Final Project Report

By
James Kuczynski James_Kuczynski@student.uml.edu
Neel Tripathi Neel_Tripathi@student.uml.edu
Michael Forsyth Michoal_Forsyth@student.uml.edu
Nicholas Forsyth Nicholas_Forsyth@student.uml.edu
Robert Vecchione Robert_Vecchione@student.uml.edu

Adapted from IEEE documentation standards (particularly IEEE-1016 and IEEE-830)

Table of Contents

1. Introduction

This document will cover the description of the Software Engineering one group two data aggregator project. The sections of this document will detail the design and application of the project.

1.1. Document Description

The following descriptions outline a basic introduction and overview to the project.

1.1.1. Introduction

This document is intended for computer scientists who will use the system. Inside the document is the design considerations, architectural strategies, system architecture, policies and tactics, and detailed system design of the project.

The project is to design and develop a portable, cross-platform application to manage the administration of data received from various hardware devices. Lightweight messages will be processed immediately, while larger data will be stored in a database for future analysis.

There are a limited number of programs designed to aggregate data from hardware devices, and to the best of our knowledge no currently commercial models utilize ROS.

1) The proposed application will provide a user-friendly GUI which will allow the user a comprehensive method of managing designated hardware devices.
2) The proposed application will manage data transfer between sensor devices and a cloud, and will present analysis results to the user in real time.
3) Information will be displayed using graphics that provide a transparent visualization of user data.

1.1.2. System Overview

The system will allow users to log into the system, add devices, and add users through a GUI window. The system was designed with the collaboration of sensor device input and cloud database storing in mind. The approach to the design was to decide the crucial components to the independent system and complete those first. The collaborative components will be created in future meetings with other groups.

2. Design Considerations

This section describes many of the issues which need to be addressed or resolved before attempting to devise a complete design solution.

2.1. Assumptions and Dependencies

These are the assumptions and dependencies regarding the software and its use for this project. Related software and hardware that we used is the Kinect 360 Microsoft hardware and compatible software packages. The operating systems involved in this system are Linux, Ubuntu, and Android. The end-user should be able to interact with GUI interface that will store their information and the devices they have registered. Possible changes in functionality are adding to the existing system that is operational on Ubuntu with an Android application in the future.

1) *Hardware Architecture* -- This application will run on any platform running Ubuntu 12.04-15.04. However, performance may be impacted depending on the hardware of a given device. However, it was developed and tested on the most recent Long Term Support version of Ubuntu, which is 14.04. Therefore, we strongly suggest using this version.
2) *Operating System* -- This application will support Ubuntu 12.04-15.04. Other UNIX/Linux-based systems may receive limited support. However, we place NO warranties or endorsement, express or implied, on the aforementioned statement, and will not assume liability for any direct, indirect, special or consequential damages arising from such use.
3) *Internet* -- The system will need access to wireless Internet in order to transmit messages, either with TCP/IP or Bluetooth.
4) *Sensor Device* -- The Kinect v1, as described on Microsoft's website, is a "physical device with depth sensing technology, a built-in color camera, an infrared (IR) emitter, and a microphone array, enabling it to sense the location and movements of people as well as their voices."

2.2. General Constraints

Constraints on this project are listed below:

- For this project the hardware or software environment are free software and the provided hardware sensors (Kinect).
- The user must be able to read English and understand how to interact with a GUI.
- The developers were limited to the resources provided by the University and accessible for free online.
- All of the code was ensured to work before being implemented in the final product and merged and complies to a general practice rubric.
- The sensor must be able to connect to the system through internet, Bluetooth, or USB. The program must be able to display to the screen the information in the form of a GUI.

