AtLongLat

Location aggregation and developer platform

Jeff Weinstein
Stephen Pike

Faculty Advisor: Zachary G. Ives

April 24, 2009
Abstract

AtLongLat (http://atlonglat.scpike.net) lets users share their aggregated location, from many devices, with web services via a developer platform focused on privacy and security. Consumers can safely broadcast their location and control the granularity and availability of their data. Application developers can use a device independent query for accessing location filtered by the user’s privacy settings.

AtLongLat receives and stores user locations from devices and other sources. These sources are authenticated and registered to a user using a handshake mechanism. The web server calculates a location estimate using the provided aggregation algorithm. Applications are linked to a user by the same authentication mechanism. These applications can request a user’s location subject to the user’s privacy settings. Users may control when applications have access as well as location proximity. Settings are customizable on the application level. 3rd party application programming interfaces (APIs) are available for sources and applications to send and request data to and from AtLongLat. AtLongLat is surfaced to users and developers through a web portal. The 3rd party APIs are available on any platform that can perform URL requests.

Motivation

Consumers have many sources of location data including devices capable of GPS, cellular triangulation, and Wi-Fi router look up. They also have tagged location data such as calendar appointments and direct user input. Consumers are interested in the best estimate of their location. There is increasing interest in using location aware services as well as broadcasting location to friends and family. It is difficult to manage one’s privacy and security given the many devices and services available.

Application developers want the best estimate of a user’s current location to create interesting web services. The best estimate may not be gleaned from the device an application is currently running on as the device might not have location awareness. Developers and consumers also need a common privacy and security format that is acceptable and transparent. Currently, application developers query location directly from a device. This query is made at the time the
application needs the data. There are very few devices with push delivery. A developer is rarely interested in the method by which location is determined. The problem can be abstracted so a service can run on any desktop, laptop, or mobile and still provide location awareness.

**Related Work**

Our investigation of related work consisted of reviewing the market for location aware services and applications, the growing number of devices with location sensors, and offerings of location aggregation and application interface layers. Mobile devices and location based services are gaining in popularity and ubiquity. The need for location data is growing and demonstrated by the success of services such as Loopt (Loopt, 2008), Where (Where, 2008), and Whrrl (whrrl, 2008). Loopt is a social network site which connects friends nearby. Where delivers notifications of interesting places near you. Whrrl helps people locate their friends and share experiences of nearby restaurants and events. Most recently, Google has released Google Latitude which shares a user’s location with his or her friends. It uses GPS location data from smart phones other sources to update one’s location. (Google, 2009)

Mapping applications from Google, Yahoo, and Microsoft also demonstrate the need for location data. Photo sharing sites like Yahoo’s Flickr (Yahoo, 2008) and Google’s Picasa (Google, 2008) leverage location to geotag photos. Popular messaging and status services like Twitter (Twitter, 2008) tag updates with location when available.

There are a growing number of devices with location sensors. According to comScore the size and growth of the Smartphone market will exceed the laptop market for the next 5 years. The Smartphone market is predicted to grow at 30% annually. Cellular triangulation has been on other handsets for some time. There are currently over 3 billion mobile handsets. Users are increasingly interested in using their handsets in tandem with location services – Mobile Map use has grown 89% since last year in the United States, and 49% in Europe. (comScore, 2008)

There are a few companies working on location aggregation and application interface layers similar to AtLongLat. Yahoo’s Fire Eagle offers a service for applications to access and update location data along with a light privacy and security profile. They do not attempt to aggregate multiple sources to estimate a best estimate from multiple inputs but simply take the
most recent location update. They offer an extensive API for developers to access and update a user’s location.

Skyhook Wireless combines GPS, cellular triangulation, and Wi-Fi-router look up to produce a best estimate of location. The Wi-Fi look up matches current Wi-Fi router’s MAC addresses to a hard-coded list of locations for routers. Skyhook has created a mapping of Wi-Fi routers to location by driving a GPS enabled car and documenting router positions. This hard-coded mapping is stored locally on a device. Their services are used to produce their best estimate of a user’s location. It’s currently the mechanism for the iPhone and iPod Touch’s Locate Me feature.

