

# Prototyping Location Based Services

*Richard Bailey, Sebastian Becerra, W. Keith Edwards*

College of Computing  
Georgia Institute of Technology  
801 Atlantic Dr  
Atlanta, GA 30332 USA  
{psy, becerra, keith}@cc.gatech.edu

## ABSTRACT

Developing a Location Based Service (LBS) is a complex process requiring experience in several areas of computer science: location sensing, client/server communication, and HCI, as well as the domain of the service's implementation. As a result, creating a LBS from scratch is outside the reach of most developers—much more difficult, for example, than creating a simple web page. This paper describes a framework we have constructed to ease the creation of LBS's.

## INTRODUCTION

In recent years, Location-Based Services (LBS) have become an increasingly active research area. Earlier work has reported on specific LBS examples [1][3], specific location sensing technologies useful in the creation of LBS's [2], adoption concerns [5], privacy concerns [6], and so on. Despite this effort, most LBS's are created by groups with intricate technical knowledge—often the same groups that developed a particular location sensing technology. Thus, creation and use of LBS's has not spread much beyond the realm of research.

However, recent developments in location sensing technologies may pave the way for easier development and deployment of LBS's. For example, the Place Lab system [4] can attain location accuracy sufficient for a range of applications without the deployment of costly fixed infrastructure, and without the need for additional hardware (beyond basic WiFi or GSM radios) on client devices. Approaches such as this both reduce the cost of deploying LBS's and widen their availability; however, they do little to reduce the software complexity of service implementation.

This paper describes our work aimed at reducing the complexity of building LBS's. Our architecture uses off-the-shelf location tracking (currently Place Lab, although extensible to other approaches) in the client. Location-based services are combinations of back-end functionality (in the form of Java and PHP) coupled with front-end computation (in the form of Javascript code that executes in standard web browsers). Our framework uses a custom

AJAX-driven messaging solution for communication between client web browsers and LBS's.

This basic architecture gives us the foundation for a wide-area deployment of LBS's. Individual services can run on arbitrary computers, much as current web servers can be hosted on desktops, laptops, and managed systems. Normal, unmodified web browsers serve as LBS clients, using our client-side code for rich communication with back-end services.

Normally, the creation of a web-oriented LBS is an arduous task. Developers must create the back-end logic of the service, typically through a server-side programming environment such as Java Servlets or PHP. Depending on the needs of the service, this code may need to communicate with a back-end database for persistent storage.

Two common approaches are to either use standard HTML mechanisms, or to use Javascript and AJAX-style code. Both of these approaches have drawbacks. Standard HTML-based front-ends have limited interactivity, and only coarse-grained communication with the back-end server. For example, if a service wishes to track the user's location on a map in realtime, constant refreshes must be performed to update the page contents. AJAX-style applications (such as Google Maps) have the ability to maintain constant (or on-demand) communication with the server as well as the ability to perform UI updates without the need to refresh an entire page.

Thus, we have created a *toolkit* to ease the creation of LBS's by developers. This toolkit combines back-end functionality (executing on the machine hosting the LBS) as well as front-end functionality (executing on the client's web browser), and uses AJAX-style messaging for communication. This toolkit manages messaging between the users of the system and the service, handles the process of sending the user's positional data to the service, and provides a number of other features useful for constructing a wide range of LBS's.

In the remainder of this paper we describe the overall web-oriented architecture we have developed, as well as our toolkit design.

## OVERVIEW

In our model, LBS's can be hosted on arbitrary machines in the public Internet. Also, rather than having a specialized client application that provides the front-end for a single

particular service, we use standard web browsers to provide the front-end to *any* service.

### Getting the Client's Location

One key requirement of this architecture is that a mechanism must exist that can (1) determine the client's location and (2) communicating that location to the LBS backend. One approach is that taken by Place Lab, which provides a local web proxy that attaches headers `X-Lat` and `X-Long` to all outgoing HTTP traffic. This approach has three drawbacks: first it requires a user to configure a proxy server when using their browser of choice to access services. Second, it may conflict with any existing proxy settings that might be required, for example, in a corporate setting. Finally, and most importantly, any website that you visit while using the proxy may learn the user's location.