- The user interface must receive user input from the keyboard and interface with a database to store it
- Data from the user is stored in a Database to reference usernames, passwords, and registered devices
- Each username and account requires a password to help provide some security and privacy to the user’s information
- The memory used by the project must be less than the maximum of the allocated database
- The interface must perform in real-time and allow the user to see the changes as they are happening (loading or refreshing)
- The project connects to the Internet and Bluetooth/USB for the sensors
- When the program is run it must fulfill all of the previously stated requirements and accomplish the objective of this project. We will test this to ensure that the end product validates its original purpose.
- Code quality control is ensured by having members peer review each others code and suggest corrections or optimizations.
- This application will only be compatible with Linux and must be run using an IDE. NetBeans was used to develop and run the application, and as such, is recommended as the IDE of choice.

2.3. Goals and Guidelines

In this section the goals, guidelines, principles, or priorities which dominate or embody the design of the system's software are detailed.

- The design emphasizes storing into memory the username and associated devices.
- The entire project must be stored and accessible from one location (like GitHub). This allows multiple users to interact and make changes remotely as well as making it much easier to compile, run, and test the project as a whole.
- The goal of the project is to provide a way for the user to store the data from a sensor so that they can look back and compile it.
- The priority of the project is to make a deliverable working code by the end of the project. The reason is that the project hinges on the fact that the user will be able to interact and utilize it.
- The ease-of-use for the end user is the main priority for this iteration of the application.
- The process of interacting with a GUI to log in and register devices must be simple and intuitive.
- The GUI must respond to user events with clear and descriptive updates.
- The further development of this application was a priority for this iteration of the application. The application will have additional system interactions added to it.
- The system should have a flexible framework and allow integration with new systems.

2.4. Development Methods

Agile methodology was used as the main approach to development of this application. Using the agile method, the group organized scrum meetings for the project to work on the backlog. The software was developed in increments and was subject to changing requirements specified by the customer, in this case, the professor. Agile fit the best because we had limited time to work on the project and the project was subject to change as the semester went on.

3. Architectural Strategies

These are some of the design strategies that we have used for this project:

- **Programming Languages and Libraries**
  - C++ programming language
  - CMake -- version of make
  - XML -- for CMake package.xml and .LAUNCH configuration files
  - JSON -- for storing user data
  - g++ -- the GNU C++ compiler
  - SQLite -- a common light-weight SQL-stype database
  - QT -- A cross-platform graphics API
  - QJSON -- a JSON serialization library
  - ROS -- a meta-operating system which can be used to manage sensor devices
  - OpenNI -- an open source driver package for the Kinect v1
- **Future plans for extending or enhancing the software**
  - Since this system is a generic application which aggregates data between a sensor and cloud, it may be of use for future projects.
- **Models and diagrams**
  - Please see the section "Detailed System Design" to view the models and diagrams.
- **Error detection and recovery**
  - The back-end checks user input for validity. For example, the system will check already registered devices, to prevent the user from accidentally registering the same device twice. Also, the back-end will make sure no mandatory fields are left blank during the registration process.
- **Distributed data or control over a network**
  - The data is distributed over the sensor, data aggregation, and cloud subsystems. Theoretically these subsystems should work together.

4. System Architecture

The data aggregator system should function with input from a system of sensors and output to a cloud database. The system integration of the application is in the following context model, where the application takes part in the data aggregator system. The input from the sensor system is collected and the data is displayed or stored it in the cloud database system.

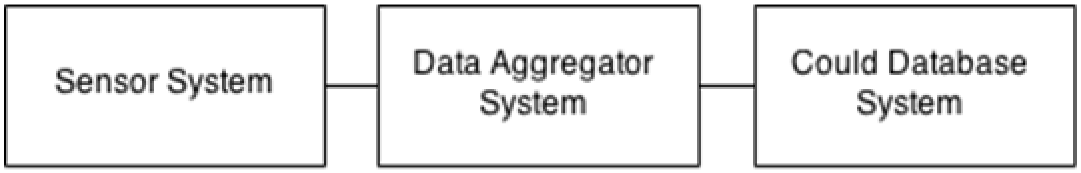

Figure: Context Model of the System.

