Research on Mobile Location Service Design Based on Android

Xianhua Shu, Zhenjun Du, Rong Chen
School of Information Science and Technology
Dalian Maritime University
Dalian, China
xiansimba@163.com

***Abstract*—Android platform is a new generation of smart mobile phone platform launched by Google. Android provides the support of mobile map and location service, which is probably a concern of vast numbers of developers. So far, the development of mobile map and location applications is complex and difficult, and is often required to pay high copyright fees to map makers. Android is free and open, providing an easy-to-use development kit containing flexible map display and control functions. This paper introduces the architecture and component models of Android, and analyzes the anatomy of an Android application including the functions of Activity, Intent Receiver, Service, Content Provider, and etc. Based on Android, the design method of a location-based mobile service is then presented. The design example shows that it's so easy to implement self-location, to draw the driving trace, to perform query and to flexibly control the real-time map on Android.**

Keywords-Android; location based service; driving trace

I. INTRODUCTION

The Open Handset Alliance released the Google Android SDK on November 12, 2007 [1]. The conception of the Android platform is attracting more and more programmers in mobile computing fields. Android is a package of software for mobile devices, including an operating system, middleware and core applications. The Android SDK provides powerful tools and APIs necessary to develop applications on the Android platform using the Java programming language.

Android platform is of open system architecture, with versatile development and debugging environment, but also supports a variety of scalable user experience, which has optimized graphics systems, rich media support and a very powerful browser. It enables reuse and replacement of components and an efficient database support and support various wireless communication means. It uses a Dalvik virtual machine heavily optimized for mobile devices [2].

Android also supports GPS, VideoCamera, compass, and 3d-accelerometer and provides rich APIs for map and location functions. Users can flexibly access, control and process the free Google map and implement location based mobile service in his mobile systems at low cost. Android platform will not only promote the technology (including the platform itself) of innovation, but also help to reduce development costs, and enable developers to form their mobile systems with unique characteristics.

The architecture of Android framework and the anatomy of an Android application are addressed in section II and section III. Based on the analyses, the design of a location-based mobile service on Android is then presented in section IV. And the last section gives the conclusion.

II. ANDROID ARCHITECTURE

The Android architecture and its main components are shown in Fig.1 as follows [3] [4].

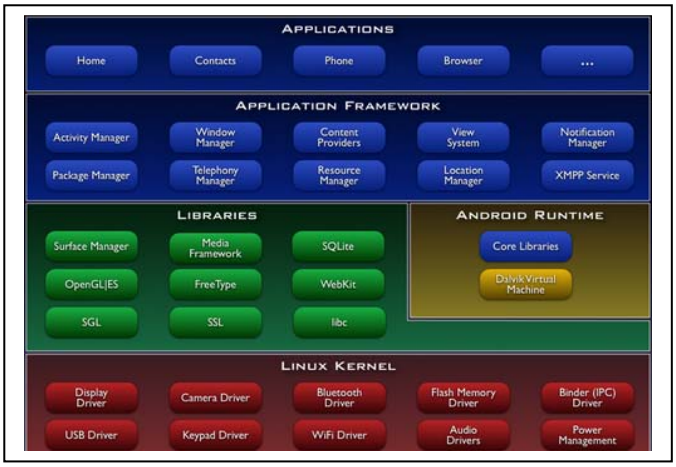

Figure 1. Android architecture.

A. Applications

A set of core applications are on the top level in the framework, including an email client, a SMS app, a calendar, a maps-application, web browser, contacts-app, and many more. All apps are written using the Java programming language.

B. Application Framework

Developers have full access to the same framework APIs used by the core applications. The application architecture is designed to simplify the reusing of all components. This mechanism allows every component to be replaced by the user. Underlying all applications is a set of services and systems,

This work has been partially supported by National Natural Science Foundation of China (No. 60775028), Dalian Science & Technology Program (No. 2007A14GX042) and Dalian Maritime University Youth Foundation (DLMU-ZL-200803).

