Department of Computer Science and Engineering
The University of Texas at Arlington

Detail Design Specification

Project: Home Irrigation Control System

Team Members:
Belachew Haile-Mariam
Gautam Adhikari
Jeremiah O'Connor
Tung Vo

Last Updated: 2/26/2015

TABLE OF CONTENTS

Document Revision History

Revision Number	Revision Date	Description	Rationale
0.1	2/14/15	ADS first draft	Created the skeleton for the sections
0.2	2/23/15	Merging Sections	Giving consistency to the document
0.3	2/24/15	Adding Table and Figure numbers	Making the document easier to navigate
1.0	2/26/15	Final Draft Revisions	Incorporated team walkthrough
1.1	3/5/2015	Adding data flow between USB/Serial Interface and Valve Command Processor	Implementing changes from DDS Presentation
2.0	3/6/2015	Figure and table numbers update and details added to Server Layer	Implementing changes from DDS review

List of Figures

List of Tables

1. Introduction

This Detail Design Specification (DDS) document will describe the high-level design of the Home Irrigation Control System (HICS) project. This document will breakdown every component to its lower component and describes all the details about the data flows and the role of each component. This DDS will expand layers proposed in our ADS: Sensor Layer, Hardware I/O Layer, Interface Layer, and Server Layer. It will provide the actual data types, data flows, dependencies, and features along with some pseudo code that demonstrates the functionality of the module. This DDS will begin with explaining each layer, sub system, and module that makes up HICS. It will provide the traceability matrix and outline the requirements to make sure the goal of the project is achieved.

1.1 Product Concept

HICS is an intelligent home irrigation control system that utilizes soil moisture sensors to measure the amount of moisture present in the user's lawn and use this information to water the lawn in an efficient way. The rain sensors and the temperature sensors help prevent watering when unnecessary. The sensors, hardware and their proper integration and programming make HICS a machine that's smart enough to save people time, effort, and money on their home irrigation system. Its purpose is to replace an existing sprinkler control system and allow users to control their home irrigation remotely through a web application that will scale to fit on a computer or mobile device.

HICS uses a central control unit to read soil moisture levels and set watering schedules accordingly by interfacing directly with the users existing irrigation valves. The control unit allows users to remotely schedule or monitor their lawns by communicating with the HICS web application over the Internet. Through the web application, users will be granted complete access to all the features that HICS offers while providing an intuitive and easy to use interface.

1.2 Product Scope

HICS is designed to provide an intelligent home irrigation solution that conserves water and allows users to control their home's irrigation using their own personal computer or mobile device. It is designed to integrate seamlessly into an existing irrigation system and gives users the option to expand their system to support additional valves and sensors. The target audiences for HICS are users with existing sprinkler systems and irrigation/landscaping companies who are looking for a smarter product to sell to their customers. Another major audience for HICS will be people who are concerned about conserving water in their homes. Water conservation enthusiasts will appreciate the numerous options that HICS provides to increase the efficiency of their sprinkler systems.

2. Architecture Overview

This section briefly describes the architectural vision of Home Irrigation Control System (HICS) and breaks down the subsystems into modules. HICS consists of four layers: the Sensor Layer, Hardware I/O Layer, Interface Layer, and Server Layer. Each subsystem will be explained and broken down to modules while briefly explaining their roles. The following diagram (Figure 2-1) illustrates the system architecture of HICS as well as the overall dataflow.

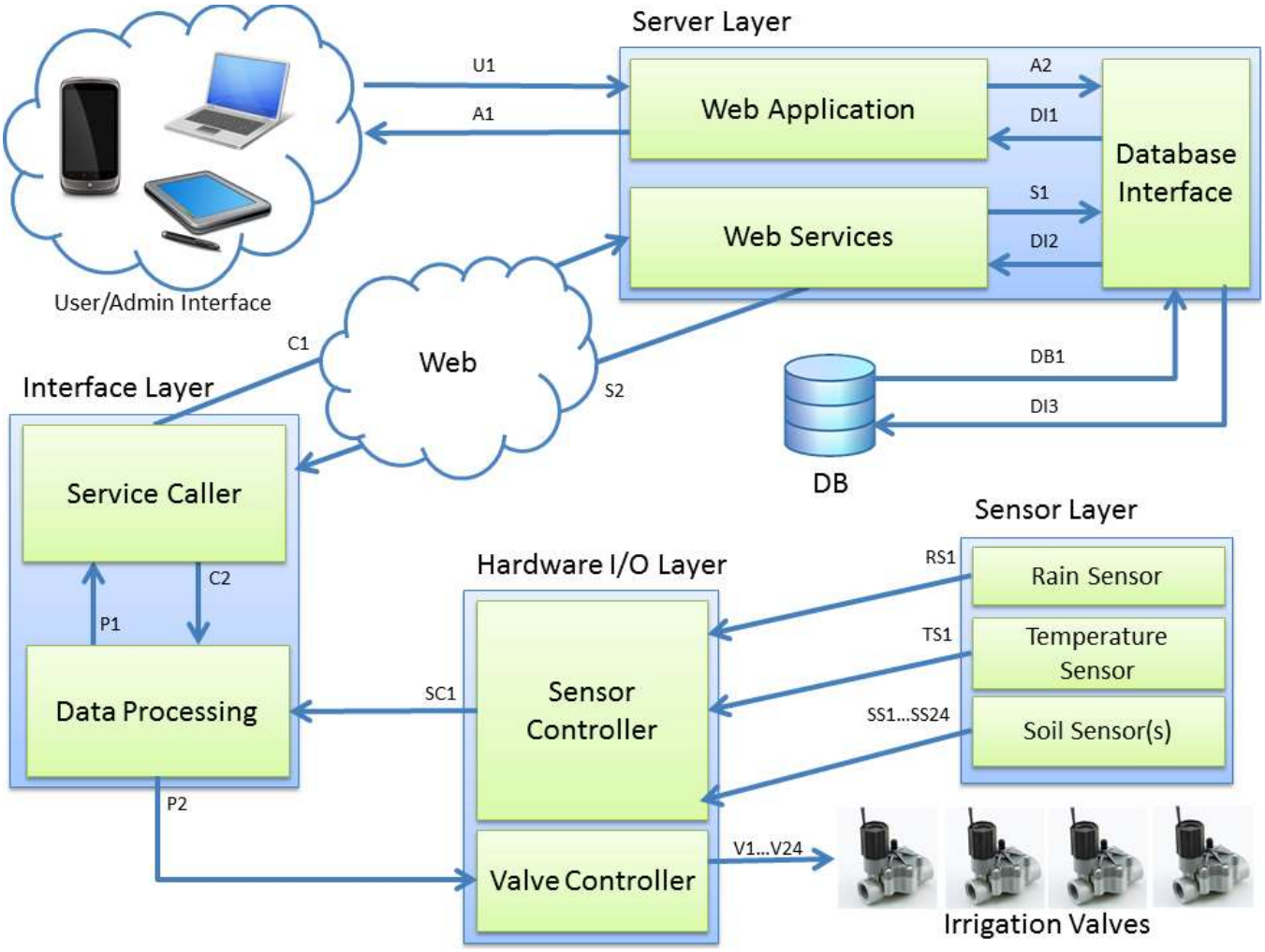

Figure 2-1 Architecture Overview Diagram

2.1 Sensor Layer

The Sensor Layer has three subsystems: the Rain Sensor Subsystem, Temperature Sensor Subsystem, and Soil Moisture Sensor Subsystem. The Sensor Layer is responsible sending all the different environment condition readings to the Hardware I/O Layer. Each Sensor Layer subsystem communicates independently of the other subsystems.

2.1.1 – Rain Sensor Subsystem: The Rain Sensor Subsystem will detect precipitation amounts by collecting rainfall. This subsystem will interface with the Sensor Controller Subsystem and will send an alert signal to stop all watering cycles if it detects rain.

2.1.2 – Temperature Sensor Subsystem: The Temperature Sensor Subsystem will measure the current temperature of the environment. This subsystem will interface with the Sensor Controller Subsystem to send data about the current temperature in the area.

2.1.3 – Soil Moisture Sensor(s) Subsystem: The Soil Moisture Sensor Subsystem will measure the current soil moisture levels at its corresponding irrigating zone. This subsystem will interface with the Sensor Controller Subsystem and send data consisting of the current soil moisture levels. Each soil moisture sensor is associated with its own irrigation valve.

2.2 Hardware I/O Layer

The Hardware I/O Layer collects information from the environment sensors and also controls the activities of the irrigation valve(s). The Hardware I/O Layer includes a Sensor Controller Subsystem and a Valve Controller Subsystem.

2.2.1 – Sensor Controller Subsystem: The Sensor Controller Subsystem will interface directly to all the sensors, which includes the rain sensor, soil moisture sensor(s), and temperature sensor. The main component of the Sensor Controller is a microcontroller that reads the signal transmitted from the sensors.

2.2.2 – Valve Controller Subsystem: The Valve Controller Subsystem will interface directly with the irrigation valve(s). It will also communicate with the Data Processor Subsystem to receive control command to turn the valves on or off. The main component of the Valve Controller is a microcontroller, which will receive the commands from the Data Processor Subsystem and use them to control the operation of the irrigation valve(s).

2.3 Interface Layer

The Interface Layer consists of two subsystems: the Service Caller Subsystem and Data Processor Subsystem. The Interface Layer is responsible for processing all communication between the Hardware I/O Layer and the Server Layer. The communication between the two layers consists of input readings from the Hardware I/O Layer as well as control commands from the Server Layer.

2.3.1 – Service Caller Subsystem: The Service Caller Subsystem will be a mechanism within the Interface Layer to make HTTP calls to the Web Services Subsystem. It will take the response from the web services and relay it to the Data Processing Subsystem.

2.3.2 – Data Processing Subsystem: The Data Processing Subsystem will be used to accept incoming data from the Sensor Controller Subsystem, format it into a JSON message, and pass it along to the Service Caller Subsystem. It will also be used to do the inverse of this process—accept JSON data from the Service Caller Subsystem, parse it into individual valve control signals, and send these signals to the Valve Controller Subsystem.

2.4 Server Layer

The Server Layer has three subsystems: the Web Application Subsystem, Web Services Subsystem, and Database Interface Subsystem. The Server Layer handles communication and control between the web application/web services and the Interface Layer. The Server Layer has access to the database and uses the internet to exchange data between all interfaces.

2.4.1 – Web Application Subsystem: The Web Application Subsystem is the primary interface with which the user communicates with the rest of the system. This subsystem will accept input from the user via a web browser that will be stored in the HICS database. The Web Application Subsystem also provides the GUI which is used to present information to the user.

2.4.2 – Database Interface Subsystem: The Database Interface Subsystem is the central hub for data communication between the Server and Interface Layers. The behavior of each interfacing layer is dictated by the information it sends and receives from the Database Interface Subsystem. Data that is communicated to the Database Interface Subsystem must update and save correctly to ensure the system behaves as expected.

2.4.3 – Web Services Subsystem: The Web Services Subsystem serves as the communication link between the interfacing systems and the database. The subsystem takes requests from the Service Caller Subsystem and either stores or retrieves data depending on the request type.

2.5 Module Decomposition

This section provides a visual breakdown of each sub-system of the architecture diagram into the smallest modules. It shows the inter-relation and dependencies between each module.

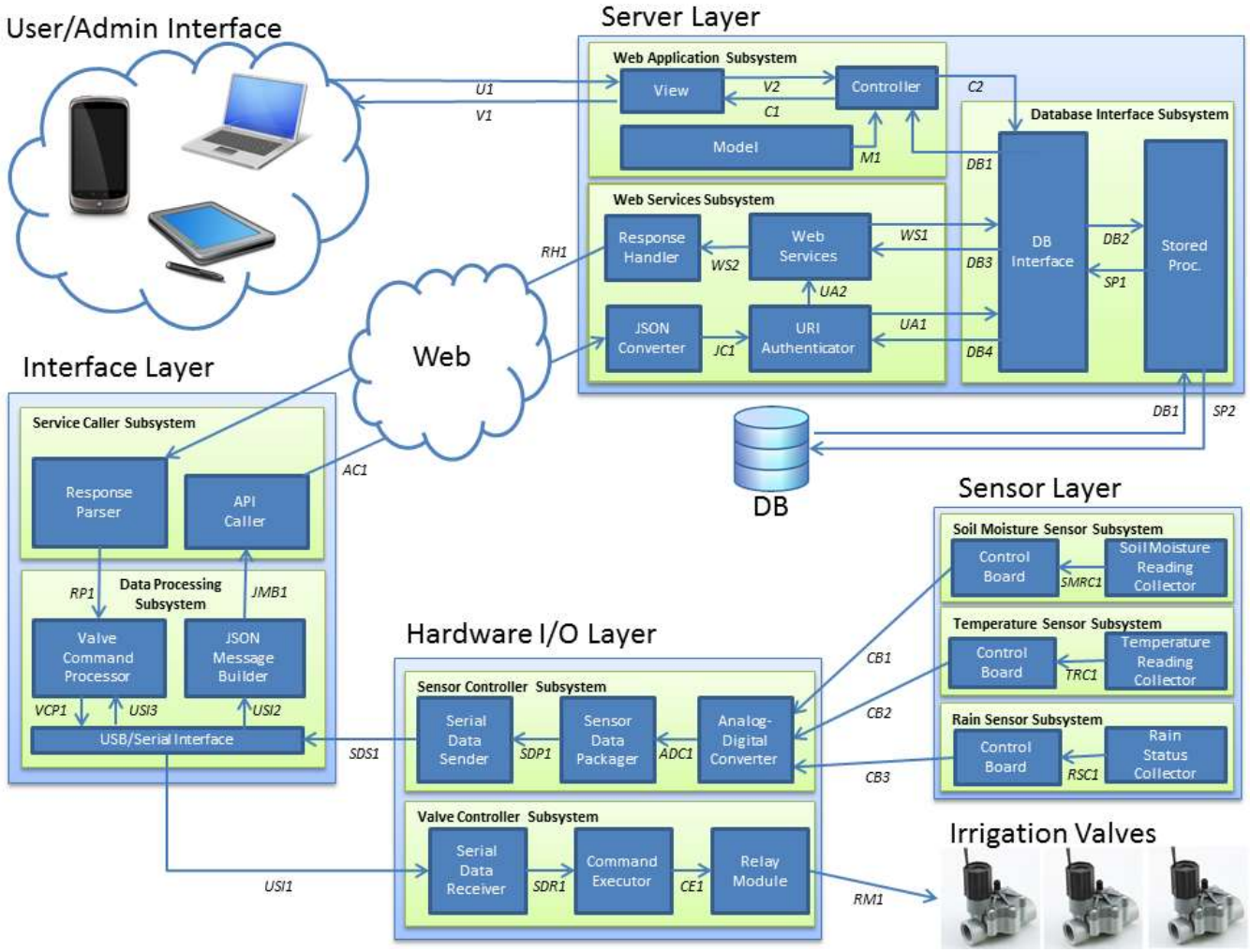

Figure 2-2 Module Decomposition Diagram

Sensor Layer

Rain Sensor Subsystem

2.5.1 – Rain Status Collector: The Rain Signal Collector is responsible for detecting the presence of rain. If the collector collects enough precipitation, the signal collector transmits an alert signal to the rain sensor Control Board module.