The end function of the system's components is described simply by the process model in section 6. The following are features of the program: the user can log on, add devices and users, access either the input from the sensors or stored data, display the collected data in different forms, and log off. This decomposition was chosen as the function of the system flows through the processes from input to output.

5. Policies and Tactics

Below are the design policies and tactics that affect the details of the interface and implementation of the various aspects of the system.

- For this project, we chose to use the GNU g++ C++ compiler. The libraries utilized were QT, a graphical library, QJON, which is a JSON library specifically designed for easy integration with QT, and SQLite, a light-weight database.
- Some engineering trade-offs that the group made are: We chose to use the Kinect instead of the Pebble watch for several reasons. First, none of the members of our subgroup have, or even have access to, a Pebble watch while we do have access to a Kinect. We chose the Linux operating system over Android because not only do none of us have experience with Android development, but we also have only limited experience with Java, which would have made it almost impossible.
- We attempted to use fairly strict programing guidelines, in order to make our project as a whole easy to modify in the future. Instance variables began with a descriptive name, followed by an abbreviation of the data type. We documented code using JavaDoc and Doxygen as our documentation standard. However, since we used an Agile process, the documentation is sparse in areas.
- We used elements of the layered design pattern. The lowest classes provide base functionality to the classes above them. This tiered system goes up until it reaches the GUI.

- This software was tested by a collection of volunteers taken from the software engineering community (i.e. some of the people who work at Professor Holly's Robotics Lab).
- The source code, documentation, and detailed installation instructions, are all listed on out GitHub page. As this project involves the aggregation of data between sensor and cloud, this software could be reused as part of a robotics project. As one of our team members works in a research lab, parts of this code may be used in the future.
- The project directories are kept in the following way:
    - **build** *(directory)* -- contains build configuration files for the NetBeans IDE.
    - **dist** *(directory)* -- contains an executable
    - **install** *(directory)* -- contains a shell script for cloning the repository and installing the dependencies on the users machine. This directory also contains a ".desktop" file, which will enable the user to add a icon linked to the executable to the desktop.
    - **moc** *(directory)* -- auto-generated QT code.
    - **qss** *(directory)* -- contains cascading style sheets to change the GUI design.
    - **src** *(directory)* -- contains the header and source files.
    - **.dep.inc** *(file)* -- this file is a NetBeans configuration file
    - **Makefile** *(file)* -- this file is a makefile generated by NetBeans
    - **Se_logo.png** *(file)* -- icon files used by the program. These only serve aesthetic purposes.
- For instructions on building and running the project, read section 7, which contains these. Also, we recommend reading the "README.md" at https://github.com/DeepBlue14/Software_Engineer_91.411_2/tree/master/2_DataAggregator , which contain more detail.

6. Detailed System Design

Each subsection of this section will refer to and contain a detailed description of the subsystem software. The discussion covers the following subsystem software attributes:

- **Classification**
  The subsystem is the data aggregator. This subsystem collects the data from the user directly through a GUI and interacts with the database to store the information.

- **Definition**
  The data aggregator provides the users with the ability to register users and devices, start/stop data analysis, and view results.

- **Responsibilities**
  The primary purpose of this component is to provide a user interface which the user can use to receive data from devices, process the data, display results, and

send the results to a database for long-term storage. The user can also register multiple devices, and register other users if he has an administrator account.

- **Constraints**
  This application will run in real time, so the device the user chooses must posses sufficient internet speed. Also, the system which the Kinect is connected to must have sufficient resources and processing capabilities to publish data over a TCP/IP or Bluetooth socket in real time. In its current phase, the system requires a Linux environment. However, since the program dependencies (QT, QJSON, SQLite) are cross-platform, future versions could be cross-platform, for any Windows/Linux/Android, etc.

- **Composition**
  This subsystem is composed of a program that interacts with the Kinect to run a GUI and the input boxes/database that receives the data.

- **Uses/Interactions**
  The relationships between the different classes are shown below.

