globsyn

Taking People To The Next Level

globsyn

globsyn
finishing school

Location & Sensors

Location & Sensors

- Topics to be covered in this session:
    - Location & sensors

Location & Sensors

- Location and maps-based apps can be built using the classes of the android.location package and the Google Maps Android API.

- Use sensors on the device to add rich location and motion capabilities to your app, from GPS or network location to accelerometer, gyroscope, temperature, barometer, and more.

Location & Sensors Contd.

- Location Services:

  Through the android.location package classes, the applications access to the location services supported by the device.

  The central component of the location framework is the LocationManager system service, which provides APIs to determine location and bearing of the underlying device

  The method getSystemService(Context.LOCATION_SERVICE) returns a handle to a new LocationManager instance.

Location & Sensors Contd.

Once your application has a LocationManager, your application is able to do three things:

- Query for the list of all LocationProviders for the last known user location.

- Register/unregister for periodic updates of the user's current location from a location provider (specified either by criteria or name).

- Register/unregister for a given Intent to be fired if the device comes within a given proximity (specified by radius in meters) of a given lat/long.

Location & Sensors Contd.

Google Maps Android API:

- Can add maps to your app that are based on Google Maps data, automatically handles access to Google Maps servers, data downloading, map display, and touch gestures on the map.
- Use these API calls to add markers, polygons and overlays, and to change the user's view of a particular map area.
- It is not included in the Android platform, but available on any device with the Google Play Store running Android 2.2 or higher, through Google Play services.
- To integrate Google Maps into your app, you need to install the Google Play services libraries for your Android SDK.

Location & Sensors Contd.

- MapView Class: displays a map with data obtained from the Google Maps service.

- It captures key presses and touch gestures to pan and zoom the map automatically, handling network requests for additional maps tiles.

- Provides all of the UI elements & methods necessary for users to control the map,

Location & Sensors Contd.

Location strategies:

- Knowing where the user is allows your application to be smarter and deliver better information to the user.
- Utilize GPS and Android's Network Location Provider to acquire the user location.
- GPS is most accurate, but it only works outdoors, quickly consumes battery power, and doesn't return the location as quickly as users want.
- Android's Network Location Provider determines user location using cell tower and Wi-Fi signals, providing location information that works indoors and outdoors, responds faster, and uses less battery power.

Location & Sensors Contd.

Challenges in Determining User Location:

- Multitude of location sources GPS, Cell-ID, and Wi-Fi can each provide a clue to users location. Determining which to use and trust is a matter of trade-offs in accuracy, speed, and battery-efficiency.
- User movement: Need to re-estimate the user location changes often.
- Varying accuracy: Location estimates coming from each location source are not consistent in their accuracy. A location obtained 10 seconds ago from one source might be more accurate than the newest location from another or same source.

Location & Sensors Contd.

Requesting Location Updates:

- Callbacks are used to get user location in Android
- Calling the requestLocationUpdates(), & passing it a LocationListener, you receive location updates from the LocationManager ("Location Manager").
- LocationListener must implement several callback methods that the Location Manager calls when the user location changes or when the status of the service changes.

Location & Sensors Contd.

Requesting User Permissions
In order to receive location updates at runtime from NETWORK_PROVIDER or GPS_PROVIDER, you must request user permission by declaring either the ACCESS_COARSE_LOCATION or ACCESS_FINE_LOCATION permission, respectively, in your Android manifest file. For example:

```
<manifest ... >
    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATIO
N" />
    ...
</manifest>
```

Location & Sensors Contd.

Defining a Model for the Best Performance

- Due to the less than optimal accuracy, user movement, the multitude of methods to obtain the location, and the desire to conserve battery, getting user location is complicated.
- Must define a consistent model that specifies how your application obtains the user location.
- This model includes when you start and stop listening for updates and when to use cached location data.

Location & Sensors Contd.

