CIS 422 Project 2: *Freedge Tracker*
Software Design Specification

Ginni Gallagher (gmg), Ellie Kobak (erk), Kalyn Koyanagi (kek),
Liza Richards (ljr), Madison Werries (mgw)
03-06-2022 - v2.01

Table of Contents

1. SDS Revision History

Date	Author	Description
4-30-2019	ajh	Created the initial document template.
2-8-2022	ljr	Made initial document.
2-13-2022	ljr	Wrote first draft of document
2-16-2022	kek	Updated sections 4 and 5
2-16-2022	ljr	Updated section 3 and 5.3
2-17-2022	gmg	Updated section 4 and 6
2-17-2022	mgw	Added terminology section; updated section 4
2-17-2022	kek	Updated section 5
2-17-2022	ljr	Updated section 6 and 7
2-24-2022	erk	Created version 2.01
3-01-2022	gmg	Updated sections to reflect current design
3-04-2022	ljr	Updated sections 4, 5, 6, and 7
3-05-2022	gmg	Reviewed and proof-read
3-06-2022	mgw	Updated static and dynamic diagrams from original version

2. Terminology and Definitions

Throughout the documentation, distinct terms are used to define the types of users who will be utilizing the system and denote the objects and components involved in the system. For clarity and ease of use, the definitions of these terms have been included here.

Administrator
This term is used to indicate an individual who oversees the operations of the Freedge Organization itself, such as the Freedge cofounder, Ernst Oehninger.

Caretaker
A “caretaker” is an individual who manages a public freedge in their community. They are the primary contact responsible for the construction, maintenance, and ownership of a particular freedge.

Freedge - The Organization
When the term “Freedge” is capitalized, it is meant to refer to the nonprofit organization itself rather than one or more freedges.

Freedge - A Physical Object
When the term “freedge” is *not* capitalized, it refers to a community fridge that may be owned and operated by one or more caretakers. The freedge may or may not be registered with Freedge.

Community Fridge
This is a term interchangeable with “freedge.”

Fridge
The word “fridge” is used to describe the physical refrigerator component of a freedge. This distinction is important when discussing things like sensor data from the fridge.

3. System Overview

This software provides Freedge administrators with an automated way to create and maintain an organized database of information about the community fridges which have registered with the organization. The system allows administrators to more easily determine if community fridges are still active or not through the use of a notification system.

The system is organized into five different interfaces responsible for interacting with each other to interpret the usage and communication of data for a community fridge. Each component handles one of the following:

1. Creating and maintaining a database containing information about the freedges.
2. Creating objects to store and access freedge data for easy use by other modules.
3. Reading and parsing freedge data from a specified file provided by the user.
4. Sending notifications to freedge caretakers to check for activity status.
5. Providing an interface to allow administrators to navigate the system.

The components that read and sort freedge information work closely with those that load freedge data into a database. The information is imported as a CSV file and is then parsed to be loaded into the database. The components that are tasked with reading, loading, and storing freedge data and sending notifications to freedge caretakers work closely, as the notifications will be sent to caretakers using freedge and caretaker information provided by the files imported into the freedge database. The visual interface module links all the components of the system together to work as a complete system.

4. Software Architecture

4.1. Components

1. **Freedge Database** - Contains up-to-date information on individual freedge that has been registered with the Freedge Tracker system, including the caretaker's information, the address of the freedge, and the status of the freedge.

2. **Freedge Class -** Holds all of the information about a single freedge.

3. **Caretaker Notification Interface** - The Caretaker Notification Interface sends notifications to one or more freedge caretakers via a pop-up visual window, prompting the caretaker for a response to be processed by the system.[1]

4. **Caretaker Contact Information Parser** - Processes a file imported by the user which contains information about the freedges and their caretakers who have registered with the Freedge organization. The parser prepares the information to be added and organized in the database.

5. **Administrator Interface -** Manages and updates the Freedge Tracker system's visual display in response to user input via interactions such as clicking on display buttons and clicking on rows in the database.

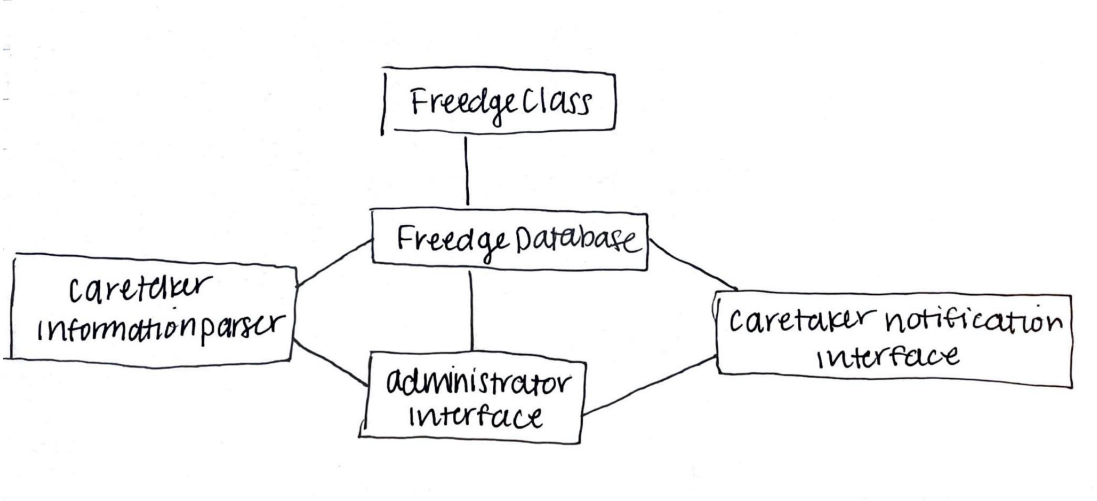

Figure A: A basic overview of the Software Architecture of the System

The boxes show modules, and the lines show which modules communicate with one another

[1] The version of the Caretaker Notification System implemented in the Freedge Tracker prototype does not send notifications to other devices. The prototype mimics the goal for future implementations, which would provide the functionality to send notifications and receive responses via SMS texting or email to freedge caretaker devices worldwide.

4.2. Component Interactions

The Administrator Interface is responsible for all the functionality of the system and acts as a driver for the running program. The Administrator Interface presents the user with several options, all displayed in a menu. These options correspond with the notification and database features of the Freedge Tracker.

