Title: Location-Based Social Software for Mobile Devices

Date: April 28, 2004

Inventors:

Dennis Crowley
New York, NY 10002
dens@dodgeball.com

Alex Rainert
Brooklyn, NY 11231
alex@dodgeball.com

Abstract
The service is best described as "location-based social software for mobile devices" – ways in which people can use their cell phones to find out who and what is nearby. Users opt-in with their location by sending a text message from their mobile phone. This location can be a venue (bar or restaurant), public park, landmark or street address. This location is translated into a geographic coordinate (GPS latitude and longitude), which is then used to deliver information about who and what is nearby. As multiple users opt-in with their locations, it is possible to send users messages based on their relationship and/or proximity to other users.

There are three main components to the service. The first is a web-based social software component which allows users to determine who their friends are, set preferences as to which of these friends they want to send messages to, as well as invite new users to join the service. The second is a city guide based on user-generated content. Users are encouraged to add venues to our database, write comments and reviews of these venues and also describe venues in terms of the "attributes" they offer (e.g. pool table, darts, happy hour, dancing). The third is the mobile component, which receives, parses and sends text messages back to users' phones letting them know who and what is nearby.

Core Functionality:
The service is both (a) a method for collecting opt-in location-tracking information, and (b) a method for determining relevant content to a user based on relationship, affinity and/or proximity. The key features of the service include the ability for a user to:

1. Look up information on venues (address, cross street, phone number)
2. Search for "attributes" that are nearby and tied to venues (e.g. nearby pool tables, nearby happy hours)
3. Broadcast their current location to their first-degree friends
4. Broadcast messages to anyone within a given geographic proximity and time frame
5. Broadcast messages to all first-degree friends
6. Listen for events happening within a given geographic proximity and time frame
7. Be alerted when friends-of-friends are within a given geographic proximity and time frame
8. Be alerted when people who are compatible (based on their profile) are within a given geographic proximity and time frame

Core Components:

1. **Incoming text message parser.** Messages that are sent from users' phones are routed to an email inbox. Software constantly monitors this inbox, checking for new messages. When a new message arrives, the software parses the message into its component parts based on MIME headers: To, From, Subject, Body. Since the format of incoming messages varies from carrier to carrier and phone to phone, custom code is needed to parse MIME messages in many instances. Once the message has been successfully parsed, it is passed onto a piece of software that can interpret the user's intent based on the content of the message.

2. **Extracting data from MIME types.** Once the message is successfully parsed, the To, From, Body messages are then analyzed to determine (a) what city the user is in, (b) who sent the message and (c) what is the user trying to do. The "To" field is used to determine which city the user is currently in. Each city has a unique address to which

users send messages - e.g. nyc@dodgeball.com vs. sf@dodgeball.com - which enables the service to deliver content that is relevant only to that city which the user requested. The "From" field allows us to tell who the message is coming from in the form of an email address (e.g. 9175551212@vext.com). We parse this email address, splitting the data at the "@" sign, using the data to the left side to determine who the user is and using that data to the right side to determine if the message is coming from a mobile device. The data on the left side is compared to a database table that can match the users' phone number to their profile (username, preferences who their friends are, etc.). Data on the right side is compared to a database table of all the accepted email hosts (the service filters out emails that are not arriving from mobile devices, sending a "your device is currently not supported" message to these users). Finally, both the "Subject" and "Body" fields are parsed to extract the command that the user is trying to send.

3. **Processing the users' request.** Once the "To", "From" and "Subject" fields are parsed, these three elements are used to determine the user's intent and send a response back to the user. The service allows users to perform a variety of tasks – for example, looking up venue details or broadcasting a message to friends. Users distinguish their intention by following the guidelines of our text-messaging query language. By using special ASCII characters (see below) in certain ways, users can perform complex actions and queries using a simple interaction. In determining the user's intent, the first thing the service does is to parse the message contents in order to tie the message to a specific action. Depending on the user's status ("Is this message from a registered user or a non-registered user?"), the status of the venue ("Does it exist in the database? Is it geocoded?"), and the user's profile ("Does the user have any friends in their profile? Are they approved to send and receive messages?"), a response is generated and sent to the user.

    a. **Check-in:** A user "checks-in" by sending a message with an "@" plus the name of the venue they are currently located at (e.g. "@ Ace Bar"). If the user is a registered user, the service will compile a list of all of that user's friends (pulled from a database of approved connections between two users – see below "Manage Friends") and send a message to each of these user's mobile phones and/or email inboxes (depending on the user's preferences); therefore, alerting that user to the current whereabouts – venue name, address, time and date - of the user who checked in. ("Dens is at Ace Bar (531 5th Street, btw Ave A & B) at 11:30pm. Reply with @venue name to check-in!"). The user will receive a message back that says "Okay, we just sent a message to 55 of your friends letting them know you are at Ace Bar!" The geographic coordinates of this user (latitude and longitude) are added to the database and are used to determine this users' proximity to other users who have checked-in within a given time-frame. A "check-in" is no longer relevant after three hours. The outgoing messages are formatted so that replies go back to the service and not the user.