Flow for obtaining user location

- ♣Start application.
- ♣start listening for updates from desired location providers.
- ♣Maintain a "current best estimate" of location by filtering out new, but less accurate fixes.
- ♣Stop listening for location updates.
- ♣Take advantage of the last best location estimate.

Location & Sensors Contd.

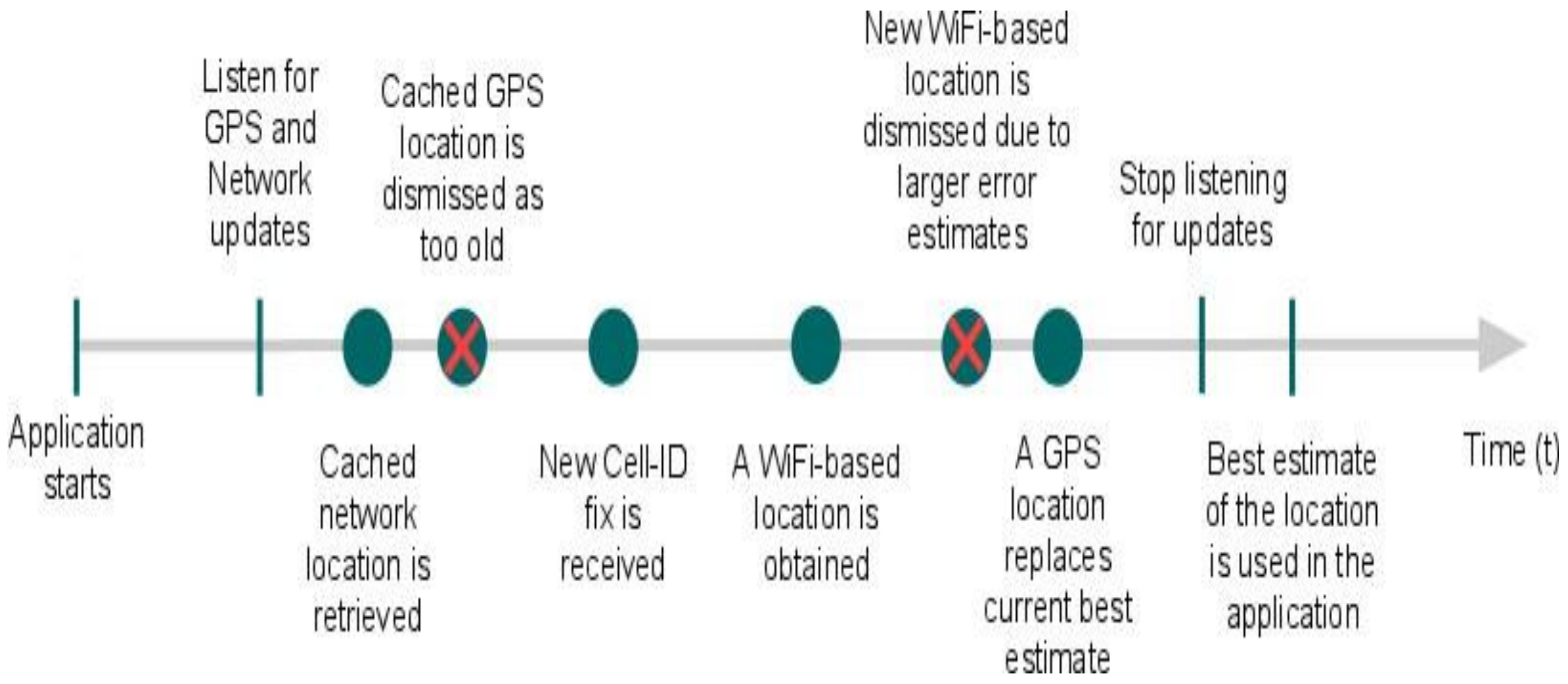
Listen for GPS and Network updates
Cached GPS location is dismissed as too old
New WiFi-based location is dismissed due to larger error estimates
Stop listening for updates
Application starts
Cached network location is retrieved
New Cell-ID fix is received
A WiFi-based location is obtained
A GPS location replaces current best estimate
Best estimate of the location is used in the application
Time (t)

Figure 1. A timeline representing the window in which an application listens for location updates.

