2015-03-01

Hydrologic Data Sharing Using Open Source Software and Low-Cost Electronics

Jeffrey Michael Sadler
Brigham Young University - Provo

Hydrologic Data Sharing Using Open Source Software and Low-Cost Electronics

Jeffrey Michael Sadler

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Daniel P. Ames, Chair
E. James Nelson
Gustavious P. Williams

Department of Civil and Environmental Engineering

Brigham Young University

March 2015

ABSTRACT

Standards-Based Data Sharing using Open Source Software
and Low-Cost Electronics

Jeffrey Michael Sadler
Department of Civil and Environmental Engineering, BYU
Master of Science

While it is generally accepted that environmental data are critical to understanding environmental phenomena, there are yet improvements to be made in their consistent collection, curation, and sharing. This thesis describes two research efforts to improve two different aspects of hydrologic data collection and management. First described is a recipe for the design, development, and deployment of a low-cost environmental data logging and transmission system for environmental sensors and its connection to an open source data-sharing network. The hardware is built using several low-cost, open-source, mass-produced components. The system automatically ingests data into HydroServer, a standards-based server in the open source Hydrologic Information System (HIS) created by the Consortium of Universities for the Advancement of Hydrologic Sciences Inc (CUAHSI). A recipe for building the system is provided along with several test deployment results. Second, a connection between HydroServer and HydroShare is described. While the CUAHSI HIS system is intended to empower the hydrologic sciences community with better data storage and distribution, it lacks support for the kind of "Web 2.0" collaboration and social-networking capabilities that are increasing scientific discovery in other fields. The design, development, and testing of a software system that integrates CUAHSI HIS with the HydroShare social hydrology architecture is presented. The resulting system supports efficient archive, discovery, and retrieval of data, extensive creator and science metadata, assignment of a persistent digital identifier such as a Digital Object Identifier (DOI), scientific discussion and collaboration around the data and other basic social-networking features. In this system, HydroShare provides functionality for social interaction and collaboration while the existing HIS provides the distributed data management and web services framework. The system is expected to enable scientists, for the first time, to access and share both national- and research lab-scale hydrologic time series in a standards-based web services architecture combined with a social network developed specifically for the hydrologic sciences. These two research projects address and provide a solution for significant challenges in the automatic collection, curation, and feature-rich sharing of hydrologic data.

Keywords: Open-source hardware; Sensor networks; Environmental monitoring; Open-source software; Time series data; Digital objects; Social networks; Metadata; Web systems

ACKNOWLEDGEMENTS

I'm grateful for Dr. Daniel P. Ames and his tireless effort, confidence, and sacrifice and for his example that extends far beyond his professional life. I am grateful for my parents who have always loved and believed in me and who gave me the opportunities to arrive to this point. I'm grateful for my wife, Camilla, who has always stood beside me and believed in me. Finally I am grateful to God, my Heavenly Father. He has given me all that is good in my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Collected, transmitted, and stored observational data are invaluable in field of hydrology. They are needed for the understanding, modeling, and predicting of hydrologic phenomena (Baker 1936). With recent technological advances, the ability to collect, curate, and effectively exchange hydrologic data has increased dramatically (Conner et al. 2013; Hicks et al. 2013; Michener et al. 2012; Tarboton et al. 2009). However, there are still many areas for improvement in systems, software, and methods for accomplishing these tasks. In this thesis, I address the following challenges in the area of hydrologic data: 1) the cost of hydrologic data collection hardware and data management software, 2) automatically providing for the long-term storage, interoperability, and discovery of hydrologic data, and 3) the need for a web-based environment that hydrologists can use to share and collaborate around their data.

To illustrate the need for solutions to the challenges above, I will use a simple example. Suppose a hydrologist is hired by a city to develop a hydrologic model. The city needs the model to better understand the local watershed's response to rainfall events, thus being able to predict a storm's effect on public infrastructure such as roads and bridges, as well as private property such as homes and commercial buildings. To calibrate and then validate this model, the hydrologist would need hydrologic data, specifically rainfall and stream data (streamflow and/or stream stage). In the process of developing a hydrologic model, the needed data may go through a number of different stages. Michener and Jones (2012) provide a useful model for understanding the stages

scientific data may undergo in their "…transformation…into information." This model is called the "data life cycle" and is summarized in Figure 1.

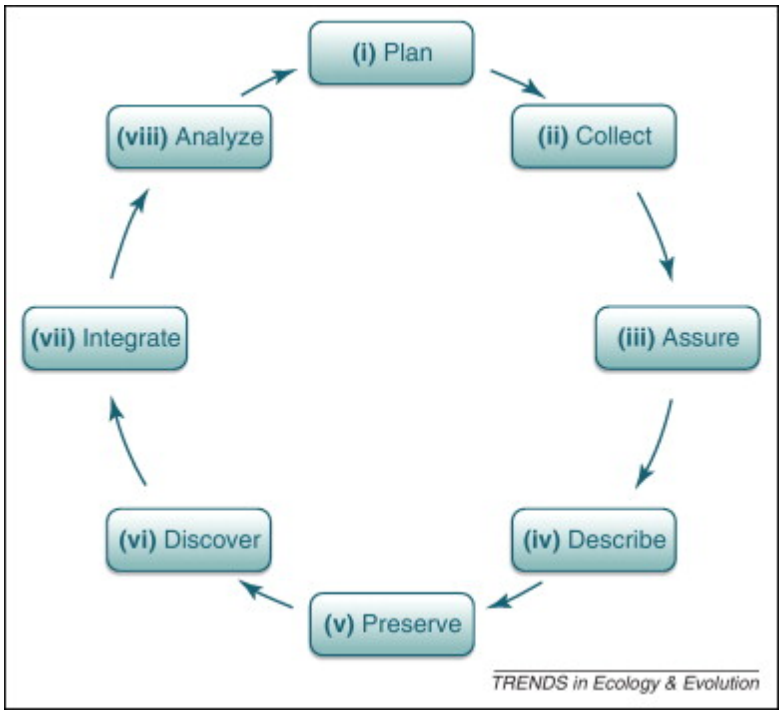

Figure 1. Data life cycle (Michener and Jones 2012)

This thesis consists of two main chapters that focus on different aspects of the hydrologic data life cycle. The first chapter, "A Recipe for Standards-Based Data Sharing using Open Source Software and Low-Cost Electronics," focuses on hydrologic data collection, curation, and exchange, stages 2, 5, and 6 of the data life cycle respectively. Data collection can have significant costs due to the cost of commercial hardware systems and/or hardware maintenance. The solution described in Chapter 2 addresses this problem by using low-cost, mass-produced parts to provide data logging and data transmission functionality. Additionally, the challenge of curating and making discoverable the collected data is addressed. Without proper curation, hydrologic data and

its accompanying metadata can quickly be lost. Without data standards and supporting cyber-infrastructure, their discovery and exchange can be hampered. The solution described for these challenges is the automatic ingestion of the data transmitted by the low-cost equipment into an open-source software, HydroServer. HydroServer is part of a standards-based service oriented architecture system designed specifically for the exchange of hydrologic time series and acts as an endpoint for web services for data discovery and retrieval. The system therefore, provides a framework for collecting hydrologic data and making that data automatically discoverable and retrievable in a standards-based system.

The third chapter, "Time Series Data as a Social Media Object," is centered on enhancing the sharing of hydrologic data to improve their integration and analysis, stages 7 and 8 of the data life cycle respectively. While much work has been done in hydrologic data exchange, "Web 2.0" functionality for sharing hydrologic data is less developed. To address this need, a connection between HydroServer and HydroShare, an online scientific collaborative environment, was established. In this way, the social media functionalities of HydroShare that facilitate and foster collaboration and enhanced data sharing are applied to time series datasets stored on HydroServers. Within the HydroShare environment, time series datasets stored in HydroServers can be easily discussed and shared. Finally, I give a conclusion for this thesis in Chapter 4.

The contributions of this thesis are a framework for low-cost environmental data logging and transmission that provides automatic data curation, discoverability, and interoperability, and the ability to interact with standards-based time series datasets as social media.

2 A RECIPE FOR STANDARDS-BASED DATA SHARING USING OPEN SOURCE SOFTWARE AND LOW-COST ELECTRONICS

2.1 Chapter Introduction

It is generally accepted that to understand and address environmental phenomena, environmental observations are required (Baker 1936). While important, it is often very costly to install and maintain systems that collect environmental data at high spatial and temporal resolutions; thus cost is a common barrier to exhaustive environmental monitoring. This barrier affects many organizations including researchers, educators, and government agencies around the world. For example, more than 200 USGS gauging stations have been recently discontinued, with dozens more threatened with the same fate due to funding issues (U.S. Geologic Survey 2014).

High costs are especially problematic when sensors are deployed for a brief time period, in a high-risk environment, or in developing countries with limited resources. A short-term investigation of an area requiring several nodes may be needed for an environmental model calibration (Quinn et al. 2010). In these a cases, an investment of several thousand dollars for equipment to be used only for short-term deployments may be infeasible. When several nodes are needed, costs, and thus infeasibility, rise quickly (Younis and Akkaya 2008). Another example where the expense of sensor systems may be an especially significant barrier is when the equipment is likely to be, or intended to be, destroyed (Kean et al. 2012; Oliveira and Rodrigues 2011). A final example is the collection of data in developing countries where the capital required to invest in expensive monitoring and telemetry equipment may be in short supply (Basha and Rus

2007). For example, since flooding is one of the costliest natural disasters, stream stage and rainfall data are invaluable to countries that are especially affected by these events such as many of the developing countries in Latin America (Baker 1936; Berz 2000; Charvériat 2000). The costs associated with purchasing, implementing, and maintaining equipment needed to collect stream level and rainfall data can be burdensome to developing economies.

In recent years, the advancement of open-source hardware has sparked significant reductions in the cost of scientific equipment in many technical areas. A key contributor to this movement is the Arduino, an open-source, programmable microcontroller. This easy-to-use electronics prototyping platform is adaptable to both simple and complex tasks (Arduino 2014). For example, an Arduino has been coupled with 3D printers to manufacture custom equipment for experimental laboratories, significantly reducing costs (Anzlone et al. 2013; Pearce 2012). Similar technology has been used to address broad problems such as energy availability and efficiency (Mousa and Claudel 2014; Pourmirza and Brooke 2013). More specifically, researchers have used low-cost open-source hardware in environmental monitoring to measure parameters such as latrine usage, snow depth, soil moisture, air quality, marine environments, and even pathogen levels (Clasen et al. 2012; Fedi et al. 2013; Lundquist and Lott 2008; Mitchell et al. 2014; Trevathan et al. 2012).

In addition to hardware costs, difficulties with the interoperability can threaten the value of environmental data. As science becomes increasingly data-intensive and collaborative, the need for easily-shareable data grows as well (Michener et al. 2012). The ability to share heterogeneous data from a variety of sources can be greatly enhanced through the use of data structure and sharing standard formats (Giuliani et al. 2011). While the importance of data sharing is generally accepted and can be facilitated through industry standards, in many cases the effort needed to make data shareable is a considerable hindrance (Borgman 2012).

Lower financial and technical barriers for data collection have increased the amount of data that can be affordably collected (Henson et al. 2013). Recently, plans for managing and sharing data collected for research have become heavily emphasized by funding institutions such as the U.S. National Science Foundation (NSF) (Tenopir et al. 2011).

The effective and creative use of newly developed and more affordable technologies to increase the spatial and temporal resolutions of environmental data is an active area of research (Hut 2013; Newman et al. 2012; van de Giesen et al. 2014; Zaslavsky et al. 2013). A number of low-cost hardware solutions to collect and transmit environmental data have been developed and described (Baker 2014; Wickert 2014).

An open hardware system developed by the Stroud Research Center, is used to collect data in the Christina River Basin Critical Zone Observatory. With Arduino boards and other open-source electronics such as Zigbee radios, the system uses several custom sensor modules including a radio reporting stream gauge, soil moisture sensor systems, pressure transducer readout, and respirometer controllers. These data are made available in WaterML1.1 format via WaterOneFlow web services created by the Consortium of Universities for the Advancement of Hydrologic Sciences Inc. (CUAHSI) (Hicks et al. 2013).

My goal is to improve upon the methods used in the cases above for managing and sharing collected data. For example, Fedi et al. (2013) and Trevathan et al. (2012) do not describe a specific plan to make the collected data publicly available, and the latrine usage data described by Clasen et al. (2012) were stored on an SD memory card but not made publicly available. The data storage and sharing schema of Hicks et al. (2013) is quite extensive, with streaming data viewable via a custom website and accessible through web services, but lacks implementation of Open Geospatial Consortium (OGC) standards.

To improve the management and shareability of the data collected by low-cost systems, my system automatically inserts observed data into a customized, OGC-compliant CUAHSI HydroServer (Kadlec and Ames 2012). This approach leverages the capabilities and standards of the relatively mature CUAHSI Hydrologic Information System (HIS) (CUAHSI 2014). Hence this solution provides a lightweight, low-cost connector that could be used to automatically ingest data collected by existing and future low-cost data collection systems into a federated, standards-based software system for sharing time series data.