        Once messages are sent alerting the user's friends to his/her whereabouts, the database is queried to return a list of all users who have also checked-in within a given geographic distance from that user within a given time frame. (Current constraints are 3 hours, 10 blocks – though this is customizable by the user and/or external variables. See "dynamic proximity" below.) This list of currently "checked-in" users is then compared to this user's list of friends-of-friends. (The list of friends-of-friends is compiled from a database query that returns a list of all the people that a given user is friends with, combining it with a list of people that these users are friends with and then removing any duplicate results). If one or more friend-of-a-friend matches is found within the given proximity and time constraints (again: 3 hours, 10 blocks), then each of the users is sent a message that alerts the other to his or her presence. These messages read: "Dens is at Ace Bar. You know Dens through Alex. Reply to say hi." If the recipient of the user has a camera phone (as

specified in that user's preferences) the message the user receives will also contain a photo of the user with whom they are connected. The system will send a text message (with or without a photo) for each of the friend-of-a-friend matches generated based on that user's check-in. If three friend-of-a-friend matches are made, six messages are sent, etc. Each of these outgoing messages is formatted so that replies go back to the person to whom the user is being introduced in order to facilitate communication between the two users.

Once friend-of-friend messages are sent, the service will then compare the user's geographic location with the geographic location of all checked-in users, again compiling a list of all users who are within a set geographic proximity and timeframe, though this time comparing the user's profile with the profiles of other nearby users. If a match is made, regardless of if or how the users are connected (read: regardless whether they are friends, friends-of-friends or not connected at all), each user is sent a message alerting the user to his or her presence, calling attention to the profile element that they have in common. Profile comparison is currently based on a "crush list" – the ability for users to designate (from the web site) users that they find attractive. If a user "checks-in" and a person from his or her crush list is within the given geographic proximity and time frame, then the user who created the crush (read: created the link between the two users) is sent a message that reads "One of your crushes is within 10 blocks of you. We let them know where you are so be on the lookout!" while the person being "crushed upon" receives a message that reads, "Dens has a crush on you. Head over to Ace Bar (531 5th Street, btw Ave A & B) if you think he's cute." If the recipient has a camera phone (as specified in that user's preferences) the message the user receives will also contain a photo of the user that has a crush on them. If both users have a crush on each other and are found to be within the set geographic proximity and timeframe, each user is sent a message that says "Dens has a crush on Genja. Genja has a crush on Dens. You two should make out. Reply to introduce yourself." Each of these outgoing messages is formatted so that replies go back to the person to whom the user is being introduced.

b. **Listen:** The listen command allows a user to opt-in with their location without broadcasting their whereabouts to their friends (syntax: ".Luna Lounge). Though the user's geographic coordinates (based upon the venue at which they tell us they are currently located) will be added to the database, no messages to friends or friends-of-friends are sent alerting people to the user's location. However, if friend-of-friend matches are made (see above), the user who sent the "listen" command will receive an alert of nearby friends-of-friends, though the person they are connected to will not. The same goes for the "crush list" – if a user sends a listen command, they be alerted to the proximity / presence of their crush, though a message will not be sent to the user who is being "crushed upon".

c. **Shout:** The shout command allows users to either (a) broadcast a message to any user within a given geographical proximity and time frame or (b) broadcast a message to all of the users with whom they are friends. The difference is in the syntax of the message sent. If a user sends "Ace Bar!hello there" then the message "hello there" will be sent to any user within a set geographical proximity and time frame (proximity being determined by the latitude and longitude of the venue specified), regardless of how these users are connected via friends or friends-of-friends. If a user sends "!party at my apt tonight" (note: no venue is specified) then the message "party at my apt tonight" will be broadcast to all users with whom this user is friends with, though it will not be sent to this users' friends-of-friends. In the case of the former, all recipients of the message will see "Dens is @ Ace Bar and says:

'hello there ' @ 11:59pm". In the case of the latter, recipients will see "Dens says: 'party at my apartment tonight' (11:59pm)".

d. **Query:** Two search queries allow users to return information about the venues around them. If a user sends "Ace Bar?" the service will look up the information in the database and return the venue name, address, and cross street (e.g. "Ace Bar is at 531 5th Street (btw Ave A & B.). Reply w/ @venue name to check-in!" Users can also search for nearby "attributes" by sending the name of the venue they are near, as well as what they're looking for. For example, sending "Ace Bar?pool table" will return a list of all of the places closest to the geographic coordinates of Ace Bar that have either (a) "pool table" attribute in the database or (b) a user-submitted comment that contains the word "pool table". The message received by the user looks like "We found 8 matches for 'pool table' near Ace Bar: Black Star, Max Fish, Whiskey Ward." In both of these scenarios, the user's geographic coordinates are not made available and messages are not sent to friends of friends of friends.