Location & Sensors Contd.

Deciding when to start listening for updates

Might want to start listening for location updates as soon as your application starts, or only after users activate a certain feature. Consumes a lot of battery power, & not allow for sufficient accuracy. You can begin listening for updates by calling requestLocationUpdates():

```
String locationProvider = LocationManager.NETWORK_PROVIDER;
// Or, use GPS location data:
// String locationProvider = LocationManager.GPS_PROVIDER;
locationManager.requestLocationUpdates(locationProvider, 0, 0,
locationListener);
```

Location & Sensors Contd.

Getting a fast fix with the last known location

You should utilize a cached location by calling getLastKnownLocation(String):

```
String locationProvider =
LocationManager.NETWORK_PROVIDER;
// Or use LocationManager.GPS_PROVIDER

Location lastKnownLocation =
locationManager.getLastKnownLocation(locationProvider)
;
```

Location & Sensors Contd.

Deciding when to stop listening for updates

To improves the accuracy of the estimate, use a short gap between when the location is acquired and when the location is used,
Listening for a long time consumes a lot of battery power, so as soon as you have the information you need, you should stop listening for updates by calling removeUpdates(PendingIntent):

```
// Remove the listener you previously added
locationManager.removeUpdates(locationListener);
```

References

Maintaining a current best estimate

Should include logic for choosing location fixes based on several criteria depends on the use-cases of the application and field testing.

A few steps to validate the accuracy of a location fix:

- Check if the location retrieved is significantly newer than the previous estimate.
- Check if the accuracy claimed by the location is better or worse than the previous estimate.
- Check which provider the new location is from and determine if you trust it more.

References

Adjusting the model to save battery and data exchange:

- **Reduce the size of the window** used to listen for location updates means less interaction with GPS and network location services, thus, preserving battery life. But it also allows for fewer locations from which to choose a best estimate.

- **Set the location providers to return updates less frequently:** By increasing the parameters in requestLocationUpdates() that specify the interval time and minimum distance change.

- **Restrict a set of providers:** Depending on where your application is used or the desired level of accuracy, you can choose the Network Location Provider or GPS, instead of both.

Location & Sensors Contd.

Common application cases where the user locations is used in your application:

- Tagging user-created content with a location
- Helping the user decide on where to go
- Providing Mock Location Data : To test your location-based features by mocking location data in the Android emulator. Such as using Eclipse, DDMS, or the "geo" command in the emulator console.

Location & Sensors Contd.

- Using Eclipse

Select Window > Show View > Other > Emulator Control.

- Using DDMS

Manually send individual longitude/latitude coordinates to the device.

- Use a GPX file describing a route for playback to the device.

- Use a KML file describing individual place marks for sequenced playback to the device.

Location & Sensors Contd.

Using the "geo" command in the emulator console

- Launch your application in the Android emulator and open a terminal/console in your SDK's /tools directory.
- Connect to the emulator console: telnet localhost <*console-port*>
- Send the location data: geo fix to send a fixed geo-location.
- This command accepts a longitude and latitude in decimal degrees, and an optional altitude in meters.
- geo nmea to send an NMEA 0183 sentence.
- This command accepts a single NMEA sentence of type '$GPGGA' (fix data) or '$GPRMC' (transit data).

Location & Sensors Contd.

Sensors:

- Most Android-powered devices have built-in sensors that measure motion, orientation, and various environmental conditions.
- Are capable of providing raw data with high precision and accuracy,
- Can access sensors available on the device and acquire raw sensor data by using the Android sensor framework.
- The sensor framework provides several classes and interfaces that help you perform a wide variety of sensor-related tasks.

Location & Sensors Contd.

- 2 types of sensors:
  - **Hardware-based sensors:** Physical components built into a handset or tablet device.Derive their data by directly measuring specific environmental properties, such as acceleration, geomagnetic field strength, or angular change.
- Ex: TYPE_ACCELEROMETER(for Motion detection (shake, tilt, etc.).