The HIS enables standards-based storage, sharing, publication, and discovery of time-series data (Horsburgh et al. 2009). There are over 85 data sources currently cataloged in the CUAHSI Central Catalog, some of which use sensor data streams. My work builds on and adds to such capabilities by enabling real-time data streaming from low-cost hardware through a custom HydroServer that supports the OGC Web Feature Service (WFS) and the newly adopted OGC WaterML2.0 standard (Taylor et al. 2013). My custom HydroServer, called "HydroServer Lite", is built on the open source Linux-Apache-MySQL-PHP (LAMP) software stack, requiring only a simple computer and inexpensive, or free, webhosting. Because it does not use closed-source hardware or software, the system is extremely inexpensive and light-weight to deploy and hence can be highly suitable in rapid deployment, low-cost, short term, underfunded, or citizen science monitoring efforts (Buytaert et al. 2014). This paper is the first complete description of the integration of low-cost environmental data collection equipment and a free, open-source data management system that follows industry standards (i.e. WaterML1.0 and WaterML2.0).

The remainder of the paper describes the methods used to develop a low-cost data logging and transmission system configured to automatically input collected data into a HydroServer – including a recipe for others to follow to create a similar system. I also outline methods for testing

the success of the hardware, the interoperability of the data, and, the ease-of-configuration of the hardware. The results of each of these assessments are given, followed by a brief conclusion.

2.2 Methods

2.2.1 Hardware

In selecting hardware components for the environmental data logging and transmission system, my primary aims were to build a unit that was self-contained, low-cost, and relatively easy to build (e.g. with minimal soldering). The module also had to accommodate the automatic ingestion of collected data into the CUAHSI HIS. These criteria guided design decisions. The two main electronics components I used were an Arduino Uno and a SIMCOM SIM900 Quad-band GSM GPRS Shield. These are shown in Figure 2 along with the case, temperature and humidity sensor, and solar power system I used.

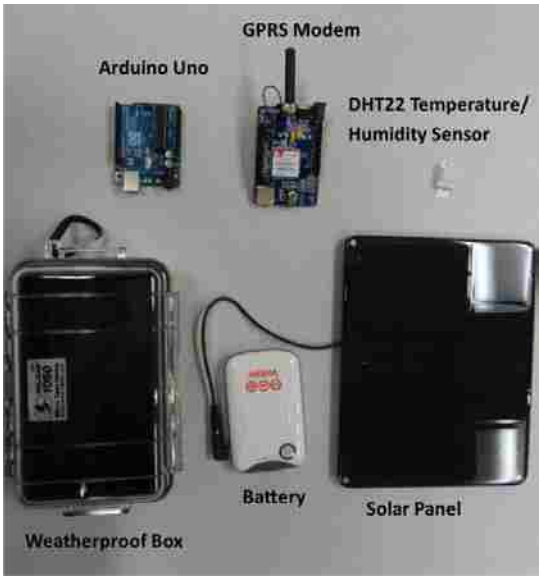

Figure 2. Arduino, solar panel, battery, sensor, and GPRS modem

I chose the Arduino Uno (Arduino 2014) as an easy-to-use and inexpensive prototyping platform. I initially used a DHT22 temperature and humidity sensor, and later made the system compatible with SDI-12 (serial data interface at 1200 baud) (www.sdi-12.org/index.php) sensors to allow the use of any of the many SDI-12 sensors available, many of which are generally well-respected and trusted as manifested by their use in organizations including the USGS (Fitzpatrick 2010). The SDI-12 sensor used in this research was a KPSI pressure transducer water level sensor that meets USGS standards.

I chose to use the cell phone network to eliminate the need for expensive long-range radio telemetry, and to avoid the spatial limitations of short-range radios like Zigbees; the system need only be deployed in a location with cell phone coverage. Furthermore, cell phone coverage is spreading rapidly everywhere, including in developing countries which could greatly benefit from low-cost sensor and data-transmission systems (Thomson et al. 2012). For power, a 6W solar panel with an accompanying 4000mAh lithium ion battery were purchased for a total of $88 USD. This constituted the largest portion of the total cost.

2.2.2 Software

To automatically insert the collected data into the custom HydroServer, an HTTP GET request is transmitted over the GPRS network. On the server, a PHP script (i.e. “parsetest.php”) parses the data into an SQL query which inserts the metadata and data into the MySQL version of the Observations Data Model (ODM) (Horsburgh, et al., 2009) with front-end access through HydroServer. This workflow is shown in Figure 3. HydroServer provides a simplistic, web-based user-interface for adding, editing, and deleting raw data and metadata in the ODM (Conner et al. 2013; Kadlec and Ames 2012). Notwithstanding its low cost and simplicity, HydroServer provides

many useful features, including a web map interface showing the locations of data-collection sites, graphical and tabular representations of the collected data, and download functionality.

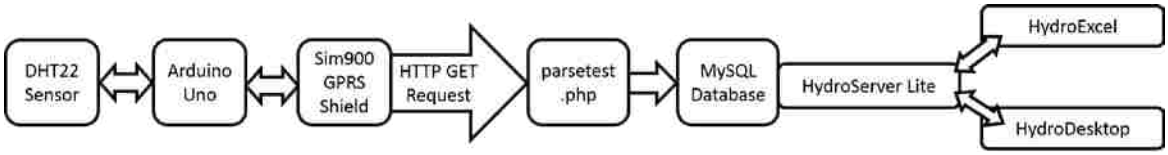

Figure 3. Workflow of data from sensor to user

In addition to being free and simple to use, my custom HydroServer is an endpoint for web services including CUAHSI WaterOneFlow services. The HIS provides a well-defined system and several software tools to make data easily discoverable and downloadable in standards-compliant WaterML1.1 format through WaterOneFlow web services. Through these services the data on a HydroServer can be accessed through REST and SOAP services, making the data discoverable and downloadable by client tools such as HydroExcel and HydroDesktop (Ames et al. 2012). As an endpoint for WFS HydroServer makes the data interoperable across platforms such as ArcGIS and Quantum GIS. Additionally, with WaterML1.1 and WaterML2.0, data on a HydroServer can be included in international catalogues such as the Global Earth Observation System of System (GEOSS) catalog.

2.2.3 Implementation Recipe

This section is an implementation guide for the system describing the HydroServer installation and giving step-by-step guidance for the hardware configuration. To use the PHP based HydroServer, the primary requirement is basic web-hosting with access to a MySQL

database and permissions to run PHP scripts. Once proper hosting is acquired, the HydroServer source code must be downloaded and copied onto the webserver domain. The source code and details about installation and configuration for use with the hardware in this paper are available at the project's CodePlex site (HydroServer Lite 2014).

Regarding hardware, I used two independent but similar configurations. The first test case used the DHT22 sensor and secondly I used the SDI-12 pressure transducer. Assembling the system hardware was relatively simple, requiring minimal soldering and very few tools (i.e. drill, soldering iron, solder, pin-wire connector, male-to-male jumpers). All of the parts were purchased online and are listed, excluding sensors, along with their prices and online purchasing locations in Table 1. Table 2 lists the same information for the two sensors I used.

Table 1. Individual item costs, purchasing locations, and total cost of hardware for logging, transmission, case, and solar power.

Item (where to purchase)	Cost
Arduino Uno (http://www.amazon.com/Arduino-UNO-board-DIP-ATmega328P/dp/B006H06TVG)	$16
SIMCOM SIM900 Quad-band GSM GPRS Shield (http://www.ebay.com/itm/like/151121095914?lpid=82)	$35
Miniature breadboard (https://www.sparkfun.com/products/12045)	$4
SIM card and phone plan (https://www.ptel.com/phones)	$15
MicroSD Card (http://www.amazon.com/b?node=3015433011)	$10
Voltaic 6W Solar Panel and 4000 mAh Battery (http://www.voltaicsystems.com/6-watt-kit)	$90
Pelican 1050 Clear Micro Case (http://www.bhphotovideo.com/bnh/controller/home?O=&sku=257883&gclid=CjwKEAiAj-)	$15
Total Cost	$181

Table 2. Costs and online locations of sensors used

Item (where to purchase)	Cost
DHT22 Temperature and Humidity Sensor (https://www.sparkfun.com/products/10167)	$10
SDI-12 Pressure Transducer (http://www.meas-spec.com/product/t_product.aspx?id=9023#)	~$700

The hardware system can be assembled in six steps:

1. Attaching the GPRS shield to the Arduino
2. Preparing the weatherproof box
3. Connecting the sensor to jumpers/wires
4. Wiring the sensor jumpers to GPRS shield via the mini-breadboard
5. Uploading sketch (custom source code) to Arduino
6. Connecting the system to power source

First the GPRS shield was attached to the Arduino board. This was done by simply inserting the shield male pins into the Arduino female pins so that the shield sits on top of the Arduino board. Second, since the sensors have to be outside of the weatherproof box, I drilled a hole large enough for the wires to fit through, about one centimeter wide. Thereafter, the wiring was done through this hole. Third, to connect the sensors to the GPRS shield I used solderless breadboards to minimize soldering. For the DHT22 I needed to use a connector to connect the pins of the sensor to wires. The SDI-12 pressure transducer came pre-wired, but I needed to solder male-to-male jumpers with a gauge small enough to fit into the breadboard. With the connector from the DHT22 pins and jumpers soldered onto the SDI-12 pressure transducer wires I was able to make the connections into the breadboards. Fourth I used male-to-male jumpers between the

breadboards, connected to the sensors and the GPRS shield. The connections to the sensors are shown in Figure 4 and Figure 5.

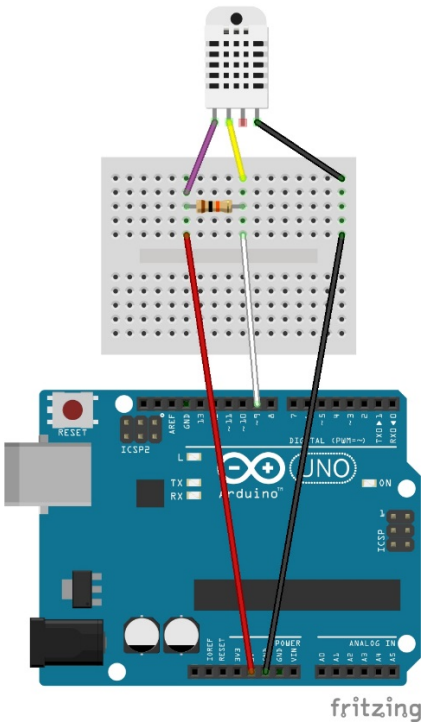

Figure 4. Wiring diagram for DHT22 and Arduino

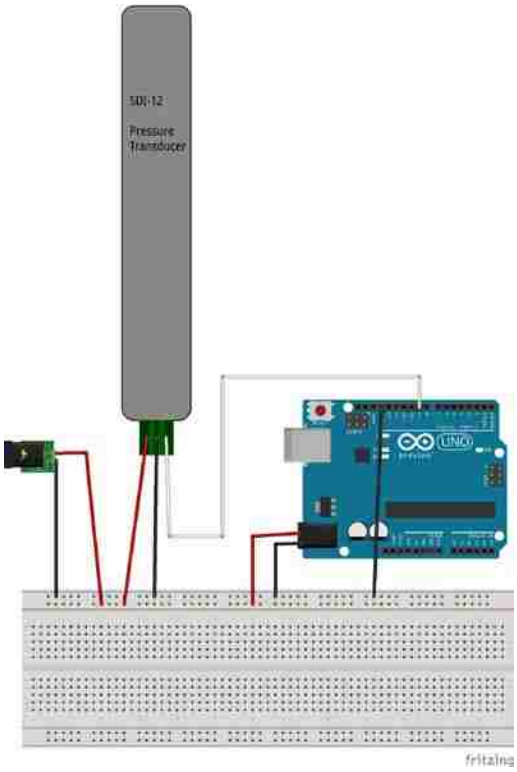

Figure 5. Wiring diagram for SDI-12 pressure transducer and Arduino

Fifth, the sketch (source code) was uploaded using an A/B USB cable. A sketch is the program that the Arduino runs. Sketches are written in a programming language derived from C++ and are uploaded using the free Arduino IDE (Arduino 2014). I used existing Arduino libraries to interface with the GPRS shield and the two sensors. For interaction between with the GPRS shield, the built-in Software Serial library was used. A third party open source library was used for the DHT22 sensor (Adams 2012). The compatibility with SDI-12 sensors was provided by a library written by the Stroud Research Center (Hicks et al. 2013; Stroud Research Center 2014). To make

the Software Serial library and the SDI-12 libraries run at the same time, a small modification had to be made to both codes. This change, more detailed instructions, pictures, and all the sketches that I used can be accessed on GitHub (Sadler 2014).