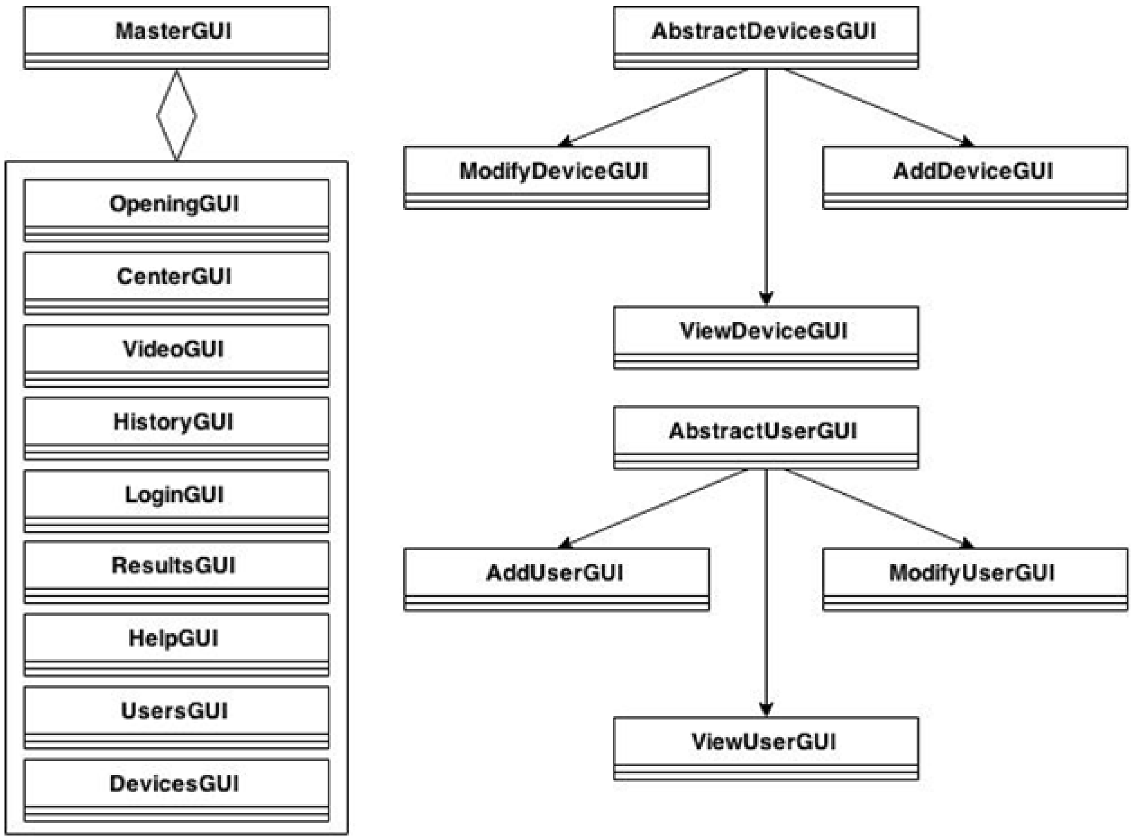

Figure: System Class Layout

- **Resources**

The resources that are used are the software libraries listed in section 8, a database to store the information from the user, and a library to interact with the Kinect.

- **Processing**
  The components will be pre-initialized and receive user input to function. Once user input is received, the state of the GUI will be appropriately updated. The user will determine which part of the application is functioning as well as when such functions end.

- **Interface/Exports**
  Computer GUI that interacts with the user receives strings as input for username and password. Once that input has been verified the system will be able to receive input strings for additional devices to be added to the user's account/database.

The next two models will show the functional processes of the application.

