Location Aware Reminders for iOS

Supervised by Russell Beale

Alex Layton - 1080395

25 March, 2013

Abstract

Mobile apps have become increasingly popular with the rise of various mobile operating systems and their respective app stores. These app stores allow users access to a multitude of apps from utilities to games, and as the capabilities of these phones increase the allow more complex apps to be built using the sensors built into the device. A popular mobile operating system is Apple's iOS, which was updated in iOS5 to include a Reminders app. This app features the ability to add geo-fenced location reminders which alerts the user when they are in the specified location. These geo-fenced reminders have few uses, whereas a reminder which could preempt when to alert the user based on their current location and their destination has a wide range of uses.

Contents

0.1 Introduction

0.1.1 Motivation

With the increasing popularity of mobile devices their are more opportunities to create mobile apps which can be distributed through various app stores on a number of mobile operating systems. The operating systems boast millions of users, which collectively download billions of apps from the hundreds of thousands available on their devices app store. Consider Apple's iPhone, which runs iOS. Their latest version of iOS - iOS6 has around 300 million users and their App Store offers over 800,000 apps that have been downloaded a total of 4 billion times (cite apple). This shows what a great opportunity these app stores are and allow smaller developers to distribute their app on a scale thats usually only seen by larger developers. Although their is the possibility of mass user adoption, it is also easy to go unnoticed amongst the other hundreds of thousands of apps.

As the adoption of these devices increases they are being used to complete more and more tasks. These tasks can be important such as calendaring in order to allow the user to organise their time or setting alarms and reminders. Traditionally a reminder simply alerts the user at a set time with the given reminder. As the capabilities of mobile devices have increased and the introduction of GPS and GLONASS (maybe a footnote) sensors have allowed reminders to make use of location data, which can be seen in Apple's Reminders app where instead of a reminder being triggered by a time it is triggered by a location, specifically when the users current location is within a certain distance from the desired location. This technique is known as geo-fencing (cite geo-fencing) and is essentially creating a virtual perimeter around a location so once that area is entered a task can be performed, whether it is downloading something in the background or in this case alerting the user with a reminder. The problem with these geo-fenced reminders is that they do not take time into consideration and as a result are limited in scope - a better solution would be one that considers both location and time.

By taking into account both location and time these reminders could be used to remind the user dependant on how far away they are from their destination and how long until they need to be there. For instance, if a user set a reminder for half an hour in the future, and it takes 15 minutes to reach that destination, providing the user doesn't move they should be alerted in 15 minutes in order for them to reach their destination on time. The scope of this functionality is much greater than the previously mentioned geo-fenced reminders, which could be used in any application that involves the user reaching a destination before a deadline. This project looks at the implementation of these reminders in the form of an iOS app.

0.1.2 Aims

There are several aims involved in this project. First and foremost the main aim of this is to implement the preemptive reminder functionality in an iOS app. The other main aim is to release the proposed functionality in an app available on the App Store. Another aim of the project is to create the preemptive reminder functionality as a library which would allow for other apps to easily make use of these reminders. Since

using location services on a device can drastically drain the battery an aim is to try and make the app as efficient as possible by only using location when needed and using these services in a low power mode if possible. The created app to demonstrate the reminders functionality should also have a good user interface, as well as good user experience when adding location reminders.

0.2 Analysis and Specification

0.2.1 Existing Apps

It is important to research existing apps when building an app for a mobile device. As previously mentioned, with hundreds of thousands of apps available on the various distribution channels there is the possibility that the proposed functionality has already been implemented. By looking on the different app stores its possible to see what applications are already available. The mobile platforms with the biggest user bases are iOS and Android so its logical to target their respective stores - the App Store (cite app store) and the Google Play Store (cite play store) as the basis for researching existing apps.

On iOS there exists the previously mentioned Reminders app, which features the geo-fenced reminders functionality. Another app available on the app store is Checkmark (cite checkmark). This app builds upon the functionality in Reminders by allowing the user to set the radius of the geo-fence around a location as well as allowing the user to set a timer to alert them a specified amount of time after reaching the geo-fenced area. Geo-Reminders (cite georeminders) is another app that offers geo-fenced reminders.

For Android the existing apps are very similar in functionality to the ones available on iOS. Milwus Location Alarm (cite app) is an example of geo-fenced reminders.

From the research conducted into existing apps, there are currently no apps that make use of the proposed functionality.

0.2.2 Objective-C

In order to build an iOS app an understanding of the Objective-C programming language is required. The language builds upon the C language by adding object-oriented patterns. It allows for method and classes as found in other objected-oriented languages while making use of core C APIs. As this report will cover the implementation of an app, it may be useful to demonstrate some parts of the Objective-C language.

```
#import <Foundation/Foundation.h>

@interface HelloWorld : NSObject
+ (void)hello1;
- (void)hello2;
@end

@implementation HelloWorld
```

```
+ (void)hello1
{
    NSLog(@"Hello, World");
}
- (void)hello2
{
    NSLog(@"Hello, World");
}
@end

int main(void)
{
    [HelloWorld hello1];
    HelloWorld *hw = [[HelloWorld alloc] init];
    [hw hello2];
}
```

All Objective-C classes have an interface, this is where all the public methods are declared for that class. The method 'hello1' is declared with a '+' which shows that it is a class method, while 'hello2' declared with a '-' is an instance method. In this example the interface is declared inline, but it is more common to find a '.h' file for the interface while the implementation is found in a '.m' file.