To adapt the Arduino code for a specific use, several variables in the sketch must be modified. These variables are all included in the HTTP GET sent by the GPRS shield. A proper request includes the URL corresponding to the desired HydroServer, the source id, the site id, and the variable id. It is noted that, though all the correct parameters (URL, source id, etc.) would be required for a malicious or negligent data entry into the system, there is no explicit authentication for the submission of data with the current configuration.

The final step is to connect the device to a power source. With the DHT22 I used the solar power device from Voltaic Systems (see Table 1). The panel and battery are easily connected with the adapters included with the shipment. The Arduino connects to the battery using the same A/B USB cord that connects the Arduino to the computer. To set the battery to "always-on" mode, the proper setting for continuous monitoring, I pressed and held the power button for six seconds. With the pressure transducer I used a wired power source. While the DHT22 sensor was powered directly from the Arduino board, the SDI-12 required an external power supply. In this case, I used a wall-wart to power the rails on the breadboard. The rails then powered jumpers from the sensor and wires connected to a 2.1mm center-positive plug inserted into the Arduinos power jack (see Figure 5).

2.2.4 Hardware Testing

Three deployments of the system were conducted to demonstrate proof-of-concept of the low-cost environmental sensor, the link to HydroServer, and the interoperability of collected data made possible by web services. Each deployment took place in Provo, Utah, USA and in each

deployment, data were collected every 15 minutes. In the first two deployments, the data were transmitted every hour but in the third deployment the data were sent every 15 minutes. After each deployment, the transmission success rate was recorded and reported. In the first deployment, the sensor system was deployed on the side of a building where temperature and relative humidity data were collected using the DHT22 sensor. The deployment was powered using the solar panel and battery described above. In this deployment the collected data were received by the webserver and simply saved to a text file.

In the second deployment, the sensor system was placed on the roof of a three-story university building on the Brigham Young University campus. As in the first deployment, temperature and relative humidity values were collected by the DHT22 sensor and the unit was powered by the solar panel and battery. Unlike the first deployment, however, the data in this deployment were inserted automatically into the ODM and thus were made accessible in HydroServer.

The third deployment was conducted on the Provo River. There, the water level in a stilling basin was recorded by the SDI-12 pressure transducer. The location was the Harbor Drive gauging station in Provo run by the Central Utah Water Conservancy District (CUWCD). The sensor used by the CUWCD is a guided-wave radar level sensor, and their data is transmitted via radio. In this deployment, the low-cost sensor system was powered using the CUWCD power source. The values collected and transmitted by the CUWCD were compared with the values collected by the system presented in this paper as a method of proving validity.

2.2.5 Data Interoperability Testing

To validate the open hardware/open software interconnectivity via custom PHP scripts and the HydroServer ODM database schema used in the second and third deployments, data search

and discovery tasks were performed using two distinct data access systems: HydroDesktop and the open source geographic information system, Quantum GIS (QGIS 2014).

HydroDesktop was used to test WaterOneFlow and WaterML 1.1 web services access to the collected data. With the WSDL URL of the HydroServer, HydroDesktop was used to gather the metadata on the server including the Harbor Drive metadata (site geolocation, variables collected at the site, etc.). Once the metadata were gathered, the sites were discoverable data sources within the program. The Harbor Drive site was discovered and represented as a point on an online basemap using the HydroDesktop search function based on spatial extents, variable, and time frame (i.e. "Utah County", "Stream Level", and "7/1/14-8/31/14" in this case). HydroDesktop provides several data viewing and processing tools including statistical tools such as an R portal, and a plotting tool. After the Harbor Drive site was discovered on the HydroDesktop map, the data were downloaded into the program by clicking a link in the Harbor Drive site popup window. The data were then plotted using the HydroDesktop plotting function.

Within Quantum GIS, my objective was to view the Harbor Drive site as a GIS feature through WFS and to access the WaterML2.0 document of the time series data at the site. To use WFS in Quantum GIS, the WFS URL provided by HydroServer was input into the 'Add WFS Service' function. The service added features according to collected variable. I added all the sites where water level was collected. After the service call was complete, these points were added to the map as a new feature set. The attribute table of the new feature set, including the Harbor Drive site, contained URLS for each site corresponding with WaterML2.0 time series documents. Results of the HydroDesktop and Quantum GIS testing are provided in the Results and Discussion section below.

2.2.6 Usability Testing

To test the usability of the system, data from a user survey were gathered from a K-12 teacher's workshop. In the 105 minute workshop, sixteen teachers were organized into four groups and were given instructions and materials to assemble a temperature/humidity sensor system. At the end of the workshop the teachers were given surveys. Fourteen of the sixteen teachers returned the survey. Eight of the teachers taught high school classes (grades 9-12) and six taught in middle schools (grades 6-8). All but one teacher indicated that they regularly taught sciences and one answer was left blank. The survey questions were focused on competence, difficulty, perceived educational value of the technology, interest, barriers, and applications in the classroom.

2.3 Results and Discussion

Results of the hardware, data-interoperability, and usability tests described above are provided in this section.

2.3.1 Hardware Testing Results

First Deployment

The first deployment, shown in Figure 6 was conducted over a ten-day period during which, 1696 data points were collected and transmitted via HTTP request to the webserver. The data, including the timestamp, were stored in a text file on the webserver. With ample sunlight on these days, the sensor was able to collect and transmit data without losing power throughout the test period. A graphical representation of the temperature and humidity data is shown in Figure 7. After retrieving the data, I noticed that the temperature values were invalid when the temperature was below zero, due to a problem in the library I used. That said, I was not overly concerned with this because the focus of this deployment was not on the actual environmental data collected by

the sensors. Rather, I focused on the success of the data logging and transmission and the interoperability of the data once received on the server.

Figure 6. Sensor in first deployment location. The temperature sensor, not visible here, is external to the case.

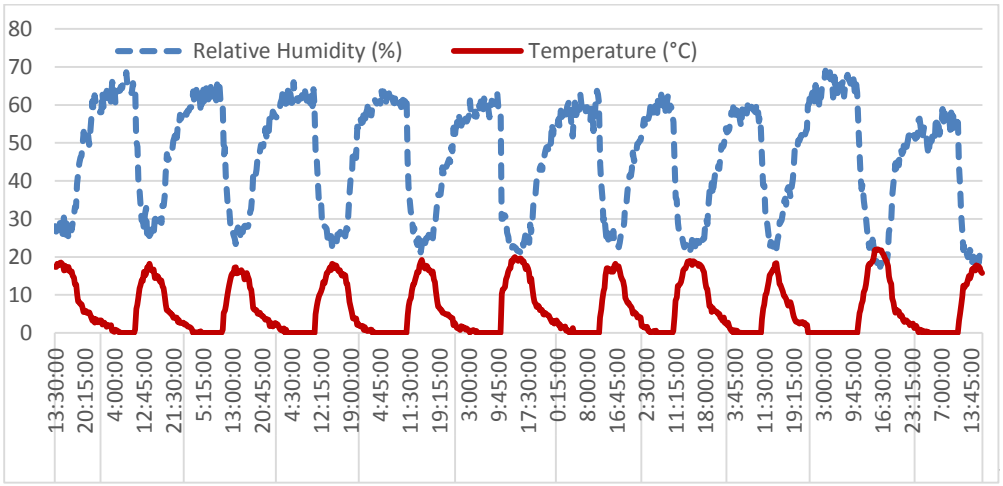

Figure 7. Temperature and humidity values collected by sensor in first deployment

In the ten-day period, there were 30 instances in which the data were not successfully transmitted to the server. This a transmission success rate 96%. Of the 30 instances, a failure in consecutive hours only occurred three times. As an attempt to address this issue, a redundancy was written into the Arduino code. With the redundancy, at each hour the GPRS modem transmitted the data for the past two hours. In this way, each set of data was transmitted twice in case of a problem with one of the connections.

Second Deployment

The second deployment was conducted over a 22-day period in which 4,248 data points were collected for both relative humidity and temperature. Only 8 transmission failures occurred out of the 2,132 attempts. This corresponds to a 99.6% transmission success rate. This increase in success rate, 96% to 99.6%, suggests that the redundancy was effective. The data in this deployment were successfully inserted into the MySQL ODM and were thus accessible by HydroServer and the CUAHSI HIS. A graphical representation, produced by HydroServer, of the data collected in the second deployment is shown in Figure 8.

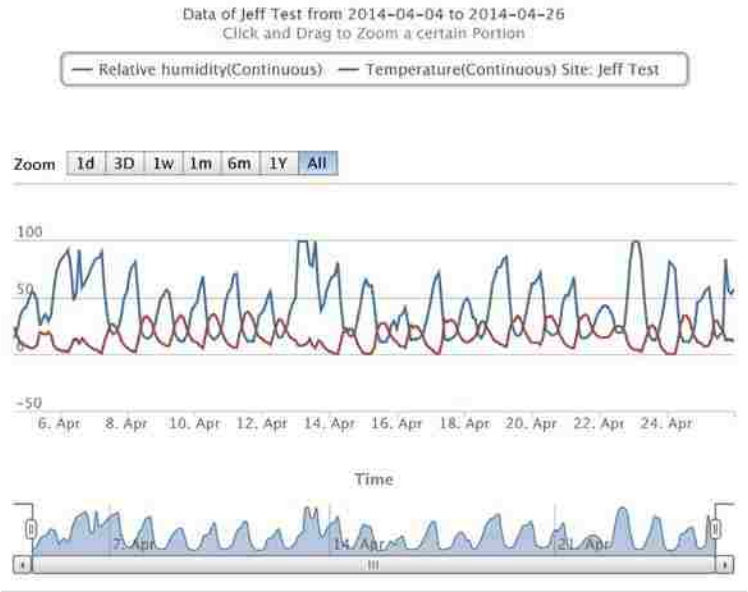

Figure 8. Plot of data collected in second deployment produced by HydroServer.

Third Deployment

The third deployment was conducted over a 40-day period in which 3,380 data points of stream level were collected. The overall transmission success rate for this deployment was 88.16%. In the forty days, twice the transmission stopped and needed to be manually reset. These two periods of no data lasted for 48 hours and 45 hours respectively. The reason the transmission failed in these periods is uncertain. Several factors may have contributed, including an error in the cell phone network, loss of power, or some malfunction in the GPRS board. Not taking into account the two extended periods of transmission failure, the transmission success rate was 97.58%. The data collected by the low-cost sensor and the data collected by the CUWCD sensor are shown in Figure 9. As seen in the figure, the values are very similar. When plotted against each other, Figure 10 results. The r-squared value of the two data series is very high, 0.9963.