The CSV file input to the system is first parsed by the Caretaker Contact Information Parser and then stored in the system's database. The Caretaker Notification Interface receives contact information data for each freedge and its caretaker from the database. In a fully implemented version of this system, the Caretaker Notification Interface will use the contact information data to send notifications to freedge caretakers. If no response to the notification is received by a particular freedge caretaker, the system will update that freedge location to an "inactive" status in the Freedge Database. The update will also be made on the Administrator Interface, so users can see the activity status of that particular freedge has changed.

4.3. Design Rationale

The rationale behind the choice of these components is to maintain modularity to allow for easier maintenance while eliminating potentially redundant code. The compartmentalized files also allow for easy testing and debugging, as every component can be tested individually. The current software architecture minimizes the need for transferring Freedge data between components, which ensures user data security and efficiency. Maintaining efficiency is a major element of the architecture, as speed will often be a top priority for a system that is being constantly updated every time a notification is sent out and a response is received.

5. Software Modules

5.1 Freedge Database

The module's role and primary function

The Freedge Database contains up-to-date information on each individual freedge that has been registered and uploaded with the Freedge Tracker system. The database includes the following information for each freedge:

a. The freedge ID, project name, and network name.
b. The freedge caretaker's name, contact information, and preferred method of contact.
c. The location of the freedge itself, and when it was registered with the organization.
d. The status of the freedge (active, temporarily inactive, or permanently inactive).
e. Whether or not a caretaker has agreed to be contacted with regards to the status of their freedge.
f. The last time the status of the freedge was retrieved and updated.

The initial database is created by uploading a csv file containing information about each of the freedges which have been registered with the organization.

Interface Specification

The Freedge Database's main purpose is to keep a centralized and up-to-date database of all of the information collected by Freedge for all registered freedges. This is accomplished through the following steps:

1. Obtain database information from the caretaker contact information parser.
2. Create a new database, or update a previously created database, containing the most up-to-date information.
3. Update the database when the caretaker notification interface alerts of changes to any freedge's activity status.