2.5.2 – Control Board: The rain sensor Control Board is responsible interpreting the analog voltage signal that transmits from the rain collector. Once the signal has been interpreted it is relayed to the ADC module.

Temperature Sensor Subsystem

2.5.3 – Temperature Reading Collector: The Temperature Reading Collector reads the temperature of the environment and reports the signal to the temperature sensor Control Board module.

2.5.4 – Control Board: The temperature sensor Control Board interprets the analog signals being produced by the temperature sensor and reports the information to the ADC module.

Soil Moisture Sensor Subsystem

2.5.5 – Soil Moisture Reading Collector: The Soil Moisture Signal Collector reads the amounts of moisture present in the soil and sends the signal to the soil moisture sensor Control Board module.

2.5.6 – Control Board: The soil moisture sensor Control Board is responsible for interpreting the analog signals being produced its corresponding soil moisture sensor and reports the information to the ADC module.

Hardware I/O Layer

Sensor Controller Subsystem

2.5.7 – Analog-Digital Converter: This module collects all the readings from the sensors and converts their signals from analog to digital so that the information can be further processed and packed.

2.5.8 – Serial Data Packager: This Serial Data Packager module is responsible for converting and packaging the raw sensor readings into readable data.

2.5.9 – Serial Data Sender: This module sends the collected sensor data over a USB cable to the Raspberry pi. This connection is the primary link between the Arduino and Raspberry Pi.

Valve Controller Subsystem

2.5.10 – Serial Data Receiver: This module receives the serial data sent by USB/Serial Interface and converts it into usable command data.

2.5.11 – Command Executer: This module executes the command that was received from the Serial Data Receiver module to the relay module.

2.5.12 – Relay Module: The Relay Module converts the command from the Command Executer to an electronic signal that will toggle the irrigation valve(s) on or off.

Interface Layer

Data Processing Subsystem

2.5.13 – USB/ Serial Interface: This module initiates the connection between the Interface Layer and the Hardware I/O layer. The connection and communication between the layers is done over a USB cable.

2.5.14 – JSON Message Builder: The JSON Message Builder module converts the data received from the USB/Serial Interface into a JSON Object and formats the URI request as well.

2.5.15 – Valve Command Processor: This Valve Command Processor module parses the JSON object to determine what command is being relayed. The module also maps the command to a serial data command that is to be transferred to Valve Controller Subsystem through the USB/Serial Interface.

Service Caller Subsystem

2.5.16 – API Caller: This module builds the HTTP request and makes the appropriate call to the HICS API. The JSON sensor readings object is attached to the request in the body and the HICS system identifier is attached in the parameters.

2.5.17 – Response Parser: This module parses the HTTP response sent back by the HICS API and creates a JSON object from the body that is sent to the Valve Command Processor.

Server Layer

Web Services Subsystem

2.5.18 – JSON Converter: This module converts the JSON data from the HTTP request into corresponding domain objects that the web services will understand. It also parses out the HICS system id from the request parameters for authorization.

2.5.19 – URI Authenticator: The URI Authenticator module is responsible for authenticating the HICS system id that was attached to the API call. If authorization is successful if routes the call to the appropriate Web Services method.

2.5.20 – Web Services: The Web Services module contains all the API methods needed for the Service Caller Subsystem to communicate back and forth between the database.

2.5.21 – Response Handler: This module takes the updated command state or status code and relays it to back to the system that initiated the request. It returns the data or the

Web Application Subsystem

2.5.22 – Model: This module is defines the blueprint for the database tables. It provides a new model or loads a model with existing data and set rules on how that data is accessed.

2.5.23 – Controller: The Controller module handles all the dataflow between the HTML views and the database. It is also responsible for loading views and handling all page routings.

2.5.24 – View/UI: This module is the graphical interface that the user interacts with to control or view the status of their HICS system.

Database Interface Subsystem

2.5.25 – DB Interface: The DB Interface module is the interface that is responsible for retrieving and storing data in that database. The module contains all the domain services that handle the logic for the CRUD functionality of each database table.

2.5.26 – Stored Procedures: This module acts like a middleman for communication with the database and is responsible for authenticating pre-requesting queries. The definitions inside the module are made to ensure the security of the data being communicated to and from the database.

2.6 Module Producer Consumer Matrix

The table shown in Table 2-3 shows the data flow between each individual module. Some modules are less complex have only a single association with another module. For example, Model has data going to only controller, so the data is only flowing one way. Other modules such as the DB Interface are associated with a number of different modules and are therefore more complex. In general the more data flows that interact with a module the more complex and important that module is.

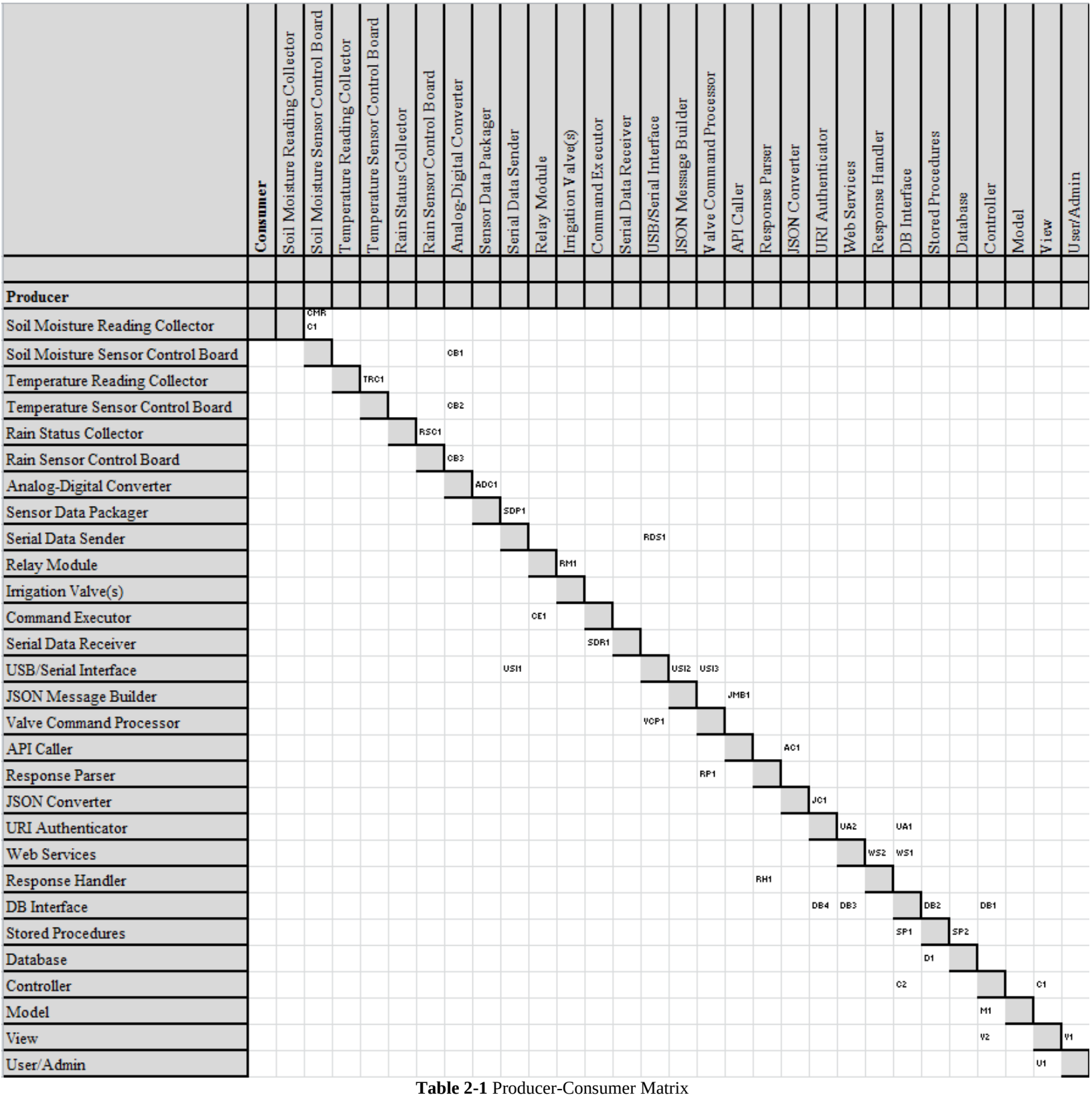

Table 2-1 Producer-Consumer Matrix

3. System Hardware Description

3.1 Raspberry Pi

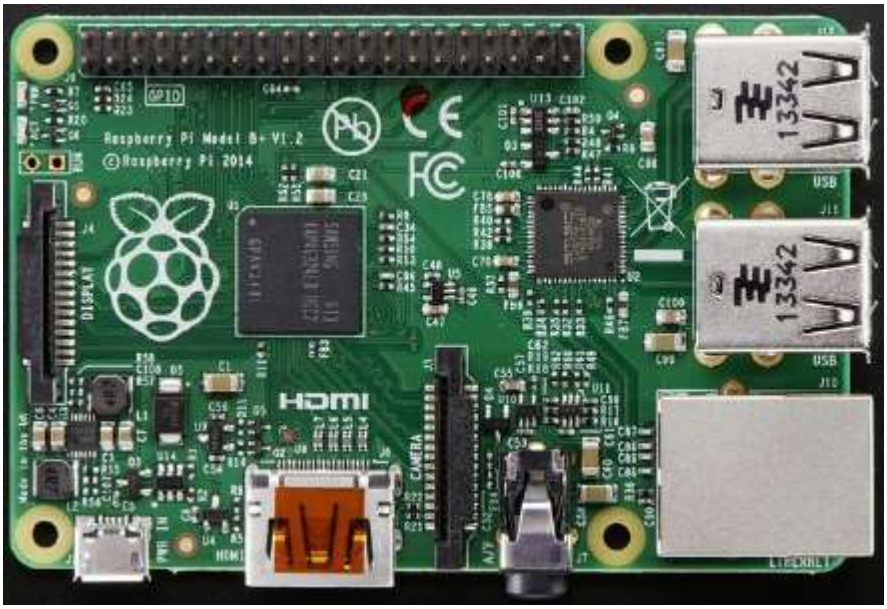

Figure 3-1 Raspberry Pi

3.1.1 – Purpose: The Raspberry Pi will be at the core of HICS acting as the central communication unit. The Raspberry Pi will be responsible for the communication between the hardware components to the web application.

3.1.2 – Quantity: HICS will require a single Raspberry Pi microcontroller.

3.1.3 – Interface: The Raspberry Pi will connect to the Internet through Ethernet port RJ45 for a stable connection. This connection is used to handle communication between the Raspberry Pi and the HICS API. The Pi will also connect to the Arduino microcontroller through a USB cable to send out the commands to control the irrigation valves in addition to receiver the sensor readings.

3.1.4 – Specifications:

Model	Raspberry Pi B+
SoC	Broadcom BCM2835
CPU	700 MHz ARM1176JZF -S core
GPU	250 MHz Broadcom IV
RAM	512
USB	4x 2.0 Onboard USB ports
Video Out	HDMI up to 1920x1200 resolution, PAL, NTSC
Audio Out	3.5mm Jack, HDMI
Storage	Onboard SD/MM/SDIO card slot
Networking	10/100mbps Ethernet, RJ-45
Peripherals	GPIO, UART, SPI, IIC +3.3V, +5.0V
Power	700 mA, 5V via MicroUSB or GPIO header
OS	Raspbian

Table 3-1 Raspberry Pi Specifications

3.2 Arduino Mega 2560

Figure 3-2 Arduino Mega

3.2.1 – Purpose: In the HICS system the Arduino Mega acts as the primary controller unit. The Arduino Mega will connect the soil moisture sensors, temperature sensor, and rain sensor. The Arduino Mega will also execute commands to turn on and turn off the irrigation valves. With 16 analog ports and 54 digital ports, the Arduino will handle all of the analog inputs, as well as the digital output and inputs.

3.2.2 – Quantity: Because the HICS system must handle up to 24 soil moisture sensors and 24 irrigation valves, it will require two Arduino Mega microcontrollers.

3.2.3 – Interfaces: Arduino Mega will physically connect to the soil moisture sensors and temperature sensor to collect the analog signals from the sensors. It will also connect directly to the rain sensor to collect its signal. The Arduino Mega will also interface with the Relay Board to control the operation of irrigation valve(s).

3.2.4 – Specifications:

Model	**Arduino Mega ATmega1289**
Analog Input Pin	16
Digital I/O Pins	54 (of which 14 provide PWM output)
SRAM	8K
EEPROM	4KB
Clock Speed	16MHz
UART Port	3
I2C	Port
USB port	1
Operating Voltage	5V

Table 3-2 Arduino Mega Specifications

3.3 Relay Switch Module

Figure 3-3 Relay Switch Module

3.3.1 – Purpose: Relay switches are electrical switches controlled by other switches by allowing a small current flow circuit to control a higher current circuit. For HICS, the relays will be used to control the irrigation valves that need to be turned on and off through a physical switch. Using 3.3V from the Arduino Mega's I/O pins, the relay will control the main power operating up to 250V.

3.3.2 – Quantity: HICS will require 3 single 8-channel relay modules that add up 24 channel controls.