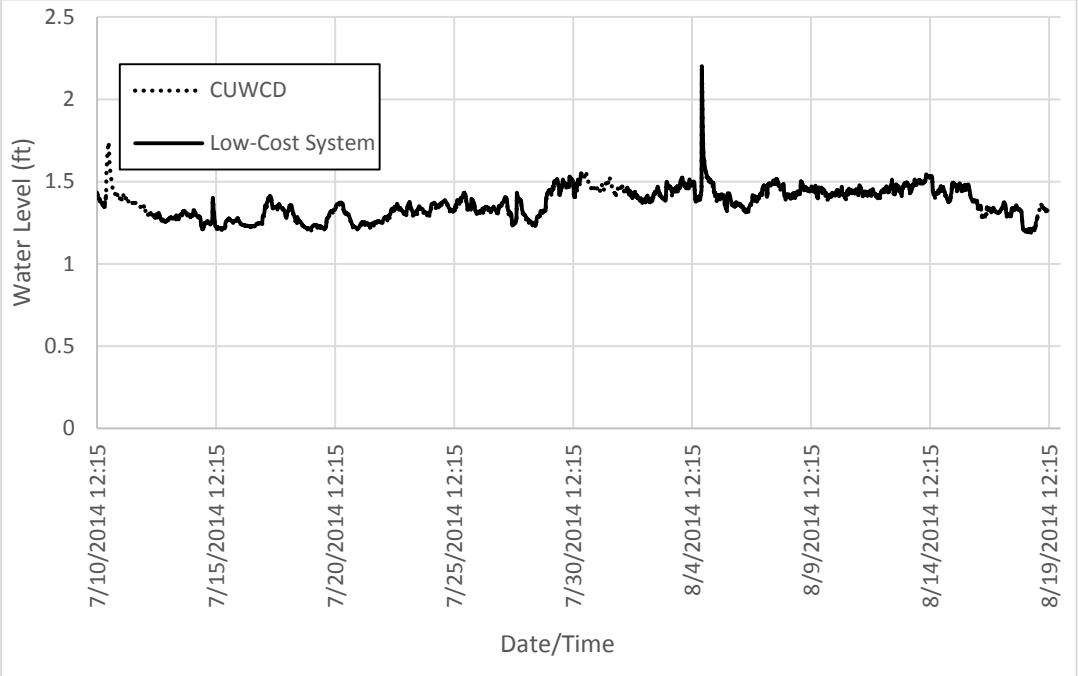

Figure 9. Stream level as reported by low-cost system and CUWCD system

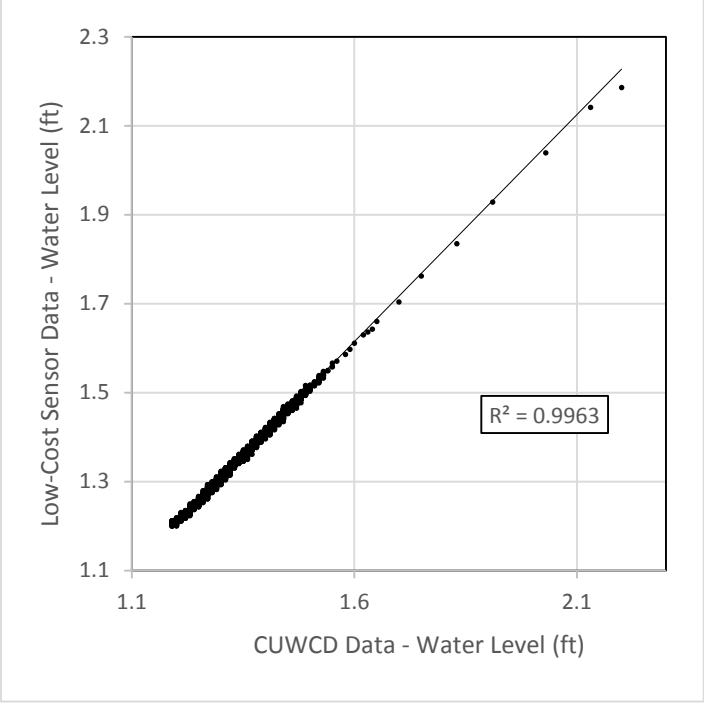

Figure 10. Low-cost system and CUWCD system values plotted against each other

2.3.2 Data-Interoperability Testing Results

Data interoperability testing using HydroDesktop and Quantum GIS yielded positive results indicating that the custom, OGC standards-based HydroServer, was successfully coupled with custom Ardiuno and PHP scripts. In both cases (HydroDesktop and Quantum GIS), all Harbor Drive data stored within the ODM database were retrieved from the custom HydroServer with zero percent loss.

Figure 11 shows a screen capture of the HydroDesktop software having completed a data search in the area of Harbor Drive. Successful execution of the search is illustrated by the appearance of an icon at the proper latitude and longitude of the Harbor Drive site. An informative "pop up" bubble with the site name and a data download link appears when hovering the mouse cursor over the icon. By clicking the download link, I was able to successfully retrieve the WaterML 1.1 document representing the data logged at this site. A plot of these data, as generated by HydroDesktop, is shown in Figure 12. Retrieval and visualization of the Harbor Drive dataset via HydroDesktop indicates successful "closing the loop" from data collection, logging, transmittal, server storage, and web-services based access.

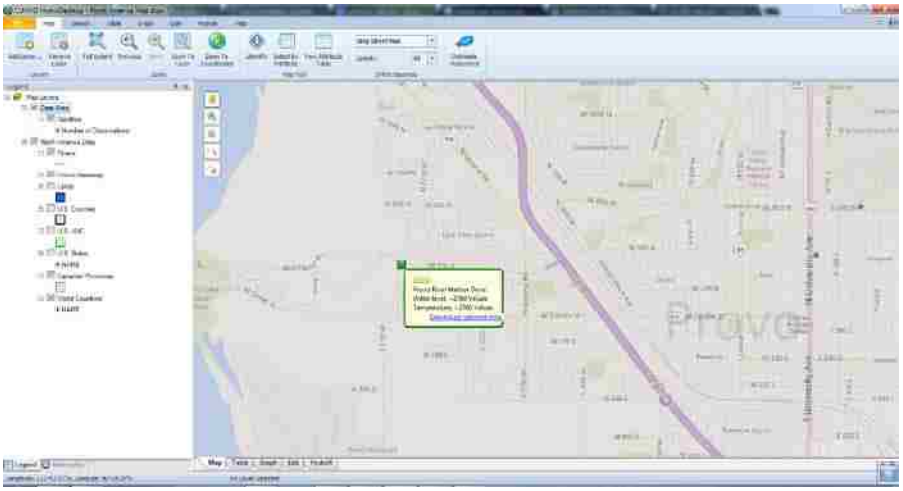

Figure 11. Harbor Drive site in HydroDesktop

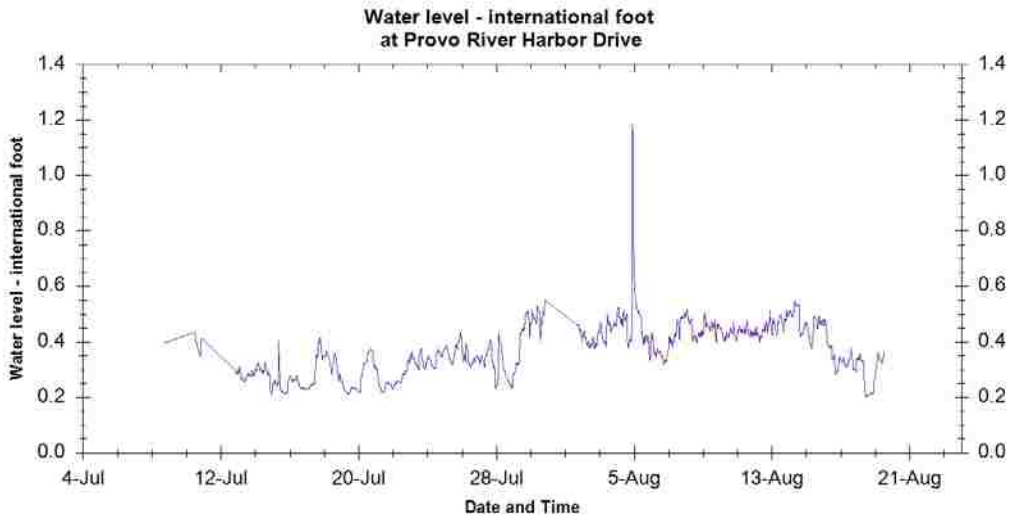

Figure 12. HydroDesktop plot of data at Harbor Drive site

Successful data discovery using Quantum GIS is illustrated via the software screen capture shown in Figure 13. Here I registered the WFS feature data end-point within Quantum GIS and successfully retrieved a single point feature that is displayed in the map at the correct Harbor Drive location. Next I open the Quantum GIS attribute table viewer for this point feature and observe the site metadata (stored as feature attributes) including a direct link to the OGC compliant WaterML 2.0 document containing the Harbor Drive data. The link is provided as a REST URL endpoint. This URL was then used to successfully retrieve the WaterML 2.0 document from the custom HydroServer.

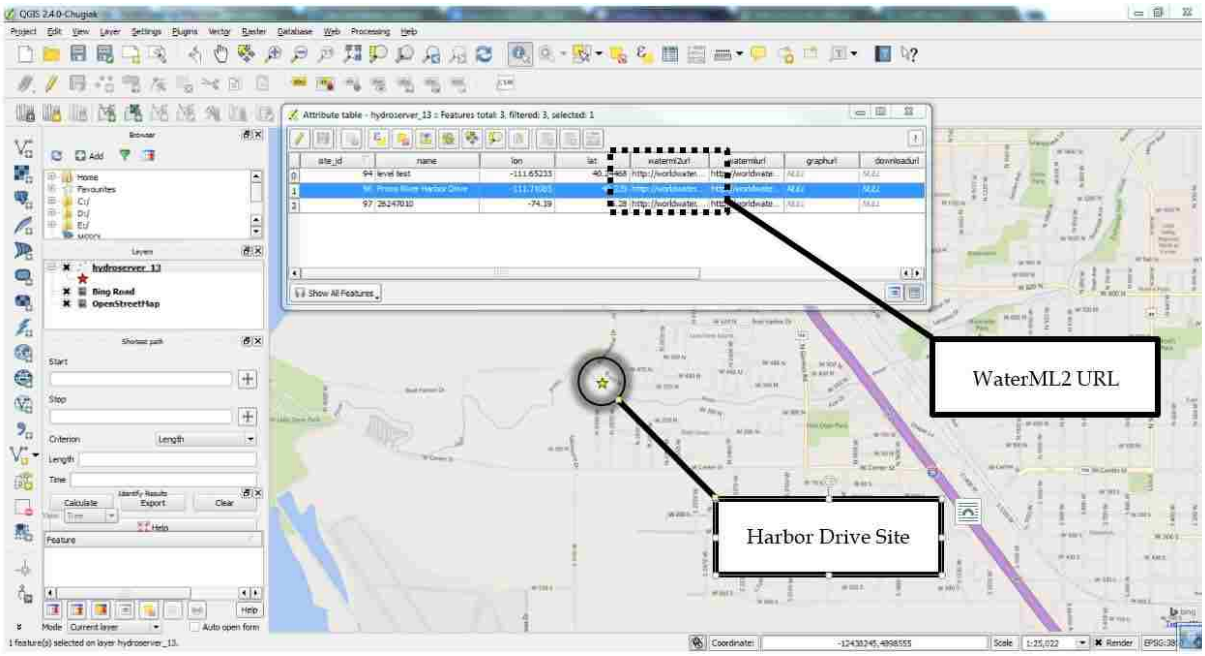

Figure 13. Harbor Drive site and WaterML2 link in QGIS

2.3.3 Usability Workshop Results

The usability workshop gave some insight into the difficulties in assembling a low-cost hardware system for environmental data collection. In the usability workshop survey both quantitative and qualitative questions were used. The quantitative questions were based on a six number scale (1-6). On the scale one corresponded to “less confident,” “less difficult,” “less interested,” and “less valuable,” and six to “more confident,” etc. Because of the small sample size I cannot draw any quantitative conclusions; however, some qualitative results were interesting. Perhaps most notable were the teachers’ responses when asked about the most difficult part of the exercise. The majority (11 of 14) reported that getting the software working was the most difficult part. This part of the exercise included downloading the libraries needed for the DHT22 sensor and the real-time clock. To use these libraries, they had to be unzipped, renamed, and placed in the libraries folder of the Arduino IDE. This process was difficult for the teachers. You may find

it easier *not* to use odd page breaks in the text but rather insert blank pages where needed at the end of chapters to ensure that the next chapter begins on an odd page.

2.4 Chapter Conclusions

Hydrologic data logging and transmission hardware as well as software to curate and make collected data shareable can be prohibitively expensive. Often data interoperability is a problem. This paper presents a solution including a recipe and test cases using open hardware for data logging and transmission and the automatic ingestion of its data into the free and open source CUAHSI HIS. Excluding sensor, the total cost of the data logging and transmission hardware, including solar powering system, was $181. The system was successful as a low cost solution for gathering environmental data, viewing that data in near-real-time, and storing the data in a standards-based system allowing for easy access and sharing through the widely used WaterOneFlow web services and WFS services.

The results did reveal some shortcomings. The failure of data transmission, likely caused by either loss of power or loss of cell phone reception, was the primary area that could be improved in the system. Despite some transmission failure, the lowest transmission success rate of the system was over 88%. Programming Arduino to react according to the HTTP Response or using SMS for data transmission are possible solutions to this problem. The autonomy of the GPRS shield was also a problem in the third deployment. To make the hardware deployment more usable, the workshop results suggest focusing on the software needed to run the Arduino. Providing a downloadable version of the software, sketch, and any other needed external libraries would be a way to simplify this process in the future.

The results serve as a proof-of-concept for a low-cost hardware system that automatically inserts data into a standards based hydrologic information system for data distribution. Future work in this area should include additional testing and refinement of the approach presented. Long-term testing would need to be performed to assess long-term deployment reliability. The use of many nodes on the same system is another area that should be tested. Finally, the integration of collected data into CUAHSI's newest software, HydroShare, as a Referenced Time Series resource would be a beneficial use of this technology.

2.5 Chapter Acknowledgements

I thank Measurement Specialties in Virginia for the generous donation of the KPSI 500 Pressure Transducer and Central Utah Water Conservancy District for their sharing of their stilling well, data, and time. This work was partially supported by the National Science Foundation under collaborative grants OCI-1148453 and OCI-1148090 for the development of HydroShare (http://www.hydroshare.org). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

3 HYDROLOGIC TIME SERIES DATA AS SOCIAL MEDIA

3.1 Chapter Introduction

The emergence of the so-called fourth scientific research paradigm, data-intensive science, has brought with it significant challenges and opportunities that are magnified by rapid increases in the availability of data and data management tools. (Hey et al. 2009; Piasecki et al. 2010). In recent years, scientific data have increased in quantity due largely to the proliferation of sensor networks and remote sensing technologies (Borgman et al. 2007). Additionally, the heterogeneity of scientific data raises challenges for collaborating scientists of various disciplines (Edwards et al. 2011). Such challenges are alleviated in part through the development and adoption of formalized data standards and standards-based data management tools as has been done in the hydrologic sciences community (Botts et al. 2008; Valentine et al. 2012). The ability to aggregate and distribute heterogeneous data across disciplines enables the use of ever more sophisticated models addressing ever more complex challenges such as global climate variability, the effects of urbanization, and decreasing availability of water (Michener et al. 2012).