Xtify (xtify, 2008) is self described as “a mobile application to bring the power of location to web applications.” Xtify allows web applications to communicate alerts and notifications to the devices through a push system. This allows an application to notify a user when he or she breaches some area. They are not focused on aggregation methods.

Technical Approach

Overview

AtLongLat solves the brokerage problem between users trying to share their location and developers attempting to access it. The service gathers location from multiple devices, protects that data with privacy controls, and then exposes the filtered location information via a developer platform. The frameworks and technologies used in the project are described in the Enabling Technologies section below. The System Components section outlines the four layers of the system.

Enabling Technologies

AtLongLat’s architecture takes advantage of many open source web technologies. The API uses the OAuth protocol to verify and authenticate data from devices and applications. The reverse geocoding is implemented using the Google geocoding platform. The server runs on the Django web framework, which provides web API calls and database abstraction.
OAuth Protocol

OAuth began in 2006 as “an open protocol to allow secure API authorization in a simple and standard method from desktop and web applications.”\(^1\) It gives web service developers a mechanism to expose external APIs that require user authorization without releasing user credentials. A brief discussion of the protocol will outline the OAuth process and the benefits for AtLongLat. The OAuth implementation used was built on the open source django-oauth platform written by David Larlet.\(^2\)

To use the OAuth protocol, developers must register their applications with the web service to receive a consumer token. They use the token to sign requests to the service. When a user wants to share their location with a 3rd party a three step process ensues:

1. A user visits the 3rd party site and wants to link with AtLongLat. The external site signs and sends a request to AtLongLat for a request token. This request is sent to http://atlonglat.scpike.net/oauth/request_token/
2. AtLongLat responds with a request token and a unique authorization URL on AtLongLat’s website. The user is directed to AtLongLat’s website, and is prompted to enter credentials and approve the link. AtLongLat redirects the user back to the 3rd party site.
3. The application exchanges the approved request token for an access token, which it can then use to authenticate and make requests to the API. This exchange occurs when the 3rd party sends a request to http://atlonglat.scpike.net/oauth/access_token/

This process provides two benefits to AtLongLat. First, users are able to grant access to applications while only entering their credentials on AtLongLat’s website. This helps solve some of the trust issues between users and developers. Since the credentials are never sent to the application, users can turn off access to their location from the 3rd party at any time without changing their login information.

The second benefit is security. OAuth provides verification of the origin of requests and protection against man-in-the-middle attacks. While OAuth does not require encryption of requests, it does force request signing. Information is exchanged in plain text but is coupled with

\(^1\) http://oauth.net/
\(^2\) http://code.welldev.org/django-oauth/wiki/Home
a signature built using the token’s secret. A generated unique value is included in the signature to guarantee requests can only be made once and not from malicious parties.

The signing of requests does not prevent data from being intercepted but rather verifies the integrity of the data and prevents request spoofing. No immediately identifiable information is passed along with the requests. Attackers could determine when two requests pertain to the same user but would not have information to his or her identity.

**Reverse Geocoding**

The system stores data at varying levels of granularity. Sources, however, are only permitted to send readings in latitude and longitude format. Bridging the gap between these two formats, the aggregation framework uses a geocoding library to calculate the intermediary location data. The Geopy Python module\(^3\) interfaces with the Google reverse geocoding platform and Python methods. The response from the reverse geocoding is parsed into country, state, zipcode, and city information.

**Django Web Framework**

Designing and building the architecture required for AtLongLat involves database access and server side programming. The Django Web Framework\(^4\) allowed for a very high-level approach to system design. Django is a Python web framework that maps URL requests to methods. Server side programming was written in Python which is a familiar, powerful language. Also, Django provides a mapping between Python classes and SQL database objects.