Models

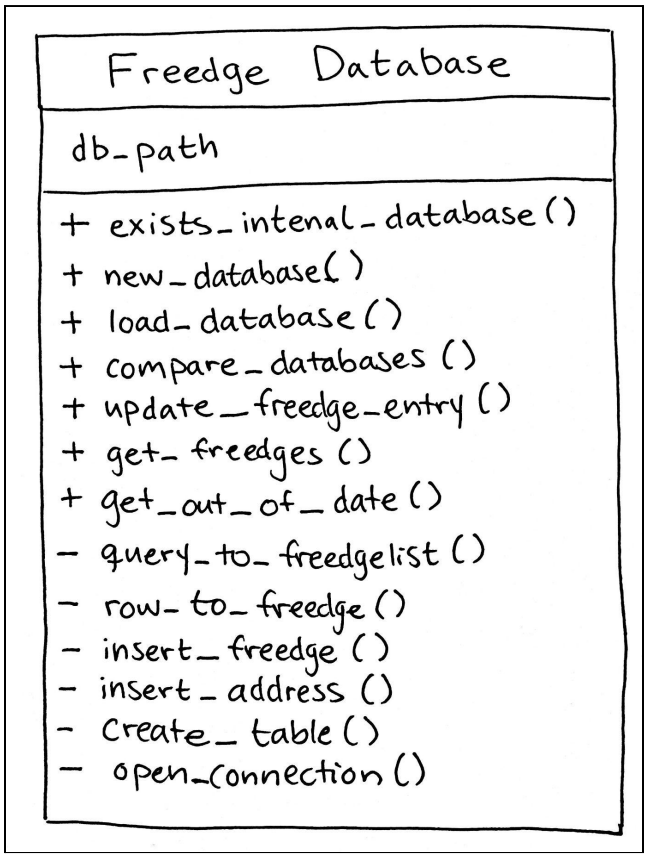

Figure 1: A static UML class diagram of the Community Fridge Database

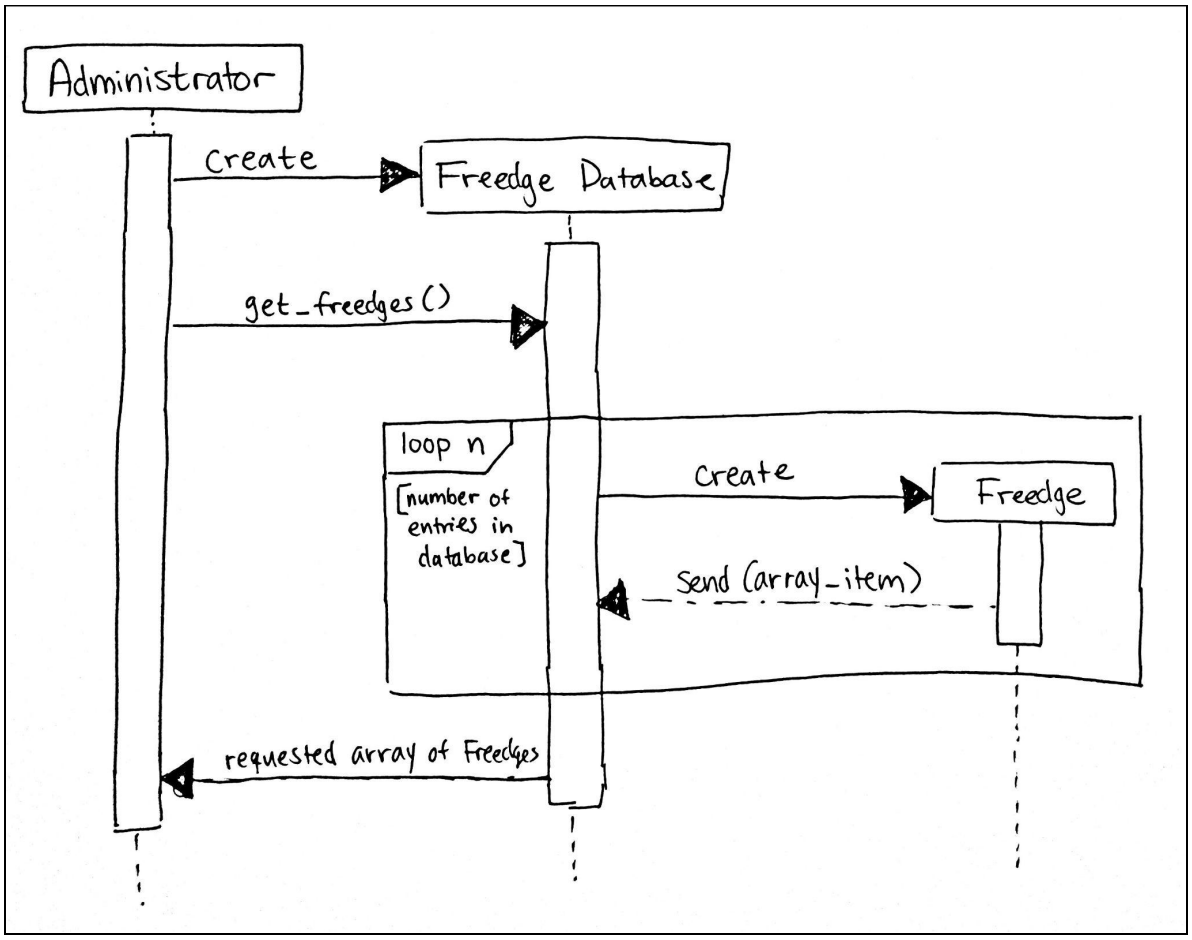

Figure 2: A dynamic UML Sequence diagram showing the creation of a Community Fridge Database

Design rationale

The Freedge Database is the core of the Freedge Tracker system, allowing for the creation, modification, and maintenance of a centralized system to store all the data relevant to freedges and their caretakers who have registered with the Freedge Organization. This database is important because it is what all other modules rely on in order to function properly.

5.2 Freedge Class

The module's role and primary function

The Freedge class maintains all of the information about a single freedge, including information about the caretaker of the freedge as well as its current status. Its primary function is to provide an easy way for the other modules to interact with single entries from the Freedge Database module.

Interface Specification

The primary function of the Freedge class is to provide an easy way for the other components in the system to interact with the data stored in the database. This includes functionality that allows for:

1. The creation of new Freedge objects to pass back and forth between the other modules to allow for usage in an object-oriented manner.
2. A function which retrieves how long it has been since the status of the freedge was updated.
3. A function to convert the information stored in the Freedge instance into a displayable string for use by the Administrator Interface.
4. A method to update the status of the Freedge.
5. A method to modify the date of the last update to the current date.