3.1.1 Scientific Data Exchange

Recent advances in water data cyberinfrastructure have drastically simplified the storage and exchange of water related data by individual researchers and large agencies alike (Beran and Piasecki 2009; Conner et al. 2013; Horsburgh et al. 2009; Morgenschweis 2011; Tarboton et al. 2013). DataONE is an informative example of these systems as a platform designed to facilitate

scientific discovery and sharing by cataloging existing data repositories (Michener et al. 2012). Similarly, MyExperiment, iPlant, and SEAD are projects focused on the curation and sharing of scientific workflows, biologic data, and sustainability data respectively (Goble et al. 2010; Goff et al. 2011; Plale et al. 2013).

Another system gaining in use is the Hydrologic Information System (HIS) developed and maintained by the Consortium of Universities for the Advancement of Hydrologic Sciences Inc. (CUAHSI). The HIS is designed for the storage and exchange of hydrologic time series data. A service oriented system, the HIS consists of three main parts: HydroServer, HIS Central, and HydroDesktop (Ames et al. 2010; Ames et al. 2012; Tarboton et al. 2009). HydroServer hosts the data and provides a front-end interface to the database where the data are stored (Conner et al. 2013). Additionally, the PHP/MySQL version of HydroServer accommodates the insertion of automatically transmitted sensor data via web application programmer interface (API) (Kadlec et al. 2015; Sadler et al. 2015) and supports OGC Web Feature Services (Michaelis and Ames 2012) and WaterML 2 (Taylor et al. 2010). The second component, HIS Central, catalogs the time series data stored on registered HydroServers making them discoverable to client tools. Finally, with HydroDesktop, the main client tool of the HIS, users can discover, visualize, and analyze HIS time series data (Ames et al. 2012). To exchange the data between the components CUAHSI has developed WaterOneFlow web services and the WaterML1.1 format. Incrementally, however, servers are moving to the Open Geospatial Consortium (OGC) standard, WaterML2 (Taylor et al. 2013) for data transfer and OGC Sensor Observation Service (SOS) (Botts et al. 2008) for server queries.

3.1.2 Data Sharing

Current web technologies include many services for efficient data exchange such as email, cloud services, and file transfer protocol (FTP). CUAHSI HIS is an example data exchange service intended specifically for scientific data. While systems and technologies for scientific data exchange have been well-developed, major scientific advances may require more in terms of data sharing than the mere exchange of data. I suggest that a more complete and useful notion of scientific data sharing should include at least the following components:

1. A mechanism to efficiently archive, discover, and retrieve of data (Piasecki et al. 2010)
2. Extensive creator and science metadata including versioning, licensing, and provenance information (Bechhofer et al. 2013)
3. A persistent digital identifier such as a Digital Object Identifier (DOI) (Paskin 2008)
4. A system for facilitating scientific discussion and collaboration around the data.

In the hydrologic sciences, much has already been accomplished regarding the first three components of this list, as evidenced by the volume of research already cited herein. However, very little headway has been made in terms of facilitating the open scientific discussion and collaboration around data. One notable exception is the recent policy of the online journal, PLOS-One requiring all authors to make their data publicly available upon publication (Winter 2014). This data publishing requirement combined with the open discussion model supported by the journal web site (i.e. readers can post comments on any article) provides a framework for open collaboration on data associated with publications.

A similar collaboration and data sharing support system for data associated with pre-publication research has the potential of increasing the speed of scientific advances (Tarboton et al. 2013). An example of such a system is iPlant. In the iPlant system users upload research data, and can collaborate by sharing that data with other iPlant users or groups. Any file or folder on the system can be discussed in a structured comment management system (Goff et al. 2011). Similarly, D4Science.org users can share "updates" to which datasets can be attached on a public news feed. These updates can be discussed and "favorited" (Assante et al. 2014). In this way iPlant and D4Science provide a space for discussion and clarification regarding individual or groups of datasets which are not necessarily associated with published journal articles. This type of collaboration is expected to enhance and accelerate the research and publication process as scientists can present and share their research data in a low-risk environment. The social networking features noted here are similar to those found in most common social network websites (i.e. providing comments, indicating favorites). Additional social-network functionality that could, if applied to scientific research domains, aid research productivity includes the concepts of "following" and forming groups.

3.1.3 HydroShare as a Scientific Social Media Platform

HydroShare is an NSF-funded partner project to CUAHSI HIS and is being developed as a data sharing system for static datasets (e.g. files on disk rather than dynamic database-driven HIS datasets). It is designed to provide scientists each of the four functionalities of a data-sharing network described above including the ability for datasets to be treated as social media. The HydroShare project is focused on providing a simple web-based data sharing platform for hydrologists to discover and access hydrologic data and models. The HydroShare project intends to "…make sharing of hydrologic data and models as easy as sharing videos on YouTube or

shopping on Amazon.com" (Tarboton et al. 2014). An additional goal of HydroShare is to provide users with cloud and/or distributed computing resources for data and model analysis.

A major use case for HydroShare is shown in Figure 14. In the figure, a primary investigator (1) has collected data whether by observation or by instrumentation. This person requires the help of his or her collaborators in the process of analyzing and modeling with the collected data. The primary investigator enters the data into HydoShare. In HydroShare he or she can give access to the data to each of his or her collaborators (2, 3, and 4). The researcher team now uses HydroShare's analysis and modelling tools to iterate over the data until they have required results. The results and the data to reach them can then be published (5) at which point the data sets become immutable and are assigned persistent identifiers. This type of use case has driven, at least in part, the development and design of the HydroShare system. In addition to these collaborative workspace functions, the HydroShare design also supports the concept of data as social media. As such, data can be interacted with by third parties who can comment on a dataset, mark a dataset as a favorite, receive notifications when new data are published by the data owner, and contact the data creator for further discussion.

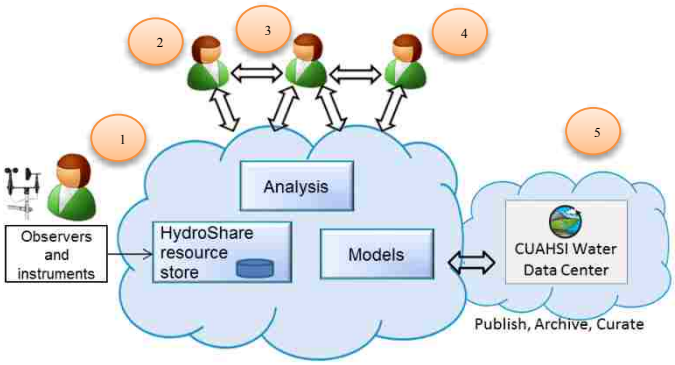

Figure 14. Standard HydroShare use case including data publication, retrieval, and analysis

The core functionality of HydroShare is facilitated primarily by two third-party software frameworks: the Django web application framework and Integrated Rule-Oriented Data System (iRODS). The HydroShare web interface and social media functionality are built using Django – a Python-based web application development framework (https://www.djangoproject.com/). Django provides a database abstraction layer which, in the case of HydroShare, connects to a PostgreSQL database where social metadata and application-level information is stored. The iRODS system – a distributed, scalable file management system (https://github.com/irods/irods) stores the HydroShare resources (e.g. the actual files containing science data) together with science metadata, and the user accounts. The iRODS layer also manages federated data – allowing HydroShare to scale through the use of cloud-based or other servers. An overview diagram of the HydroShare architecture is shown in Figure 15.

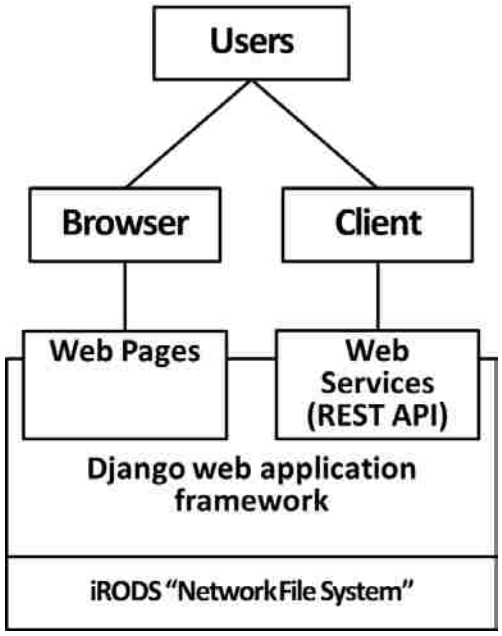

Figure 15. Overview of HydroShare architecture

The design of HydroShare is based around the concept of a "resource" as a basic unit of data. HydroShare resources are designed in terms of a formal Resource Data Model based on the Open Archives Initiative's Object Reuse and Exchange (OAI-ORE) standard (Lagoze et al. 2008). A major objective in designing the Resource Data Model was to accommodate the creation and integration of different, specific data types, while nevertheless maintaining metadata common to all data types. HydroShare uses the Dublin Core metadata standard (http://www.dublincore .org/documents/dcmi-terms) as the base metadata definition for all resource types. The Dublin Core metadata terms are generic enough to apply to any resource and important enough to merit inclusion in all resources.

While each resource data type in HydroShare requires some Dublin Core terms, certain resource data types require additional metadata terms. For example, if a user were adding time series data to HydroShare, the measured variable should be specified and associated with the data to maintain their usefulness and integrity. Since "measured variable" is not one of the Dublin Core terms, a new metadata term would be needed. This type of metadata extension is accommodated in HydroShare by allowing each data type its own data model wherein extended metadata terms can be established. This is accomplished through the use of Django apps. Each data type in HydroShare is contained within its own Django app which is included in the main HydroShare site. In this way HydroShare provides for extensive science metadata.

In addition to the Dublin Core metadata, common to all resource data types is the creation of a science metadata document and the use of the BagIt format. The science metadata document is an XML file to which the Dublin Core and extended metadata terms are written. An example science metadata document is shown in Figure 16. BagIt is a data compression and archiving specification originally developed for use by the U.S. Library of Congress (Boyko et al. 2011). A

Bag is created for each HydroShare resource. Each Bag has a standard file structure that includes the resource content (e.g. resource files), the scientific metadata document, and the resource map document. An example of this file structure is shown in Figure 17. It is the BagIt format, which standardizes the system and science metadata documents and ensures the consistent maintenance of the science and creator metadata.

A number of resource types have been proposed and/or developed for HydroShare including: generic resource, geographic feature resource, static time series resource, raster data resource, LiDAR resource, document resource, and aggregate resource. A resource may be a single observation (e.g. an aerial image or single streamflow measurement) or a collection of data (e.g. a NetCDF file with multiple observations or a streamflow time series). In each of these cases, the HydroShare resource is stored in iRODS as a file on disk. In this way, HydroShare is not entirely unlike any other file sharing system (e.g. FTP), though highly augmented with structured metadata, data type specific value added functionality, and social networking capabilities.

Figure 16. Example science metadata document

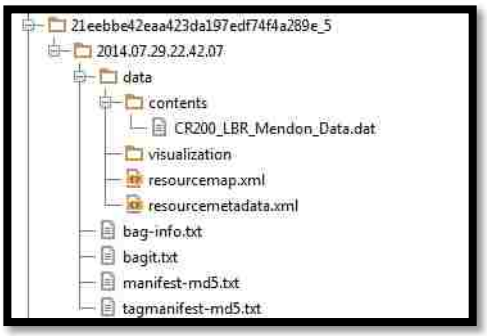

Figure 17. BagIt file structure in HydroShare

HydroShare exposes social media functionality to provide a system for scientific discussion and collaboration. In HydroShare participating scientists create user profiles which can include a profile picture, a listing of research interests, and attached CV. A HydroShare user has several options for sharing his or her resources which include sharing with no one, allowing only certain users to view, edit, and/or own the resource, and publishing the resource for access by all HydroShare users and non-users alike. In coming releases HydroShare will also support forming custom groups made up of users who may be collaborating on the same project or who may all have similar research interests. Another social media functionality is the ability to discuss and rate resources. A HydroShare user can comment on all resources to which he or she has access regardless of whether the resource is a public resource. User profiles, groups, and informal communication around HydroShare resources facilitate an open environment of scientific discussion and collaboration (Tarboton et al. 2014).

Kaplan and Haelin (2010) characterize social media in terms of six categories according to self-presentation/self-disclosure and social presence/media richness. HydroShare's social media design combines elements of social networks (i.e. creation of groups, and a user profile which allows for professional self-disclosure), collaborative projects (i.e. collaboration around content to

produce a more complete and better result), and content communities (i.e. upload, sharing, and discoverability of research data). This combination social network, collaborative project, content community is uniquely suited to aid hydrologists in their individual and collaborative research.

