Design Document

PinPointer: An Android App to Find Your Way

Anindita Das
dasanuiit@gmail.com

Architecture

Fig. 1 shows the basic design structure for the PinPointer program. Note that the goal is to develop an App that provides a compelling and intuitive user experience for different usage scenarios and covers the entire range of requirements from location identification to memorization aid and communication. Such an App must be designed with the user interface in mind from the ground up. This design requirement is reflected directly in Fig. 1: each software component (shown with a rectangular box) in fact represents a Java class, and each Java class corresponds to a screen that the user actually sees by clicking the different buttons in the App. The design of the system in this manner is deliberate: we conceived the program as a collection of "features" coinciding with the experience that the user will wish to see, and developed the code and logic around those features.

The different program components shown in Fig. 1 implement the following functionality.

- **MyCarFinder:** Implements the "Home Screen", with a list containing the collection of saved locations, together with buttons to save the current location or manually enter a new location.
- **SaveLocation:** Implements the functionality to permit saving the current location. Provides facilities to take a picture of the location and add notes which can be associated with the location as a memory refresher.
- **EnterLocation:** Similar to **SaveLocation** but designed for manually entering locations. Since the location manually entered is typically not the current location, a photograph of the current location is irrelevant. On the other hand, it is tedious and likely impossible to manually enter *GPS coordinates*. Instead, the user must

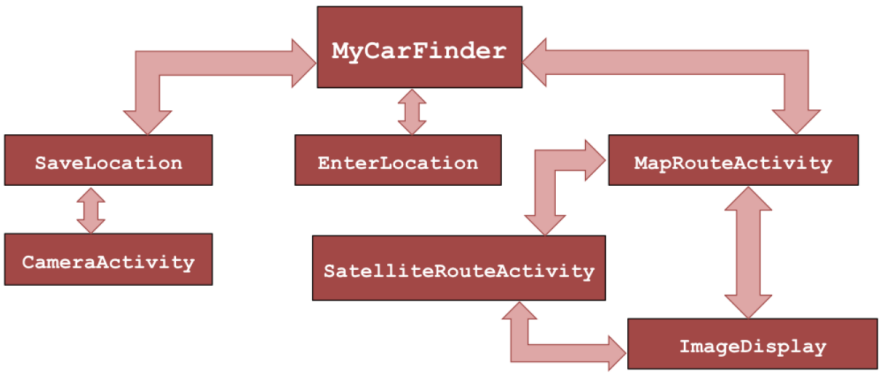

Figure 1: Block diagram showing the different program components for PinPointer and their communications. The rectangular boxes represent the different components and the block arrows show how they interact with each one another.

be permitted to enter *addresses* which are translated to GPS coordinates through geocoding.

- **CameraActivity:** Invoked by **SaveLocation**, this component communicates with the camera application to take and store a picture.
- **MapRouteActivity, SatelliteRouteActivity:** These two components govern the different navigation views, the map and satellite views. Each invokes the other, permitting the user to switch between the two views.
- **ImageDisplay:** Invoked by the two mapviews above, this helps display the picture taken for a location while saving it. At the navigation screen, the image is provided as an icon by the map. However, one may want to see a magnified view of the image. This is done by **ImageDisplay**, invoked by clicking the image icon. A subsequent click takes you back to the mapview with the iconified image.

Timeline

The PinPointer involves approximately one (wo)man-month of effort. The effort can be broken up roughly into the following categories.

- Getting up to speed with Android App implementation (1 week)
- Conception and basic design (3 days)
- Understanding different available APIs (4 days)

- Basic Implementation (1 week)
- Fine-tuning implementation and debugging (1 week)
- Documentation and re-structuring (2 days)

Testing Plan

Our testing strategy involves two components:

- Full-system testing to see if the different user functionalities are working in unison, and if not, which components has problems. This is made easier by the design choice to map each program component to a unique item on the user interface screen.
- Traditional testing/debugging of individual system components.

PinPointer is a user-facing App: it is meant to solve the location and navigation problems for the end-user with a smartphone. The easiest and most convenient way of doing "full-system" testing of the App therefore is by just acting as the user, installing it on a smartphone, and using it "in field" for different user scenarios. This is expedited in part because the author has herself recently moved to a new and unfamiliar city, thereby providing the App with an ideal "customer": a user who frequently gets lost in a new place and needs to find her way. (Indeed, the design of the PinPointer is motivated in part on the author's frustration at finding her car after shopping or dinner in Downtown Portland.)