3.3.3 – Interfaces: The Relay switch modules will interface directly with the Arduino to receive control command. They will also be hardwired to the irrigation valves to turn them on and off.

3.3.4 – Specifications:

Model	SaintSmart 8-Channel Relay
Operating Voltage	5V
Operating Current	20mA
Relay Current/Voltage	AC250V 10A, DC 30V 10A
Interface	Arduino Raspberry Pi, ARM, PIC, DSP

Table 3-3 Relay Switch Specifications

3.4 Soil Moisture Sensors

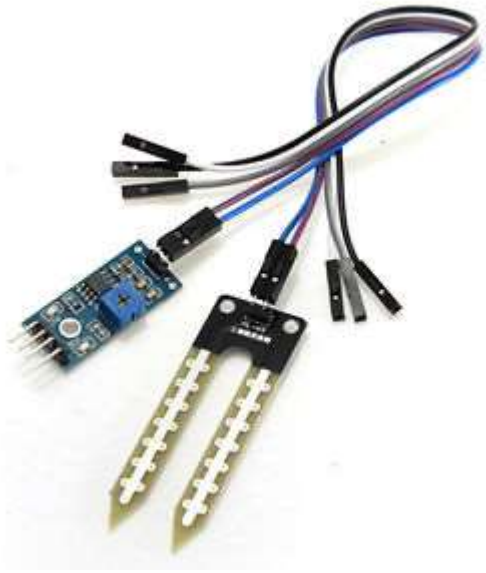

Figure 3-4 Soil Moisture Sensors

3.4.1 – Purpose: The soil moisture sensors will collect soil moisture levels in the ground. Based on the readings from soil moisture sensors, HICS will decide to turn on or turn off the irrigation system.

3.4.2 – Quantity: HICS will require 24 soil moisture sensors that are responding to 24 irrigation valves.

3.4.3 – Interfaces: The soil moisture sensors will physically interface to the Arduino Mega, which will collect the analog inputs from the sensors.

3.4.4 – Specifications:

Model	TOOGOO Soil Moisture Sensor
Operating Voltage	3.3V-5V
Digital Output	Yes
Analog Output	Yes
Adjustable Sensitivity	Yes

Table 3-4 Soil Moisture Sensor Specifications

3.5 Temperature Sensor

Figure 3-5 Temperature Sensor

3.5.1 – Purpose: The temperature sensor will collect the current temperature levels at the location where HICS is installed. Based on these readings from the temperature sensor, HICS will decide to turn on or turn off the irrigation valve(s) due to freezing conditions.

3.5.2 – Quantity: HICS will require a single temperature sensor.

3.5.3 – Interfaces: The temperature sensor will physically interface to the Arduino Mega, which will collect the analog inputs from the sensor.

3.5.4 – Specifications:

Model	DS18B20 Temperature Sensor
Operating Voltage	3.3V-5V
Analog Output	Yes
Adjustable Resolution	Yes

Table 3-5 Temperature Sensor Specifications

3.6 Rain Sensor

Figure 3-6 Rain Sensor

3.6.1 – Purpose: The rain sensor will detect rain conditions at the location where HICS is installed. Based on the signals transmitted from the rain sensor, HICS will decide to turn on or turn off the irrigation system due to rain conditions.

3.6.2 – Quantity: HICS will require a single rain sensor.

3.6.3 – Interfaces: The rain sensor will physically interface to Arduino Mega, which will collect the digital input from the sensor.

3.6.4 – Specifications:

Model	DS18B20 Temperature Sensor
Operating Voltage	3.3V-5V
Digital Output	Yes
Adjustable Resolution	Yes

Table 3-6 Rain Sensor Specifications

3.7 AC Power Module

Figure 3-7 AC Adaptor for Raspberry Pi

3.7.1 – Purpose: The AC adaptor will supply the Raspberry Pi with 5 volts of power at 2.5 amperes.

3.7.2 – Quantity: HICS will require a single AC adapter for the Raspberry Pi.

3.7.3 – Interfaces: The AC adaptor will connect to the Raspberry Pi via its micro USB port.

3.7.4 – Specifications:

Model	CanaKit 2.5A Micro USB Power Supply
Voltage	5V
Current	2.5A

Table 3-7 AC Power Specifications

4. System Software Description

This section details the software technologies that make up the HICS web application and control unit software. The two main software components are web technologies and low level hardware programming. Each peice of software in HICS functions independently of one another yet each relies heavily on the other to function properly. An individual breakdown of each of these software components is explained in more detail below.

4.1 HICS Web Application

The HICS web application will be built using ASP.NET MVC 5 running on the .NET Framework 4.5. This environment allows us utilize Visual Studios to easily manage all files and external libraries for the application. Our HTML views will be built and rendered using the Razor View Engine 3.0. The other frontend components will be designed extensively using the jQuery and Twitter Bootstrap libraries for styling and functionality. The controller side of the application, along with the API (web services), will be created using C#. The DB interface backend of the web application will operate using Entity Framework 6 which we will use to utilize and define our database context and setup repository services for each domain class. Entity Framework 6 will also be used handle all the mapping between the domain relationships that are defined in database tables.

4.2 HICS Hardware Software

The software that will be running on the hardware, in HICS, is specific to each microcontroller. The Raspberry Pi will be running an offshoot OS of Linux called Rasbian. The Rasbian OS supports the Processing 2.2.1 programming language which will be used for the data packaging and API calls from the Raspberry Pi. The other microcontroller we will be using will be an Arduino. The Arduino will not run a specific OS but will instead be loaded with software developed in Arduino’s native C based language to run as soon as the unit receives power.

5. Sensor Layer

The purpose of the Sensor Layer is to gather data from the sensors to monitor different environment conditions. The Sensor Layer consists of a Rain Sensor, a Temperature Sensor, and Soil Moisture Sensors. For each kind of sensor there will be a subsystem responsible for collecting the different environment conditions.

5.1 Rain Sensor Subsystem

The Rain Sensor Subsystem will detect precipitation amounts by measuring the current rainfall. This sensor will interface with the Sensor Controller Subsystem and will send an alert signal to stop all watering cycles if it detects rain.

5.1.1 – Rain Status Collector

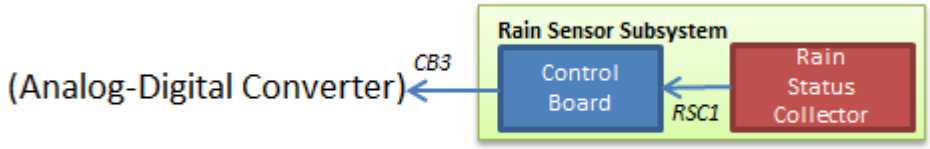

Figure 5-1 Rain Status Collector Module

Prologue

The Rain Status Collector module will be a sensor whose circuitry will measure whether or not it is being exposed to rain. It will report this information to the Control Board module when a specific amount of precipitation has been collected.

Interfaces

Source	Sink	Input to Sink	Return from Sink
Rain Status Collector	Control Board	Analog voltage	None

Table 5-1 Rain Status Collector Interfaces

External Data Dependencies

- Alert signal depends on a certain amount of precipitation to be collected

Internal Data Dependencies

None

Pseudo Code

N/A

5.1.2 – Control Board

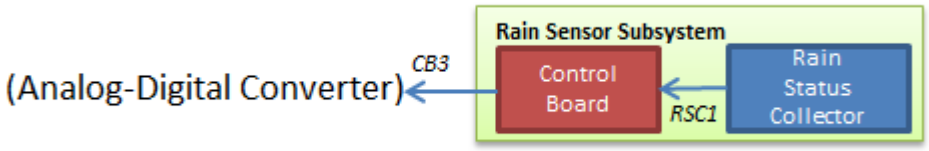

Figure 5-2 Rain Control Board Module

Prologue

The Control Board module will interpret the analog voltage coming from the Rain Status Collector Module and will pass it on to the Analog-Digital Converter Module in the Sensor Controller Subsystem. It will convey this data as both an analog value between 0V and 5V, and as a high/low digital value. This module has on-board led status indicators and a potentiometer for adjusting the sensitivity.

Interfaces

Source	Sink	Input to Sink	Return from Sink
Rain Status Collector	Control Board	Analog voltage	None
Control Board	Analog-Digital Converter	Analog voltage	None

Table 5-2 Rain Control Board Interfaces

External Data Dependencies

None

Internal Data Dependencies

An onboard potentiometer must be used to adjust the sensitivity.

Pseudo Code

N/A

5.2 Temperature Sensor Subsystem

The Temperature Sensor Subsystem will measure the current temperature of the environment where the HICS unit is installed. This sensor will interface with the Sensor Controller to send data about the current temperature in the area.

5.2.1 – Temperature Reading Collector

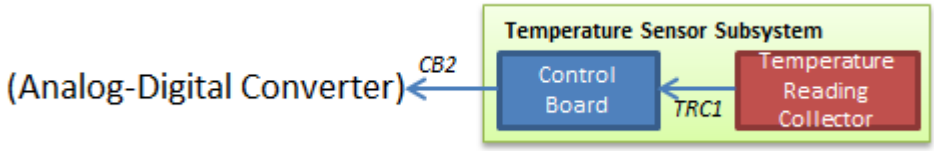

Figure 5-3 Temperature Reading Collector Module

Prologue

The Temperature Reading Collector Module will be a sensor whose circuitry will measure the temperature of its environment. It will report this information to the Control Board Module.

Interfaces

Source	Sink	Input to Sink	Return from Sink
Temperature Reading Collector	Control Board	Analog voltage	None

Table 5-3 Temperature Reading Collector Interfaces

External Data Dependencies

- Placement of the sensor will affect the readings accuracy

Internal Data Dependencies

None

Pseudo Code

N/A

5.2.2 – Control Board

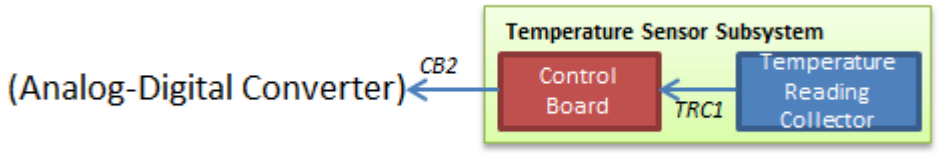

Figure 5-4 Temperature Control Board Module

Prologue

The Control Board Module will interpret the analog voltage coming from the Temperature Reading Collector Module and will pass it on to the Analog-Digital Converter Module in the Sensor Controller Subsystem. It will convey this data as both an analog value between 0V and 5V, and as a high/low digital value. This module has on-board led status indicators and a potentiometer for adjusting the sensitivity.

Interfaces

Source	Sink	Input to Sink	Return from Sink
Temperature Reading Collector	Control Board	Analog voltage	None
Control Board	Analog-Digital Converter	Analog voltage	None

Table 5-4 Temperature Control Board Interfaces

External Data Dependencies

None

Internal Data Dependencies

An onboard potentiometer must be used to adjust the sensitivity.

Pseudo Code

N/A

5.3 Soil Moisture Sensor Subsystem

The Soil Moisture Sensor(s) will measure the current soil moisture levels at its corresponding irrigating zone. Each sensor will interface with the Sensor Controller and send data consisting of the current soil moisture levels. Each Soil Moisture Sensor is associated with its own irrigation valve.

5.3.1 – Soil Moisture Reading Collector

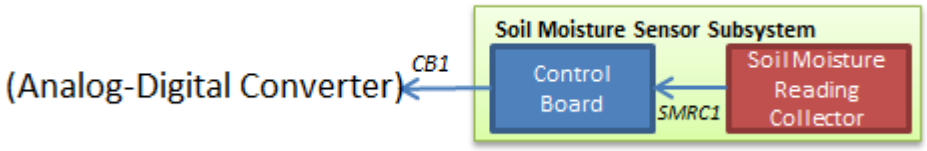

Figure 5-5 Soil Moisture Reading Collector Module

Prologue

The Soil Moisture Reading Collector Module will be a sensor whose circuitry will measure the moisture levels of its environment. Its intended use is to be placed within the user's soil to measure its moisture content. This information will be reported to the Control Board Module.

Interfaces

Source	Sink	Input to Sink	Return from Sink
Soil Moisture Reading Collector	Control Board	Analog voltage	None

Table 5-5 Soil Moisture Reading Collector Interfaces

External Data Dependencies

- Proper placement of the sensor in the soil

Internal Data Dependencies

None

Pseudo Code

N/A

5.3.2 – Control Board

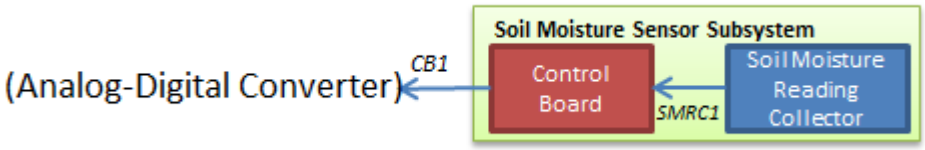

Figure 5-6 Soil Moisture Control Board Module

Prologue

The Control Board Module will interpret the analog voltage coming from the Soil Moisture Reading Collector Module and will pass it on to the Analog-Digital Converter Module in the Sensor Controller Subsystem. It will convey this data as both an analog value between 0V and 5V, and as a high/low digital value. This module has on-board led status indicators and a potentiometer for adjusting the sensitivity.

Interfaces

Source	Sink	Input to Sink	Return from Sink
Soil Moisture Reading Collector	Control Board	Analog voltage	None
Control Board	Analog-Digital Converter	Analog voltage	None

Table 5-6 Soil Moisture Control Board Interfaces

External Data Dependencies

None

Internal Data Dependencies

An onboard potentiometer must be used to adjust the sensitivity.

Pseudo Code

N/A

6. Hardware I/O Layer

The Hardware I/O Layer collects information from the environment sensors and also controls the activities of the irrigation valve(s). The Hardware I/O Layer includes a Sensor Controller Subsystem and Valve Controller Subsystem.

6.1 Sensor Controller Subsystem

The Sensor Controller Subsystem will interface directly with the Rain Sensor Subsystem, Soil Moisture Sensor Subsystem, and Temperature Sensor Subsystem. The main component of the Sensor Controller Subsystem is an Arduino microcontroller that will read in the signal transmitted from the sensors. Once the information has been read in from the sensors, the Sensor Controller Subsystem will pass this information along to the USB/Serial Interface Module in the Data Processing Subsystem.