978-1-4244-3693-4/09/$25.00 ©2009 IEEE

including a rich and extensible set of ActivitiesViews that can be used to build an application, including grids, lists, textViews editIntroductionTexts, Spinners, Buttons, an embeddable web browser and even an MapView which can be put into every app within very few lines of code; Content Providers that enable applications to access data from other applications (such as Contacts), or to share their own data; a automatic Resource Manager, making non-code resources accessible from code; a Notification Manager that enabling all applications to show custom alerts in the upper status bar; an Activity Manager managing the life of each applications and providing a useful navigation backtrack.

C. Libraries

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 in Fig.1.

D. 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 given by the OS, and owns its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM is executing files in the .dex (Dalvik Executable) format which was optimized for minimal cpu-and-memory-usage. The Virtual Machine is register-based, and runs classes compiled by a Java language compiler that have been transformed at compile-time into the .dex format using the "dx" tool, that are shipped with the SDK. The Linux Kernel can run multiple instances of the Dalvik VM, also providing underlying functionality such as threads and lowest-level memory management.

E. Linux Kernel

Android relies on Linux (Kernel version 2.6) for core system services such as memory management, process management, network stack, security, and driver model. The core also acts as a hardware abstraction layer between the applications and all the hardware.

III. Anatomy of an Android Application

There are four building blocks to an Android application: Activity, Intent Receiver, Service, Content Provider. Not every application needs to have all four, but a user's application will be written with some combination of these. Once the user has decided what components are needed for the application, they should be listed in a file called AndroidManifest.xml, which is where the components of the application are declared and what their capabilities and requirements are [5] [6].

A. Activity

Activity is the most common one of the four Android building blocks. An activity is usually a single screen in your application. Each activity is implemented as a single class that extends the Activity base class. Your class will display a user interface composed of Views and respond to events. Most applications consist of multiple screens. For example, a text messaging application might have one screen that shows a list of contacts to send messages to, a second screen to write the message to the chosen contact, and other screens to review old messages or change settings. Each of these screens would be implemented as an activity. Moving to another screen is accomplished by a starting a new activity. In some cases an Activity may return a value to the previous activity - for example an activity that lets the user pick a photo would return the chosen photo to the caller.

When a new screen opens, the previous screen is paused and put onto a history stack. The user can navigate backward through previously opened screens in the history. Screens can also choose to be removed from the history stack when it would be inappropriate for them to remain. Android retains history stacks for each application launched from the home screen.

Android uses a special class called Intent to move from screen to screen. Intent describes what an application wants done. The two most important parts of the intent data structure are the action and the data to act upon. Typical values for action are MAIN (the front door of the application), VIEW, PICK, EDIT, etc. The data is expressed as a Uniform Resource Indicator (URI). For example, to view a website in the browser, you would create an Intent with the VIEW action and the data set to a Website-URI.

There is a related class called an IntentFilter. While an intent is effectively a request to do something, an intent filter is a description of what intents an activity (or intent receiver, see below) is capable of handling. An activity that is able to display contact information for a person would publish an IntentFilter that said that it knows how to handle the action VIEW when applied to data representing a person. Activities publish their IntentFilters in the AndroidManifest.xml file.

Navigating from screen to screen is accomplished by resolving intents. To navigate forward, an activity calls startActivity (myIntent). The system then looks at the intent filters for all installed applications and picks the activity whose intent filters best matches myIntent. The new activity is informed of the Intent, which causes it to be launched. The process of resolving intents happens at run time when startActivity is called, which offers two key benefits: Activities can reuse functionality from other components simply by making a request in the form of an Intent. Activities can be replaced at any time by a new Activity with an equivalent.

B. Intent Receiver

You can use an IntentReceiver when you want code in your application to execute in reaction to an external event, for example, when the phone rings, or when the data network is available, or when it's midnight. Intent receivers do not display a UI, although they may display Notifications to alert the user if something interesting has happened. Intent receivers are also registered in AndroidManifest.xml, but you can also register them using Context.registerReceiver method. Your application does not have to be running for its intent receivers to be called; the system will start your application, if necessary, when an

intent receiver is triggered. Applications can also send their own intent broadcasts to others with Context.broadcastIntent method.