For individual components (once a fault has been isolated by full-system testing to a specific component) we resort to traditional program testing/debugging techniques, *e.g.*, making use of output logs and using standard debugging interfaces.

Tools and APIs

The program makes effective use of a number of available APIs. Fig. 2 shows some of the critical APIs. The APIs are used to provide the following functionality:

- **Camera API:** This API manages the actual camera hardware, and used for snapping pictures with PinPointer to be associated with the current location.
- **Location API:** This API provides access to the system location service (GPS). We use this to determine the coordinates of current location to be saved.

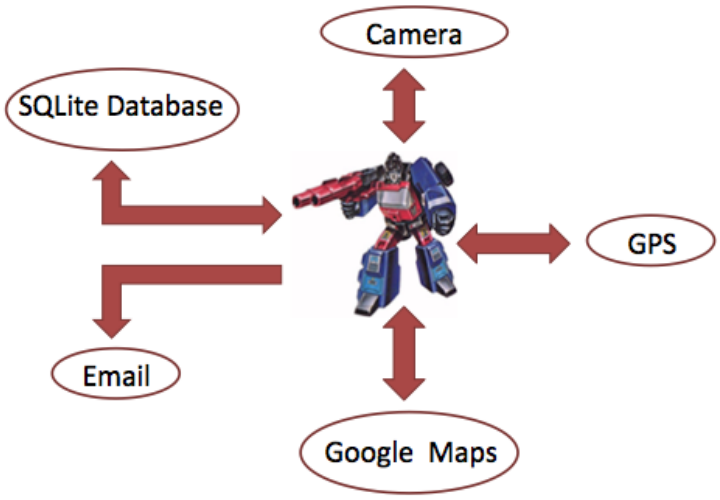

Figure 2: Key APIs for PinPointer. The picture in the center is the PinPointer logo.

- **GeoCoder:** While a GPS location provides precision and is a good aid to *navigating* to a specific place, a user finds it more intuitive if presented with an address rather than latitudes and longitudes. Likewise, the user, when entering a location manually, will likely prefer to enter an address rather than GPS coordinates. The Geocoder API provides the key functionality to turn an address to GPS coordinates and vice versa (respectively referred to as forward and reverse geocoding). PinPointer internally stores locations in terms of GPS coordinates but presents them to users as addresses.

Performance Issues

The following are some of the key performance issues with PinPointer.

- Since GPS is used (and enabled) for a significant amount of time, the battery is drained quickly.
- Sometimes the location estimate obtained from GPS is not always accurate. Accuracy varies even if the same location is saved a few times within a space of seconds. Determining how many times the App should sample a location with GPS involves a trade-off among accuracy, speed, and battery efficiency.

- Storing the image in SQLite database will be a performance hit, so the image file has to be stored in external storage like the SD card.

Technical Risks

There are many technical risks in building an App like PinPointer, which makes critical use of communication with a number of APIs. The following are two critical ones.

- Both the Geocoding and Google Maps API have a query limit (2500 and 1000 unique requests per person per day respectively). This is typically not a problem for Geocoding (one hardly expects to store and retrieve 2500 locations per day), but can be a problem with Google Maps, particularly if one "refreshes" the location several times when navigating to a destination.
- PinPointer depends critically on the availability of GPS signal to record a location. This limits the locations in which PinPointer can actually be used. For instance, within a covered, enclosed parking garage, the application of PinPointer is limited by the unavailability of GPS signal.

Internationalization Plan

Currently, PinPointer is developed with the US market in mind. There is no envisaged difficulty with its use in other countries where map and routing informations are available. However, to do so, our geocoding must support address translation for such countries. In our current work, we have used forward geocoding through `Locale.US`; this API assumes that the address being provided is a US address and returns the corresponding GPS coordinates. For use in other countries we need to use forward geocoding with other locales. Of course, our App will not be of use in a country for which routes and maps are currently unavailable.

IP Issues

We expect to put PinPointer in public domain. We believe this will be useful both for other users to study and modify the App thus augmenting it with new functionality which the author may not have anticipated, and to help increase the customer base. We do not expect any patent infringement since the work was done using public APIs on a standard platform with Java.