Google Glass HUD

Agustin Marquez

Alvaro Correa

Justin Callanan

Sandra Rubio

Abstract

This project is an end-to-end solution for collecting, storing, and displaying data from wireless sensor networks. Although the system can be adapted to handle a wide variety of sensor data, as a proof of concept, data was gathered from Xbee wireless mesh networks with attached sensors. This collection was performed using the Processing programming language on a base station computer. MongoDB, a document-oriented database system was used to store the data on a remote server. To provide a user interface for viewing the data within MongoDB, an app was developed for Google Glass which includes voice activation and a variety of output formats.

This system was designed to be adaptable, scalable, and extensible. Processing is capable of handling serial data from a wide variety of devices. Because MongoDB is document-oriented and thus only semi-structured, it can accept different kinds of data sent from Processing with little to no modification, and because it runs on a remote server, it can provide the user with data even when the base station is not active. Google Glass was chosen as a front end to demonstrate an unobtrusive and intuitive way to view the data, but hypothetically, the system could include applications for other mobile devices or a web interface.

Table of contents

Introduction

This paper covers the multitude of technical concerns and challenges in creating this system from scratch. Because of the collaborative nature of the project, this paper has been broken into sections, and in some cases, subsections, in which each group member has written on the specifics of their respective piece of the system.

There are two subsections to the introduction, the first of which discusses the process of finding a suitable way to approach each portion of the project, and the second of which covers the ways in which problems that arose were resolved. Following this, there are three sections, which cover Processing, MongoDB, and Google Glass, the three pieces of the system, which gather, store, and display data, respectively. Finally, there is a brief summary of the project, which includes some thoughts on the directions the project could take if it were expanded, and a page of technical references used in the project.

Introduction – First Approach

We started our project spending time in understanding our project and thinking about the optimal solution for it. This part includes a research about the Xbee. We conclude that with this radio module we could gather data from the sensors and work with it in the cloud. Once the information was there, we should prepare it as a .JSON file and import that file from Google Glass application.

In this first approach we propounded a program coded on Processing language and based on Xbee Libraries to get the data from the sensors, keep it on a .txt file using "saveString();" method and send it to the cloud through a HTTP POST connection. So the next step was figure out which services should we be using in the cloud. And our first thought was to use Amazon Web Services. We had worked with it some time ago so at this point, it was our best shot. It could be easy just gather the data, save it on a file and send in to the cloud from the Processing program, and then, once all the information was in the cloud, we could manage it to translate the .txt file to a .JSON file to sort the data properly.

But Amazon Web Services (S3 bucket) was not enough for our project. It had a wonderful functionality to translate one file to another, but we couldn't use it to send the data to the Google Glass because each file just had a few readings, and we needed real time communications. So we chose MongoDB, a cross-platform document-oriented database system. Here we can manage the JSON files and make them able for the Google Glass in an easier and faster way.

Google Glass proved to be quite a challenging platform to develop for. Because it is still in its early stages, the APIs and documentation available for it are changing very quickly, and a major update was rolled just as the project was getting started.

Introduction – Problems & Solutions

- There were some issues with gathering the data properly in Processing. The Xbee libraries did not work as intended.
  - This was solved by using alternative libraries.

- There were some problems with the HTTP POST libraries as well.
  - In spite of the fact that the S3 bucket and MongoDB were working perfectly, the HTTP POST issue was resolved by sending the data directly to MongoDB rather than using the S3 bucket.

- Google pushed the Android 4.4 update out to Google Glass shortly before work was started on the final piece of the project, which changed numerous APIs for Glass.
  - The app needed to be written using an interface framework provided in the online documentation, which was adapted for use in the project by Val Scarlata.
  - This was complicated further by incompatibilities between the two most widely used Android IDEs, Android Development Tools (ADT) and the newer Android Studio.

- There were some issues in the Glass interface caused by the sheer volume of data involved.
  - This was ultimately resolved by placing limits on the number of records the app pulls from MongoDB.

Processing

Once the measures have been taken by the sensors and sent with the Xbees with the help of an Arduino, we need to manipulate the data in order to present it in real-time in the google glass.

The first step is taking the data that really matters and discard unnecessary redundancy that is introduce by the Xbee. In this moment we will just have to store six bytes, enough to have the measure and the identification of each sensor. The Xbee is connected to the computer via USB, and the processing will receive all this data via Serial.

Now that we have the necessary data, we have to prepare it to be stored in the data base. We organize it as .json file in which we include the name of the sensor, the value measured and the time when the measurement was done.

Finally the data is prepared to be sent to the data base. We have used the mongoDB, we store all the data in a table called sensors. From there all the measures will be available for the mobile application.

MongoDB

From the beginning we identified that we will need a backend for our application in order to let our clients push data, and store it, and to have this information ready for the Google Glass users. The most important part of the backend will be the database, where we will store our data and it will be critical to us to keep this information safe and reliable.

