Software Design Description

for

the Irrigation Recommendation System

Version 1.3

Prepared by Raulie Raulerson

12/5/2017

Table of Contents

Table of Contents ii

Revision History ii

1. Introduction 1

1.1 Purpose 1

1.2 Scope 1

1.3 Dependencies, Constraints, and Assumptions 1

1.4 Document Conventions 2

1.5 Definitions, Acronyms, and Abbreviations 3

1.6 References 4

2. System Features 5

2.1 Essential Features 5

2.2 Non-Essential Features 6

3. Design Overview 7

3.1 Description of the Problem 7

3.2 Technologies Used 7

3.3 System Architecture 8

3.4 System Operations 9

4. Data Design 11

4.1 Internal Software Data Structure 11

4.2 Data Format for Uploads 12

5. Architectural and Component-Level Design 12

5.1 System Structure 12

5.2 IRS Interface Design 13

5.3 Graphical User Interface (GUI) Design 15

5.4 GUI Draft Wireframe 21

6. Testing 21

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Raulie Raulerson | 7/31/2017 | Initial version | 1.0 |
| Raulie Raulerson | 8/1/2017 | Added sections 3-5 | 1.1 |
| Raulie Raulerson | 8/2/2017 | Updated section 5 | 1.2 |
| Raulie Raulerson | 12/5/2017 | Revised entire document to reflect design changes | 1.3 |

# Introduction

## Purpose

The purpose of this document is to transform the requirements listed in the Software Requirements Specification (SRS) into a design that can be implemented in the Construction phase of the Software Development Life Cycle (SDLC). This design document will outline the interfaces, class design, design constraints, major design decisions, potential test cases, and component-level design for the Irrigation Recommendation System (IRS).

## Scope

The IRS product will consist of three major components: 1) a Graphical User Interface (GUI) that will enable a user to upload their specific soil moisture sensor (SMS) and weather data, 2) the classes that configure and train (or load) the recurrent neural network (RNN) and that generate an irrigation recommendation report, and 3) the DeepLearning4J (DL4J) API that house the RNN that will analyze and make a recommendation based on the data input. Please note that component number 3 listed above is critical to the success of the project; however, it is an open-source Java library that is already able to be utilized and will not need to be constructed for this project. Furthermore, the design for the project can be broken down into the essential features for implementing the product and those that are non-essential.

## Dependencies, Constraints, and Assumptions

### **Dependencies**

The following dependencies will be assumed in the development of the IRS project:

* Training data will be linked to a specific crop and soil type by the user when a recommendation is requested.
* Mac systems will be able to run a shade Java Archive (JAR) file produced by Maven.
* Dependencies amongst files and libraries for the DeepLearning4J API and the underlying Java source code will be listed in the project object model (POM) file created by Maven and all files that are needed for a particular build will be in the POM.xml file.

### **Constraints**

The major constraint for the IRS project will be time. Time could possibly constrain the implementation of the non-essential features listed 2.2. This constraint will be avoided by paying strict adherence to the detailed timeline that has been developed.

### **Assumptions**

The following assumptions will be made in the development of the IRS project:

* Assume that users know how to obtain data from their sensors as well as nearby weather stations.
* Assume that users know how to put data into correct format.
* Assume that users know how get their data into CSV format.
* Assume that users have access to the Internet.

## Document Conventions

Sections, sub-sections, and any additional headings within those sections/sub-sections use the following typographical convention in this document:

**1. Section**

**1.1 Sub-section**

**1.1.1 Heading in sub-section**

All acronyms that will be used in this document are listed in section 1.5.

## Definitions, Acronyms, and Abbreviations

### **Acronyms**

API – Application Programming Interface

CR – Change Request

CSV – Comma-Separated Values

DL4J – DeepLearning4J

EC – Electrical Conductivity

ET – Evapotranspiration

FDACS – Florida Department of Agriculture and Consumer Services

GUI – Graphical User Interface

IDE – Integrated Development Environment

IRS – Irrigation Recommendation System

JAR – Java Archive

JRE – Java Runtime Environment

OS – Operating System

POM – Project Object Model

RNN – Recurrent Neural Network

SBDTS - Software Budget and Detailed Timeline Spreadsheet

SDD – Software Design Description

SDLC – Software Development Life Cycle

SPMP – Software Project Management Plan

SMS – Soil Moisture Sensor

SRS – Software Requirements Specification

STD – Software Testing Document

UI – User Interface

VWC – Volumetric Water Content

## References

**Project Definition Report**

|  |  |
| --- | --- |
| Version | 1.2 |
| Date | 10/5/2017 |
| Author | Raulie Raulerson |