The implementation for both of the methods is straight forward, with them both printing 'Hello, World'. You may notice that the function to print the string is prefixed with 'NS' as well as the class we are inheriting from. This is due to the fact that all classes within the Foundation framework were created for the NextStep platform and as a result still carry the naming convention from this platform. Class prefixes are common in Objective-C for instance the framework for user interfaces on iOS is called 'UIKit' and as a result all classes in this framework being with 'UI'. This project will include similar prefixing of class names.

Like other programming languages the main method is where the code is executed. Method calls in Objective-C are wrapped in square brackets, with the first word inside the bracket being the class and the second being the method call. As you can see, since 'hello1' is a class method the class does not need to be instantiated while the call to 'hello2' requires the class to be allocated memory and initialised.

Delegation is a widely used programming pattern within Objective-C. Since delegation will be used throughout the project it may also be useful to introduce this concept. According to the Apple Developer Documentation (cite docs), a delegate is an object that acts on the behalf of another object in order to respond to an event received by this other object. For example, a class may hold an instance of a text field (UITextField) and the class may wish to be notified when the user presses the enter key on their keyboard, so the application can behave accordingly. The documentation shows that the text field has a protocol reference - UITextFieldDelegate and that the textfield has a property called delegate that can be any object that subscribes to this delegate protocol. In order for the class that has the text field to be notified it should set the delegate property of the text field as itself and implement the following delegate method;

```
- (BOOL)textFieldShouldReturn:(UITextField *)textField
```

This means that now whenever the enter button is pressed while using the text field, the text field instance calls the above method on the instance that is stored in the delegate property. A better example of delegation is;

```
@protocol ADelegate <NSObject>
- (void)aDelegateMethod;
@end

@interface A : NSObject {}
@property id<ADelegate> delegate;
- (void)start;
@end

@implementation A
- (void)start
{
    //do something here then call
    if (_delegate) [_delegate aDelegateMethod];
}
@end

@interface B : NSObject <ADelegate>
- (void)aMethod;
@end

@implementation B
- (void)aMethod
{
    NSLog(@"Hello, World");
}

- (void)aDelegateMethod
{
^^INSLog(@"Hello, World");
}
@end

int main(void)
{
^^IA *aObj = [[A alloc] init];
^^IB *bObj = [[B alloc] init];
^^Ia.delegate = b; //register b as a's delegate
^^I[a start]; //on some event [delegate aDelegateMethod] is called
}
```

The tools to write and run Objective-C code are freely available and developers can install the Xcode IDE which provides a toolchain to allow Objective-C code to be compile, run (or simulated), tested and profiled. Xcode requires a machine running Mac OS X, but Objective-C code can be compiled on any machine that can run the llvm compiler. In order for developers to release their apps on the App Store, they must be provisioned and submitted to Apple for approval. An Apple developers license is required in order to do this.

0.2.3 Literary Review

The Apple Developer Center (cite dev centre) provided all the necessary literature to enable the implementation of this project. This website provides all the necessary documentation to use Apple provided frameworks as well as presentations from developer conferences introducing newly implemented functionality in frameworks and new features in the iOS operating system. Apple also provide example implementations in order to demonstrate presented functionality.

Since the proposed reminder functionality requires location it was important to research into the existing location frameworks and APIs available on iOS. The WWDC presentation - 'Staying on Track with Location Services' (cite presentation) provides a good high level insight into the location technologies used within iOS. This presentation identified several factors needed for the implementation of location reminders. One of these factors is background execution, since reminders are essentially useless if the app can only be run in the foreground. In order to be allowed to execute in the background the app needs to register as an app that can run in the background. The presentation demonstrates how to register as a background app by adding the following to the applications information property list file.

(add plist code)

Although it is possible to run standard location services while in the background, there exists methods within the CoreLocation framework to allow for monitoring of significant changes. These significant location changes make use of cell tower triangulation instead of wi-fi, GPS and GLONASS which is used my the standard location changes.

TODO: Add section on delegation to objective c and then talk about location update thingy

Another factor in this project is using maps. Since reminders will use location, using maps will provide a way to give user specified locations context. Apple also provide a MapKit presentation from their WWDC sessions called; 'Getting Around Using MapKit' (Cite presentation). This presentation introduces the new maps released in iOS6 as well as how to implement them into an app. Since MapKit uses the CoreLocation framework itself in order to show the users current location on a map, it is easy to use the two frameworks together.

0.3 Design

0.3.1 User Interface

0.3.2 User Experience

0.4 Implementation and Testing

0.5 Project Management

0.6 Results

0.7 Conclusion