4. **Social software web site.** Users register for the service and manage both their settings and relationship to other users via a website. This website shares similar core functionality with other social networking services such as Friendster, Orkut, Tribe, and LinkedIn. As is the case with the aforementioned sites, the web-site serves as a front-end for a database that defines the relationships that users have with one another. The core feature of this "relationship" database is that User A is friends with User B and User B is friends with User A, therefore users A and B are connected to one another. Through database queries, the data set described above can also determine which users are friends-of-friends (that being, User A is friends with User B who is friends with User C. Therefore, User A and User C are friends-of-friends). The website gives users the ability to invite new friends to join the site and approve and/or deny relationship connections. In comparing our site to other sites in the social networking genre, there are two key differentiators:

    a. **Manage Friends.** When a User A requests the friendship of User B, he or she will send a "friend request". The two users are not connected as "friends" until both have approved the request (and there are two entries in the database – one per user - one connecting User A to User B, the other connecting User B to User A). Since a key feature of the service is alerting your friends to your presence, there needs to be a way to further segment existing friends into (a) the people you want to broadcast your location to and (b) people you do not want to broadcast your location to. Our service allows users to "break" one end of the "User A connected to User B" connection without (a) deleting the connection and (b) allowing the other user to see that they have been "blocked". This is accomplished by adding another field to the database that allows a user to block one side of a user-to-user relationship without destroying or deleting the connection in the database. The end result is that when a user "checks in", they are broadcasting not to their raw friend list, but rather broadcasting their location to everyone except those individuals who they have chosen to block.

    b. **Crush list.** Users have the ability to browse through the "ambient profiles" (see below) of other users on the site – browsing through listings of friends and friends-of-friends. The "crush list" allows a user to select up to a certain number individuals (currently 5) that he/she finds to be attractive so that the user will be alerted to their crushes' proximity if they "check-in" to a location that is within a set geographic proximity of the user's. Users have the ability to add to or edit their crush list at any time. A separate database table is maintained to track who has a crush on whom, and to deliver this information in real time to users.

Key Differentiators:

1. **Opt-in location tracking.** Users need to first tell us where they are before we can tell them who and what is nearby. "Opt-in" solves some of the problems associated with "Big Brother"-esuqe, passive tracking / surveillance systems.

2. **Text message query language.** The service offers a wide variety of functionality to the user in a very simple user-interface (SMS/text messaging). This is accomplished by using our unique text message query language. For example, users can look up the address of a particular venue ("Luna Lounge?"), look for nearby pool tables ("Luna Lounge?pool table"), broadcast their location to their friends ("@Luna Lounge'"), listen for nearby events (".Luna Lounge"), broadcast messages to anyone within 10 blocks ("Luna Lounge!hello there") or broadcast messages to all of their friends ("!movies anyone?').

3. **Broadcast location to friends.** The service can broadcast a user's location to others who are within their social network. This feature is tied to one-degree friendships.

4. **Alerts based on relationship.** The service will alert you if friends-of-friends are within a given geographic proximity and time frame. This feature can be tied to relationships that are outside the user's one-degree social network.

5. **Alerts based on profile matching.** The service will alert you if your profile matches any users who are within a given geographic proximity and time frame. This feature is relationship-independent, meaning that a user can match with another user even if they are not connected within two degrees. (Note: Crush list is essentially profile matching).

6. **Location expires after a set period of time.** When users "check-in" we capture their location – and offer them relevant information about what is going on around them – for a period of three hours. After three hours the "check-in" expires. If the user "checks-in" at a different venues before the three hours have expired the newer check-in overwrites the earlier one.

7. **Dynamic proximity.** The applications defined here are set to trigger events when a user "checks in" within a set geographic proximity. By default, this geographic proximity is set to (0.80467km, or roughly 10 city blocks). Proximity can also be determined by factors such as weather conditions (temperature / precipitation) – allowing the service to dynamically adjust how far the user will travel given the current weather conditions and have that reflected in the types of messages the user receives.

8. **Manage friends.** The ability to filter your personal network based on different criteria / filters. For example, users whom you want to alert of your location (read: broadcast your location) vs. users you want to hide your location from. This system is described above in more detail.

9. **Ambient Profile.** User profiles not explicitly created, bur rather created over time as users contribute content to a geographically targeted community. Unlike other social networking services, profiles are constantly changing depending on how the user contributes to the system. Users' profiles dynamically update depending on where the user has most recently checked-in, what venues the user has comment on and which friends (and friends-of-friends) this user has recently been associated with.

10. **Location aggregation.** The service also functions as method to aggregate location data across multiple users. Described above are examples of matchmaking and data delivery applications that can be provided when the location of mobile users is known by a central server. By aggregating the location of users (across multiple mobile carriers), one provider can host multiple applications or provide location data for multiple location-based services for any number of third-parties.

Appendix A :: Recent Press (highlighting prior art status)