Models

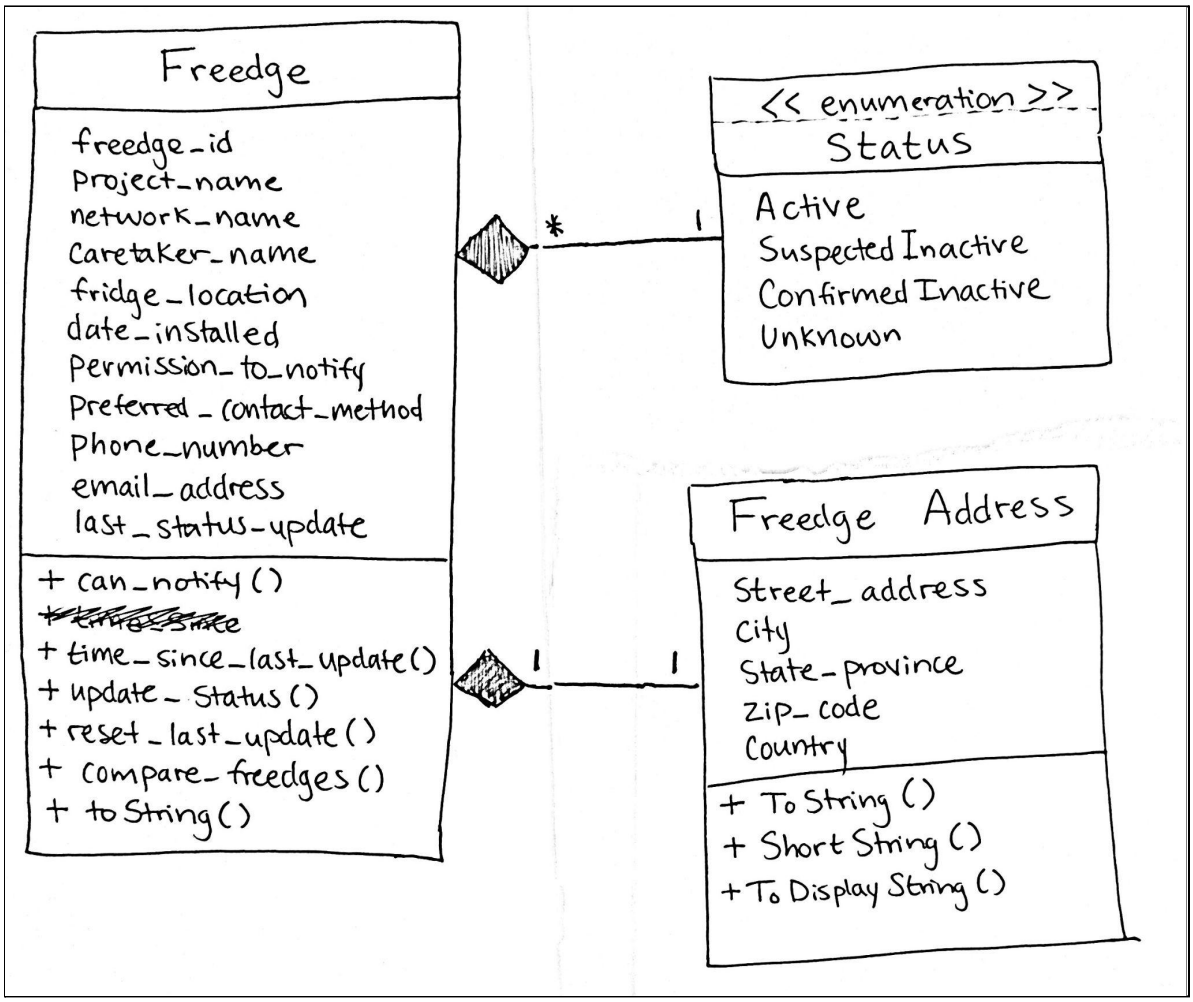

Figure 3: A static UML class diagram of the Freedge Class

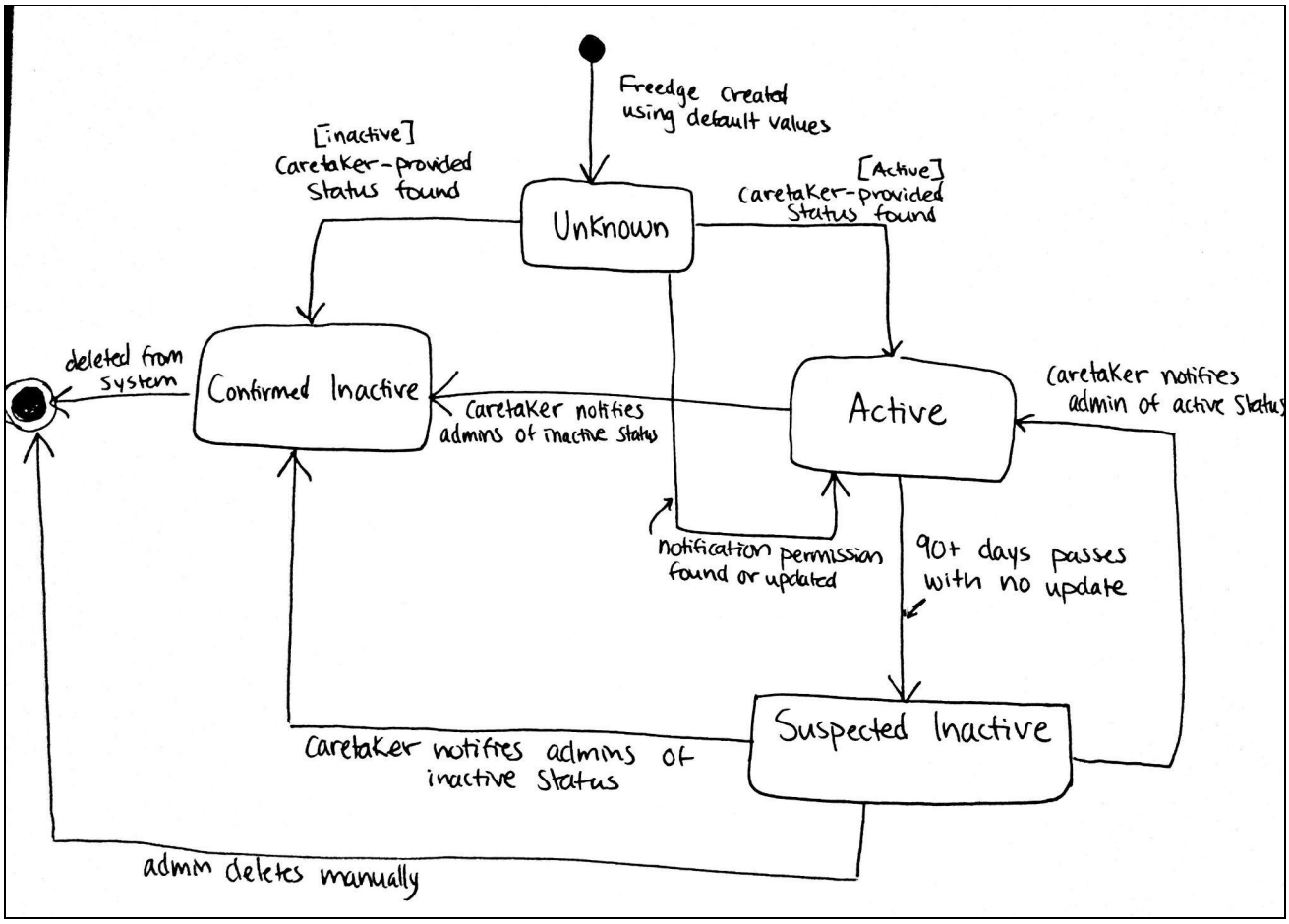

Figure 4: A Dynamic UML State Diagram showing the changing Status of a Freedge

Design rationale

The reason for including the Freedge Class module is to allow for seamless interaction between the modules. Rather than having the Administrator Interface and the Notification Interface interact directly with the database, all of the entries in the Freedge Database are instead converted into Freedge Class objects first. This not only helps maintain the data stored in the database more safely, but it also allows for the implementation of the database to act independently of the other modules.

5.3 Freedge Information Parser

The module's role and primary function

The Information Parser is responsible for parsing a user inputted csv file into a format which can be properly loaded into the Freedge Database.

Interface Specification

The primary purpose of this module is to parse through the user inputted file, and ensure that the Freedge Database has the most up-to-date information. This is done through the following:

1. Read in freedge information from a csv file.
2. Remove all of the whitespace from a field name.
3. Obtain the headers of the file in a list which consists of the project name, network name, date of installation, caretaker name, activity status, contact information, notification permission, and preferred notification method.
4. Set the format of each freedge address to be consistent and place them in a list.
5. Generate lists containing each freedge's information indicated and sorted by the order of the header names above.
6. Send the organized data to the Freedge Database.

