Newmont Mining Mobile Tracker

Alec Geatches, Alex Turner, Esteban Chavez, Cole Kembel

Colorado School of Mines Summer Field Session

June 19, 2012

I. Introduction

As one of the world's largest gold producers, Newmont mining has operations in many countries including Peru, Indonesia, and Ghana to name a few. With about 43,000 employees, Newmont is dedicated to ensuring the safety of its employees. Since Newmont has operations in potentially high-risk environments, knowing the location of working employees is imperative, especially when emergencies arise. The problem with current tracking solutions involves carrying an extra device and having to manually turn on the transmitter. Often times, employees may forget to turn on their device. Thus, Newmont proposed an idea to modify the current tracking process.

The client's proposal included using workers' phones to periodically update their current location to a database. The locations of all workers could then be viewed via a specified website. This would require making a mobile application for Android and iOS devices, and also a web interface with a map for viewing employee locations. The mobile application would provide functionality for automatically sending updates at a frequency ranging from every 15 minutes to every 60 minutes. The mobile application should also provide functionality for being able to manually send a location update at the user's request. The mobile application will not be released at either the Android or iOS app store, but instead be privately dispersed by Newmont to its employees. The final product will first be used for testing. Since most of Newmont's employees use Blackberry devices, the actual implementation of this tracking system would not be practical for all personnel at this time.

II. Requirements

Functional Requirements:

1. The mobile application must retrieve GPS coordinates of the user's current location and send these coordinates to the database if the user has Internet connectivity.
2. The mobile application should provide a manual update method that attempts to send a location update when selected.
3. The mobile application should provide an automatic update method that routinely sends a location update at time intervals set by the user. These time intervals range from every 15 minutes to every hour.
4. The mobile application should store updates locally when the user's phone does not have signal or access to the server. Cached updates are sent upon next transmission.
5. The mobile application must be able to run in the background of the device.
6. Users updates can be viewed at the website specified by the client.
7. The website should have a user authentication screen.
8. The website's home screen should show the latest update of every user on a world map.
9. The website should have a sidebar with a list of employees.
10. The website should provide filters to show which employees have updated recently and which employees have not updated recently.
11. Selecting an individual user from the world map should provide a link to a map with that user's past locations.

Non-functional Requirements:

1. Develop a native mobile application for both Android and iOS devices.
2. The Android application should be written in Java with the Eclipse IDE.
3. The iOS application should be written in Objective-C with the Xcode IDE.
4. The server used is an Amazon EC2 Ubuntu web server set up by the client.
5. The database used is a PostgreSQL spatial database set up by the client.
6. Google Maps, Open Layers, or ArcGIS should be used for map display.
7. The project should use a shared repository for version control of developed code.

III. System Architecture

Figure 1 shows a broad overview of the system architecture behind the Newmont Mobile Tracker.

Control flow from the mobile application perspective:

The employees will be interacting with the mobile application, where they can send regular or emergency location updates. For the sake of the diagram, there is no distinction between the two types of updates because control and data flow is the same for both.

On the mobile application interface (iOS/Android Interface), when an employee selects "send", a request is sent to the GPS Location Manager to get the devices current latitude and longitude.

From here there are two possibilities: the GPS can fail to receive coordinates, or the GPS can successfully gather a location. If the GPS fails, the user is alerted that the update failed via the status bar in the application interface. If the GPS succeeds, the user's location is passed off to the local cache.

The local cache is meant to store data on the device until a network connection is available at which time data can be dumped to the web server. If no connection is available, the data stays in the cache and the user is alerted that the update failed. If a connection is available, the data gets sent to the server and the user is notified that the update was successful. The data that was sent to the web server also gets removed from the cache at this point.

The web server mobile API accepts data, does some simple bookkeeping if necessary, and dumps the data to a PostgreSQL spatial database.

The PostgreSQL PostGIS Spatial database simply accepts device and location information and stores it in the appropriate tables. This is the end of control and data flow from the mobile application perspective.

Control flow from the website perspective:

The safety administrator for Newmont will be monitoring the website which will have the locations of all employees who are currently in the field.