TYPE_LIGHT (for Controlling screen brightness.) etc,

Location & Sensors Contd.

- **Software-based sensors:** Not physical devices, although they mimic hardware-based sensors. Derive their data from one or more of the hardware-based sensors and are sometimes called virtual sensors or synthetic sensors. The linear acceleration sensor and the gravity sensor are examples of software-based sensors.

- Ex: TYPE_ORIENTATION (for Determining device position.) etc.

Location & Sensors Contd.

Sensor Framework:

- Is part of the android.hardware package and includesthe following classes and interfaces:

- SensorManager: This class to create an instance of the sensor service. Which provides various methods for accessing and listing sensors, registering and unregistering sensor event listeners, and acquiring orientation information.
- Sensor: This class to create an instance of a specific sensor. This class provides various methods that let you determine a sensor's capabilities.

Location & Sensors Contd.

- SensorEvent : This class to create a sensor event object, which provides information about a sensor event. A sensor event object includes the following information: the raw sensor data, the type of sensor that generated the event, the accuracy of the data and the timestamp for the event.

- SensorEventListener : This interface to create two callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes.

Location & Sensors Contd.

- 2 basic task of sensor-relatedAPIs:
  - ♣**Identifying sensors and sensor capabilities at runtime** is useful if your application has features that rely on specific sensor types or capabilities

The API also provides methods that let you determine the capabilities of each sensor, such as its maximum range, its resolution, and its power requirements.

To identify the sensors that are on a device you first need to get a reference to the sensor service. To do this, you create an instance of the SensorManager class by calling the getSystemService() method and passing in the SENSOR_SERVICE argument.

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
```

Location & Sensors Contd.

- Monitoring Sensor events:

To monitor raw sensor data you need to implement two callback methods that are exposed through the SensorEventListener interface: onAccuracyChanged() and onSensorChanged().

- The Android system calls these methods whenever the following occurs:

♣ Asensor's accuracy changes. Accuracy is represented by one of four status constants: SENSOR_STATUS_ACCURACY_LOW, SENSOR_STATUS_ACCURACY_MEDIUM, SENSOR_STATUS_ACCURACY_HIGH, or SENSOR_STATUS_UNRELIABLE.

Location & Sensors Contd.

- A sensor reports a new value. A SensorEvent object contains information about the new sensor data, including: the accuracy of the data, the sensor that generated the data, the timestamp at which the data was generated, and the new data that the sensor recorded.

- Handling Different Sensor Configurations: There is no specific standard sensor configuration for devices. 2 options for ensuring that a given sensor is present on a device:
  - Detect sensors at runtime and enable or disable application features as appropriate.
  - Use Google Play filters to target devices with specific sensor configurations.

Location & Sensors Contd.

3 broad categories of sensors:

- **Motion sensors:** These sensors measure acceleration forces and rotational forces along three axes. Includes hardware based sensors (accelerometers, gyroscopes. gravity sensors), either hardware-based or software-based (gravity, linear acceleration, and rotation vector sensors)
    - Useful for monitoring device movement, such as tilt, shake, rotation, or swing.
    - The movement is usually a reflection of direct user input or it can also be a reflection of the physical environment in which the device is sitting

Location & Sensors Contd.

♣**Position sensors:** These sensors measure the physical position of a device.

2 hardware based sensors that let you determine the position of a device:

- Geomagnetic field sensor
- Proximity Sensor:

Orientation sensor is software-based and derives its data from the accelerometer and the geomagnetic field sensor.

Location & Sensors Contd.

- **Environmental sensors:** Hardware based sensors are available only if a device manufacturer has built them into a device.

  - These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity.
  - Includes barometers, photometers, and thermometers.
  - Environment sensors return a single sensor value for each data event.

References

http://developer.android.com/guide/topics/sensors/index.html

Taking People To The Next Level . . .