C. *Service*

A Service is code that is long-lived and runs without a UI. A good example of this is a media player playing songs from a play list. In a media player application, there would probably be one or more activities that allow the user to choose songs and start playing them. However, the music playback itself should not be handled by an activity because the user will expect the music to keep playing even after navigating to a new screen. In this case, the media player activity could start a service using the Context.startService method to run in the background to keep the music going. The system will then keep the music playback service running until it has finished. Note that you can connect to a service with the Context.bindService method. When connected to a service, you can communicate with it through an interface exposed by the service. For the music service, this might allow you to pause, rewind, etc.

D. *Content Provider*

Applications can store their data in files, a SQLite database, preferences or any other mechanism that makes sense. A content provider, however, is useful if you want your application's data to be shared with other applications. A content provider is a class that implements a standard set of methods to let other applications store and retrieve the type of data that is handled by that content provider.

IV. LOCATION BASED MOBILE SERVICE DESIGN

Flexible map display and control functions and location support are provided in Android for mobile system design. This section analyses MapView, MapActivity and Location-Based API on Android, and presents a simple design example to show the location-based mobile service design on Android.

MapView is used to display a view of the map. It can accept the keyboard events such as onKeyDown and onKeyUp to support the map movement and the zoom feature. It also supports multi-layer Overlay and user can draw coordinates, pictures and strings on the map. MapView is set up only by MapActivity. Because MapView uses the file system and network in the background, all of these threads are in the control of the Activity life cycle. Before useing the map, an apikey for the Map service needs to be applied from Google.

Android defines UI by the layout. MapView is required to be added into the layout, as follows:

```
<com.google.android.maps.MapView
   android:id="@+id/myMap"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:enabled="true"
   android:clickable="true"
   android:apiKey="application apikey" />
```

Other necessary views can then be added, such as an EditText to enter an address and a button to carry out queries as shown in Fig.2.

Figure 2. Address search interface.

We then use the Android Location-Based API to collect user current position and display that location on the screen, and use Google Maps to display the current user location on the cell phone.

Now create a LocationManager from which we can get the coordinate values:

```
LocationManager lm = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
```

Next, set up a GeoPoint and assign to it the latPoint and lngPoint values that we retrieved from the GPS, and use the controller to move the map to current location:

```
GeoPoint p = new GeoPoint((int)(latPoint*1E6), (int)
(lngPoint*1E6));
myMapController.animateTo(p);
myMapController.setZoom(10);
myMap.invalidate();
```

After running the program, you may use the DDMS tool in Android SDK to send geographical coordinates by KML to simulate the GPS route in Fig.3.

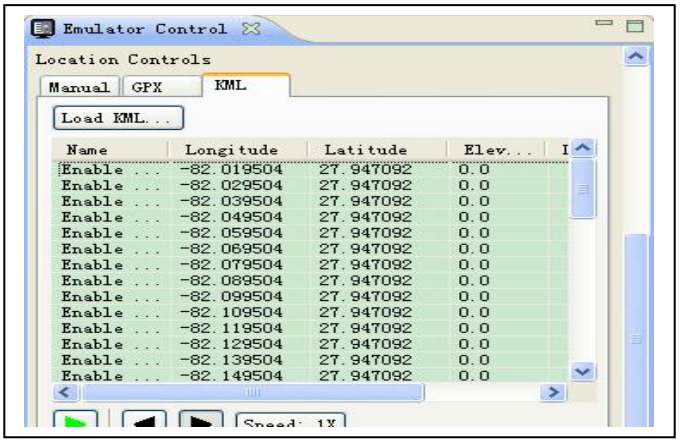

Figure 3. Location controls.