6.1.1 – Analog-Digital Converter

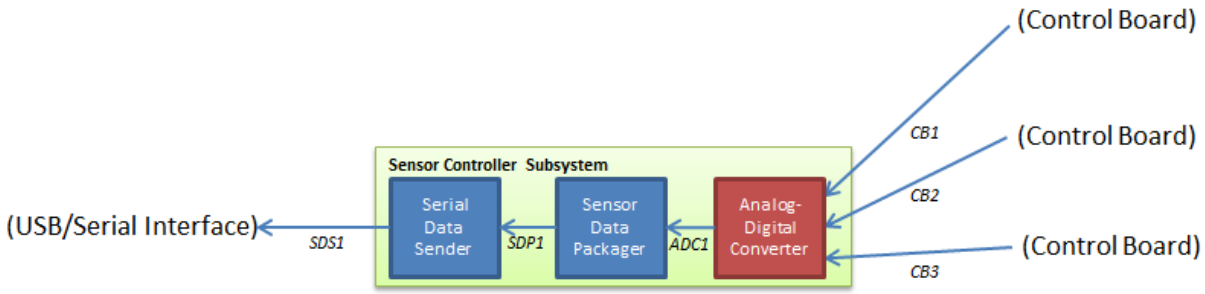

Figure 6-1 Analog-Digital Converter Module

Prologue

The Analog-Digital Converter is a mechanism onboard the Arduino for reading in analog signals and representing them as digital values in a program. The Arduino will read in the analog inputs from the various Control Board Interface(s) and will store them as local variables within its memory.

Interfaces

Source	Sink	Input to Sink	Return from Sink
Control Board(s)	Analog-Digital Converter	Analog voltage	None
Analog-Digital Converter	Sensor Data Packager	Float/double values for sensor readings	None

Table 6-1 Analog-Digital Converter Interfaces

External Data Dependencies

- Physical connection from each sensor to the Arduino
- Arduino Programming Language standard libraries

Internal Data Dependencies

None

Pseudo Code

```
/*
Reads an analog input on pin 0, converts it to voltage, and prints the result to the serial monitor.
Attach the center pin of a potentiometer to pin A0, and the outside pins to +5V and ground.
*/
// the setup routine runs once when you press reset:
void setup() {
	// initialize serial communication at 9600 bits per second:
	Serial.begin(9600);
}
void loop() {
	// read the input on analog pin 0:
	int sensorValue = analogRead(A0);

	// Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):
	float voltage = sensorValue * (5.0 / 1023.0);
}
```

6.1.2 – Sensor Data Packager

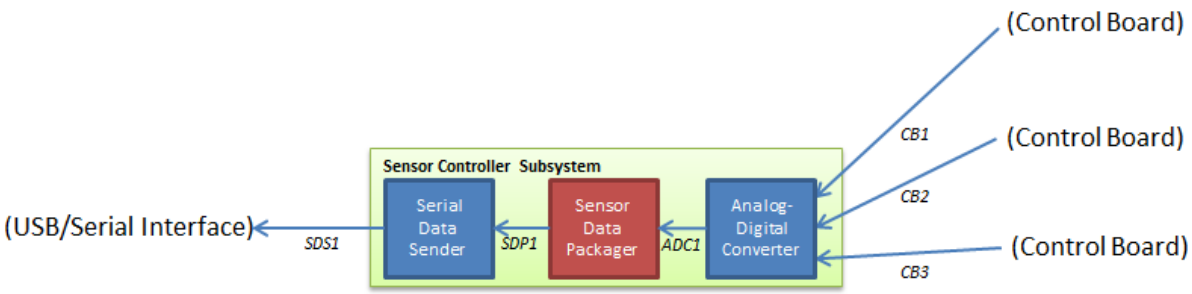

Figure 6-2 Sensor Data Packager Module

Prologue

The Sensor Data Packager Module will convert a piece of floating point data into a string of information. This is done so that the data can be sent over a USB/Serial Interface by the Serial Data Sender Module.

Interfaces

Source	Sink	Input to Sink	Return from Sink
Analog-Digital Converter	Sensor Data Packager	Float/double values for sensor readings	None
Sensor Data Packager	Serial Data Sender	String of sensor reading values	None

Table 6-2 Sensor Data Packager Interfaces

External Data Dependencies

- Arduino Programming Language standard libraries

Internal Data Dependencies

None

Pseudo Code

```
void setup() {
        // initialize serial communication at 9600 bits per second:
        Serial.begin(9600);
}

void loop() {
        String message = “Sensor 1:”;
        message = message + voltage1;
        String message = “Sensor 2:”;
        message = message + voltage2;

        ...
}
```

6.1.3 – Serial Data Sender

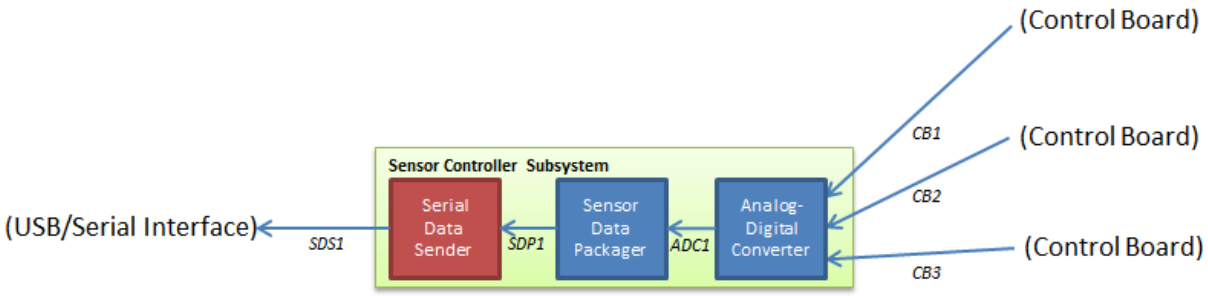

Figure 6-3 Serial Data Sender Module

Prologue

The Serial Data Sender Module is responsible for writing arrays of data onto the serial port. This data is then read by the USB/Serial Interface Module in the Interface Layer. The physical connection for this communication is a USB 2.0 cable.

Interfaces

Source	Sink	Input to Sink	Return from Sink
Sensor Data Packager	Serial Data Sender	String of sensor reading values	None
Serial Data Sender	USB/Serial Interface	String of sensor reading values	None

Table 6-3 Serial Data Sender Interfaces

External Data Dependencies

- USB Connection must be established between the Arduino and Raspberry Pi
- Arduino Programming Language standard libraries

Internal Data Dependencies

None

Pseudo Code

```
void setup() {
        Serial.begin(9600);
}

void loop() {
        Serial.write(45); // send a byte with the value 45
        //send the string “hello” and return the length of the string.
        int bytesSent = Serial.write(“hello”);
}
```

6.2 Valve Controller Subsystem

The Valve Controller Subsystem will interface directly with the irrigation valve(s). It will also communicate with the Data Processor Subsystem to receive control commands to turn the irrigation valve(s) on or off. The main component of the Valve Controller is an Arduino microcontroller, which will receive the commands from the Data Processing Subsystem and use them to control the operation of irrigation valve(s).

6.2.1 – Relay Module

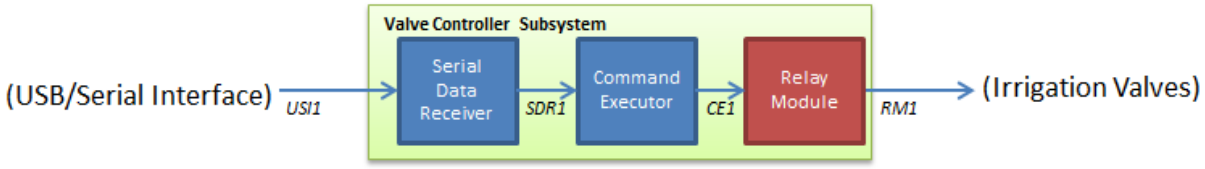

Figure 6-4 Relay Module

Prologue

The Relay Module will take commands from the Arduino and convert them into electronic signals that can interface with large, real world devices. In HICS, the relay modules will interface with solenoid irrigation valve(s).

Interfaces

Source	Sink	Input to Sink	Return from Sink
Command Executor	Relay Module	0V(Low) or 5V(High)	None
Relay Module	Irrigation Valve(s)	0V(off) or 24V(on)	None

Table 6-4 Relay Module Interfaces

External Data Dependencies

- Irrigation Valves must be properly connected to the Arduino unit

Internal Data Dependencies

None

Pseudo Code

N/A

6.2.2 – Command Executer

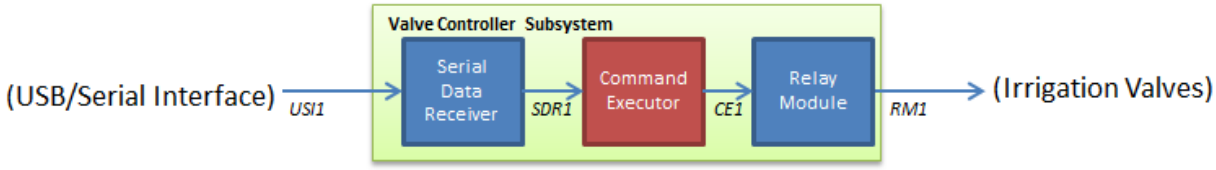

Figure 6-5 Command Executor Module

Prologue

The Command Executer Module will set the digital pins on the Arduino to be either High or Low, depending on the specific command.

Interfaces

Source	Sink	Input to Sink	Return from Sink
Serial Data Receiver	Command Executor	Valve/pin number and boolean value	None
Command Executor	Relay Module	0V(Low) or 5V(High)	None

Table 6-5 Command Executer Interfaces

External Data Dependencies

- Arduino Programming Language standard libraries

Internal Data Dependencies

None

Pseudo Code

```
// Valve connected to digital pin 13
int valvePin = 13;
void setup() {
        pinMode(valvePin, OUTPUT);     // sets the digital pin as output
}

void loop() {
        // sets the Valve on
        digitalWrite(valvePin, HIGH);
        // waits for a second
        delay(1000);
        // sets the LED off
        digitalWrite(valvePin, LOW);
        // waits for a second
        delay(1000);
}
```

6.2.3 – Serial Data Receiver

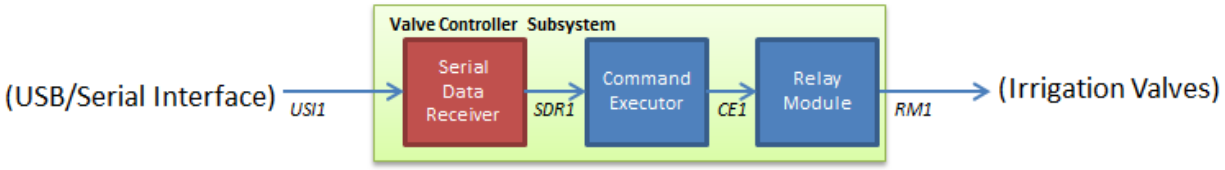

Figure 6-6 Serial Data Receiver Module

Prologue

The Serial Data Receiver Module reads data from the serial port and converts it into a string. Additionally, this module is responsible for parsing the valve number and open/closed state information from this string and passing that information to the Command Executer Module.

Interfaces

Source	Sink	Input to Sink	Return from Sink
USB/Serial Interface	Serial Data Receiver	Byte (to be read in as a String)	None
Serial Data Receiver	Command Executor	Valve/pin number and boolean value	None

Table 6-6 Serial Data Receiver Interfaces

External Data Dependencies

- USB Connection must be established between the Arduino and Raspberry Pi
- Arduino Programming Language standard libraries

Internal Data Dependencies

None

Pseudo Code

```
int incomingByte = 0;   // for incoming serial data
void setup() {
        //opens serial port, sets data rate to 9600 bps
        Serial.begin(9600);
}

void loop() {
        while (Serial.available() > 0) {
                char recieved = Serial.read();
                inData += recieved;
                // Process message when new line character is received
                if (recieved == '\n') {
                        //parse inData
                        //send parsed information to command Executor

                        // Clear recieved buffer
                        inData = "";
                }
        }
}
```

7. Interface Layer

The purpose of the Interface Layer is to process and relay all communication between the Hardware I/O Layer and Server Layer to report sensor readings and issue control commands. It consists of two main subsystems—the Service Caller Subsystem and the Data Processing Subsystem. The design for each of the subsystems is explained in detail below.

7.1 Service Caller Subsystem

The Service Caller Subsystem is responsible for sending sensor readings of the user's home irrigation system and receiving the command responses from the Web Service Subsystem. In the Service Caller Subsystem the sensor information is packaged as a JSON message and relayed to a HICS API method. When the Web Service Subsystem receives the update information, it will store the readings and respond with command changes from the previous request. The returned command information will be passed to Data Processing Subsystem, which will handle how it is executed.

7.1.1 – Response Parser

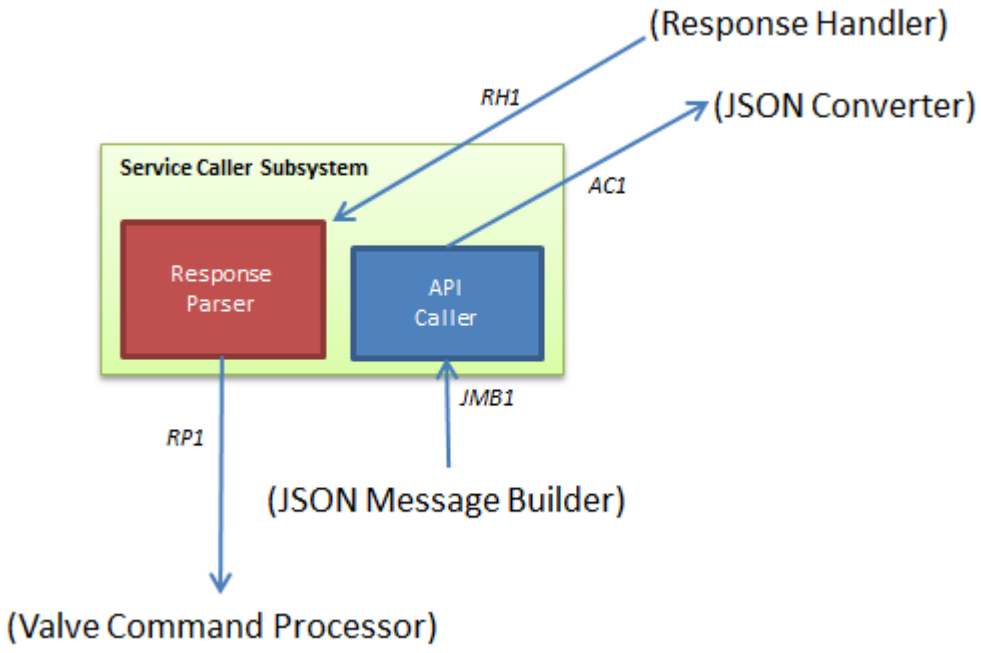

Figure 7-1 Response Parser Module

Prologue

The Response Parser Module will primarily be responsible for receiving the HTTP response from the Web Services Subsystem and determining what information gets sent to the Valve Command Processor Module. At this module the response body, represented by a JSON object, will be parsed and checked for commands. If the response only contains a status code then no information is output from the module. If the Response Parser Module finds any commands in the response the JSON object is then sent to the Valve Command Processor for processing.