Models

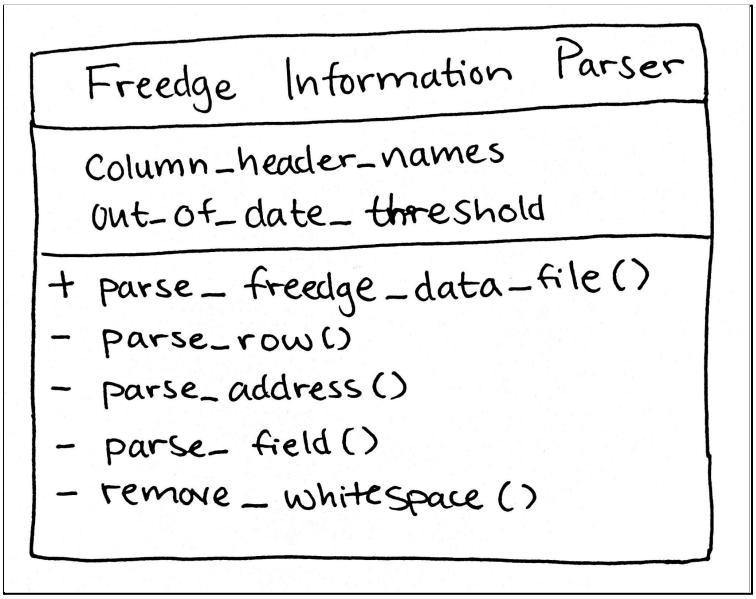

Figure 5: Static UML Class Diagram for Contact Information Parser

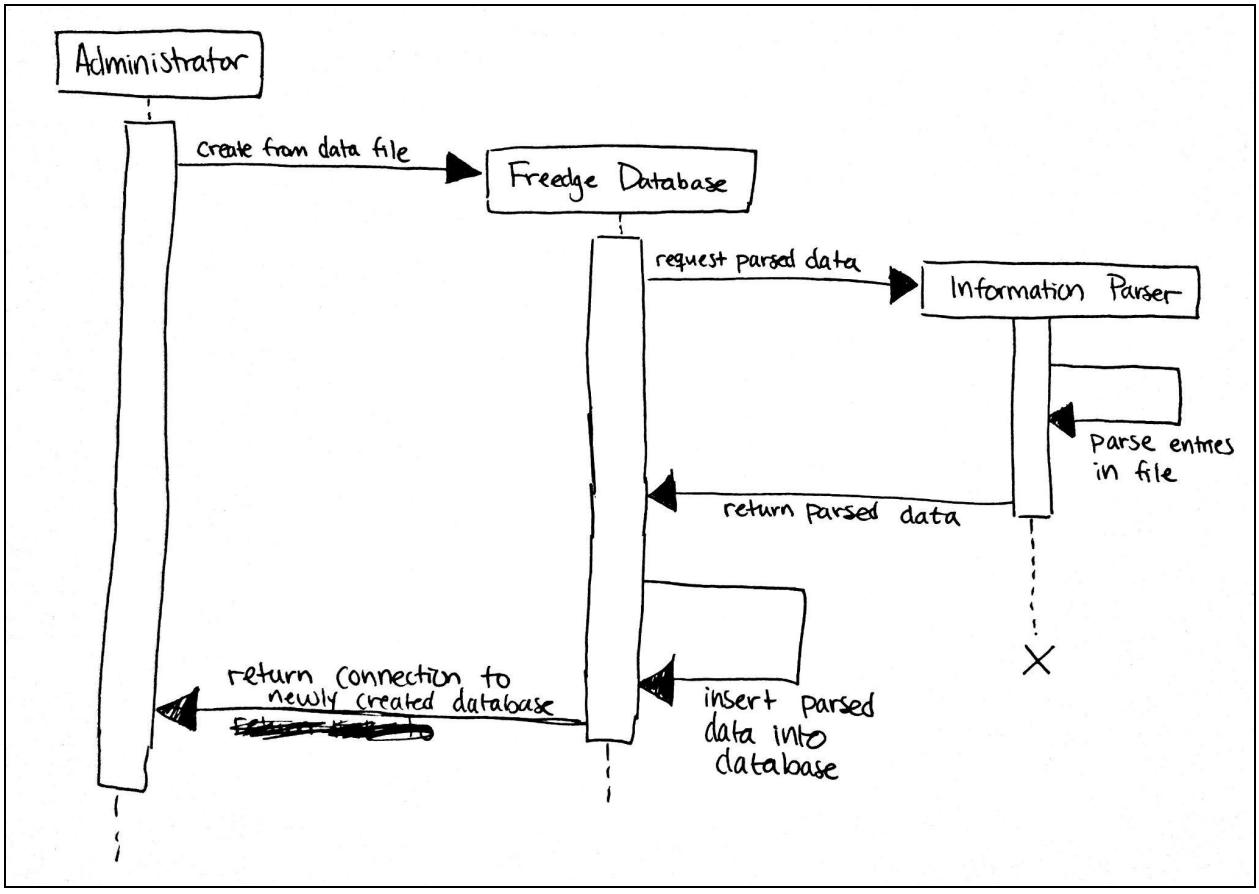

Figure 6: Dynamic UML Sequence Diagram for Contact Information Parser

Design rationale

The caretaker contact information parser models (shown in figures 3 and 4) demonstrate how the methods provided by the information parser interact with the database in order to store the necessary information for each freedge caretaker within the system. The parser sorts through an imported file and stores all the caretaker information so that it is organized, consistent, and ready for use by other system modules.

5.4 Caretaker Notification Interface

The module's role and primary function

The following list explains the different responsibilities of the caretaker notification interface:

1. Obtain each out-of-date freedge caretaker's name, the freedge project name, and the status of the freedge (active or inactive) from the freedge database.
2. Send a personalized message through a visual window to the freedge caretaker using the obtained information asking for a response pertaining to their freedge's status.
3. Receive responses from caretaker indicating their freedge status and update the freedge database.

Interface Specification

The notification interface's main function is to obtain and use each out of date freedge and freedge caretaker's information to create messages asking for confirmation as to if their community fridge is still active or not. This is completed through the following steps:

1. Obtaining each out of date freedge's caretaker name, project name, and activity status from the freedge database.
2. Sending a personalized message created by the Notification Interface to each freedge caretaker in the out of date list using the information displayed on the interface. This message prompts a response from the caretaker.
3. Completing an action according to the caretaker's response:
    a. A response of "Still active" will not change the status of the freedge and leave it marked as "active" in the database.
    b. A response of "No longer active" will change the status of the freedge and update it to "inactive" in the database.