The web interface interacts with the user and the server, retrieving data from the server through user interaction.

The server handles the request by executing a SQL query on the PostgreSQL database, which in return sends the results back to the server. The server then converts the data to JSON and sends that back to the web interface. The web interface parses this JSON and displays the information in the users browser.

IV. Technical Design

Figure 2 shows the PostgreSQL database design for our web services. The *device* and *update* entities represent mobile device state and location data. Each update has a location coordinate, device time, server time, an emergency flag and a unique ID column. Each device has a unique universal ID, an owner name, and a unique ID column. The *user* entity is separate from the *device* and *update* entities, and is used for web interface authorization. Each user has a username, password hash and a unique ID.

Figure 3 shows the database design used for the iOS mobile application cache. Each stored update contains information about the device state, location, and time of the failed update.

Figures 4 and 5 show the object hierarchies and design of the Android and iOS mobile applications.

Figure 6 is a graphical flowchart for the mobile application. Although the figure depicts an iPhone, the layout and control flow is consistent between the iOS and Android platforms. The red arrow represents first application launch. Each subsequent arrow represents a possible application state. The red dots indicate user interaction.

Figures 7, 8 and 9 represent a graphical flowchart for the web interface. They depict all possible user interactions, options, and state changes on the website.

V. Design & Implementation Decisions

Mobile:

Background tasking and automatic updates were implemented differently on Android and iOS. In iOS, when automatic updates are activated, the application spawns a process to handle all future automatic updates. This new process stays active and in order for it to remain "alive", the GPS is turned on for a brief moment to register the app as active and prevent the OS from stopping it. In the Android OS, the AlarmManager class was used to schedule a service for the feature and to repeat at the user's set interval. In this situation, the service starts then stays active for the duration of the update and is disposed of once updating is complete.

Updates were handled the same way in both operating systems. Both will wait for a location. Once the location is determined, the phone attempts to send the data to the server but should it fail (no Internet connection), the data is stored on the device and will be sent at the next scheduled update or when the user chooses to manually update. This eliminates the need to constantly check for an Internet connection, which would negatively impact the device's battery life.

Server:

To avoid some complexities with different device time zones we use both device time and a server time. On the "Employee Search" bar, there is a dropdown that will allow sorting by update times. Here we sort by server time due to the fact that an earlier device time does not mean that it updated more recently (e.g. Australia is 16 hours ahead of the time in Colorado, so unless the device in Australia stopped updating for at least 16 hours, the device in Colorado will never be the most recent if sorting by device time). However, all the time data presented on the map is in device time, including the individual view where points are connected based on their device time.

We also use geometric PostGIS extras in the PostgreSQL database to store location points from the devices. We use the "Point (double x, double y)" data structure provided which can be used to do searches based on location and distances. Although we don't make use of that functionality, it has been set up so that the web application can easily add those features.

VI. Results

Extensive testing was done for both the mobile and server side of the project. The mobile application was tested on a variety of devices (iPhones, iPod touches, and a few Android devices with different OS versions). Based on the results obtained, the application ran exceptionally well. An issue that should be addressed, however, is that in rare cases the mobile application can be very inaccurate and place the user several miles away from their true location. We believe this to be caused by the inaccuracy of the cell network. Unfortunately, this is a hardware and OS issue that we were unable to fix in software. Everything on the server side of the project works perfectly fine and there seems to be no issues at all. The website is fully functional and tested on Safari, Firefox, Chrome, and Internet Explorer. The site provides all the features requested by the client and more. It is also intuitive and easy to use.

Overall, this project was a success. We were able to meet all the requirements set by our client and we are very satisfied with the results. After seeing the potential of this application, our client plans to unveil this new tracking process as a viable future alternative to the current system Newmont has in place.

Appendix A - Mobile Installation Instructions

iOS First-Time Install Instructions:
If you're installing for the first time onto a device, follow these instructions:
http://vokalinteractive.com/2011/02/18/installing-an-ad-hoc-distribution-on-your-ios-device-ipa/

iOS Reinstall Instructions:
If you've already installed the application onto the device, the procedure is more involved.