**Software Requirements Specification (SRS)**

|  |  |
| --- | --- |
| Version | 1.7 |
| Date | 12/4/2017 |
| Author | Raulie Raulerson |

**Software Project Management Plan (SPMP)**

|  |  |
| --- | --- |
| Version | 1.5 |
| Date | 12/4/2017 |
| Author | Raulie Raulerson |

**Risk Management Plan**

|  |  |
| --- | --- |
| Version | 1.5 |
| Date | 12/4/2017 |
| Author | Raulie Raulerson |

Aasen, R., Carlisle, E., Carson, N., Cerauskis, L., Jeffers, E., Green, T., . . . Vo, P. (2011, March 4). *SplitPay Software Design Document*(Rep.). doi:https://cise.ufl.edu/class/cen3031fa11/documents/examples/SDD\_Example\_1\_2011.pdf

Chagnon, T., Kokinda, J., Meinert, A., & Naegely, D. (2009, April 7). *TheraWii Software Design Document*(Rep.). doi:https://www.cs.drexel.edu/~dpn52/Therawii/design.pdf

Pressman, R. S., & Maxim, B. R. (2015). *Software engineering: A practitioner's approach* (8th ed.). New York: Mcgraw-Hill.

# System Features

## Essential Features

The features listed below are **necessary** for the IRS product to function as intended and they must be included in the final product.

### **Enter Required Data into GUI**

### The system shall allow the user to enter the crop type.

### The system shall allow the user to enter the soil type.

### The system shall allow the user to enter 1-3 soil moisture sensor depths.

### **Upload SMS and Weather Data**

* The system shall allow the user to select the “Browse” button.
* The system shall allow the user to navigate to the file destination of their choice.
* The system shall allow the user to select the file of their choice.
* The system shall allow the user to upload the file chosen.
* The system shall prompt the user when the file they chose has been uploaded.

### **Train the RNN/model**

### The system shall allow the user to upload training data.

### The system shall allow the user to select the “Train the System Using Your Crop and Soil Data” button.

### The system shall prompt the user when the RNN/model has finished optimizing/tuning its parameters based on the number of epochs and iterations that are chosen (note: the system administrator and project manager will determine the learning rate, number of epochs, and number of iterations to be used in optimizing the RNN/model later in the project).

* The system shall save the RNN’s parameters to a file destination and name as outlined in section 4.5.

### **Generate Irrigation Recommendation Report**

* The system shall allow the user to select the “Generate Irrigation Recommendation” button.
* The system shall prompt the user that the report was generated.
* The system shall prompt the user to upload another csv file to be used for generating the irrigation recommendation report.
* The system shall display the report on screen.
* The system shall inform the user where the report was saved.

## Non-Essential Features

The features listed below are not necessary for the IRS product to function as intended. However, the plan is to include them in the final product as they will ensure the product is more efficient, easier to use, and has better performance.

### **Save RNN/Model Parameters**

* The system shall allow the user to select the “Save Parameters from a Model Run” button.
* The system shall designate the file name and path of the saved parameter file.
* The system shall allow the user the choice to save the file.
* The system shall prompt the user when and where the file they chose has been saved.

### **Load Parameters from Previous RNN/Model Run**

* The system shall allow the user to upload a file with parameters saved from a previous model run.
* The system shall allow the user to select the “Load Parameters from a Previous Model Run” button.
* The system shall prompt the user when the RNN’s parameter are loaded from the file that was selected.

# Design Overview

## Description of the Problem

SMSs are a revolutionary technology that could potentially help irrigation system users conserve an abundance of water. One of the drawbacks with SMSs is the steep learning curve in analyzing and interpreting the data produced by these sensors. The IRS products proposes to make this burden easier on irrigators by using a RNN to analyze, interpret, and, ultimately, make an irrigation recommendation to the user. The recommendation will be comprised of whether to irrigate, and, if irrigation is recommended, a proposed irrigation quantity (in inches).