3.1.4 Melding HIS and HydroShare

HIS presents a useful solution to the challenge of storing and exchanging time series data in an efficient, standardized way. What is lacking, however, is the ability to interact with these time series in a web-based system that supports the expanded notion of scientific data sharing that I described above. With the current HIS web services, time series can be easily retrieved; through HydroDesktop users can discover HIS time series and retrieve them to their desktop computer for analysis. CUAHSI has also recently provided limited web functionality through http://data.cuahsi.org/. This service gives users a web map interface for discovering and downloading HIS time series. However, like HydroDesktop, the new web site currently only supports data retrieval. These tools do not offer any kind of online interaction with the data themselves, and do not provide for social media-style scientific collaboration and discussion.

The remainder of this paper presents the design, development, and testing of a new software system intended to fill the need for web-based HIS functionality, online collaboration, and social networking for HIS time series data (i.e. turning an HIS time series dataset into social media.) This connection is made by the creation of a new resource type in HydroShare that is unique from any of the existing or proposed HydroShare resource types: an HIS Referenced Time Series resource type – a resource that, rather than storing data in iRODS as files on disk, retrieves data dynamically from one of the many existing HydroServers in the HIS network (Figure 18). As

HydroShare resources, HIS time series, for the first time, will be digital social objects that can be discussed, endorsed, followed, and shared – effectively becoming social media. Furthermore, the referenced HIS time series data will also benefit from the HydroShare data curation features such as provenance and attribution metadata and a persistent identifier.

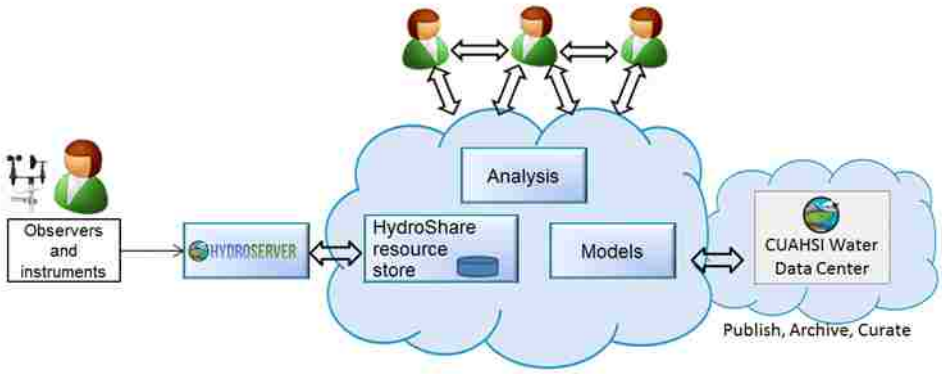

Figure 18. Overall workflow of HIS referenced time series data type

In the following sections, I describe how the HIS Referenced Time Series resource diverges from the generic resource data type in its extended metadata elements and unique creation process. I also describe my methods of testing for the success of the implementation and the results of the tests. Finally I provide some observations regarding meeting my initial objectives, potential extension of the system, and the possible applicability of lessons learned in this project to other data sharing efforts.

3.2 Methods

3.2.1 Design of a Referenced Time Series Data Type for HydroShare

This section describes the design of the HIS Referenced Time Series resource type. The differences between the HIS Referenced Time Series data type and generic resource data type are primarily differences in metadata and resource creation. The metadata differences are reflected in the HIS Referenced Time Series data model. The HIS Referenced Time Series data model consists of three main types of classes: the resource data type class (i.e. RefTimeSeries), the resource data type metadata class (i.e. RefTSMetadata), and the classes for the individual extended metadata terms (i.e. Method, QualityControlLevel, Variable, Site). The RefTimeSeries class inherits from the HydroShare AbstractResource class. This way HIS Referenced Time Series objects will have universally needed attributes such as "short_id" and "doi." In the RefTimeSeries class I have the option to add attributes specific to the data type. I added "url" and "reference_type." These attributes are included only in the Django database and are not included in the science metadata document.

The RefTSMetadata class inherits from the HydroShare CoreMetaData class and is the link between the RefTimeSeries class and the metadata term classes. In RefTSMetadata class "One-To-One" relations are made with each of the extended metadata classes (i.e. Method, QualityControlLevel, Variable, and Site). The individual metadata term classes are then included as supported metadata elements for the HIS Referenced Time Series resources. This way the terms' class methods can be used to create, update, and delete class instances associated with HIS Referenced Time Series resources. A key method of the RefTSMetadata and CoreMetaData classes is "get_xml." This method writes all of the metadata terms and their values and them into an XML document. The CoreMetadata "get_xml" method does this for the Dublin Core standard

metadata elements while the RefTSMetadata "get_xml" method must be written to do this for each of the extended metadata terms for the class instance.

Finally a class is made for each of the extended metadata terms. These classes inherit from the HydroShare AbstractMetaDataElement class. The AbstractMetaDataElement has the "create," "update," and "remove" methods. These methods however simply redirect to the child class methods and thus must be implemented in the child class. In these classes there is one required attribute: "term". Other attributes needed for further description can be added. For example, the Site class has the "term," "name," "code," "latitude," and "longitude" attributes. The class structure for the HIS Referenced Time Series data type is depicted in Figure 19.

To populate the extended metadata terms, the HIS Referenced Time Series data type requires a unique resource creation process. In contrast to the creation of a generic HydroShare resource, no files are uploaded when creating an HIS Referenced Time Series resource. Rather, a reference is needed that links to data stored on an externally hosted HydroServer. Additionally, to support WaterOneFlow services, users need a way to select a specific site and relevant variable. These differences in creation are accommodated by the HydroShare generic resource creation workflow which can redirect to a different page depending on the selected data type. In the generic resource creation workflow the "pre_create_resource" signal is sent when the creation process is first begun. This signal accepts an optional reference for a redirect page which the HIS Referenced Time Series "pre_create_resource" receiver returns. The system then redirects the user to the page where he or she can create an HIS Referenced Time Series resource with interactive controls for selecting a site and variable.

The implementation of the HIS Referenced Time Series design within the HydroShare-Django framework is presented in the results section 3.3.1 below.

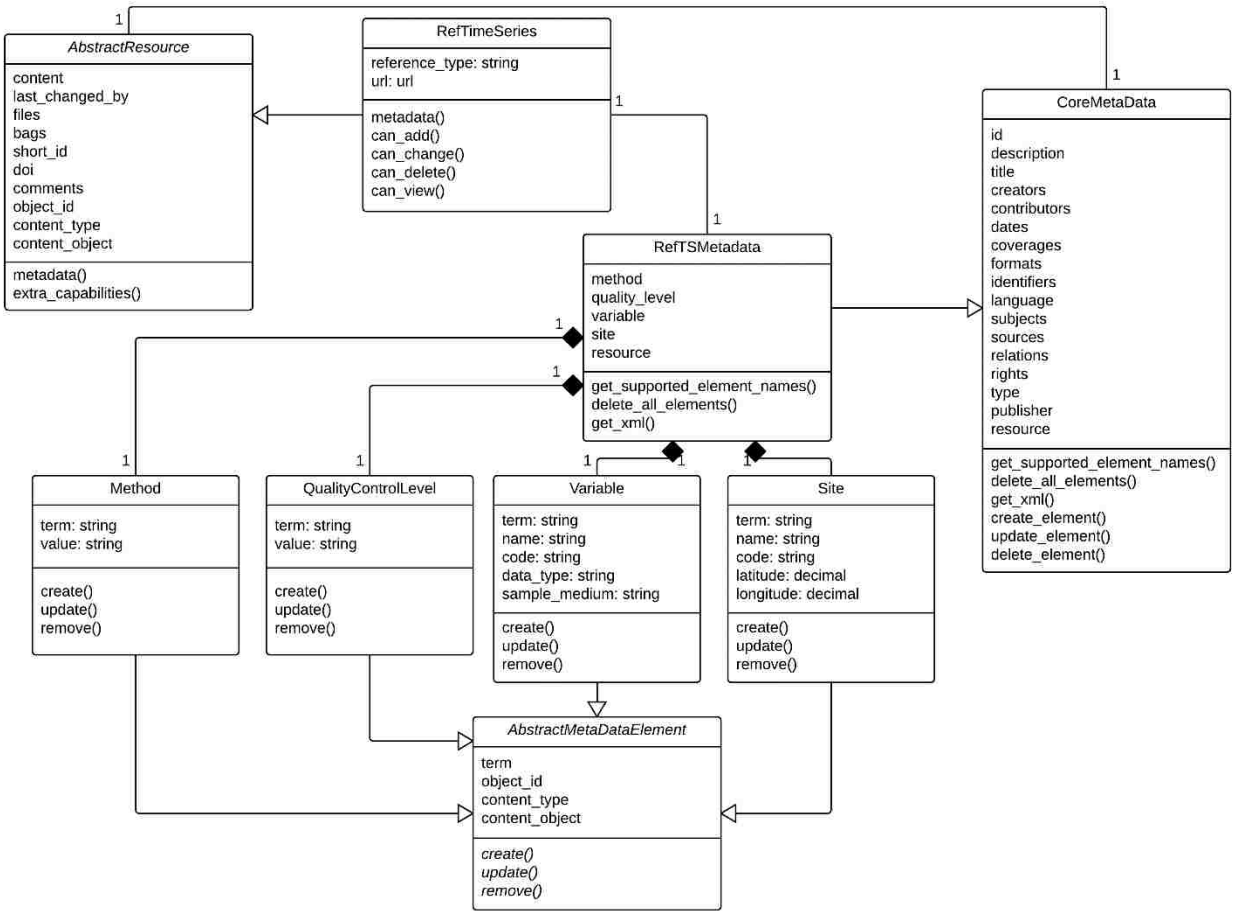

Figure 19. UML class diagram for HIS referenced time series data type

3.2.2 Case Study Testing Approach

The implementation of the HIS Referenced Time Series data type was tested in three parts: creation, access, and data integrity. Two HIS time series resources were used as test cases. One of the resources used SOAP services and the other used REST services. The two URLs used for testing are shown in Table 3.

Table 3. Reference URLS used in testing of HIS referenced time series

SOAP	http://hydroportal .cuahsi.org/SwedishMonitoringData/webapp/cuahsi_1_1.asmx?WSDL
REST	http://worldwater.byu.edu/interactive/sandbox /services/index.php/cuahsi_1_1.asmx /GetValues?location=WWO:S-PRHD&variable=WWO:JSWL &startDate=&endDate=

In both the test cases, I executed the HIS Referenced Time Series creation workflow and ascertained if the resource was created successfully. For using the SOAP services, a HydroServer storing data from Swedish water quality monitoring equipment was used. In this case I first tested the user interface for creating the resource, that is, whether the user interface process for selecting a site and a variable, given the WSDL URL, was sufficient for making a HIS Referenced Time Series resource. To test the resource creation using REST services, I used a URL from a HydroServer at http://worldwater.byu.edu/.

I evaluated the data integrity of the Referenced Time Series resource in HydroShare. This was done by creating a Referenced Time Series resource, downloading that resource and then comparing the data and metadata of the downloaded resource to the original data and metadata stored on the HydroServers. I noted any discrepancies between the two sets of data and metadata.

Finally I tested the social media functionality of the created resources. This included the sharing, commenting, and endorsing functionalities. To test the sharing, several levels of accessibility were tested. First, I tested if the resources were only accessible to the resource owner, "jsadler2," before they were shared to specific users or made public. Second, I shared one of the resources, the Water Level Provo River resource, with another user, "shaunjl." As only viewing rights were shared, I tested whether the "shaunjl" user was able to access the resource without being able edit it. I also ensured that the non-HydroShare user was still unable to access either data

set. Finally, I make the Water Level Provo River resource public and assessed whether a HydroShare user and non-user alike were able to access the data.

To test the commenting and endorsing functionality I used the public Water Level Provo River resource. A user who was not the resource owner "shaunjl," and a non-HydroShare-user made comments on the resource. A HydroShare user and non-user then endorsed those comments.

3.3 Results

This section presents the results of the implementation of the Referenced Time Series resource type in HydroShare followed by the results of case study tests outlines in section 3.2.2. These results are based on a development version of HydroShare. The Referenced Time Series resource type is expected to be included in the first full release of HydroShare version 1.0 scheduled for summer 2015. Source code for the HIS Referenced Time Series resource is available together with full HydroShare source code at http://github.com/hydroshare/hydroshare.