Models

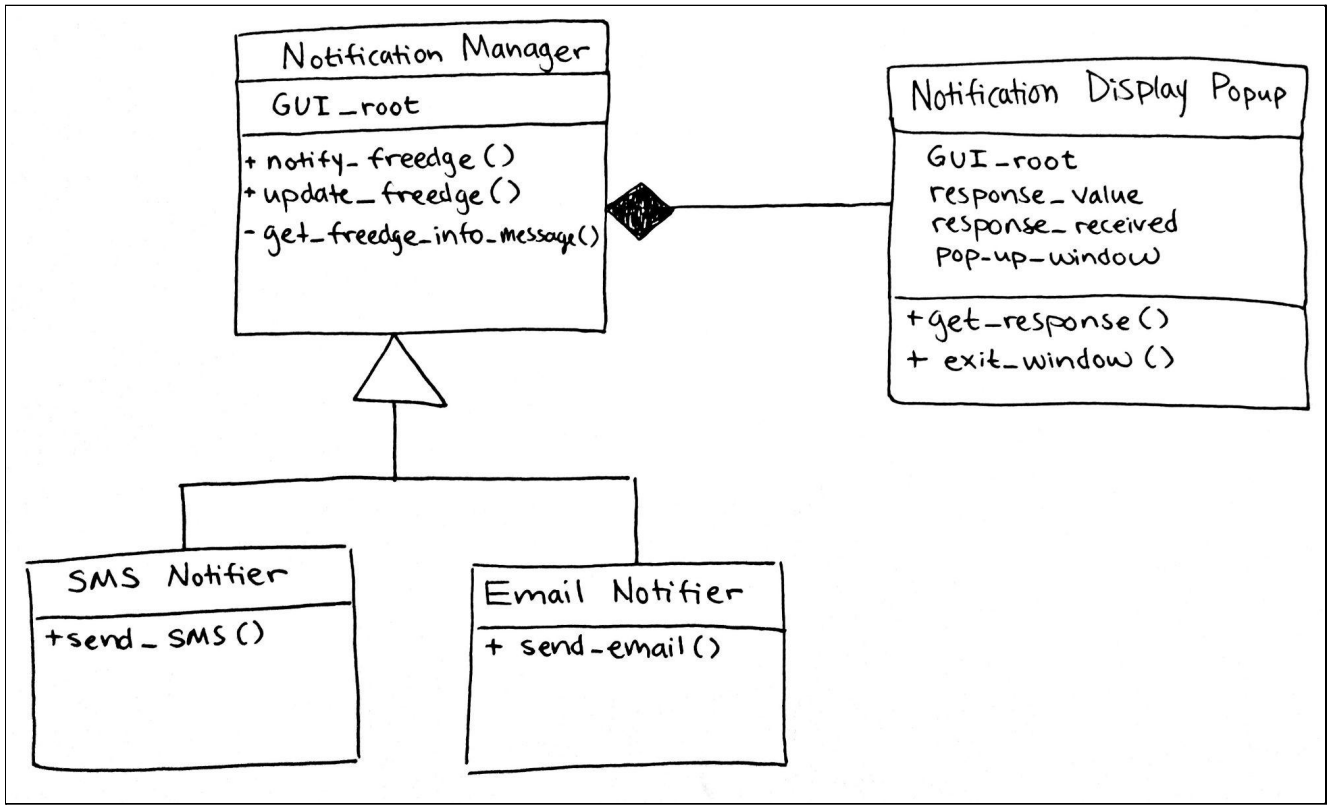

Figure 7: Static UML diagram for the Caretaker Notification Interface

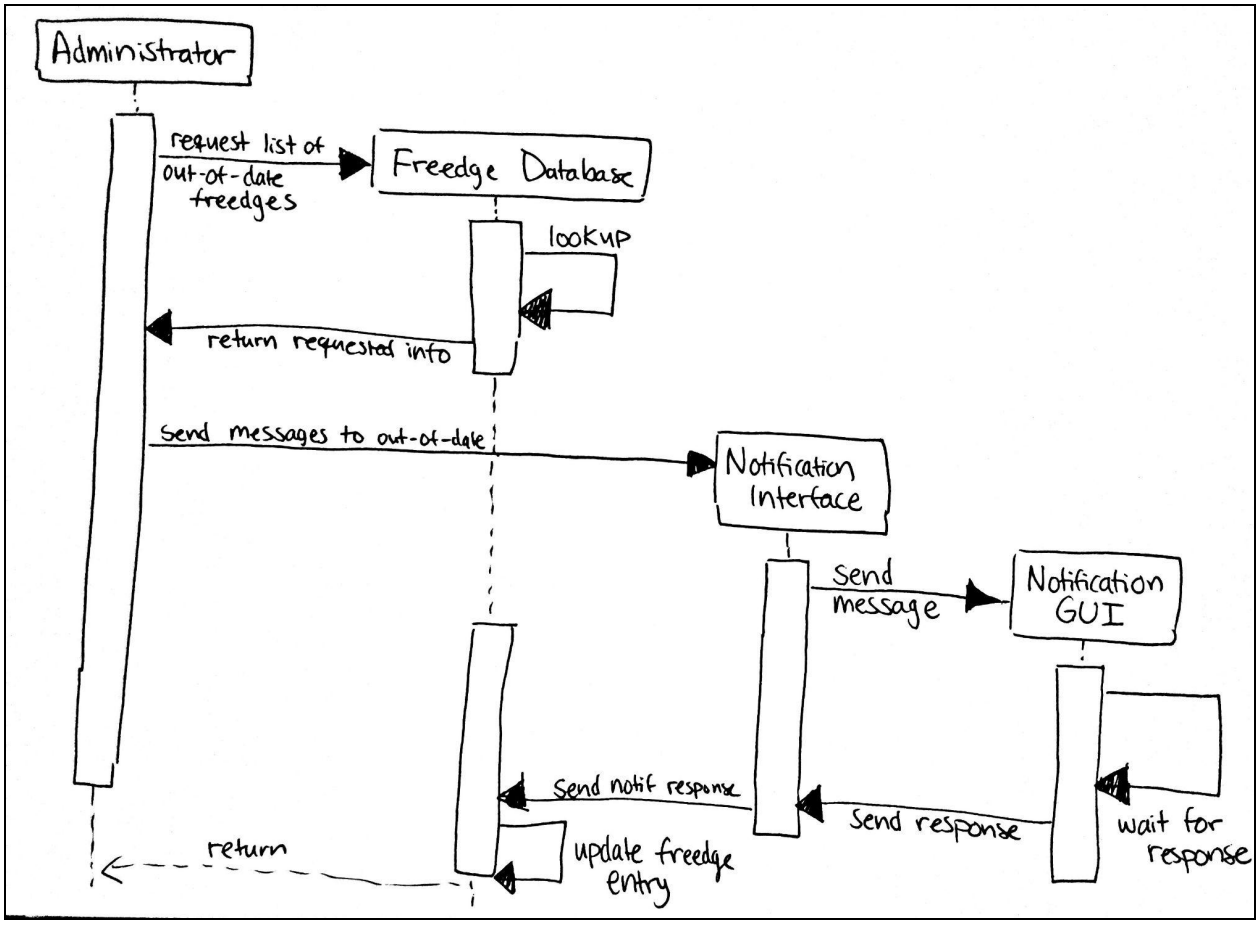

Figure 8: Dynamic UML Sequence Diagram of the Notification Interface

Design rationale

The Notification Interface allows for user interaction with the system to update freedge activity statuses. Making this component its own interface contributes to the future maintainability and evolution of the system to eventually incorporate sending SMS messaging and email notifications. The Notification Interface is important because it provides the Freedge Organization with the functionality to update and monitor all freedge activity.