## Technologies Used

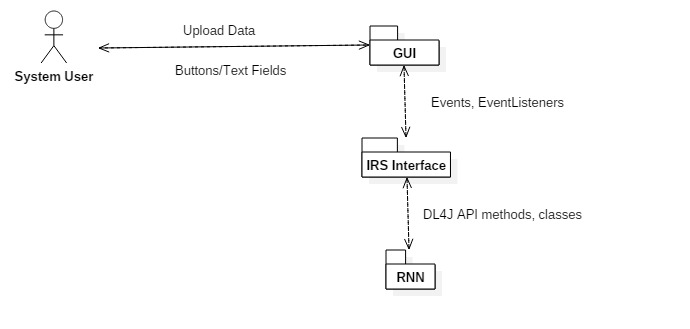
The IRS will interact with data by uploading the data directly into the RNN/Model. All data will be in Comma-Separated Values (CSV) format unless a user loads parameters from a previous model run (zip file). The system that will be developed on macOS Sierra (Version 10.12.3 or later). *Please note that the interface between the SMS and its data will be separate and independent from this project.*

The IRS software will be developed with the IntelliJ IDEA (Version 2016.3.6) Integrated Development Environment (IDE) which will use the Java Runtime Environment (JRE) version 1.8.0\_112-release-408-b6 x86\_64 and the Java Virtual Machine (OpenJDK 64-Bit Server VM by JetBrains s.r.o). In addition, Maven will be used as the software project management tool of choice in combination with the DeepLearning4J Application Programming Interface (API).

## System Architecture

The overview of the system architecture for the IRS software is shown in Figure 1 below. The system will be comprised of the following components:

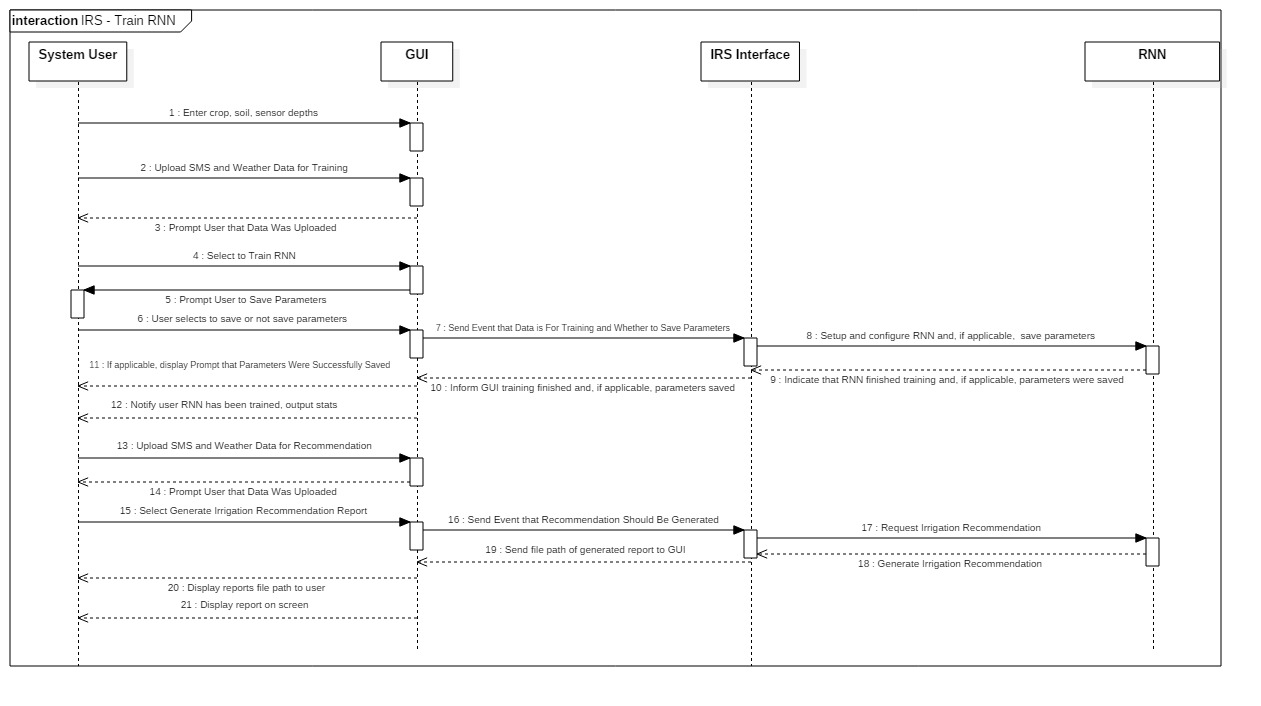
* Graphical User Interface (GUI) – this component will consist of the interface that the system user will interact with. It will consist of buttons and text fields.
* IRS Interface – this component will consist of the classes that interact with the GUI component as well as the DeepLearning4J (DL4J) API. These classes will setup the RNN/model for regression using the user’s uploaded information. It will also be responsible for generating the irrigation recommendation report that is produced.
* DL4J (RNN) – this component contains all the source code necessary for the RNN. This DL4J exists as an open-source Java library. It is only listed here because it will be instrumental to the project.