1. Delete the application from the device.
    - http://blog.redshedtechnology.com/uninstall-an-ios-application-from-your-device-iphone-ipad-ipod/
2. Delete the application from iTunes
    - http://howto.cnet.com/8301-11310_39-57379574-285/how-to-delete-unwanted-ios-apps-from-itunes/
3. Sync iOS device with iTunes
4. Now follow the *First-Time Install* instructions above to install the application with the attached files, same as last time
    - When dragging the new application into iTunes, accept any prompts to overwrite files.
5. You'll need to sync your iOS device another time after the newest build is installed into iTunes

Android instructions:
1. Make sure that your device OS is 2.3 or above
2. Check to see if installation of unknown sources is allowed (settings > security > unknown sources)
3. Download the apk file onto your phone and on your phone navigate to the file and install.

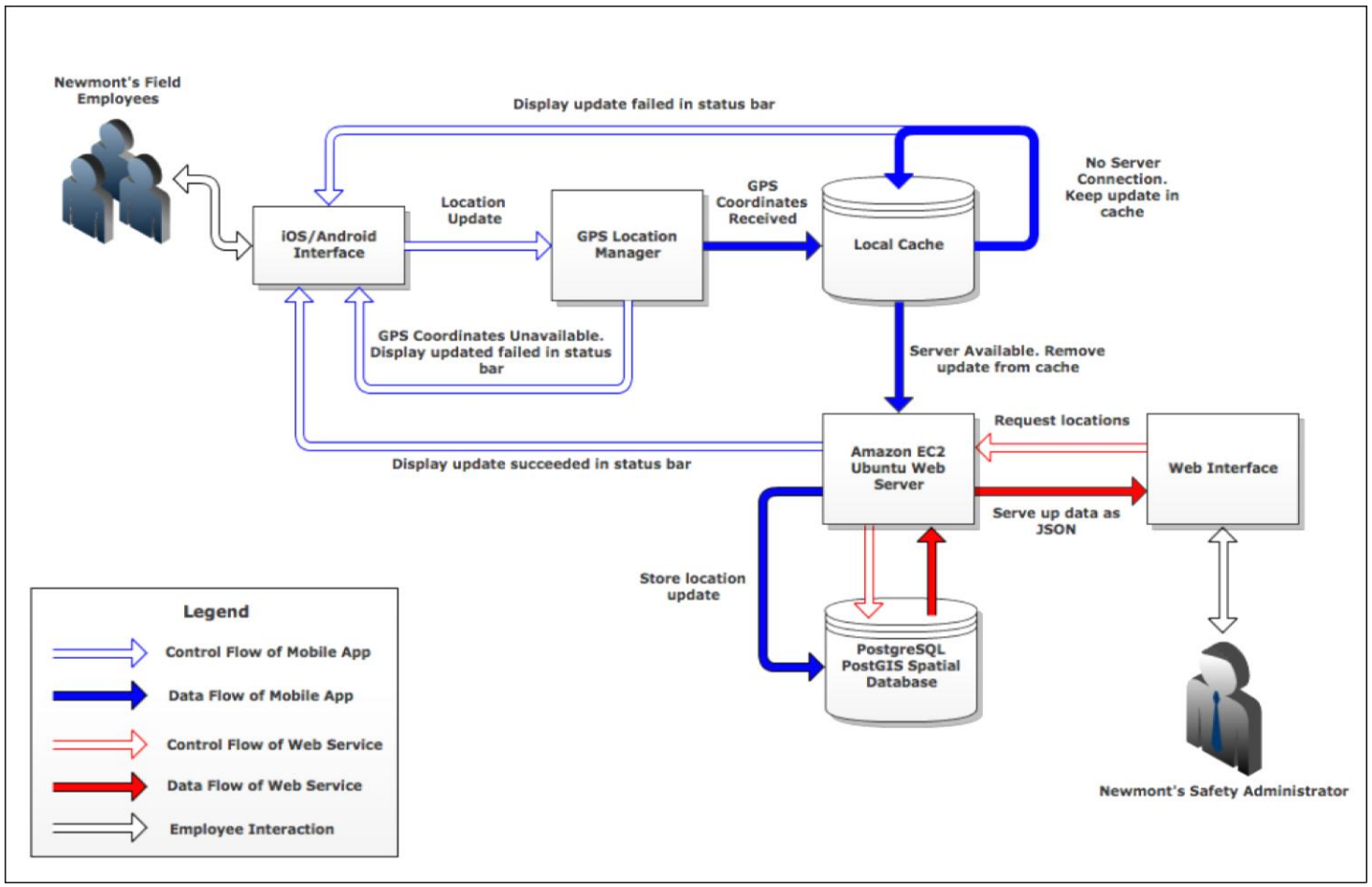
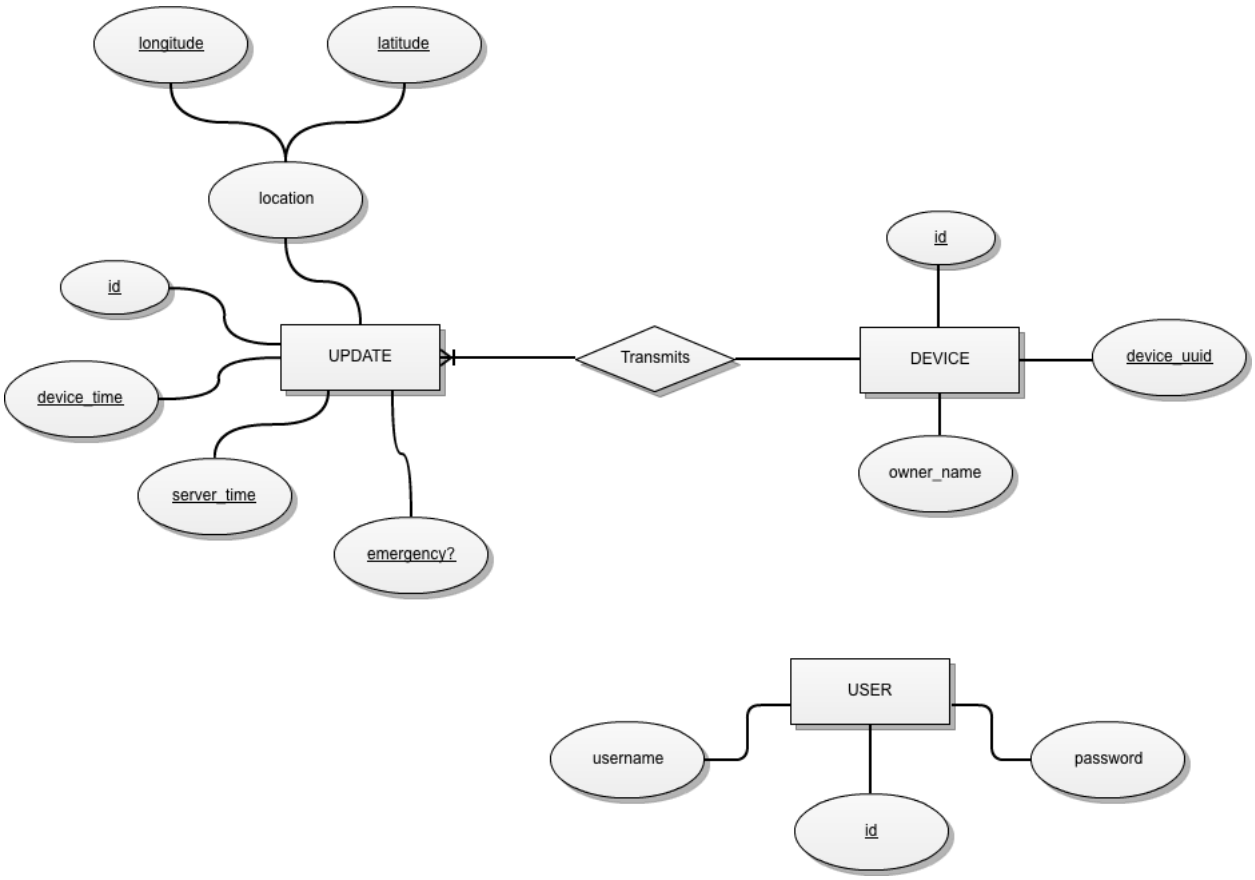
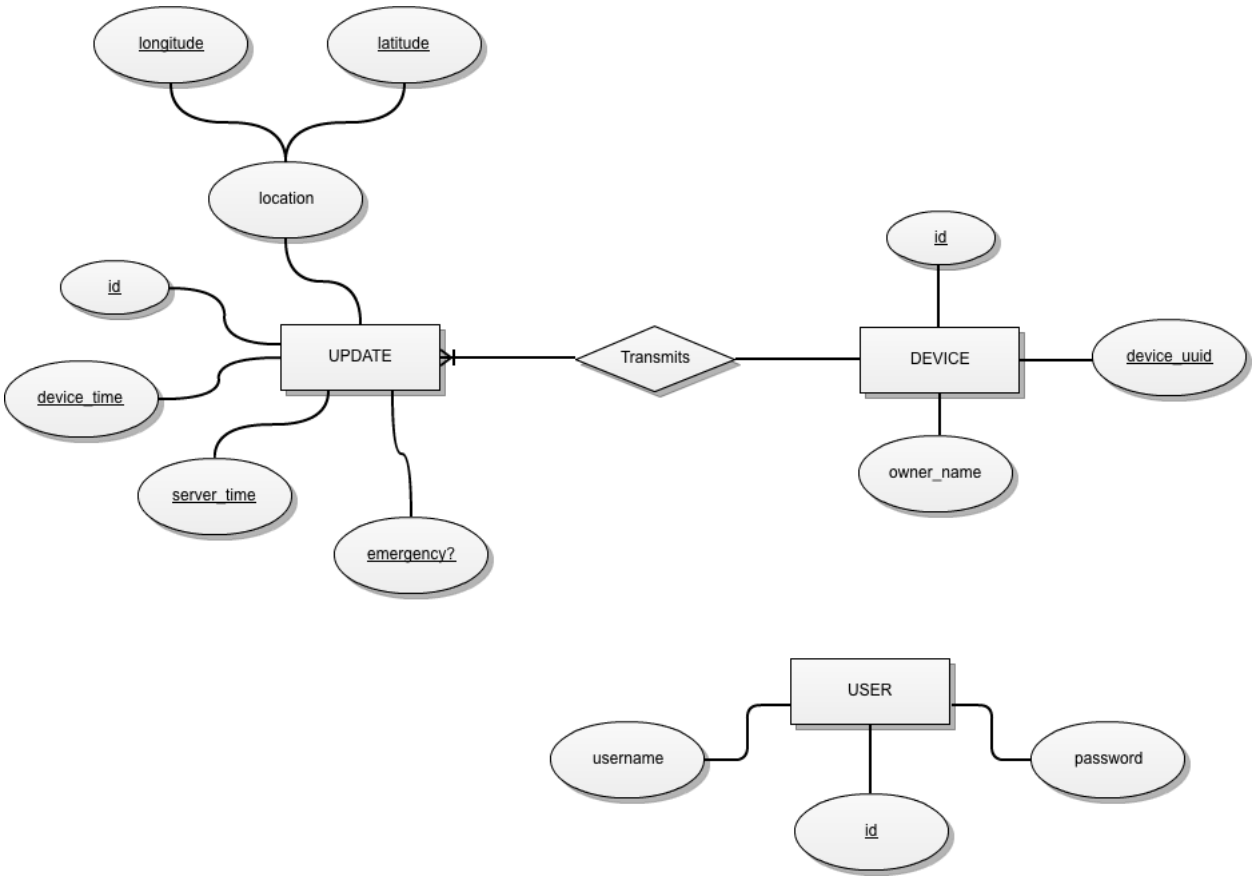


Figure 1: System Architecture


longitude
latitude
location
id
UPDATE
device_time
server_time
emergency?
Transmits
id
DEVICE
device_uuid
owner_name
USER
username
password
id


longitude
latitude
location
id
UPDATE
device_time
server_time
emergency?
Transmits
id
DEVICE
device_uuid
owner_name
USER
username
password
id