Interfaces

Source	Sink	Input to Sink	Return from Sink
Response Parser	Valve Command Processor	JSON command object	None
Response Handler	Response Parser	JSON command object or status code	None

Table 7-1 Response Parser Interfaces

External Data Dependencies

- Processing 2.2.1+
- Processing Foundation libraries

Internal Data Dependencies

- HTTP response from the API

Pseudo Code

```
responseParser(obj  response) {
        // verify response
        If (response!=null){
                // parse JSON response
                JSONArray command = response.getJSONArray("command");
        }
        // if command is found relay message to the Valve Command Processing
        for(int i=0;i<valves.size();i++){
                valves[i]=command[i];
        }
}
```

7.1.2 – API Caller

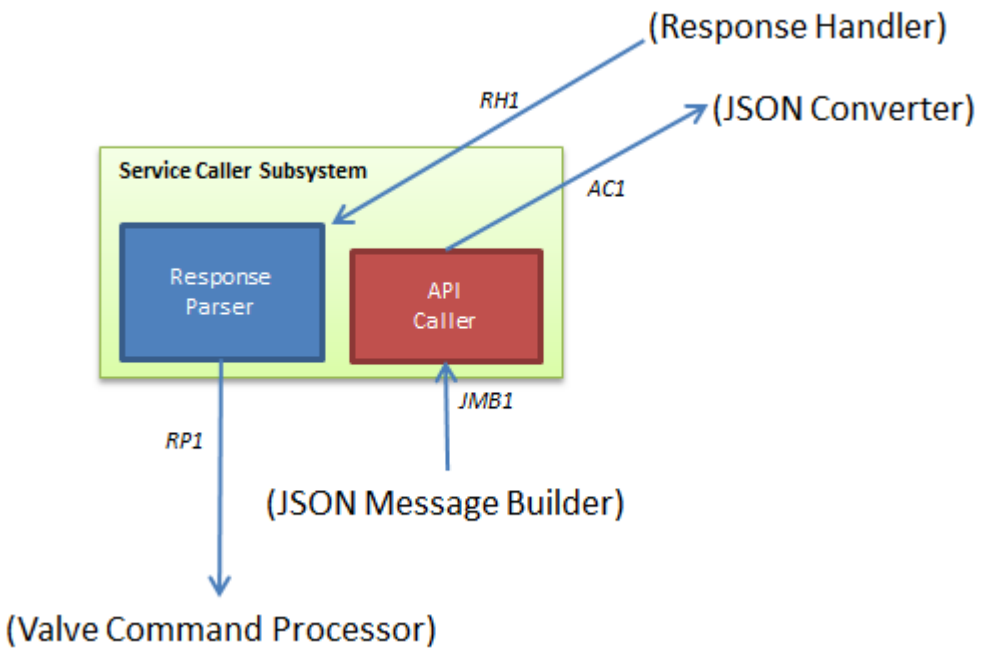

Figure 7-2 API Caller Module

Prologue

The API Caller will be responsible for formatting the information from the JSON Message Builder Module into a URI request with the message object attached as a parameter. The API Caller also executes the request call to send the request to the Web Service Subsystem. The call from API Caller Module will be made through an Internet connect to the HICS API.

Interfaces

Source	Sink	Input to Sink	Return from Sink
JSON Message Builder	API Caller	JSON object with sensor data	None
API Caller	JSON Converter	HTTP request	None

Table 7-2 API Caller Interfaces

External Data Dependencies

- Processing 2.2.1+
- Processing Foundation libraries

Internal Data Dependencies

- Predefined URIs for the HICS API methods

Pseudo Code

```
APIcaller(obj message) {
        // verify message
        If (message!=null)
        {
                // formatting message into an HTTP request
                GetRequest uri=”http://smartgrass.com/api/update”+message
        }
        // send the request to the HICS API
        uri.send()
}
```

7.2 Data Processing Subsystem

The Data Processing subsystem will be responsible for processing all the data that is communicated between the Web Services subsystem and the Hardware I/O subsystem. This system will receive update command from the Response Parser, also it will receive data sensor from I/O Hardware layer through USB connection. Then, data will be built as JSON Message before it is relayed to the Service caller subsystem.

7.2.1 – USB/Serial Interface

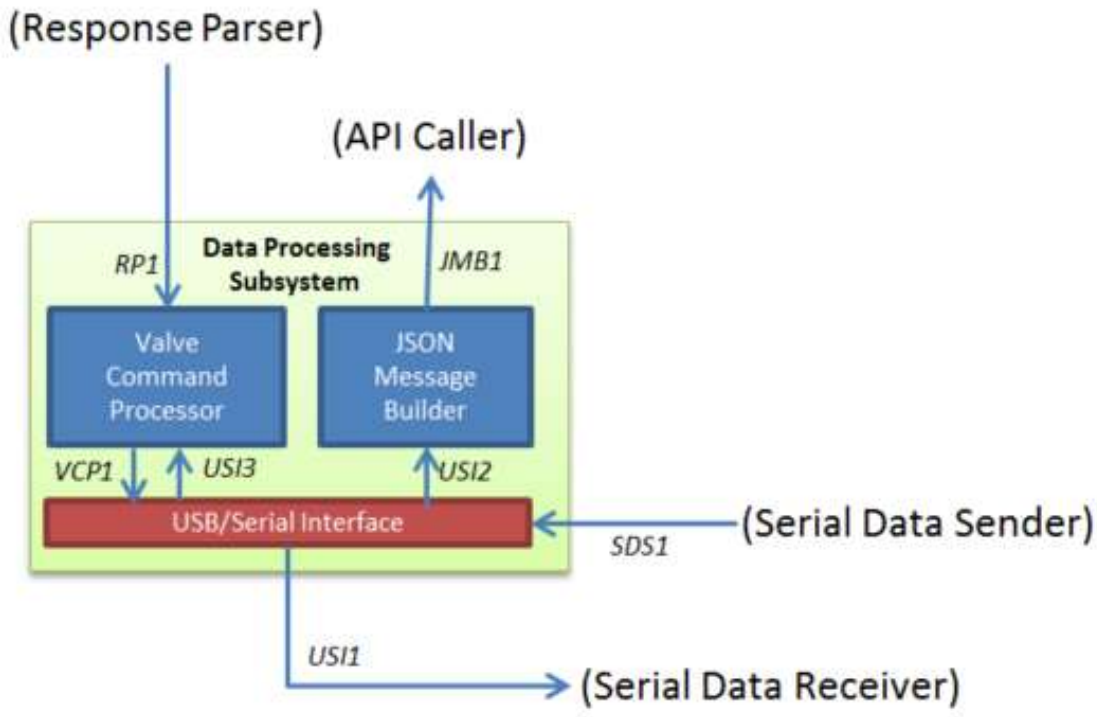

Figure 7-3 USB/Serial Interface Module

Prologue

The USB/Serial Interface will be responsible for handling communication between the Hardware I/O Layer and the Data Processing Subsystem. This module will initialize the connection between the microcontrollers and allow the data to be sent and received. The data being received will be serial data strings of sensor readings and the data being sent will be a serial data byte string that contains a control command for the irrigation valve(s).

Interfaces

Source	Sink	Input to Sink	Return from Sink
USB/Serial Interface	Serial Data Receiver	Serial data byte string with irrigation control command	None
Serial Data Sender	USB/Serial Interface	Serial data string with sensor readings	None
USB/Serial Interface	JSON Message Builder	Serial data string with sensor readings	None
USB/Serial Interface	Valve Command Processor	Temperature/Rain sensor data	Command to toggle irrigation valves on/off
Valve Command Processor	USB/Serial Interface	Serial data string with command data	None

Table 7-3 USB/Serial Interface Interfaces

External Data Dependencies

- Processing 2.2.1+
- Processing Foundation libraries
- USB 2.0 connection

Internal Data Dependencies

None

Pseudo Code

```
//USB/serial Interface
serialInterface() {
        //initialize connection between the Hardware I/O layer and the Interface layer
        port = new Serial(this, 9600);
        //allow sending and receiving data though this connection
        if (0 < port.available()) {    // If data is available to read,
                val = port.read();         // read it and store it in val
         }
}
```

7.2.2 – JSON Message Builder

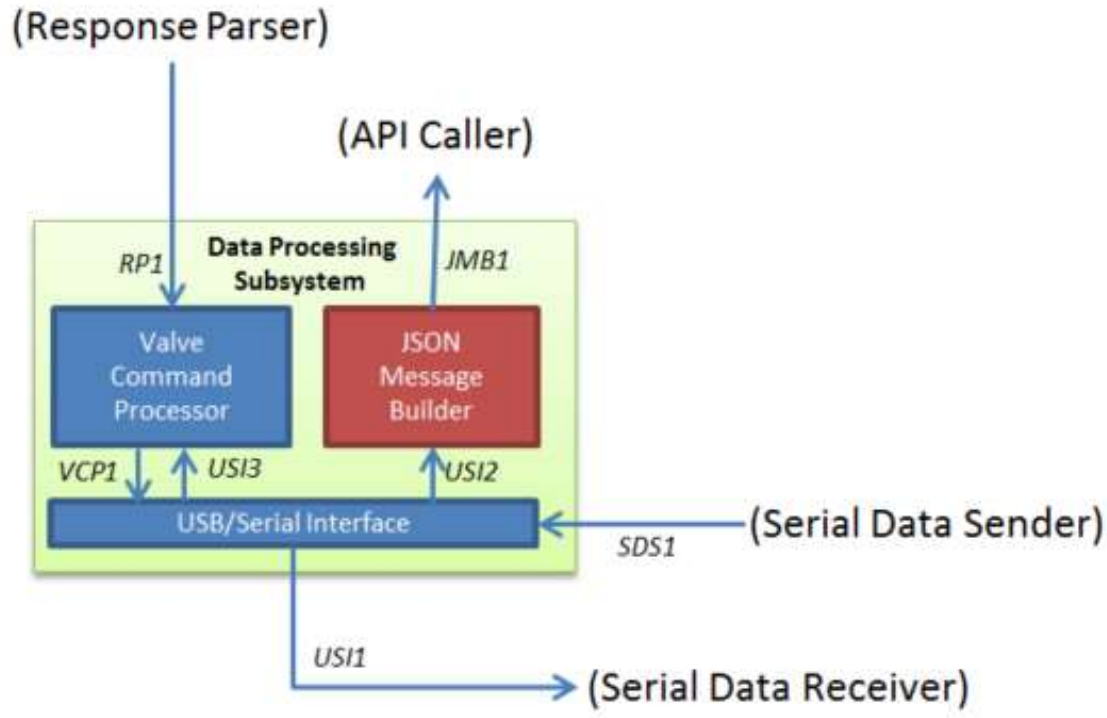

Figure 7-4 JSON Message Builder Module

Prologue

The JSON Message Builder Module will receive the sensor reading data that is passed through the USB/Serial Interface. At this module, a JSON message object is built to identify and represent all the sensor readings. This message will then be relayed to the API Caller Module to be sent to the HICS API.

Interfaces

Source	Sink	Input to Sink	Return from Sink
USB/Serial Interface	JSON Message Builder	Serial data string with sensor readings	None
JSON Message Builder	API Caller	JSON object	None

Table 7-4 JSON Message Builder Interfaces

External Data Dependencies

- Processing 2.2.1+
- Processing Foundation libraries

Internal Data Dependencies

- Data structure of sensor reading data

Pseudo Code

```
//JSON Message Builder
messageBuilder(obj  data) {
        //verify data
        If (data!=null){
                //formatting data into a JSON message
                message=data.JSONconvert();
        }
        //relay message to the API caller
                request=message;
}
```

7.2.3 – Valve Command Processor

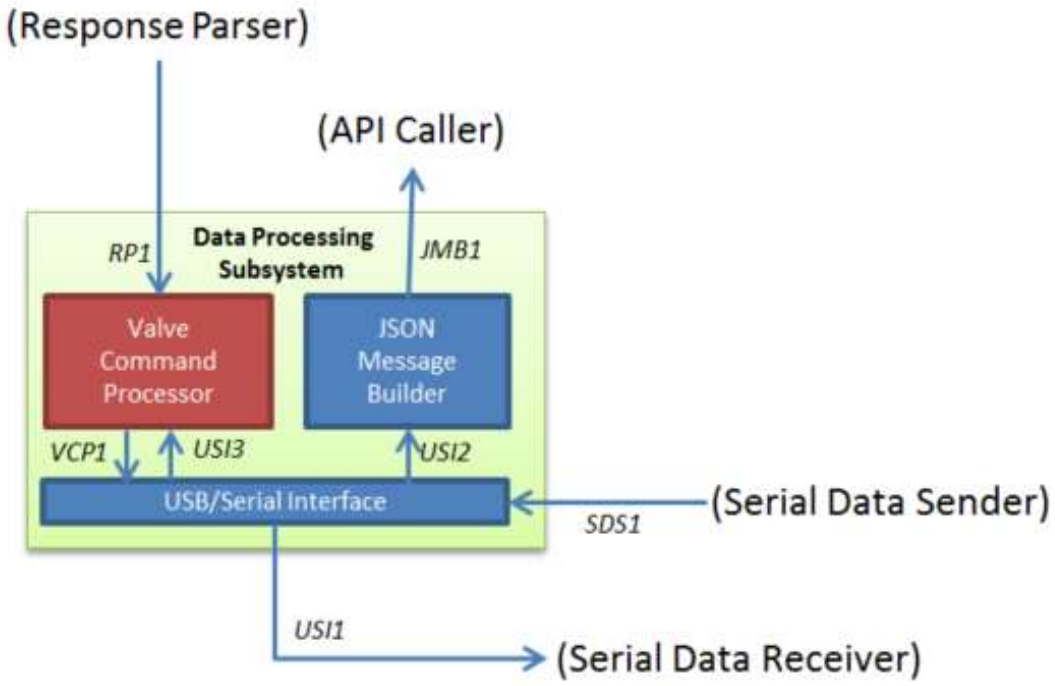

Figure 7-5 Valve Command Processor Module

Prologue

The Valve Command Processor Module will be response for receiving commands relayed from the HICS API and processing to determine what action to relay to the irrigation valve(s). The module must be able to determine what command is being sent and then translate that command into a serial data string by the Valve Controller Subsystem.