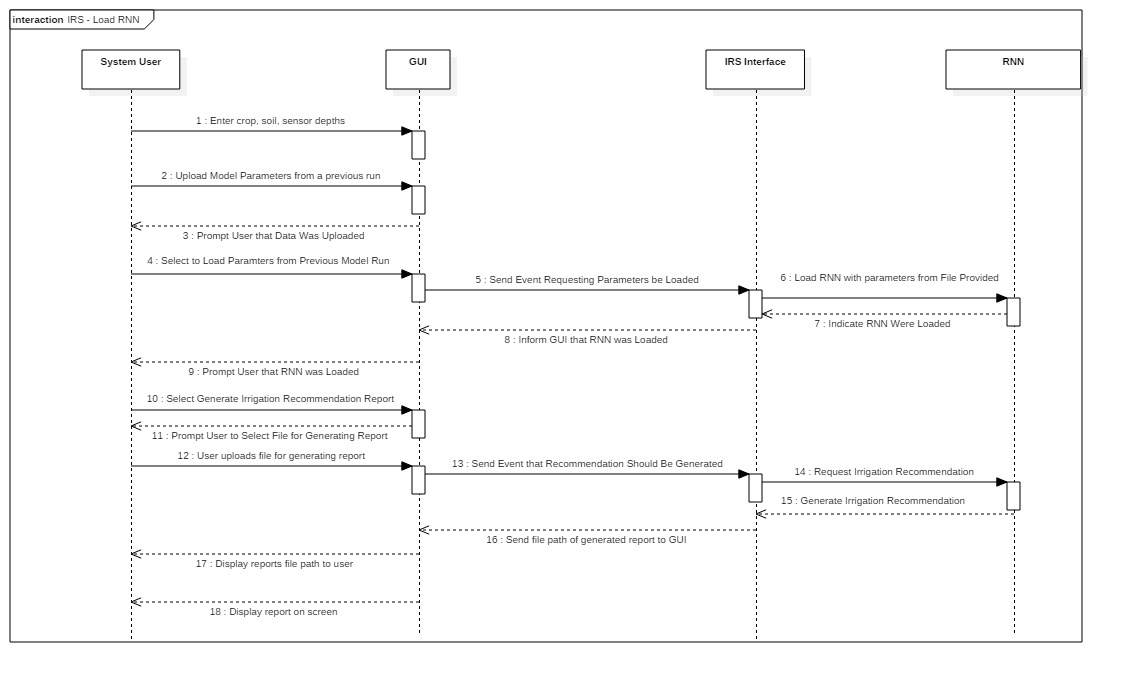
**Figure 1.** IRS Architecture

## System Operations

Figure 2 below outlines the UML Sequence Diagram for the IRS operations to train the RNN, save the parameters that are produced, and generate an irrigation recommendation report. Figure 3 displays the UML Sequence Diagram for the IRS operations that involve loading parameters from a saved file and generating an irrigation recommendation report. Please note that both these diagrams outline the functionality of the systems if errors aren’t encountered. Including errors in the sequence diagram would have complicated the diagrams, thus they were left out to make interpreting the diagrams easier. These two sequence diagrams comprise the functionality of the system that is being designed, constructed, and deployed.