**System Components**

AtLongLat consists of four layers – the source interface layer, aggregation layer, privacy layer, and application interface layer (See Figure 1). Each layer was developed as a separate, pluggable piece of the architecture to facilitate independent development and testing.

---


\(^4\) [http://www.djangoproject.com/](http://www.djangoproject.com/)
Source Interface

The source interface is located at http://atlonglat.scpike.net/oauth/update/. An authorized source can POST to this URL with an OAuth request sending a location reading. Its POST must include a latitude and longitude reading and can optionally include a timestamp. If a timestamp is not provided, the server will use the current time. Upon receiving an update request, the server will respond with an XML document giving any error information. Documentation for the server response can be found in Appendix 1.

Application Interface

The application interface layer uses the same OAuth protocol as the source interface, but returns a more detailed XML document, and does not take any special POST arguments. The service checks the application and user level privacy constraints and returns the filtered location. The XML response returns blank values for values that are not permitted. The server’s response is documented in Appendix 2.

Privacy Layer

User location information is very sensitive and a large part of this project was granting users control over what data was shared about them and at what times. Proximity control determines the location granularity applications receive. Blackout ranges are a mechanism for
restricting access during specific date and time ranges. Users can decide, on a per-application or global basis, what type of filter to be used on the location data. The levels of obfuscation filters are country, state, zip code, city, exact, and no access.

In addition to controlling the level of obfuscation, users can decide when applications are allowed to receive access through blackout ranges. Users can set, per application, an unlimited number of blackout ranges restricting what data is available at what times. The following types of ranges can be restricted using blackout ranges:

- During specific times (i.e. 9am to 5pm)
- On specific days of the week (i.e. Monday-Friday)
- Between specific date ranges (i.e. May 12 – May 29)

**Aggregation Framework**

AtLongLat is capable of running aggregation algorithms that estimate a single location from many source readings. The aggregation layer also handles reverse geocoding. Calculators are written as Python classes and do not require any knowledge of the database structure. They must implement a common interface, outlining how data will be sent and received, but are otherwise independent of the system. This allows AtLongLat to swap algorithms in and out for testing. While the only currently implemented algorithm takes the most recently received reading as the current user location, the framework allows for future expansion once larger amounts of data is collected. This can provide a testing ground for future aggregation algorithm research.
System Flow

The AtLongLat system interacts with external devices and applications and has a layered structure. The flow of location data from sources to applications is shown in Figure 2.

![System Flow Diagram]

Figure 2

Example Implementations

The main focus of this project was building the AtLongLat system, which is the central service that exposes the source and application layers. To test the system, four different implementations were written. Two of these are simple proof of concept applications – a manual source for inputting user location data and a client application for displaying the currently stored location – while the latter two are more sophisticated and are described in more detail in Figure 3. All four applications are functional.
<table>
<thead>
<tr>
<th>Name</th>
<th>Web Site</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client App</td>
<td><a href="http://clientapp.scpike.net">http://clientapp.scpike.net</a></td>
<td><strong>Application Interface</strong>: Allows a user to view his current location on Google maps as stored at AtLongLat.</td>
</tr>
<tr>
<td>Manual Source</td>
<td><a href="http://manualsource.scpike.net">http://manualsource.scpike.net</a></td>
<td><strong>Source Interface</strong>: Allows a user to update his current location by navigating to it on Google maps.</td>
</tr>
<tr>
<td>Twitter Updater</td>
<td><a href="http://twitterclient.scpike.net">http://twitterclient.scpike.net</a></td>
<td><strong>Application Interface</strong>: Queries location and syncs it with the user’s Twitter account.</td>
</tr>
<tr>
<td>iPod Application</td>
<td>N/A (runs on device)</td>
<td><strong>Application/Source Interfaces</strong>: Shows the user’s current AtLongLat position, and allows updating of the position via the location awareness feature.</td>
</tr>
</tbody>
</table>

**Figure 3**

**Twitter Updater - [http://twitterclient.scpike.net](http://twitterclient.scpike.net)**

Twitter Updater demonstrates the capabilities of the application interface layer. Twitter implements the OAuth protocol as AtLongLat does, and allows users to input up to 30 characters describing their location. The example application updates a user’s Twitter location as determined by AtLongLat. The Twitter Updater runs as a separate web service and only communicates with AtLongLat over the internet via the APIs. It uses OAuth in an efficient way, demonstrating the linking process a user must do to authorize a 3rd party application. It also handles different levels of proximity. While staying under the 30 character limit leaves out the possibility of syncing exact location, it will update the Twitter service with city, state, or country data depending on what proximity settings the user has allowed.