Database – MongoDB:

The first decision was to choose between a relational database (SQL) or a non-relational database (NoSQL), as our data doesn't keep the same schema all the time, users can push to our database different kinds of information, we chose a NoSQL database and between all the possibilities we chose MongoDB because is one of the most mature systems and has a great community support so almost every trouble we will face someone would have faced it before and it will be documented how to solve it. Another advantage of MongoDB is that it provides libraries for almost every language so it is really easy to implement in every existing system so we will be able to implement it in our system easily but more important, our client will be able to push us data very easy.

At the beginning we are just using the MongoDB authentication, creating users in the database and specifying them as parameters when connecting to the database, this is a good option for fast deployment and bootstrapping our application. The MongoDB authentication is reliable and a good option for beginning but it doesn't scale as good as we would like to so once the business model has been proved and the application grows we will create another layer between the clients and the database with a node.js server that will validate the data sent from the user and prevent from corrupt data. MongoDB itself protects the database from malicious attacks and virus.

Right now we are hosting our database server with MongoLab, a solution provided by the same company that develops MongoDB that run their servers in Amazon web services, as our product grows we can resize our service here or migrate our system to our own instances.

The data pushed from the clients, or our own solutions, will be in JSON format, as we are using a NoSQL server we store documents no queries, the user can push its own data choosing the data they push but we ask for some required parameters as the time, in 'epoch' format, their internal id, user, or

device that is generating that data, for example ‘sensor id’ and the value of this reading, so we store individual readings or values. MongoDB also stores a unique id for each document inserted in the database.

As I said before MongoDB provides a library for almost each programming language for interacting with the database and this also includes retrieving data from the server so pull information from the database is as easy as push it.

Google Glass

Google Glass was chosen as a front end for the project, for its versatility and ease of use, but it introduced a number of difficulties in the development process. The hardware and software are both nascent technologies. Developer documentation is improving rapidly with time, but because of the major operating system update that was pushed to the device shortly before work on it began, the interface implementation needed to be rethought.

Valerie Scarlata, a former professor at IIT, who has a great deal of experience in developing for Google Glass, was assisting with the project. She was able to put together the structure for the interface by adapting a code example provided on the developer website. She also acted as a consultant during the rest of the development process, and provided invaluable insight into the intricacies of the device and its software.

The function of the app is essentially to connect to MongoDB to access the sensor data stored therein, store it locally on the device, organize it in some fashion, as specified by the user, and display it to the user in a human-readable format. While conceptually this was fairly simple, there were several technical hurdles in accomplishing this.

First, connecting to MongoDB requires a database driver, which is written for Java, and is thus compatible with Android. This documentation for this driver was outdated, leading to the code that used it being rewritten several times. Also, because Android does not allow network traffic on the main thread in applications, the calls to the database needed to be written entirely asynchronously, and any interface manipulation needed to be performed in the callback functions. Because of the large amount of data being pulled down, however, the app needed to perform the interface updates while the data was still being downloaded, to avoid a long delay before it was displayed to the user. After these problems were resolved, the app functions entirely as intended.

Summary

In this project we have achieved the goal of generating real-time data, sent it from a head office, store it in a DB and present it in a mobile application. In this project we only had one central office, but this is not a must. The data can be other kind of data, and the database can store data from different parts. Also you can set a different refresh time of this data. Finally the information can be presented in many ways, not only in a mobile application.

This project has been designed to be scalable and it is open to be modified. As you can see this project is a set of modules. All of them can be changed to cover other needs.

A possible continuation of the project is a bilateral communication. This way you could control with a similar infrastructure options of regulation via mobile. A good example of this would be the control of a thermostat, changing the temperature of the room if you detect an anomalous measurement.

References

Xbee and Processing:

- https://github.com/runemadsen/HTTP-Requests-for-Processing
- http://www.faludi.com/examples/xbee-api-library-for-processing/
- https://code.google.com/p/xbee-api/wiki/Processing
- http://wiki.processing.org/w/Saving_data_to_text_files
- http://www.processing.org/reference/saveStrings_.html
- https://code.google.com/p/xbee-api/wiki/Processing
- https://stackoverflow.com/questions/2288876/how-to-configure-log4j-with-a-properties-file

MongoDB:

- https://www.mongodb.org/
- http://docs.mongodb.org/ecosystem/drivers/java/
- http://docs.mongodb.org/ecosystem/tutorial/getting-started-with-java-driver/

Google Glass:

- http://sendgrid.com/blog/google-glass-tutorial-sends-email-visually/
- https://github.com/jaredsburrows/OpenQuartz
- https://github.com/harrywye/gdkdemo
- http://glasssim.com/
- https://developers.google.com/glass/samples/gdk
- https://developers.google.com/glass/develop/patterns/immersion
- https://developers.google.com/glass/develop/gdk/ui-widgets
- https://developers.google.com/glass/develop/gdk/starting-glassware
- https://code.google.com/p/android-screen-monitor/