**Figure 2.** Train RNN Sequence Diagram



**Figure 3.** Load RNN Sequence Diagram

# Data Design

## Internal Software Data Structure

There is no proposed database for the IRS. Instead the user will upload data to the system using the GUI provided. The formats for the data that will be uploaded is displayed in section 4.2.

Parameters saved from training runs on the RNN/Model will be saved in zip files. If a user requests to load parameters from a previous model run, these parameters will be loaded using a zip file.

## Data Format for Uploads

The minimum data that will be uploaded to generate an irrigation recommendation report will be soil temperature (degrees Fahrenheit), electrical conductivity (dS/m), relative humidity (%), volumetric water content (m3/m3), daily rainfall total (inches), soil type, and the crop being grown. Below is the required data format for any data uploaded using the GUI when only 1 sensor depth is entered:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Date | VWC (m3/m3) | EC (dS/m) | Temp (F) | Rainfall (in) | Relative Humidity (%) | ET (in) |

Below is the required data format for any data uploaded using the GUI when 2 sensor depths are entered:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Date | VWC (m3/m3) | EC (dS/m) | Temp (F) | VWC (m3/m3) | EC (dS/m) | Temp (F) | Rainfall (in) | Relative Humidity (%) | ET (in) |

Below is the required data format for any data uploaded using the GUI when 3 sensor depths are entered:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Date | VWC (m3/m3) | EC (dS/m) | Temp (F) | VWC (m3/m3) | EC (dS/m) | Temp (F) | VWC (m3/m3) | EC (dS/m) | Temp (F) | Rainfall (in) | Relative Humidity (%) | ET (in) |

# Architectural and Component-Level Design

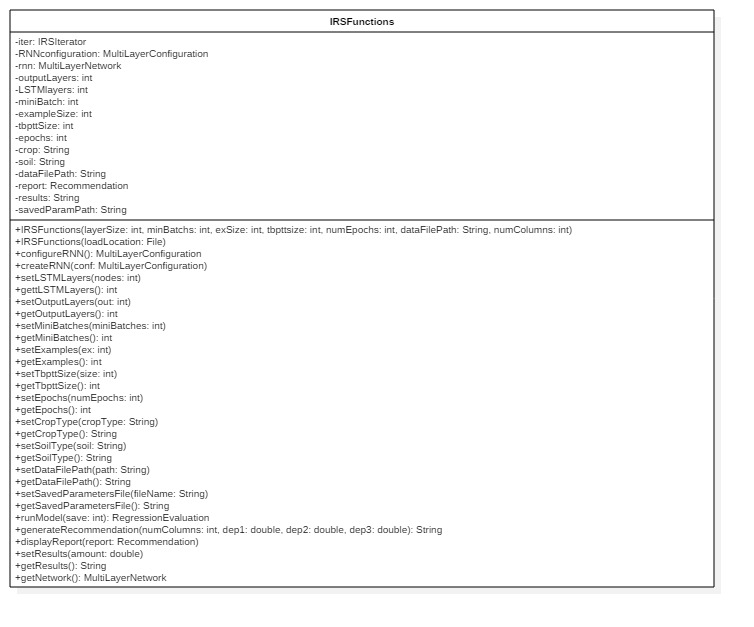
## System Structure

The scope of this project (outlined in section 1.2) consists of 3 main components. Only two of the components that were mentioned need to be designed, constructed, and deployed for this project. The components of interest are the GUI and the IRS interface that will interact with the DL4J API. The GUI will interact with the IRS interface via the ActionListener Java classes. Each of the two components will consist of multiple classes.

## IRS Interface Design

### **IRSFunctions Class**

The IRSFunctions class consists of most of the functionality that needs to be called based on the user’s selection in the GUI. This class sets up and configures the RNN, saves or loads parameters to/from a file, and creates Recommendation report object. Figure 4 displays the UML Class Diagram for the IRSFunctions class.



**Figure 4.** IRSFunctions Class Diagram

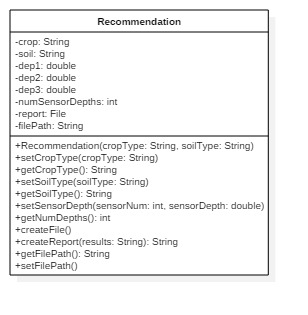
### **IRSIterator Class**

The IRSIterator class is designed to create a custom iterator for iterating through the data that is uploaded by the system user (using the GUI). This iterator will utilize a preprocessor to normalize the data that is input prior to it being fed into the RNN. The IRSFunctions class will use the IRSIterator class in configuring the RNN and train/fitting the data to the model. This class will also be used for generating an irrigation recommendation. Figure 5 displays the UML Class Diagram for the IRSIterator class.