3.3.1 Code Implementation and Resource Creation Workflow Results

The implementation of my data model was done in the models.py file in the HIS reference time series Django app. In models.py I defined all of the classes described above (RefTimeSeries, RefTSMetadata, Method, QualityControlLevel, Variable, and Site). Figure 20, Figure 21, and Figure 22 show the implementation of the RefTimeSeries, RefTSMetadata, and Site classes respectively.

```
class RefTimeSeries(Page, AbstractResource):

        class Meta:
                verbose_name = "Referenced HIS Time Series Resource"

        def extra_capabilities(self):
            return None

        @property
        def metadata(self):
            md = RefTSMetadata()
            return self._get_metadata(md)

        reference_type = models.CharField(max_length=4, null=False, blank=True, default='')

        url = models.URLField(null=False, blank=True, default='',
                              verbose_name='Web Services Url')

        def can_add(self, request):
                return AbstractResource.can_add(self, request)

        def can_change(self, request):
                return AbstractResource.can_change(self, request)

        def can_delete(self, request):
                return AbstractResource.can_delete(self, request)

        def can_view(self, request):
                return AbstractResource.can_view(self, request)

processor_for(RefTimeSeries)(resource_processor)
```

Figure 20. RefTimeSeries class definition

```
class RefTSMetadata(CoreMetaData):
    methods = generic.GenericRelation(Method)
    quality_levels = generic.GenericRelation(QualityControlLevel)
    variables = generic.GenericRelation(Variable)
    sites = generic.GenericRelation(Site)
    _refts_resource = generic.GenericRelation(RefTimeSeries)

    @classmethod
    def get_supported_element_names(cls):
        # get the names of all core metadata elements
        elements = super(RefTSMetadata, cls).get_supported_element_names()
        # add the name of any additional element to the list
        elements.append('Method')
        elements.append('QualityControlLevel')
        elements.append('Variable')
        elements.append('Site')
        return elements

    @property
    def resource(self):
        return self._refts_resource.all().first()

    def get_xml(self):
        from lxml import etree
        # get the xml string representation of the core metadata elements
        xml_string = super(RefTSMetadata, self).get_xml(pretty_print=True)
        # create an etree xml object
        RDF_ROOT = etree.fromstring(xml_string)

        # get root 'Description' element that contains all other elements
        container = RDF_ROOT.find('rdf:Description', namespaces=self.NAMESPACES)

        # inject resource specific metadata elements to container element
        for method in self.methods.all():
            hsterms_method = etree.SubElement(container, 'method' %
self.NAMESPACES['hsterms'])
            hsterms_method_rdf_Description = etree.SubElement(hsterms_method, 'Description' %
self.NAMESPACES['rdf'])

            hsterms_name = etree.SubElement(hsterms_method_rdf_Description, 'value' %
self.NAMESPACES['hsterms'])
            hsterms_name.text = method.value
```

Figure 21. RefTSMetadata class definition (partial)

```
class Site(AbstractMetaDataElement):
    term = 'Site'
    name = models.CharField(max_length=100)
    code = models.CharField(max_length=50)
    latitude = models.DecimalField(max_digits=9, decimal_places=6, null=True)
    longitude = models.DecimalField(max_digits=9, decimal_places=6, null=True)

    @classmethod
    def create(cls, **kwargs):
        if 'name' in kwargs:
            if 'content_object' in kwargs:
                metadata_obj = kwargs['content_object']
                metadata_type = ContentType.objects.get_for_model(metadata_obj)
                site = Site.objects.filter(name__iexact=kwargs['name'],
object_id=metadata_obj.id, content_type=metadata_type).first()
                if site:
                    raise ValidationError('Site for resource already exists')
                else:
                    if 'code' in kwargs:
                        site = Site.objects.create(name=kwargs['name'], code=kwargs['code'],
latitude=kwargs.get('latitude'), longitude=kwargs.get('longitude'),
content_object=metadata_obj)
                        return site
                    else:
                        raise ValidationError('code is missing from inputs')
            else:
                raise ValidationError('metadata object is missing from inputs')
        else:
            raise ValidationError('name is missing from inputs')
```

Figure 22. Site class definition (partial)

To implement the unique resource creation workflow, I used the HIS Referenced Time Series app's views.py, templates, and, unique to HydroShare, receivers.py. When the "Create Resource" button is clicked in the generic create resource page shown in Figure 23, the "pre_create_resource" signal is fired. In the HIS Referenced Time Series app, the "pre_create_resource" receiver in the receivers.py file, shown in Figure 24, receives the signal and returns the redirect URL. The system then redirects to the creation page specific to HIS Referenced Time Series shown in Figure 25.

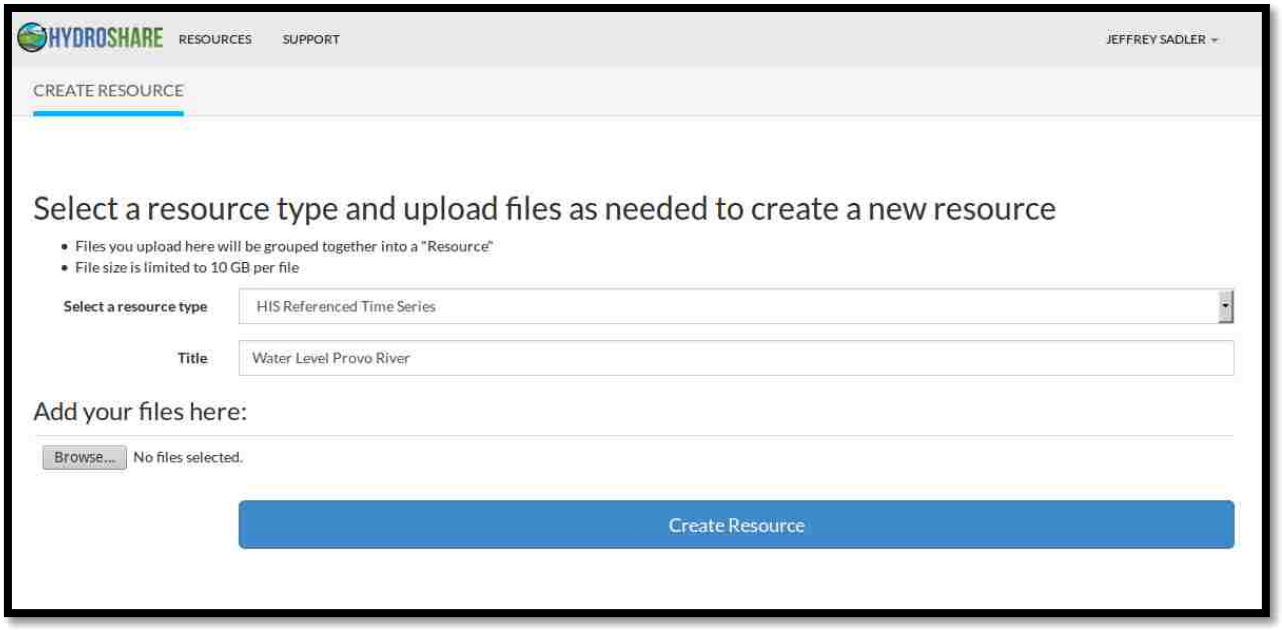

Figure 23. Generic Create Resource Page

```
from django.dispatch import receiver
from hs_core.signals import *
from ref_ts.models import RefTimeSeries

@receiver(pre_create_resource, sender=RefTimeSeries)
def ref_time_series_describe_resource_trigger(sender, **kwargs):
if(sender is RefTimeSeries):
page_url_dict = kwargs['page_url_dict']
url_key = kwargs['url_key']
page_url_dict[url_key] = "pages/create-ref-time-series.html"
```

Figure 24. Source code for the “pre_create_resource” receiver

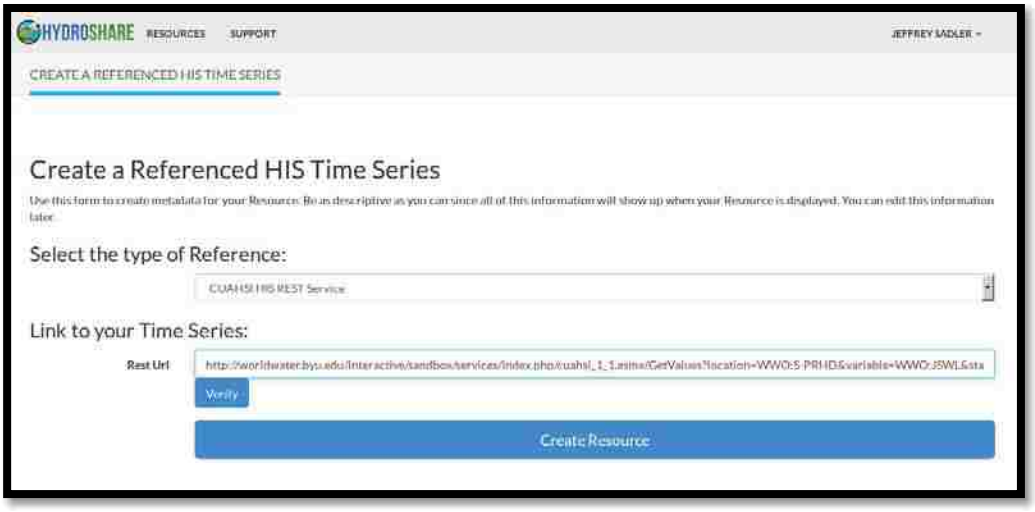

Figure 25. HIS time series resource creation page

In the HIS Referenced Time Series resource creation page, the user selects whether the HydroServer which hosts the data will be accessed via SOAP or REST services. If the user supplies a SOAP URL, HydroShare uses the WaterOneFlow methods GetSites and GetSiteInfo to retrieve available sites, and then the available variables for the selected site. When the site and variable have been selected the GetDataValues method is executed and the time series data is retrieved. The user interface for this process is shown in Figure 26. Alternatively, users can input a REST URL directly. If a REST service is used, the process is understandably simpler, requiring only the REST URL as the site and variable identifiers are provided in the URL itself. Once the time series is properly referenced with SOAP or REST services, a preview graph is shown (see also Figure 28). When the "Create Resource" button is clicked the system appends the newly entered metadata to the resource.

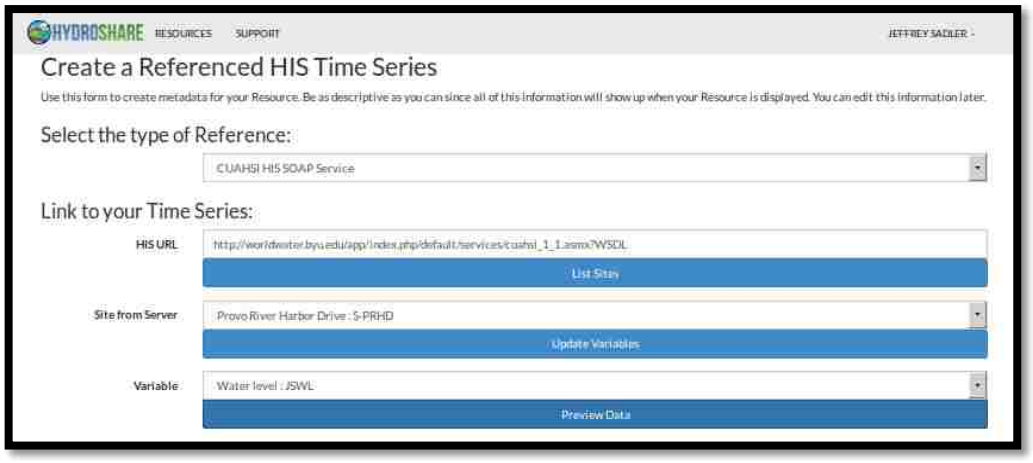

Figure 26. HIS referenced time series creation using SOAP services

3.3.2 Case Study Testing Results

In both test cases I was able to successfully create a HIS Referenced Time Series resource in HydroShare. Figure 27 shows the creation page when using SOAP services. In the figure I have selected the 'Pipbaecken Nedre' site and the 'dissolved Fluoride' variable. Figure 28 is the creation page when the user creates the resource with a REST request. This time series is water level data recorded on the Provo River, Utah, USA. In both of these cases the system was able to successfully retrieve the time series data as manifested by the successful creation and display of the preview graphs. When the "Create Resource" button is clicked, the resource is created and the system redirects to the resource landing page. Figure 29 shows a portion of the resource landing page for the water level resource. This page provides links to download individual files or the resource bag which is described above. Additionally, on this page, the resource owner can give resource access to other users and groups and can make the resource publicly available.