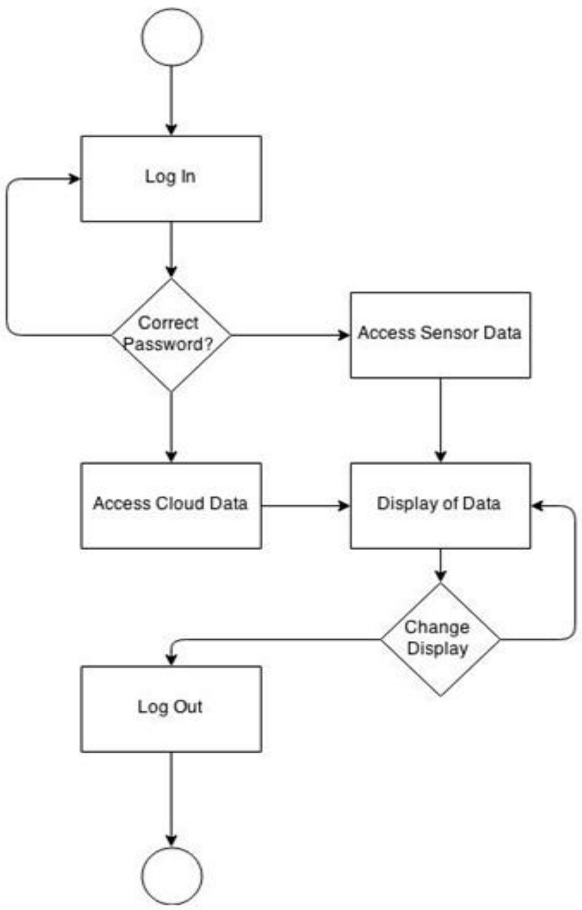

Figure: Process Model of the System

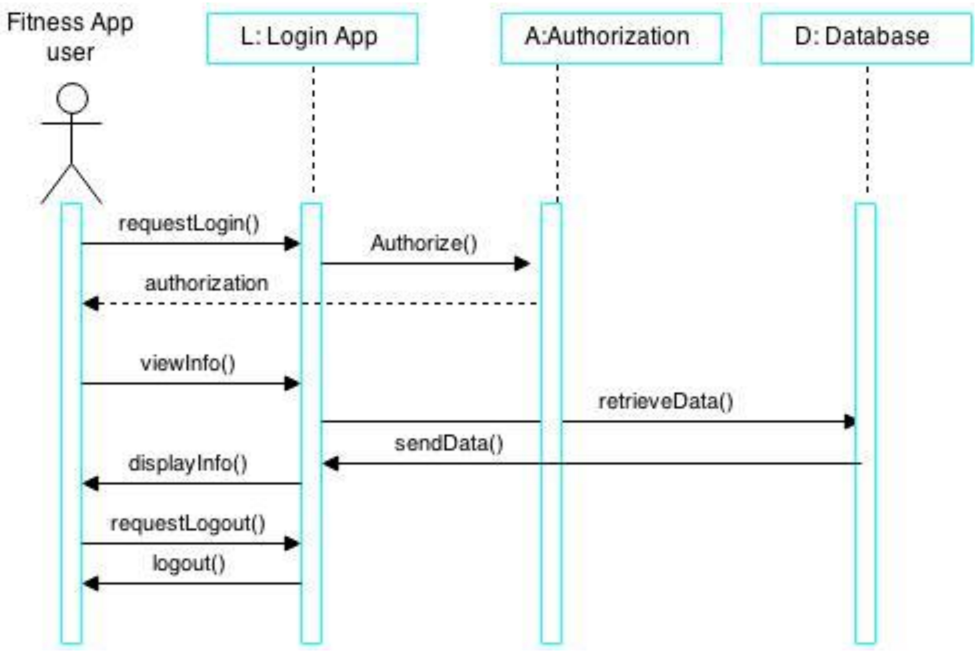

Figure: Interaction Model of the System

These models display the functional process of the application. The application starts with the log in of the user. The log in reads in a user's name and password. After logging in the user will be able to select whether to access data from the sensors or data from the database. The data, once collected, will be displayed on screen. The user will be able to go back to the selection and change the data set. The user may also add other users and sensory devices to the application. After the user is done, the user will log out of the application.