Thus, one key feature of our architecture is a mechanism that allows communication of client location to selected LBS's using existing web messaging paradigms and not requiring a proxy server. This approach is transparent to the user, preserves existing proxy settings, and can potentially maintain better user privacy, at the expense of having to install a small piece of software on the client. Given that clients would have to install Place Lab (or another location sensing system) anyway in order to use *any* LBS, we believe that this is an acceptable cost.

In our approach the client-side software runs a lightweight webserver on the local host; this web server is configured so that it will only accept connections that originate *from* the local host, and thus cannot be accessed by any remote machine. This webserver wraps the Place Lab client functionality, so that it can determine the client device's location. The execution of this webserver is transparent to the client, as is the execution of the Placelab location sensing software.

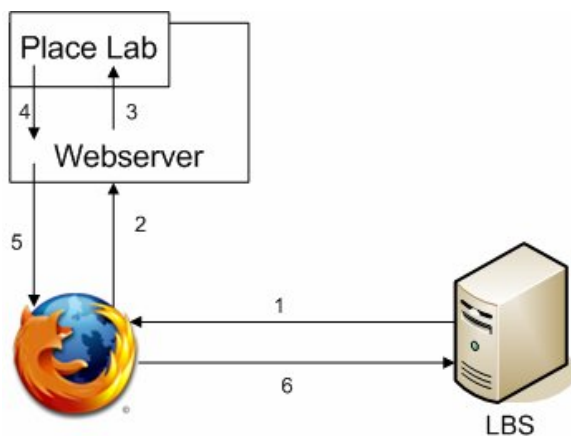


Figure 1. A graphical representation of message flow to acquire location

When the web browser on the client connects to a LBS, the LBS serves it a page that references <http://localhost/location.js> (Figure 1,1). The browser in turn requests this page from the lightweight web server running

on the local host (2), that web server determines the client's location via Place Lab (3) and generates Javascript (4) to set the latitude and longitude in predefined variables, and returns this in a resulting page (5).

Finally, Javascript in the original page returns these results back to the machine hosting the LBS, passing the latitude and longitude as parameters to the request (6).

After a one-time installation of the client software package (which includes both the lightweight web server and Place Lab), users do not have to make any further changes to their system, and existing proxy settings are unaffected. Location-based services need only include a small amount of pre-defined Javascript in order to access client location. Further, rather than passing the client's location to every web server encountered, this approach allows us to achieve greater privacy. In the default setup this is accomplished by only sending location data to a server that knows the manner in which to request it. Should further security be needed then the service developer could encrypt the data before transmission using a public-key provided with the lightweight webserver install.

Using this basic framework for transparent communication between clients and services, as well as optional authentication, we developed a toolkit framework designed to reduce the complexity of other aspects of creating LBS's.

### FRAMEWORK ARCHITECTURE

The prototyping framework exists in three parts: PHP and Javascript that are used to manage interactions between the back-end service and the browser, a Java library that acts as a superclass for common back-end service functionality, and interface classes to manage the interaction between them. To develop a new LBS, the programmer needs only write code in two contexts: *Service* (typically a Java back-end) and *Interface* (an AJAX or Dynamic HTML front-end).

### Lifecycle

The lifecycle of a LBS in our framework follows a simple three stage state machine. Stage one is waiting for a user login notification, this is a non-blocking wait. If we detect a new user login, we transition to stage two which creates a "session" for the user and then transitions to stage three. In stage three we analyze the list of active users and determine if any have logged out or have been inactive for over ten minutes (this is configurable). Alternatively, if a new user login is not detected in stage one then stage two is skipped and the system transitions instead to stage three.

It should be mentioned that beginning a session, in the context of our framework, is simply an instantiation of the service-specific `UserHandler` and the expiration, or destruction, of a session means that the `UserHandler` associated with the user will be stopped from processing messages and shut down.

### Service

Within the service portion of the framework there are two types of classes: those that the developer can (or must) extend and those that the user simply uses.