Interfaces

Source	Sink	Input to Sink	Return from Sink
Response Parser	Valve Command Processor	JSON command object	None
Valve Command Processor	USB/Serial Interface	Serial data string with command data	None
USB/Serial Interface	Valve Command Processor	Temperature/Rain sensor data	Command to toggle irrigation valves on/off

Table 7-5 Valve Command Processor Interfaces

External Data Dependencies

- Processing 2.2.1+
- Processing Foundation libraries

Internal Data Dependencies

- JSON formatted object with command information

Pseudo Code

```
//Valve Command Processor
commandProcessor(obj command) {
        //verify command
        If (command!=null){
                //passing command to the Valve Controller subsystem
                serial.write(command);

        }
}
```

8. Server Layer

The Server Layer serves at the main hub through which the user interacts with their HICS system. In addition to interfacing with the user the Server Layer is also responsible for storing and relaying data to and from every HICS unit. This dataflow is done through the HICS API which is represented by the Web Services Subsystem inside the Server Layer. The communication flow is primarily reliant on the Database Interface Subsystem that resides inside the layer.

8.1 Web Application Subsystem

The Web Application Subsystem is responsible for handling all the I/O that is provided to and from the user. The subsystem consists of a UI, a controller, and a model whose purpose is to relay data from the database to the user and vice versa. The View/UI, Model, and Controller Modules are each described more in detail below.

8.1.1 – View/UI

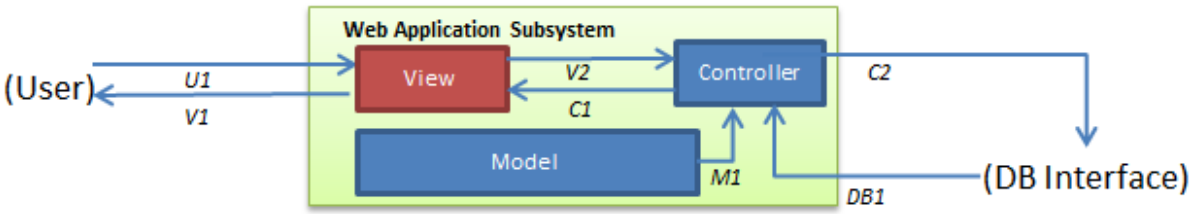

Figure 8-1 View/UI Module

Prologue

The View/UI Module is the front facing side of the Web Application that the user interfaces with directly. The module consists of HTML web pages and scripts that are used to show the user information about their HICS system as well as capture data that the user inputs. Data input is captured in the form of web forms in which the user will fill out specific information or click certain targets that will then submit the form to the controller for processing.

Interfaces

Source	Sink	Input to Sink	Return from Sink
User/Admin	View/UI	Keyboard, click, or touch event	HTML view update or alert message
View/UI	User/Admin Web or Mobile Browser	HTML view with HICS system information or user information (if admin)	None
Controller	View/UI	HTML page with new or existing view model	None
View/UI	Controller	Web form or click event	HTML view or JSON message

Table 8-1 View/UI Module Interfaces

External Data Dependencies

- ASP.NET Razor View Engine
- JQuery 2.0.3+
- Bootstrap JavaScript and CSS files
- ParsleyJS library
- Google Chrome, IE 9.0+, Firefox 3.6.9+

Internal Data Dependencies

- User/Admin mouse, touch, and keyboard events and input
- View model that corresponds to the view

Pseudo Code

```
<!-- simple registration form example -->
@model Web.Models.RegisterViewModel
@{ ViewBag.Title = "Register";}
@using (Html.BeginForm("Register", "Account", FormMethod.Post, new { @class = "form-
horizontal", role = "form" })) {
   @Html.AntiForgeryToken()
   <h4>Create a new account.</h4>
   <hr />
   @Html.ValidationSummary()
   <div class="form-group">
      @Html.LabelFor(m => m.UserName, new { @class = "col-md-2 control-label" })
      <div class="col-md-10">
         @Html.TextBoxFor(m => m.UserName, new { @class = "form-control" })
      </div>
   </div>
   <div class="form-group">
      @Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })
      <div class="col-md-10">
         @Html.PasswordFor(m => m.Password, new { @class = "form-control" })
      </div>
   </div>
   <div class="form-group">
      @Html.LabelFor(m => m.ConfirmPassword, new { @class = "col-md-2 control-label" })
      <div class="col-md-10">
         @Html.PasswordFor(m => m.ConfirmPassword, new { @class = "form-control" })
      </div>
   </div>
   <div class="form-group">
      <div class="col-md-offset-2 col-md-10">
         <input type="submit" class="btn btn-default" value="Register" />
      </div>
   </div>
}
```

8.1.2 – Model

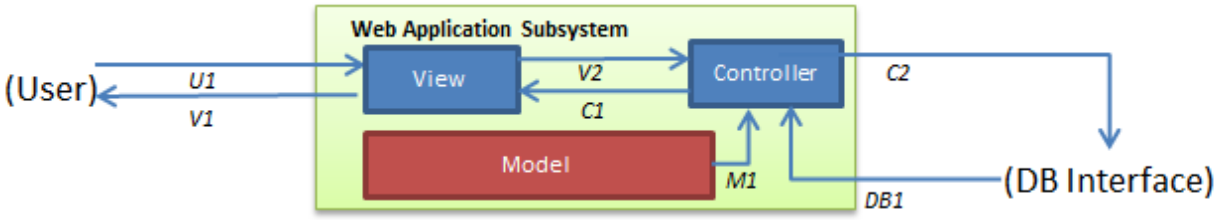

Figure 8-2 Model Module

Prologue

The Model Component is responsible for embodying the behavior of the Web Application by how the domains, or classes, are defined. The Model serves as a representation of the domains and directly manages how the data and logic are applied in relation to those domains. The Model Component helps to encapsulate the properties of the domains and also allows for the controller to customize them depending on what data it needs. The Models are also what the Razor View Engine uses to attach the users input to specific properties that can be understood by the Web Application.

Interfaces

Source	Sink	Input to Sink	Return from Sink
Model	Controller	A new view model or a model that represents existing data	None

Table 8-2 Model Module Interfaces

External Data Dependencies

- MVC 5+
- .NET Framework 4.5+

Internal Data Dependencies

- Core project library (project that holds the domains, enums, and constants)

Pseudo Code

```
// Account View Model creation example
public AccountViewModel(string role, User user, IEnumerable<Zone> zones) {
        User = new User {
                Id = user.Id,
                Username = user.Username,
                FirstName = user.FirstName,
                LastName = user.LastName,
                Role = user.Role,
                UnitId = user.UnitId,
                Active = user.Active,
                PasswordSalt = user.PasswordSalt,
                PasswordHash = user.PasswordHash,
        };
        ZoneSelectList = zones.Select(x => new SelectListItem {
                Value = x.Id,
                Text = x.Name
        });
        Role = role;
}
```

8.1.2 – Controller

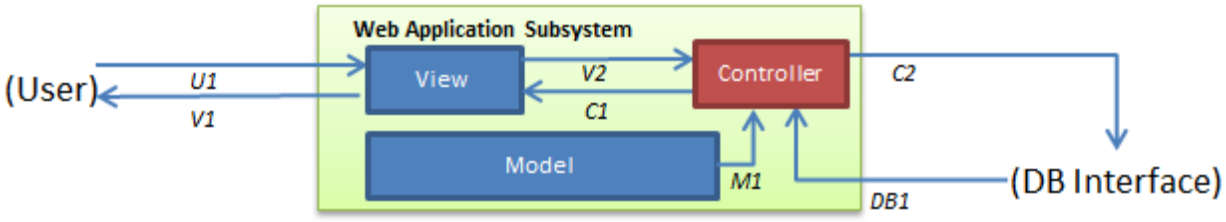

Figure 8-3 Controller Module

Prologue

The Controller Module of the Server Layer is primarily responsible for handling data transfers between the database and UI. When a user is accessing information about their HICS system the Controller is responsible for requesting that data from the database, assigning it to the correct model, and relaying the packaged view model back to the view. It is also responsible for submitting input from the user to the corresponding Database Interface. In addition the Controller also defines how navigation is handled throughout web pages and handles the authentication for user logins and requests.

Interfaces

Source	Sink	Input to Sink	Return from Sink
DB Interface	Controller	User data represented by a domain object or data structure	None
Model	Controller	A view model	None
View/UI	Controller	An updated model state or request parameters	Requested data, an HTML view, or an error message
Controller	DB Interface	A new/updated model state or request parameters	Requested data in the form of a domain object or data structure
Controller	View/UI	An HTML view with corresponding view model	None

Table 8-3 Controller Module Interfaces

External Data Dependencies

- MVC 5+
- .NET Framework 4.5+
- Newtonsoft JSON library

Internal Data Dependencies

- Core project library (project that holds the domains, enums, and constants)
- Services project library (project that holds the domain interfaces and services)

Pseudo Code

```
[HttpPost]
[AllowAnonymous]
public ActionResult Signin(AccountSignInModel model, string returnUrl) {
	try {
		if (_accountService.ValidateLogin(model.Username, model.Password)) {
			var now = DateTime.UtcNow.ToLocalTime();
			var user = _accountService.GetByUsername(model.Username);
			var userData = new UserDataModel {Id = user.Id, Role = user.Role};
			var formsAuthticket = new FormsAuthenticationTicket(
				version: 1,
				name: model.Username,
				issueDate: now,
				expiration: now.AddMinutes(20),
				isPersistent: model.RememberMe,
				userData: JsonManager.Serialize(userData),
				cookiePath: FormsAuthentication.FormsCookiePath);
			var encryptedTicket = FormsAuthentication.Encrypt(ticket);
			var httpCookie = new HttpCookie(
				FormsAuthentication.FormsCookieName, encryptedTicket);
			httpCookie.HttpOnly = true;

			// set the cookie's expiration time to ticket expiration time
			if (formsAuthticket.IsPersistent)
				httpCookie.Expires = formsAuthticket.Expiration;

			httpCookie.Path = FormsAuthentication.FormsCookiePath;
			if (FormsAuthentication.CookieDomain != null)
				httpCookie.Domain = FormsAuthentication.CookieDomain;
			HttpContext.Response.Cookies.Add(httpCookie);

			if (Url.IsLocalUrl(returnUrl))
				return Redirect(returnUrl);

			return RedirectToAction("Index", "Home");
		}
	} catch (Exception e) {
		ModelState.AddModelError("Error", "Username or pw verification failed.");
		return View(model: model);
	}
}
```

8.2 Web Services Subsystem

The Web Services Subsystem is responsible handling all interactions between the Web Application and every HICS system. The subsystem takes in information from the HICS systems in the form of environment sensor readings and command update requests. The Web Service Subsystem must relay all readings to be stored in the database and must pull any command requests that have updated since the last request.

8.2.1 – Response Handler

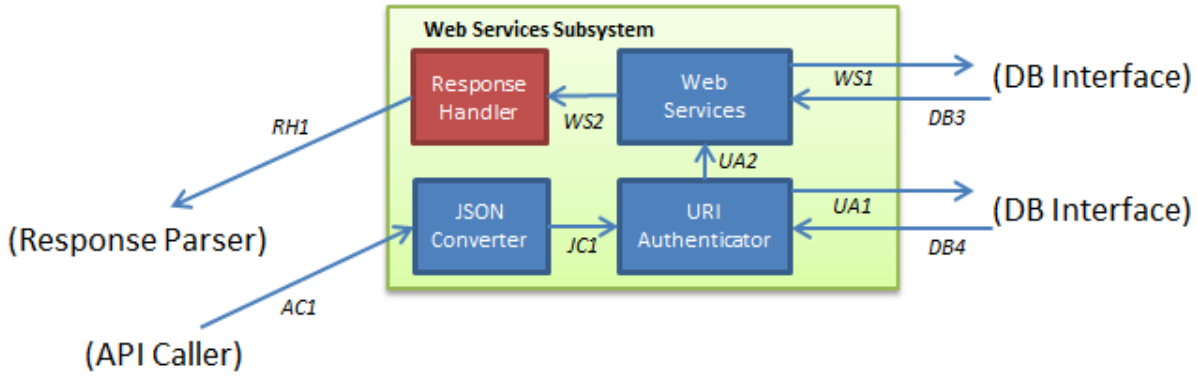

Figure 8-4 Response Handler Module

Prologue

The Response Handler Module is the component of the Web Services Subsystem that finalizes a request by responding with either the command data requested or a response code. If there was a change in the command state from the previous request (from the same system) then the updated command state is converted to a JSON object and packaged in the response to be sent back to the system. Since not all requests are for commands the Response Handler Module is also responsible for sending back status codes, along with any error messages, after a successful or unsuccessful saving of the environment sensor readings.