5.5 Administrator Interface

The module's role and primary function.

The purpose of the Administrator Interface is to allow user interaction between an Administrator and the program. The following list explains the specific responsibilities of this interface:

1. Manage the visual user display windows.
2. Control the user menu options that are contained on the main user display window.
3. Monitor all interactions between the user and the display window.

Interface Specification

The Administrator Interface enables the user to interact with all of the other modules of the system. This is done through the following steps:

1. Upon starting the system, the GUI module will prompt for the user to import a data file to use while displaying a menu with multiple options:
    a. Create or load a new database
    b. Select a freedge
    c. Notify a selected freedge or all out of date freedges
    d. Exit the system
2. When the administrator chooses to create or load a new database, they will be asked to select which csv or db file they would like to use and the information from the file will be parsed and uploaded into the visual window for administrator use.
3. When the administrator selects a freedge, they will be able to view all of the associated information in that freedge in a separate component on the main menu window.
4. When the administrator chooses a notification option, the notification interface will prompt a window to appear with a personalized message asking a caretaker to update the activity status of their freedge.
5. When the administrator chooses to exit the system, the program will stop running completely.

Models

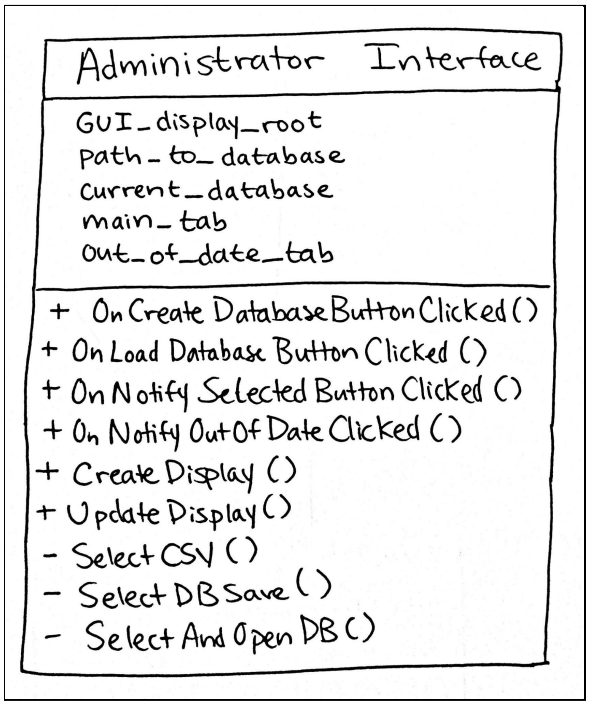

Figure 9: Static UML Class Diagram of Administrator Interface

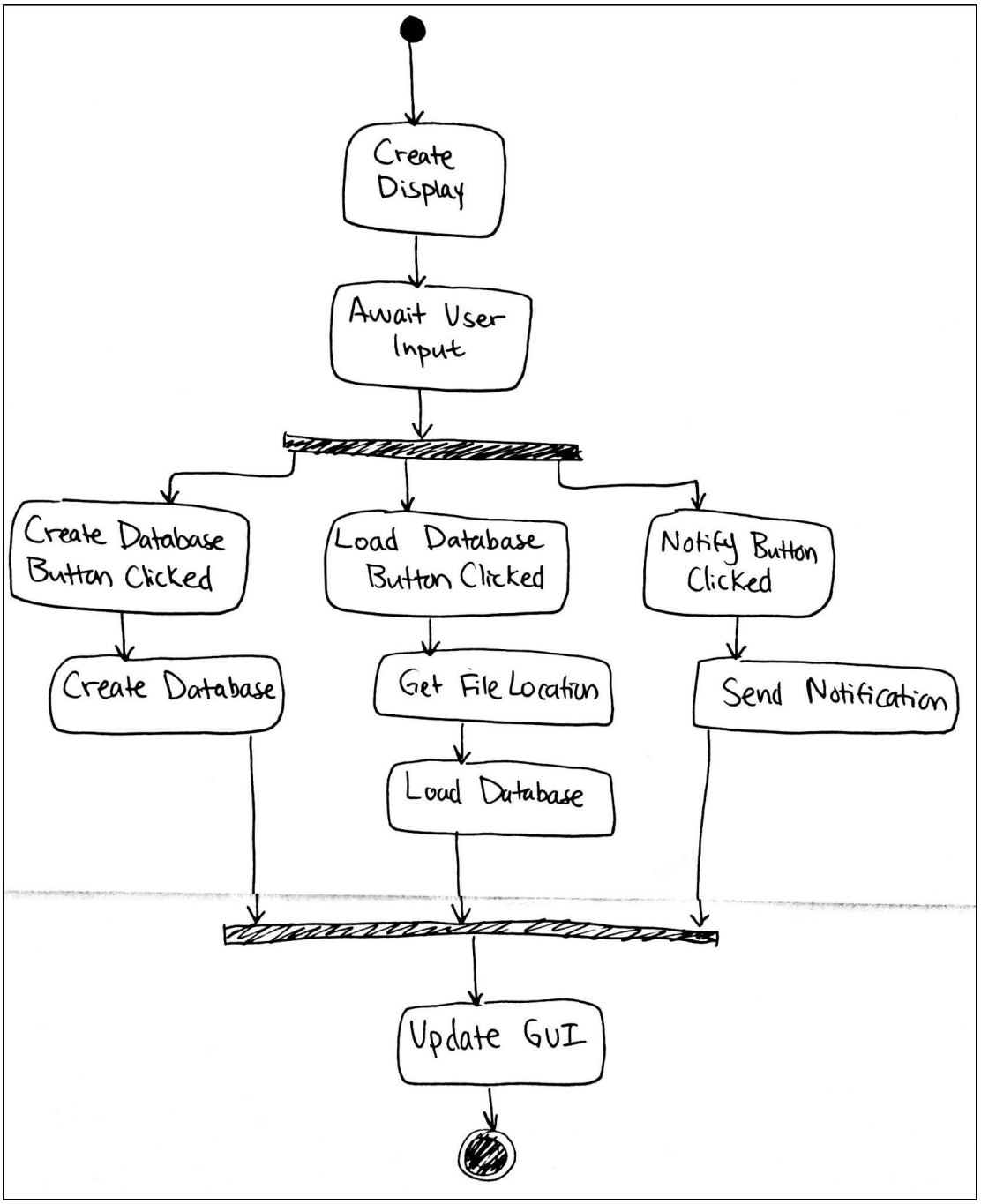

Figure 10: Dynamic UML Activity Diagram for the Administrator Interface

Design rationale

The rationale behind implementing an Administrator Interface stems from the need to host user options in one single main menu. The Administrator Interface module provides a display to users and listens for user input. Responding to this input, the module will make calls to other components of the system and await further input from the user as necessary.

6. Alternative Designs

6.1 Sensor Data Interface

An alternative design that was considered for this system was to have a freedge sensor data interface. Some of the freedges in the database were built with a Raspberry Pi sensor that keeps track of how often the fridge door was opened. This sensor data could have contributed to updating the freedge activity status. If the fridge door was opened within the last month, the fridge would be marked as active and otherwise it would be considered as inactive. This information would be obtained in a similar manner to how the other freedge data is obtained from the database. The interactions between the sensor data interface and the other components is listed below:

1. The sensor data would be obtained from contact_info_parser and inserted into freedge_database.db by freedge_database.py
2. Once this information is stored in the database, the notification interface would obtain it and use it as a factor to indicate whether or not a notification should be sent to the freedge caretaker based on the last time the fridge door was opened. If the door had not been opened within the last month, the notification would be sent and otherwise the fridge status would remain as active.
3. The status of the fridge would be updated based on the response from the user which can be viewed through the administrator interface.

Originally it was thought that the sensor information would be provided for each freedge in the database, however upon further investigation into the existing freedge database files it was noted that not all of the freedges have this sensor. Due to this inconsistency, it was decided that it would not be practical to create a system feature that does not include all the fridges.

6.2 Data Visualization Interface