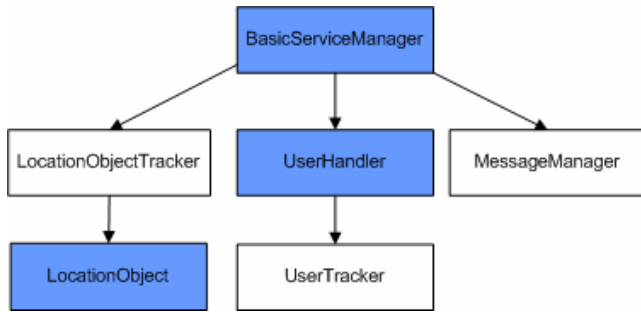


Figure 2: Classes in the framework. Blue is an indication that the developer can extend these classes to customize their service

The classes that the developer extends are shown in light blue (Figure 2) and represent portions that may be specific to the behavior of a particular LBS, of these the developer much write a service specific UserHandler.

BasicServiceManager contains information and functionality that is common to all LBS's built on our framework such as session creation and destruction. Any service that provides service-specific will extend this class.

UserHandler provides functionality for dealing with any incoming messages from the client sent with the AJAX portion of our framework. Services can extend this class to specialize their behavior to deal with new types of messages from clients. An instance of the particular service's extension of this class will be created and assigned a user as users log into the service and shutdown as user sessions expire or users logout, this creation and destruction is directly managed by the BasicServiceManager.

LocationObject is a general class that knows how to place itself into a database. Its use is completely optional for a given service – it would be used when an object in the system needs to both know its location and persist between user logins. Additionally this provides some basic location-based functionality such as finding other objects nearby. Though its usage is optional, it simplifies the creation process by allowing users to let the framework deal with all database interaction.

**User Interface**

On the client machine, our framework consists of Javascript code that executes inside a standard web browser. Since browser security models disallow using the XMLHttpRequest object to request pages from hosts different from the one serving the current page we dynamically request a script object from localhost and execute the script returned to set values. The framework's message routing terminates at a specified javascript function. When the service (UserHandler) sends a message to the end user, it gets turned into a call to the function msgRecv in main.php. The responsibility to implement

this function falls to the developer. This approach gives the developer the ability to either write a custom interface or use one of the many third party Javascript UI toolkits such as the Yahoo! UI library[7] or Rico[8] for creating and manipulating the interface. The goal is to not limit the developer to any particular interface on the client-side while providing a maximal amount of structure and thus simplifying the process of message delivery.

**Component Interface**

For managing interaction between the Service side and the User Interface the framework provides a small set of utility classes.

MessageManager's sole purpose is to send and receive messages between the UserHandler and the LBS user's browser. The messages sent through MessageManager arrive in the client-side javascript function msgRecv to be handled by the developer.

UserTracker is a wrapper for the user location tracking system we have implemented. Location data of a user, when needed, can be acquired from UserTracker simply by passing the id of the user we're looking for. Additionally User tracking can average location over time if desired.

**CASE STUDY**

This section describes the implementation of a particular service using our framework. This service, called AirPostit (AP), is a canonical geographically-based messaging service, much like GeoNotes [9]. AirPostit acts like a large bulletin board to which any interested party can post announcements. An AP "note" has a location and a range from which it can be detected, as well as the author, subject and message. A user of the AirPostit LBS connects their web browser to the service; as the user moves through the area, the browser will receive updates dynamically through the browser that display the subjects of notes visible from the current location. Should the user be interested by the subject of any of these notes, selecting a note subject will request the message body from the AP service.

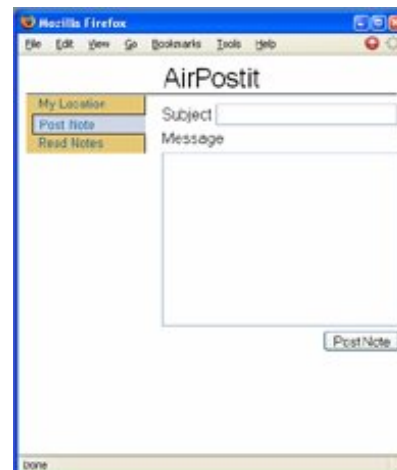


Figure 3: A very basic UI built for our AirPostit service. This was constructed using no third party toolkit.