Interfaces

Source	Sink	Input to Sink	Return from Sink
Web Services	Response Handler	An command object or a success/error string message	None
Response Handler	Response Parser	JSON object with command information or status code string	None

Table 8-4 Response Handler Module Interfaces

External Data Dependencies

- Entity Framework 6
- Microsoft AspNet.Http libraries
- Newtonsoft JSON library

Internal Data Dependencies

- Domain object(s) return from a web service call

Pseudo Code

```
public JsonResult CommandResponse(ModelBinder(typeof(ModelBinderCommandExtension))]
ICommandRequest requestModel, string status) {
        requestModel.Length = requestModel.Length < MinLength ? MinLength :
requestModel.Length;
        var response = requestModel.Select(x => new {
                Id = x.Id,
                Message = status,
                Command = x.Command,
                Active = x.Active,
                Type = x.CommandType });
        return Json(new Response(response, dataTable, requestModel.TotalCount,
requestModel.TotalCount), JsonRequestBehavior.AllowGet);
}
```

8.2.2 – Web Services

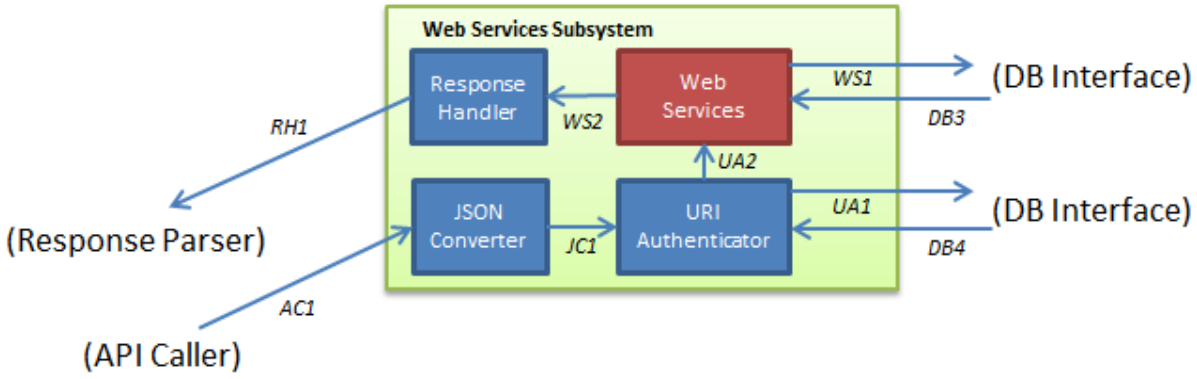

Figure 8-5 Web Services Module

Prologue

The Web Services Module is the main data processor for the HICS API. The module is composed of multiple methods that are responsible for interfacing with the Database Interface Subsystem to store environment readings as well as pull and process command statuses. If a request is to post sensor readings then the module instantiates the appropriate service repositories and invokes the corresponding save methods in the DB Interface Module. If a request is for a command state update then the module must pull the state from the database and check to see if there is a change since the previous request.

Interfaces

Source	Sink	Input to Sink	Return from Sink
URI Authenticator	Web Services	Environment sensor readings as objects and HICS system Id	None
DB Interface	Web Services	Command state object or thrown exception	None
Web Services	Response Handler	Command object or status string	None
Web Services	DB Interface	Environment sensor readings as objects and HICS system Id	Thrown exception or list of previous command states

Table 8-5 Web Services Component Interfaces

External Data Dependencies

- Entity Framework 6
- Microsoft AspNet.Http libraries

Internal Data Dependencies

- Services project library (project that holds the domain interfaces and services)

Pseudo Code

```
public Command CheckCommandState(int unitId) {
        try {
                // Instantiate all sensor services
                var unit = _hicsUnitService.GetUnitById(unitId);
                var previousRequest = _commandService.GetLastRequest(unitId);
                if (previousRequest.Timestamp < unit.Settings.LastUpdated) {
                        // Check if the state changed
                        // If state updated build new command object with updated fields
                        // and set the status to updated
                } else {
                        // Get the previous command to return
                }
                ...
        } catch (Exception ex) {
                // Attached exception message to new command and return
        }
        return command;
}
```

8.2.3 – URI Authenticator

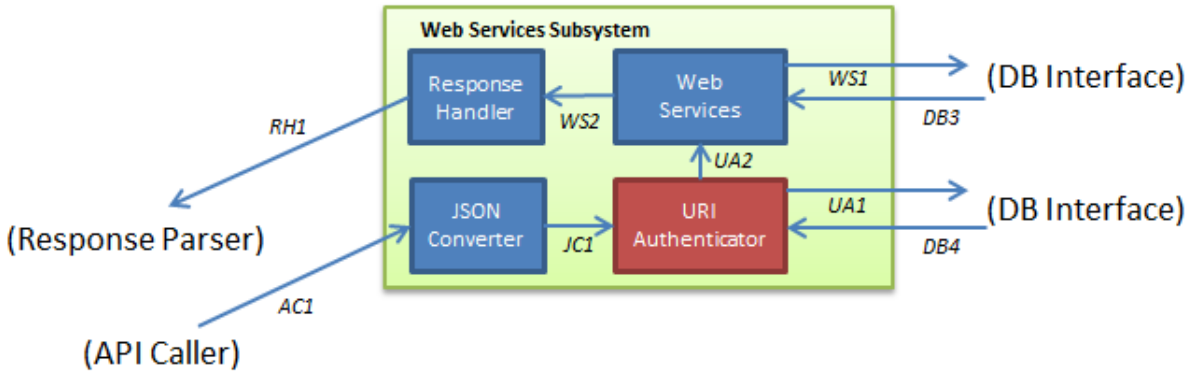

Figure 8-6 URI Authenticator Module

Prologue

The URI Authenticator module will be responsible for authenticating HTTP requests from the API Caller Module. The module must take the system id from the JSON Converter Module and query the DB Interface to check for a matching database entry. If the database returns a match then the URI Authenticator transmits the request to the appropriate Web Service method.

Interfaces

Source	Sink	Input to Sink	Return from Sink
URI Authenticator	Web Services	Sensor reading object(s) and system id	None
URI Authenticator	DB Interface	HICS system id	User domain object or null
JSON Converter	URI Authenticator	Sensor reading object(s) and system id	None
DB Interface	URI Authenticator	User domain object or null	None

Table 8-6 URI Authenticator Interfaces

External Data Dependencies

- Entity Framework 6

Internal Data Dependencies

- Services project library (project that holds the domain interfaces and services)
- HICS system id integer

Pseudo Code

```
public ActionResult AuthorizeRequest(TemperatureReading tempReading, List<SoilReading>
soilReadings, RainReading rainReading, int unitId) {
        var user = _userService.GetUserById(unitId);
        if (user == null) {
                // Terminate the request
        }
        // Redirect to web services action
}
```

8.2.4 – JSON Converter

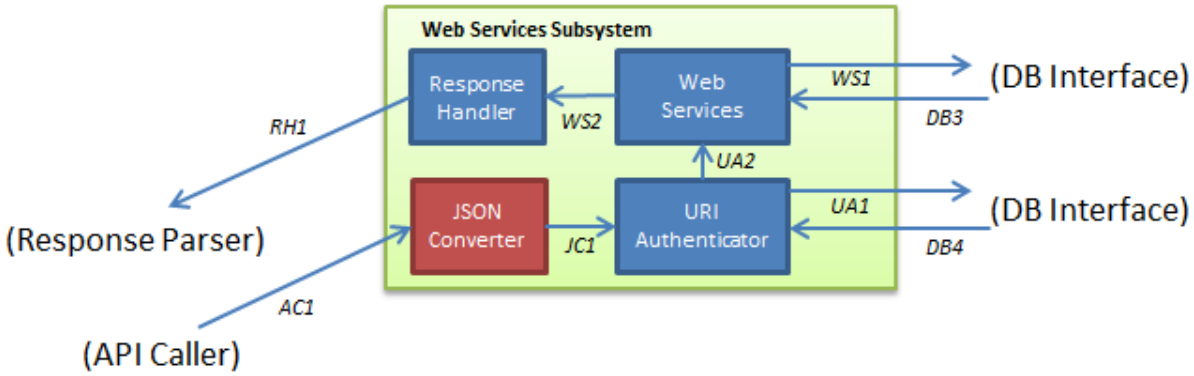

Figure 8-7 JSON Converter Module

Prologue

The JSON Converter Module will be responsible for handling the initial request from a API Caller Module. The module is responsible for parsing the HTTP request to serialize the JSON body message to its corresponding domains and extracting the system id from the request parameters. Once the request is converted the data is then transmitted to the URI Authenticator for authorization.

Interfaces

Source	Sink	Input to Sink	Return from Sink
JSON Converter	URI Authenticator	Sensor reading object(s) and system id	None
API Caller	JSON Converter	HTTP request	None

Table 8-7 JSON Converter Interfaces

External Data Dependencies

- Entity Framework 6
- Microsoft AspNet.Http libraries
- Newtonsoft JSON library

Internal Data Dependencies

- HTTP request with JSON message body and valid parameters

Pseudo Code

```
[EnableJson]
[HttpGet, OutputCache(NoStore = true, Location = OutputCacheLocation.None)]
public IHttpAction RequestHandler(object sender, RoutedEventArgs e) {
        // Serialize the request

        // Parse the message and API key

        // Send the serialized objects and system id to the authenticator
}
```

8.3 Database Interface Subsystem

The Database Interface Subsystem is the backend service for both the HICS web services and web application. The subsystem interfaces directly with the database to store and retrieve data. The subsystem is composed of a DB Interface Module which holds all the interfaces and services for the domains and the Store Procedures Module which handles the SQL queries and defines how data is transferred to and from the database.

8.3.1 – Stored Procedures

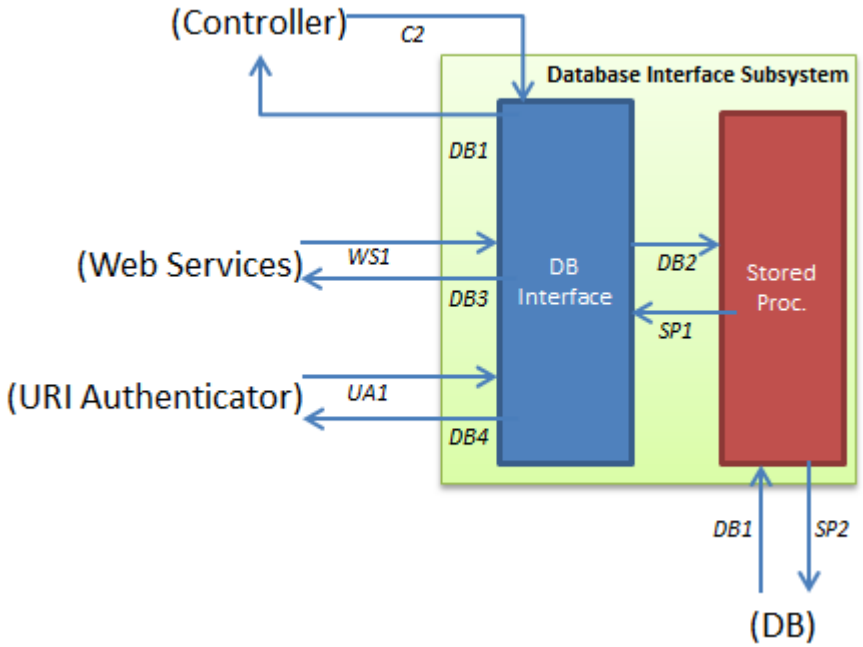

Figure 8-8 Stored Procedures Module

Prologue

The Stored Procedures Module will be responsible for handling all the database queries for the Web Service Module and Web Application. This module allows us to formally define how each component interacts with the database. The Store Procedures Module will use these definition to handle all the CRUD (create, read, update, delete) methods for every domain service. There will be a total of four store procedures, one for each CRUD method, and an additional stored procedure that will handle authentication checks from the Controller and Web Service Modules.

Interfaces

Source	Sink	Input to Sink	Return from Sink
Store Procedures	DB Interface	Requested domain object(s)	None
Stored Procedures	Database	SQL query	SQL table data

Table 8-8 Stored Procedures Interfaces

External Data Dependencies

- Entity Framework 6

Internal Data Dependencies

- Object or identity parameters needed to build the SQL queries

Pseudo Code

```
public void Insert(T entity) {
        try {
                if (entity == null)
                        throw new ArgumentNullException("entity");
                this.Entities.Add(entity);
                this._context.SaveChanges();
        } catch (DbEntityValidationException dbEx) {
                throw new Exception(dbEx.message);
        }
}
```

8.3.2 – DB Interface

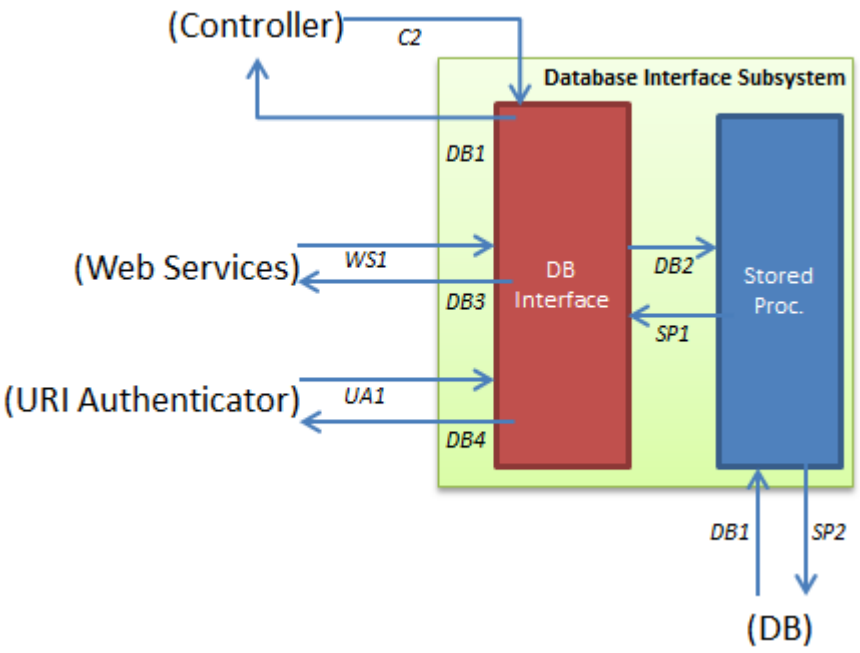

Figure 8-9 DB Interface Module

Prologue

The DB Interface Module will responsible for processing all save and pull requests to and from the database. The module contains the definitions for each of different domain interfaces and services. These services are called by the Web Services and Controller Modules through their interface definitions to encapsulate how data is being processes and communicated between the module and the database. Each domain service opens a connection with the Store Procedures Module every time it is instantiated. At the end of a service call the connection closes to ensure security and improve serialization.

