SQLite database is bundled with Android SDK (Software Development Kit). When the Android SDK is downloaded and installed, the SQLite database is ready to be used. Since no extra database setup and administration are needed, developers only have to define the SQL statements for defining and managing the database. SQLite supports

standard relational database features like SQL syntax, transactions, and prepared statements and the following data types: (i) `NULL`, a NULL value, (ii) `INTEGER`, a signed integer, (iii) `REAL`, a floating point value, (iv) `TEXT`, a text string, and (v) `BLOB`, a binary large object. Compared to server-side databases, SQLite requires only little memory at runtime (about 250 KB). SQLite database is private to the Android application which creates it. In order to share data with other applications, the applications can use Content Provider, which manages access to a structured set of data. Content providers are the standard interface that connects data in one process with code running in another process. To query a content provider, a query string uses the following URI syntax:

```
<standard_prefix>://<authority>/<data_path>/<id>
```

For example, to retrieve all the personal information stored by the AddressBook application, the URI would look like this:

```
content://AddressBook/people
```

6 Summary

Mobile application stores (or app stores) sell or provide mobile applications/services for handheld devices such as smartphones. The applications/services are not necessarily from the storeowners. Many of them are from the third parties such as independent developers. A wide variety of mobile applications is available on the stores. Popular applications include:

- *Mobile games*, which include standalone and online games,
- *Mobile offices*, which include software like spreadsheets, editors, and notebooks,
- *Mobile shopping*, which uses mobile handheld devices to purchase products, and
- *Music*, which may be the most frequently downloaded applications.

One of the best-selling apps is related to location-based services, which cover a wide range of usages such as finding the lowest-price product in a nearby store and locating the nearest gas stations. Location-based services implementation involves a variety of technologies and methods and most developers are not familiar with them. This paper explains the design and implementation of location-based services by detailing the following subjects: (i) LBS system, (ii) handheld computing, (iii) Android application development, and (iv) embeddable database SQLite. Readers may apply the knowledge learned from this article to the following applications:

- Find the interesting nearby places such as ethnic restaurants and movie theaters.
- Compare prices and perform product research.
- Advertisements and coupons are displayed based on users' locations.
- Use social networks to connect people within a short distance.

The following location-based research can also be considered:

- Detect any route anomalies. For example, an alert is generated if a pupil does not follow his/her regular route to school.
- Find travel recommendations based on route trajectories. For example, most people probably never heard the world's largest truck stop at Walcott, Iowa. With

this feature, the drivers on the highway I-80 will be notified this interesting place when they are near Walcott.
- Indoor positioning and navigation are used to help users have a better visiting experience.
- Pre-fetch and cache map tiles based on drivers' current locations so maps can be displayed in time and power consumption is reduced.

References

Android. (N.D.). *Android Developers*. Retrieved March 6, 2012, from http://developer.android.com/

Cohen, Heidi. (2011, January 20). *Mobile Marketing: 56 Must Have Facts*. Retrieved December 9, 2011, from http://heidicohen.com/mobile-marketing-must-have-facts/

Eclipse. (N.D.) *About the Eclipse Foundation*. Retrieved March 12, 2012, from http://www.eclipse.org/org/

Fleishman Hillard. (2012, February 6). *2012 Digital Influence Study*. Retrieved from March 5, 2012, from http://www.factbrowser.com/facts/4671/

Frantzell, N.-E. (2005, July 21). *An Introduction to SQLite, an Open Source Embeddable Database*. Retrieved November 11, 2011, from http://www.ibm.com/developerworks/opensource/library/os-sqlite/

GeoNames. (N.D.). *About GeoNames*. Retrieved February 17, 2012, from http://www.geonames.org/about.html

Hu, W.-C., Wiggen, T., & Yang, H.-J. (2005, May 15-18). Mobile commerce systems and payment methods. In *Proceedings of the 2005 Information Resources Management Association (IRMA 2005) International Conference*, pages 769-771, San Diego.

JiWire. (2011, October 14). *JiWire Mobile Audience Insights Report Q2 2011*. Retrieved from February 18, 2012, from http://www.factbrowser.com/facts/3196/

Kolodziej, K. & Hjelm, J. (2006). *Local Positioning Systems: LBS Applications and Services*, CRC Taylor & Francis.

Kupper, A. (2005). *Location-Based Services: Fundamentals and Operation*. Wiley.

SQLite. (N.D.a). *About SQLite*. Retrieved from March 5, 2012, from http://www.sqlite.org/about.html

SQLite. (N.D.b). *The Architecture of SQLite*. Retrieved January 12, 2012, from http://www.sqlite.org/arch.html

Steiniger, S., Neun, M., & Edwardes, A. (2006). *Foundations of Location-Based Services*. Retrieved January 13, 2012, from http://www.geo.unizh.ch/publications/cartouche/lbs_lecturenotes_steinigeretal2006.pdf

Wang, S., Min, J., & Yi, B. K. (2008, May 19-23). Location based services for mobiles: technologies and standards. In *Proceedings of the IEEE International Conference on Communication (ICC)*, Beijing, China.