According to the changed geographic coordinates, a trace is displayed on the test terminal dynamically. Fig.4 is the test result.

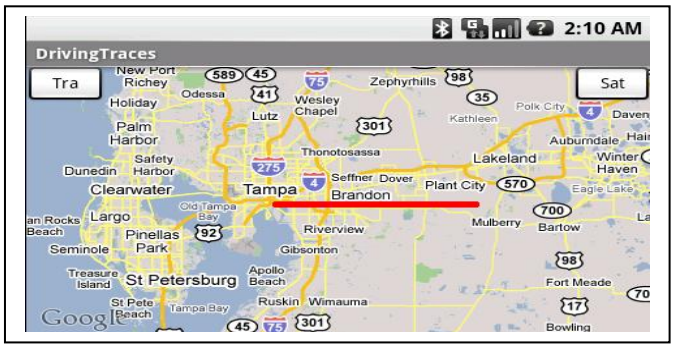

Figure 4. Driving trace result.

We can also use setZoom(int) to zoom out the map to the level needed, such as zooming out one more level as follows:

```
myMapController.setZoom (myMap.getZoomLevel ( )-1);
```

Map view types can be switched, e.g. switching to the satellite view and hiding the current traffic view, such as:

```
myMap.setSatellite (true);
myMap.setTraffic (false);
```

The switching result of map view types is shown in Fig.5.

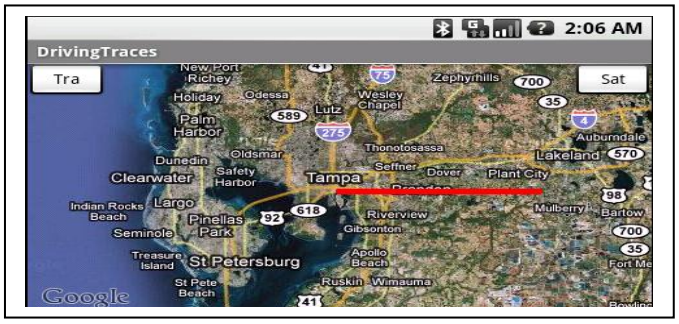

Figure 5. The satellite view (switching result).

If we want to draw landmarks, coordinates, pictures and etc on the map, our own Overlay categories need to be defined:

```
public class SitesOverlay extends ItemizedOverlay <OverlayItem> {}
```

Then override the *draw* method. For example, draw a circle at the current point, such as:

```
Point point = projection.toPixels(overLayItem.getPoint(),null);
Paint paintCircle = new Paint();
paintCircle.setColor(Color.RED);
canvas.drawCircle(point.x, point.y, 3, paintCircle);
```

Other geometric shapes can be drawn in the similar *draw* method.

We can see from the above design example that it's convenient to implement the location based service with rich map functions and the actual high running performance is also derived.

CONCLUSIONS

The feature of location based service is emphasized on Android platform. One can integrate a fully zoom and drag enabled map by adding just few lines in the java code and XML code of the Android-Default-Application. Through the research on Android architecture and application development, and from the design method and results of an application example in this paper, the availability and performance of the platform is verified and the design result also shows the easiness to implement self-location, to draw the driving trace, to perform queries and to flexibly control the real-time map on Android. The actual system also achieves high running performance. The future work is to design a more powerful mobile location-based system featured with more unique customized functions based on Android.

REFERENCES

[1] Open Hanset Alliance, http://www.openhandsetalliance.com/.

[2] Android - An Open Handset Alliance Project, http://code.google.com-/intl/zh-CN/android/.

[3] J.F. DiMarzio, Android A Programmer's Guide, Chicago: McGraw-Hill, Jul. 2008.

[4] Android Developers, http://www.androidin.com/.

[5] C. Haseman, Android Essentials, PDF Electronic Book, 2008. Available from: http://androidos.cc/dev/index.php.

[6] N. Gramlich, Android Programming , PDF Electronic Book, 2008. Available from: http://androidos.cc/dev/index.php.