Interfaces

Source	Sink	Input to Sink	Return from Sink
DB Interface	Store Procedure	Domain object(s) or query parameters	Requested domain object(s), void, or thrown exception
DB Interface	Web Services	List of command state objects	None
DB Interface	URI Authenticator	Matching user object or null	None
DB Interface	Controller	Domain object(s)	None
Stored Procedures	DB Interface	Database table data	None
Controller	DB Interface	Domain object(s) or query parameters	Requested domain object(s), void, or thrown exception
Web Services	DB Interface	Sensor reading objects or command request	List of command state objects or thrown exception
URI Authenticator	DB Interface	HICS system id	Matching user object or null

Table 8-9 DB Interface Interfaces

External Data Dependencies

- Entity Framework 6

Internal Data Dependencies

- Stored Procedures project library (project that holds the generic repository and CRUD methods)

Pseudo Code

```
public IQueryable<User> GetAllActiveUsers() {
        return _userRepo.Table.Where(x.Active);
}
```

9. Quality Assurance

9.1 Unit Testing

This section provides a detail description of the tests that will be done to assure we meet our requirements while maintaining the quality of HICS. The following sections also map out how the overall system will be tested during the progression of the project throughout the prototyping phase and into implementation.

9.2 Sensor Layer

Rain Sensor Subsystem

9.2.1 – Rain Status Collector: This module will be able to transmit a signal if it detects rain fall.

9.2.2 – Control Board: This module receives a signal from the rain collector.

Temperature Sensor Subsystem

9.2.3 – Temperature Reading Collector: This module will be able to collect the current temperature readings of the environment and transmit the signal.

9.2.4 – Control Board: This module receives the signal from the temperature collector.

Soil Moisture Sensor Subsystem

9.2.5 – Soil Moisture Reading Collector: The soil moisture signal collector reads the amounts of moisture present on the soil and transmits the signal.

9.2.6 – Control Board: This module receives the signal from the soil moisture collector.

9.3 Hardware I/O Layer

Sensor Controller Subsystem

9.3.1 – Analog-Digital Converter: This module will receive analog data from the sensors and convert the data to digital.

9.3.2 – Serial Data Packager: This module will collect all the individual sensor data readings and serialize them into a single set of readable byte strings.

9.3.3 – Serial Data Sender: This module will write the serial byte strings to the USB/Serial Interface through the USB cable.

Valve Controller Subsystem

9.3.4 – Serial Data Receiver: This module will read the data from the serial port and convert it to a string.

9.3.5 – Command Executer: This module will set the digital pins on the Arduino to low or high depending on the command.

9.3.6 – Relay Module: This module will convert the Arduino commands into electronic signals that then turn the irrigation valve(s) on and off.

9.4 Interface Layer

Data Processing Subsystem

9.4.1 – USB/ Serial Interface: This module receives packaged sensor serial data and can write serial byte strings to the Valve Controller Subsystem.

9.4.2 – Json Message Builder: This module will receive the sensor data from the USB/Serial Interface and convert it into a JSON object and format as a URI request parameters.

9.4.3 – Valve Command Processor: This module will receive the JSON object and be able to parse and determine what command is being sent. That command must then be sent to the USB/Serial Interface.

Service Caller Subsystem

9.4.4 – API Caller: This module will receive the JSON request object and create an HTTP request with the object attached in the body. The module must also attach the authorization parameters for the request and initialize the API Call.

9.4.5 – Response Parser: This module will receive the HTTP response coming back from Server Layer and parses the returned data to a JSON object. That object must then be passed to the Valve Command Processor Module.

9.5 Server Layer

Web Services Subsystem

9.5.1 – JSON Converter: This module will receive the HTTP request from the Interface Layer and convert the request body and parameters into domain objects or C# data types.

9.5.2 – URI Authenticator: This module will authenticate the API call by verifying the passed HICS system id is valid.

9.5.2 – Web Services: This module will receive the put or pull request and either store or retrieve the information from the Database Interface Subsystem.

9.5.3 – Response Handler: This module will terminate the HTTP request by either returning an updated command JSON object or status code.

Web Application Subsystem

9.5.4 – Model: This module creates new models or map models to existing domain data.

9.5.5 – Controller: This module route web traffic, load HTML views, and transmit data between the UI/View and DB Interface.

9.5.6 – View/UI: This module will receive generate the HTML for the page and capture user click, touch, or keyboard events.

Database Interface Subsystem

9.5.7 – DB Interface: This module will bind domain interfaces to their corresponding services and interface with Stored Procedure methods to store and retrieve data.

9.5.8 – Stored Procedures: This module will receive request data and authenticate the data type for secured access. The module will also execute SQL commands to the database and return or save table data.

9.6 Component Testing

Each component will be tested for their response to various inputs. Input will be given according to the dataflow on our diagram and the results will compared to the expected results. The major component tests are described in more detail below.

9.6.1 – Sensor Testing: HICS should be strong enough to bear the environment conditions like high wind, high/low temperature, and pressure. The rain sensor will properly detect rainfall, the temperature sensor will accurately read the temperature, and the soil moisture sensors will accurately report the soil moisture levels.

9.6.2 – Web Application Testing: The Web Application will be fast and responsive. It should be able to access the database information and save any required information on the views. The Web Application should be able to scale the UI properly to adhere to the device it is being accessed on.

9.6.3 – Database Testing: The database should be accessible to Web Application and Web Services when necessary. There must be a pre-requested query to access the database. There should be security check for inappropriate and unauthorized access to data.

9.7 Integration Testing

Each layer and module will be tested individually before integrating into the HICS system. Input data will be sent to each layer and the results are compared with the expected output. The Sensor Layer will be tested by providing different environment conditions to the sensors. The Hardware I/O Layer will be tested by checking if the data received by Interface Layer is in a digital package. The Interface layer will be tested to see if it can make a HTTP request and the Server Layer will be tested by accessing the database, making changes, and saving it.

9.8 System Verification Testing

To avoid any possible unknown error, HICS will be tested down to its smallest module. Each module, component, and equipment will be tested individually and then tested again after integrating. Any module that can be subjected to automated testing will be to improve throughput.

9.9 Test Cases

Test Case	Expected Result
Rain sensor detects rain	Rain status will be changed to "True". The user sees notification on the web browser.
Soil moisture level is very low (dry)	Web browser notifies the user the amount of soil moisture and suggests watering.
User schedules watering	Watering occurs on the scheduled time and it is notified to the user on the web browser.
Temperature sensor detects change	Updated temperature will be relayed and stored in the database and the new temperature will be updated on the users web browser.

Table 9-1 Test Cases Table

10. Requirements Traceability

The following sections will detail, layer by layer, how each key requirement from our SRS is fulfilled by a combination of modules. From the tables below you can see just how integrated the entire HICS system is and how requirements can span across multiple modules. The Database Management System is a requirement that incorporates a large amount of modules into its functionality. This is because the DBMS of HICS is the component that allows users to communicate with their systems and vice versa. Other requirements that rely on multiple modules are the Central Control Unit, Web Application, and Water Scheduler. Each one of these requirements is fundamental to how HICS will function and where its value comes from. The components reflect that because we've designed this system to implement everything we emphasized during our SRS and ADS. The Tables 10.1 through 10.4 give a breakdown of how each key requirement is met throughout each layer, subsystem, and module.

10.1 Sensor Layer

Requirements No.	Requirements Name	Soil Moisture Sensor S/S		Temperature Sensor S/S		Rain Sensor S/S	
		Voltage Comparator	Soil Moisture Reading Collector	Voltage Comparator	Temperature Reading Collector	Voltage Comparator	Rain Status Collector
3.1	Central Control Unit	✔	✔	✔	✔	✔	✔
3.2	Soil Moisture Sensors	✔	✔				
3.3	Web Application						
3.4	Water Scheduler	✔	✔			✔	✔
3.5	Soil Moisture Reports	✔	✔				
3.6	User Login						
3.8	Rain Sensor					✔	✔
3.10	DB Management System	✔	✔	✔	✔	✔	✔
3.14	Temperature Sensor			✔	✔		
5.1	Sensor Accuracy	✔	✔			✔	✔
5.2	Rain Detection					✔	✔
5.3	Comm. Between Web and Unit						
8.2	Browser Support						

Table 10-1 Sensor Layer Requirements Table

10.2 Hardware I/O Layer

Requirements No.	Requirements Name	Sensor Controller S/S			Valve Controller S/S		
		Analog-Digital Converter	Sensor Data Packager	Serial Data Sender	Relay Module	Command Executor	Serial Data Receiver
3.1	Central Control Unit	✔	✔	✔	✔	✔	✔
3.2	Soil Moisture Sensors	✔					
3.3	Web Application						
3.4	Water Scheduler	✔	✔	✔	✔	✔	✔
3.5	Soil Moisture Reports	✔	✔	✔			
3.6	User Login						
3.8	Rain Sensor	✔	✔	✔			
3.10	DB Management System	✔	✔	✔	✔	✔	✔
3.14	Temperature Sensor	✔	✔	✔			
5.1	Sensor Accuracy	✔	✔	✔			
5.2	Rain Detection	✔	✔	✔			
5.3	Comm. Between Web and Unit						
8.2	Browser Support						

Table 10-2 Hardware I/O Layer Requirements Table

10.3 Interface Layer

Requirements No.	Requirements Name	Data Processing S/S			Service Caller S/S	
		USB/Serial Interface	Valve Command Processor	JSON Message Builder	Response Parser	API Caller
3.1	Central Control Unit	✔	✔	✔	✔	✔
3.2	Soil Moisture Sensors	✔		✔		✔
3.3	Web Application					
3.4	Water Scheduler	✔	✔	✔	✔	✔
3.5	Soil Moisture Reports	✔		✔		✔
3.6	User Login					
3.8	Rain Sensor	✔		✔		✔
3.10	DB Management System	✔	✔	✔	✔	✔
3.14	Temperature Sensor	✔		✔		✔
5.1	Sensor Accuracy					
5.2	Rain Detection	✔		✔		✔
5.3	Comm. Between Web and Unit				✔	✔
8.2	Browser Support					

Table 10-3 Interface Layer Requirements Table

10.4 Server Layer

Requirements No.	Requirements Name	Web Services S/S				Database Interface S/S		Web Application S/S		
		JSON Converter	URI Authenticator	Web Services	Response Handler	DB Interface	Store Procedures	Controller	Model	UI/View
3.1	Central Control Unit									
3.2	Soil Moisture Sensors									
3.3	Web Application	✔	✔	✔	✔	✔	✔	✔	✔	✔
3.4	Water Scheduler			✔	✔	✔	✔	✔	✔	✔
3.5	Soil Moisture Reports					✔	✔	✔	✔	✔
3.6	User Login					✔	✔	✔	✔	✔
3.8	Rain Sensor									
3.10	DB Management System					✔	✔			
3.14	Temperature Sensor									
5.1	Sensor Accuracy									
5.2	Rain Detection									
5.3	Comm. Between Web and Unit	✔	✔	✔	✔					
8.2	Browser Support									✔

Table 10-4 Server Layer Requirements Table

11. Acceptance Plan

This section discusses the plan developed to meet the acceptance criteria set for HICS. This plan includes the necessary package and installation information required to deploy and operate HICS as well as the acceptance test plan and the acceptance criteria HICS needs to satisfy our requirements and be deemed acceptable to the customer.

11.1 Packaging and Installation

HICS will be packaged with the following items included:

- (1) Raspberry Pi micro controller
- (2) Arduino Mega 2560
- (3) Soil moisture sensors
- (1) Rain sensor
- (1) Thermal sensor
- (1) Raspberry Pi AC adapter
- (1) Power adaptor for Relay Board
- (1) User manual

Also HICS will include a Web Application which is running on a hosting service.

11.2 Acceptance Testing

The acceptance tests will be evaluated and discussed by SmartGrass to ensure every acceptance criteria is met. A further analysis and description of our acceptance tests and plan of action will be detailed in our System Test Plan document.

11.3 Acceptance Criteria

For HICS to be considered acceptable a set of acceptance criteria rules must first be met. The following list of criteria covers the high priority requirements for HICS.

- The Web Application must be able to create, edit, or delete user accounts and preferences.
- The Web Application must be able to provide a scalable UI for desktop and mobile devices.
- The system must be able to store sensor readings and display them on the UI.
- The system must be able to control the irrigation valves, and show their status on the UI.
- The Web Application must be able to allow creation of watering schedules.
- The system must be able to turn on or turn off watering automatically based on conditions in the users settings.
- The system must be able to turn off automatically when it loses the Internet connection, detects rainfall, or detects freezing conditions.

12. Appendix

12.1 Operating Systems and Libraries

12.1.1 – Raspbian OS

Raspbian is a Debian based operating system optimized for the Raspberry Pi hardware. It comes with over 35,000 packages which are precompiled in such a way that it makes the Raspberry Pi easy to install and use.

12.1.2 – GSON Library

GSON library is systems library that serializes and deserializes JSON to Java objects.

12.1.3 – JQuery Library

JQuery is a JavaScript library that allows to more customization when it comes to frontend design and interactions.

12.1.4 – Sockets Library

The Sockets library makes the communication between Python and Java via internet using TCP/IP.

12.2 Programming Languages

12.2.1 – Processing

Processing will be used for programming the micro-controller because it offers easy and quality programming.

12.2.2 – SQL

The SQL query language will be used to handle data queries with the HICS MYSQL database.

12.2.3 – C#

All Web Application and API programming will be developed using C#.

12.2.4 – HTML/JavaScript/CSS

HTML, JavaScript, and CSS will all be used to program the styling and design of the Web Application pages.

12.2.5 – Arduino Programming Language

Arduino utilizes a native programming language, much like C, to develop programs for the microcontroller. This programming language will be used for all the Arduino microcontroller programming.

12.3 Interface Table Legend

12.3.1 – Source

Source is the interface that sends data that acts as input for the sink.

12.3.2 – Sink

Sink is the interface that receives the input data coming from the source.

12.3.3 – Input to Sink

These are data such as objects and messages that are passed through parameters from the source to the sink.

12.3.4 – Return from Sink

These are data that are returned from the sink to the source.