Implementing AirPostit required extending the BasicServiceManager class. The extension of this class required only a few lines of code. LocationObject was extended in order to represent notes. When implementing this service nearly all non-trivial code was directly involved in parsing messages from the user or constructing messages to send the user in the extension of the UserHandler class.

The implementation of this service required fewer than 300 lines of Java code. Of that, approximately 75 lines were trivial (setting variables, getters, setters, et cetera) and none of the code was highly complex. In fact, most of the remaining code dealt with parsing and creating strings to be used in messages. Additionally, the time to create this service was less than four hours. While this service had a rather simple UI, more complex UIs can be added through the use of third-party Javascript UI toolkits.

### **FUTURE WORK**

In the near future we plan to release a public beta of the framework. Therefore our short-term plans are to complete developer documentation and package the framework with the required tools (Apache, PHP, MySQL, and PlaceLab). Beyond this, however, we have a number of features we plan to add to the framework.

We aim to add a batch communication framework to the client to reduce network load and increase efficiency. Such batch communication will increase responsiveness when on slow links.

In addition to this low-level implementation issue, which we generally do not expect to be visible to developers using our framework, we plan to implement a range of additional features to ease the creation of LBS's. While our framework allows the creation of services without requiring that developers deal with the complications of location sensing hardware or persisting user data, other functionality can provide additional support for LBS development. For example, we are developing an extension to our framework that will accept XML representations of data objects, and will automatically generate Javascript libraries that allow a developer to interact directly with those objects at the browser level. In the long term, our goal is to reduce the technical acumen required to build LBS's to the point where new service creation is largely done through editing XML files and writing small amounts of Javascript code.

### **SOFTWARE ACCESS**

While at the time of writing our framework is not yet in general release, any interested parties that would like a snapshot of its most recent state can contact Richard Bailey ([psy@cc.gatech.edu](mailto:psy@cc.gatech.edu)).

### **REFERENCES**

1. Brown, B., Chalmers, M., Bell, M., MacColl, I., Hall, M., and Rudman, P. "Sharing the Square: Collaborative Visiting in the City Streets," Proceedings of European Conference on Computer-Supported Cooperative Work (ECSCW), Paris, France, 2005.

2. Bahl, P., and Padmanabhan, V., "RADAR: An In-Building RF-Based User Location and Tracking System," Proceedings of IEEE InfoCom, 2000.

3. Barkhuus, L., Chalmers, M., Hall, M., Tennent, P., Bell, M., Sherwood, S., Brown, B. "Picking Pockets on the Lawn: The Development of Tactics and Strategies in a Mobile Game." Proceedings of Ubiquitous Computing (UbiComp), Tokyo, 2005.

4. LaMarca, A., Chawathe, Y., Consolvo, S., Hightower, J., Smith, I., Scott, J., Sohn, T., Howard, J., Hughes, J., Potter, F., Tabert, J., "PlaceLab: Device Positioning Using Radio Beacons in the Wild." Proceedings of Pervasive, 2005.

5. Mansley, K., Beresford, A., Scott, D., "The Carrot Approach: Encouraging the Use of Location Systems," Proceedings of Ubiquitous Computing (UbiComp) 2004.

6. Smith, I., Consolvo, S., LaMarca, A., Hightower, J., Scott, J., Sohn, T., Hughes, J., Iachello, G., Abowd, G. "Social Disclosure of Place: From Location Technology to Communication Practices," in Proceedings of Pervasive, 2005.

7. Yahoo Developer Network, "Yahoo! UI Library", 2006, <http://developer.yahoo.com/yui/index.html>

8. Rico, "JavaScript for Rich Internet Applications" 2006, <http://openrico.org/>

9. Espinoza, F., Persson, P., Sandin, A., Nyström, H., Cacciatore, E., Bylund, M., "GeoNotes: Social and Navigational Aspects of Location-Based Information Systems", Lecture Notes in Computer Science, Volume 2201, Jan 2001, Page 2