**Figure 5.** IRSIterator Class Diagram

### **Recommendation Class**

The Recommendation class is designed to receive input from the IRSFunctions class and use that input to generate a text file that contains the irrigation recommendation report that the user requested. Figure 6 displays the UML Class Diagram for the Recommendation class.

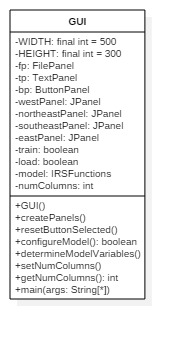


**Figure 6.** Recommendation Class Diagram

## Graphical User Interface (GUI) Design

### **GUI Class**

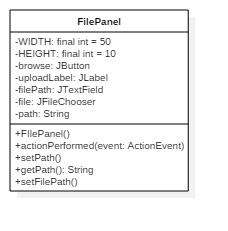
The GUI class will be the main panel that contains the frame and the 3 other panels outlined below. This panel will be responsible for communicating the user’s interactions with the GUI to the IRS interface described in section 5.2. This class will also contain the ButtonPanel class, listed in section 5.3.4, and the private ActionListener classes for the ButtonPanel. This constraint was necessary so that the GUI could act on an event that occurred when a button is selected (ensures cohesive and loosely coupled classes). Figure 7 displays the UML Class Diagram for the GUI class.



**Figure 7.** GUI Class Diagram

### **FilePanel Class**

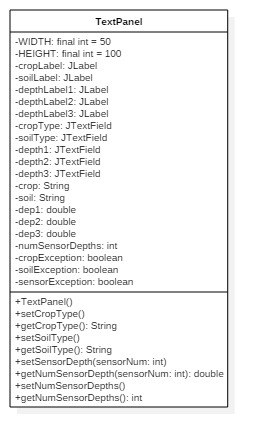
The FilePanel class will configure the panel for the GUI that will contain the label, text field, and browse button for uploading a file containing SMS and weather data. This panel will be a component of the GUI class. Figure 8 displays the UML Class Diagram for the FilePanel class.



**Figure 8.** FilePanel Class Diagram

### **TextPanel Class**

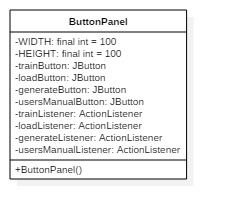
The TextPanel class will configure the panel for the GUI that will contain the labels and the five textfields for the user to enter the required data pertaining to their irrigation system. This panel will be a component of the GUI class. Figure 9 displays the UML Class Diagram for the TextPanel class.



**Figure 9.** TextPanel Class Diagram

### **ButtonPanel Class**

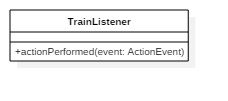
The ButtonPanel class will configure the panel for the GUI that will contain the four buttons that the user will select from to generate an action from the GUI. These buttons consist of the train the RNN/model, load model parameters to a file, save model parameters to a file, and generate an irrgitaon recommendation buttons. This panel will be a private class within the GUI class, so that events fired by the buttons can be captured and used by the GUI. Figure 10 displays the UML Class Diagram for the ButtonPanel class.



**Figure 10.** ButtonPanel Class Diagram

### **Private ActionListener Classes**

Four private ActionListener classes are contained within the GUI class. These classes listen to events fired by the JButtons in the ButtonPanel class. Each of the four classes are shown in Figure 11 below.