7. System Installation and Execution

This section describes how to correctly install and execute the necessary code for this project to work.

First, make sure that you have the correct operating system, and the GNU g++ compiler installed.

We strongly suggest that you read the README file on GitHub, it includes the instructions to install, build, and run the project in greater detail.

There are two possible ways to install the code:

Method 1:

1. Clone the repository to the desktop.
2. Install dependencies:

    a. sudo apt-get install libqt4-*
    b. sudo apt-get install libqjson0 libqjson-*
    c. sudo apt-get install sqlite3 libsqlite3-dev
3. Run NetBeans IDE, and open the "Code" directory
4. Make sure that you are linking against QT, QJSON, and SQLite libraries. *(The NetBeans configuration included will probably do this for you. If it does not, or if you are running it on a different IDE, reference the README file at https://github.com/UMLproject/Software_Engineer_91.411_2/tree/master/2_DataAggregator).* Required QT components are QtCore, QtGui, QtWidgets, QtSql, and QtXml.
5. For the "MOC Directory" option, choose "moc".
6. Press the "Clean and Build" button.
7. Press the "Run" button.

Method 2:

1. Copy and paste the content of /.../Code/install/install.bash into a *.bash file at the location you wish to install the project.
2. Run "chmod + x install.bash".
3. Run "./install.bash"
4. The script will ask you several questions (would you like to install the project, would you like to install dependencies, etc). Choose the appropriate responses.
5. Run NetBeans IDE, and open the "Code" directory
6. Make sure that you are linking against QT, QJSON, and SQLite libraries. *(The NetBeans configuration included will probably do this for you. If it does not, or if you are running it on a different IDE, reference the README file at https://github.com/UMLproject/Software_Engineer_91.411_2/tree/master/2_DataAggregator).* Required QT components are QtCore, QtGui, QtWidgets, QtSql, and QtXml.
7. For the "MOC Directory" option, choose "moc".
8. Press the "Clean and Build" button.
9. Press the "Run" button.

Optional *(setting up a desktop shortcut)*:

1. Move the file "SoftEng.desktop" (found in the "install" directory) to your desktop.
2. Open the file, and change the paths to match those on your personal system.
3. In a terminal, cd to Desktop and run "chmod +x SoftEng.desktop".

If you do this correctly, the file icon should be replaced by an icon, and when you double click it, it will launch the executable.

For more details for installation, go to the GitHub page, where a very detailed README file resides.

8. Glossary

- **QT** - A graphics library.
- **QJSON** - A JSON serialization library.
- **Ubuntu** - One of the most popular and versatile Linux distributions. Supports phone, tablet, and desktop.
- **SQLite** - A light-weight database framework
- **TCP/IP** - Transmission Control Protocol/Internet Protocol is a suite of communication protocols that is commonly used to connect hosts on the Internet.
- **Bluetooth** -- a wireless standard for transporting data using short-wavelength UHF waves. It works over short distance, and lacks the level of security that TCP/IP achieves.
- **Boost** - a C++ library.
- **ROS** -- a meta-operating system commonly used to manage sensor devices.
- **Kinect** -- sensor device capable of transmitting rgb and depth data.
- **GNU g++** -- a version of the C++ compiler.
- **OS** -- Operating system.
- **GitHub** -- A version control system, allowing members of the team to update and keep track of project updates.

9. Bibliography

- QT documentation - http://www.qt.io/
- SQLite documentation - https://www.sqlite.org/
- ROS documentation - http://www.ros.org/
- Ubuntu documentation - http://www.ubuntu.com/
- Linux documentation - https://www.linux.com/
- Boost documentation - http://www.boost.org/
- TCP/IP documentation - http://ftp1.digi.com/support/documentation/0190074_j.pdf
- OpenNI documentation - http://structure.io/openni
- OpenKinect documentation - https://github.com/OpenKinect/libfreenect