**iPod/iPhone Application**

The AtLongLat iPhone/iPod Touch application highlights the application and source interface layers. The application has 3 functions: send location determined by an iPhone query to AtLongLat, request location from AtLongLat and display location information in text and as a Google map. It uses the source API and is registered with the default user to send the iPhone’s location. The iPhone application confirms with the user that it has permission to send the data and the call is made via a URL call using the API format. The request function calls AtLongLat.
with the user’s credentials and captures the XML response. The mapping function calls the iPhone’s Google Maps application with the parameters sent from AtLongLat.

**Conclusion**

AtLongLat was designed and implemented as a location brokering service to solve location aggregation needs, device independent queries, and the privacy issues of broadcasting. It addresses the needs of consumers and application developers leveraging existing technologies. The architecture of the design will enable future development on different layers at the same time. The open 3rd party APIs are available for developers to gain access to their users’ location with abstracted privacy and security settings. The brokering service uses many types of web technologies and the API has been implemented on several platforms. The AtLongLat iPhone application sends and requests location data while the Twitter Updater sends location data to Twitter. The work required a web service backend (Django/Python/OAuth), web portals (HTML/CSS/Javascript), and sample sources and applications (iPhone Obj-C/HTML). The AtLongLat system design focused on modularity, abstraction, security, and testing capability. It solves issues facing users and application developers and provides a framework for rethinking location broadcasting over the internet.
Appendix

Appendix 1

<update>
  <error_type>{{error_type}}</error_type>
  <error_msg>{{error_msg}}</error_msg>
</update>

XML document returned by source interface.

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Error.</td>
</tr>
<tr>
<td>1</td>
<td>This is not a valid request, wrong user token combination.</td>
</tr>
<tr>
<td>2</td>
<td>This is not a valid consumer key + source pair.</td>
</tr>
<tr>
<td>3</td>
<td>Latitude Required.</td>
</tr>
<tr>
<td>4</td>
<td>Longitude Required.</td>
</tr>
<tr>
<td>5</td>
<td>The user does not have your source registered.</td>
</tr>
</tbody>
</table>

Error codes returned by source interface.

Appendix 2

<locations>
  <location>
    <userid>{{userid}}</userid>
    <latitude>{{latitude}}</latitude>
    <longitude>{{longitude}}</longitude>
    <zipcode>{{zipcode}}</zipcode>
    <city>{{city}}</city>
    <state>{{state}}</state>
    <country>{{country}}</country>
    <readable_address>{{readable_address}}</readable_address>
    <precision>{{precision}}</precision>
    <proximity>{{proximity}}</proximity>
    <timestamp>{{timestamp}}</timestamp>
    <error_msg>{{error_msg}}</error_msg>
    <error_type>{{error_type}}</error_type>
  </location>
</locations>

XML document returned by application interface.

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Error.</td>
</tr>
<tr>
<td>1</td>
<td>This is not a valid request, wrong user token combination.</td>
</tr>
<tr>
<td>2</td>
<td>This is not a valid consumer key + source pair.</td>
</tr>
<tr>
<td>3</td>
<td>The user does not have this Application registered.</td>
</tr>
<tr>
<td>4</td>
<td>This user has blacked out his location.</td>
</tr>
<tr>
<td>5</td>
<td>No Location Found for user.</td>
</tr>
</tbody>
</table>

Error codes returned by source interface.
Bibliography