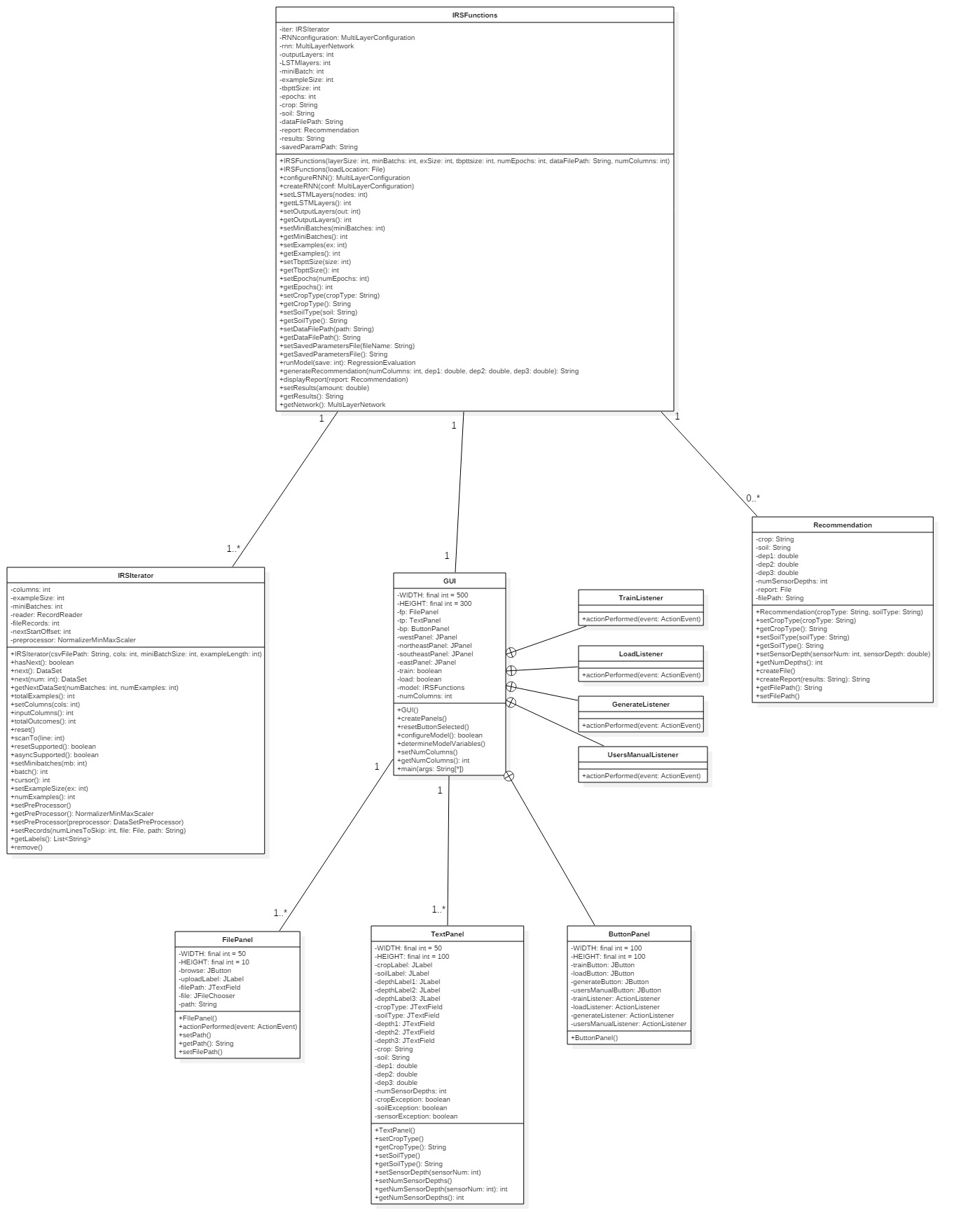








**Figure 11.** Class Diagrams for the Private ActionListener Classes Contained in the GUI Class



**Figure 12.** Entire System Class Diagram

## GUI Draft Wireframe

Figure 13 shows a final wireframe of the GUI that a system user will interact with.



#### Figure 13. Final GUI for the Irrigation Recommendation System

# Testing

The methods, constructors, and attributes listed in Figure 12 will be tested as this design is being implemented. Any duplicative methods or attributes will not be implemented and this design will be revised to reflect the change. This is commonly referred to as white box testing and is conducted during the implementation phase of the project.

Black box testing will be conducted after most the coding has been finished. This will consist of more unit testing as well as system/functional testing. This type of testing will ensure that the features listed in section 2 will be tested, and the functional requirements listed in the SRS, are implemented correctly.

Testing will also be done on deployment of the system. This will entail testing the JAR file on whether it executes on the system’s it was designed for.