An additional alternative design that was considered for our system was to use the sensor data to create graphical representations of freedge activity over time. We planned to create a separate component that obtained the sensor data from the freedge database to be used in the formation of this graph. It would have provided a visual to the user that showed the frequency of activity over the course of multiple months. The interactions between the graph interface and the other components is listed below:

1. The user selects to view an activity graph from the administrator interface window.
2. The data visualization interface collects the sensor data (6.1 contains steps on how sensor data is obtained) from the Freedge Database interface.

3. Once the sensor data is obtained, a graph would be created to show how often a fridge is accessed each month of the year.

Upon the decision to not include the freedge sensor data in our database, it was also concluded that there would be no need to include a data visualization interface.

7. Dynamic Models of Operational Scenarios (Use Cases)

There are two primary use cases for this system, which are to verify and update a freedge's status, and to build a graph to visualize how often a community fridge is accessed over time.

7.1. Use Case #1

A Freedge administrator wants to verify and update the status of a particular freedge.

The freedge status verification use case is as follows:

1. The Freedge administrator downloads the current database from the Freedge website containing each freedge caretaker and the caretaker's contact information as well as the freedge location and other general freedge information.
2. The administrator uploads the data file by selecting "Create Database" on the main menu. This file is sent to the caretaker's contact parser module.
3. The caretaker notification module receives this contact information and sends SMS messages or emails to each freedge caretaker and prompts them to give a response regarding their freedge status (active or inactive).
4. A caretaker responds with "Still active"
    a. The notification module sets the status of the freedge to "active" in the database.
5. A caretaker responds with "No longer active"
    a. The notification module sets the status of the freedge to "inactive" in the database.
6. There is no response from the caretaker.
    a. The system will register this as a non-response and status will be unknown.
7. The Freedge administrator exits the system.

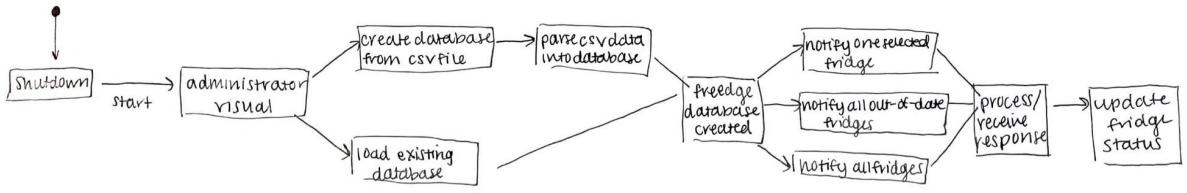

Figure 11: Diagram of System Usage
The boxes show modules, and the lines show which modules communicate with one another

7.2 Use Case #2

Creating a new database from a csv file.

Creating a new database within the system operates as follows:

1. The freedge information spreadsheet is downloaded from the Freedge website or obtained by a Freedge administrator.
2. The system recognizes that there was no database previously uploaded and the Freedge administrator is asked to choose a new csv file to upload into the system.
3. The Freedge administrator selects the file they would like to use and upload it to the system. The information in the csv file is parsed and turned into a database to be used by the administrator.
4. The Freedge administrator exits the system when done using the system.
5. Upon starting the system in their next use, the Freedge administrator will be asked if they would like to reload the database they just created or not.

8. References

Faulk, Stuart. (2011-2017). CIS 422 Document Template. Downloaded from https://uocis.assembla.com/spaces/cis-f17-template/wiki in 2018. It appears as if some of the material in this document was written by Michal Young.

9. Acknowledgements

This template was given to the UO CIS 422 class by Anthony Hornof. This template is similar to a document produced by Stuart Faulk in 2017, and uses publications cited within the document, such as IEEE Std 1362-1998 and ISO/IEC/IEEE Intl Std 29148:2018.