Figure 27. Successful data retrieval using SOAP services

Figure 28. Successful data retrieval using REST services

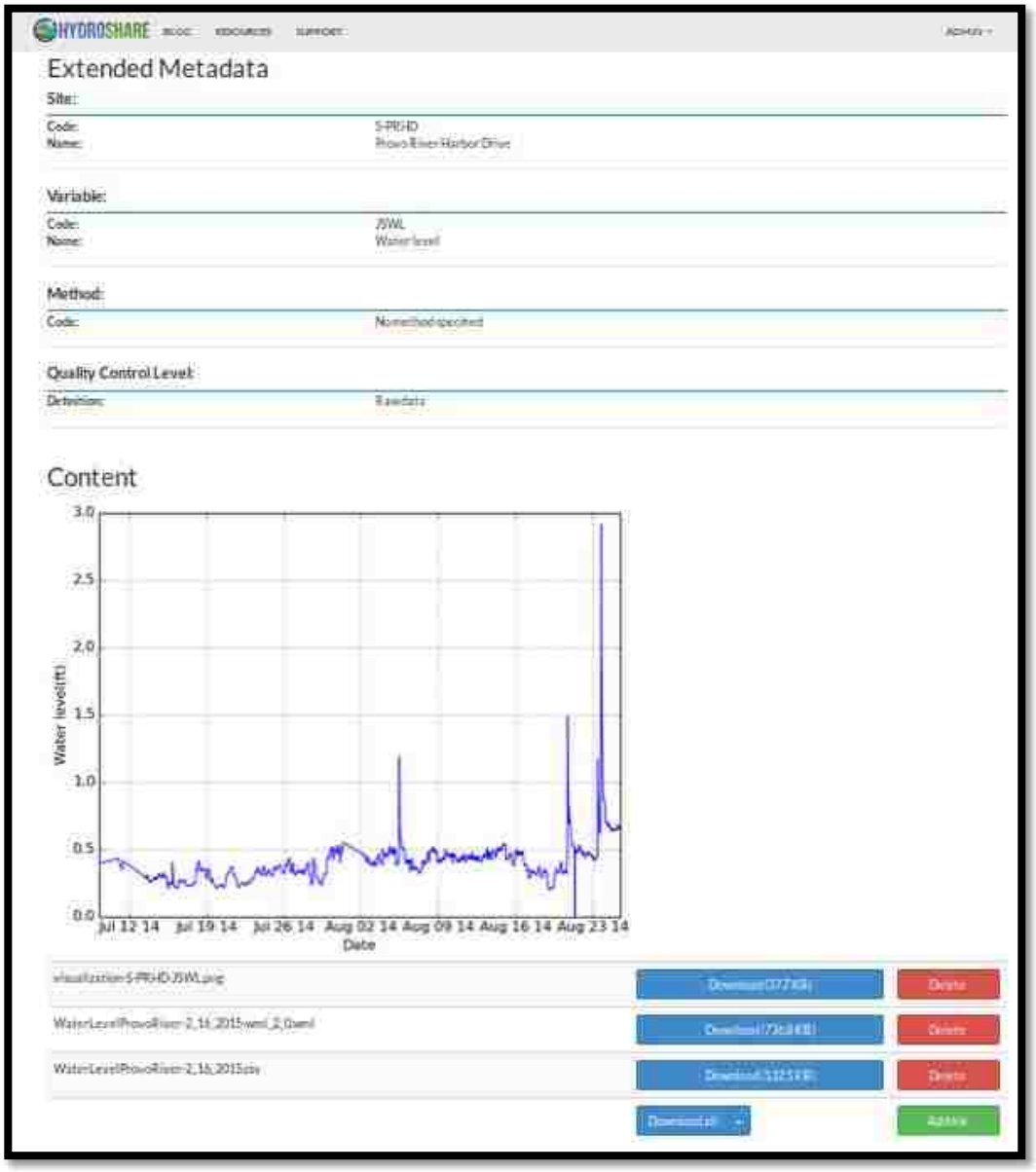

Figure 29. Resource landing page for water level time series

Data integrity was also maintained with both resources. Each of the metadata terms was correctly extracted from the service response and added to the attributes of the HydroShare resource. Figure 30 is a screenshot of the science metadata document for the "Water Level Provo River" resource including the extended metadata elements.

Figure 30. Science metadata document

The social media functionality, including sharing, commenting, and endorsing behaved as expected. The access behavior for both of the resources and their data was as expected. The owner, “jsadler2,” had complete access to his resources. Figure 31 shows the resource collection page for “jsadler2” which includes the two resources he created: “Pipbaecken Fluoride Concentrations” and “Water Level Provo River.” At this point, before either had been shared, neither of the resources appeared in the public resource collection or in the resource collection of another user, “shaunjl”. When the “Water Level Provo River” resource was shared with

"shaunjl" it appeared in his resource collection, shown in Figure 32. Although "shaunjl" was able to view the resource and access its data, he was not able to edit it in any way. Both the resources were at this point still inaccessible to the public. Finally, the "Water Level Provo River" resource was made public. Figure 33 shows the public resource collection after the "Water Level Provo River" resource was made public. As expected the "Water Level Provo River" resource is accessible to non-HydroShare-users but the "Pipbaecken Fluoride Concentrations" resource is not.

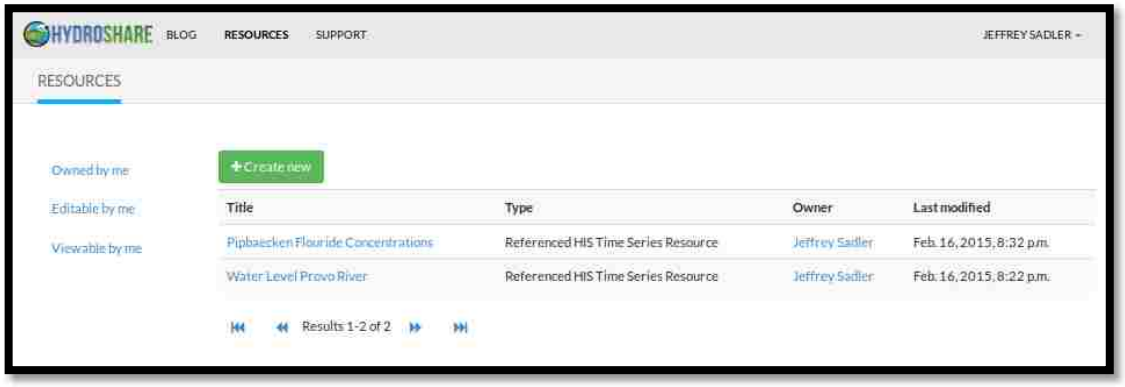

Figure 31. Resource collection for resource creator

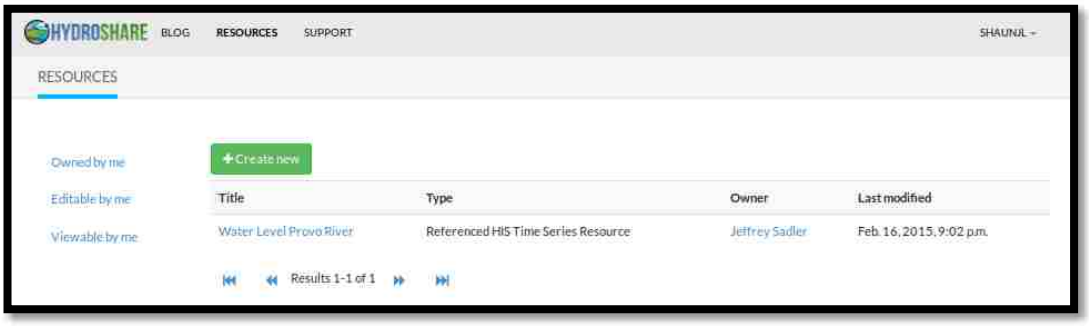

Figure 32. Collection of resources for "shaunjl"

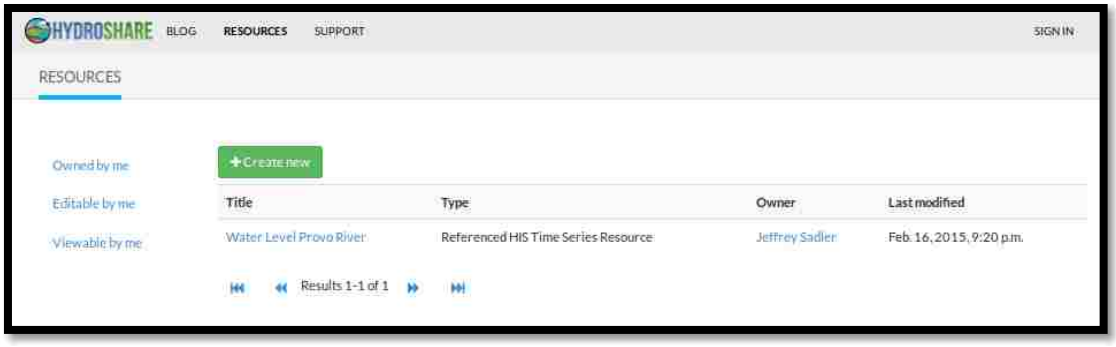

Figure 33. Collection of resources made public on HydroShare

The commenting and endorsing aspects of the social media functionality were also successful for the HIS referenced time series resources I created. Comments were made on the "Water Level Provo River" resource by a non-HydroShare-user ("John Smith"), and a HydroShare user "shaunjl." Both of these comments were then successfully endorsed. Figure 34 shows the comments on the "Water Level Provo River" resource as well as the endorsements of those comments.

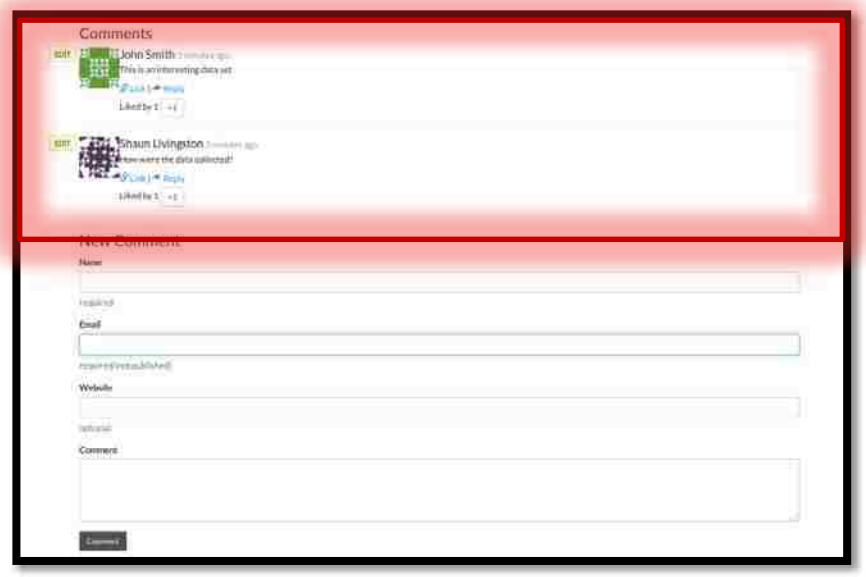

Figure 34. Comment on Water Level Provo River resource

3.4 Chapter Conclusions

I successfully designed, integrated, and tested the HIS Referenced Time Series data type for the HydroShare system. With the HIS Referenced Time Series data type users can link time series data found on HydroServers into the HydroShare system. As resources in HydroShare, HIS time series can be easily shared among other users. Additionally, HydroShare social media functionality such as commenting, endorsing, and following HIS time series resources promote clarification and discussion which can enhance and accelerate collaborative science and modeling. HydroShare also provides HIS time series thorough science and creator metadata support.

In addition to the current HydroShare features which enhance their use and sharing, planned future developments will add even more value to HIS Referenced Time Series data. Future HydroShare developments include support for creation of groups of users with common interests which will further enhance collaborative options using the system. Additionally, third party tools are now under development to access HydroShare resources (now including the HIS Referenced Time Series resources) and interact with them for visualization, analysis, modeling, etc. This will be supported through the HydroShare REST API and the External Tool data type, both of which are currently under development. Finally, as HydroShare begins to support the pre-processing, post-processing, and execution of hydrologic models, users will be able to easily integrate HIS time series resources as the data and workflows will all be contained conveniently in one system.

Beyond CUAHSI HIS, there are many repositories that offer their time series data in a standardized way including research-, national-, and international-level repositories such as DataONE, the United States Geologic Survey (USGS), and the Global Earth Observation System of Systems (GEOSS) (Butterfield et al. 2008). The HIS Referenced Time Series data type is expected to serve as a model which will be expanded and modified to support referencing time

series data stored in these and other repositories. In this way the same HydroShare's unique benefits that I have described in this paper for HIS time series datasets may be extended to many other time series datasets

The connection between HydroShare and the CUAHSI HIS through the mechanism of the new Referenced Time Series resource presented in this paper, provides hydrologists with a new, functional system for sharing and collaborating around HIS time series data – effectively turning the HIS data into social media. This collaborative tool will help scientists leverage the ever-increasing amount of data to better understand the complex water challenges facing society today.

3.5 Chapter Acknowledgements

This work was supported by the National Science Foundation under collaborative grants OCI-1148453 and OCI-1148090 for the development of HydroShare (http://www.hydroshare.org). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

4 CONCLUSIONS

The research in this thesis has successfully addressed significant challenges in hydrologic data collection, curation, and sharing. First a system using low-cost hardware, including Arduino hardware, and the free and open source HydroServer software was successfully designed and deployed to automatically transmit and store hydrologic data in a standards-based database exposed to WaterOneFlow and OGC Web Feature services. This framework provides hydrologists a complete workflow for observational including collection (with the recipe for building the sensor system) to curation and exchange (with its connection to HydroServer Lite). While this system likely would not replace a commercial grade sensor system designed for long term deployment it has unique advantages that could be further explored. For example, the system is very adaptable. The logging and transmission pieces are generic and can be used with many different sensors. Additionally, the Arduino source code is highly customizable. These features would be useful to a research group studying a diverse system and needing a flexible system to study a variety of environmental parameters. This hardware/software framework, being low-cost and hands-on, also has much potential in citizen science and educational efforts. One can easily picture a Junior High or High School science class building and deploying their own environmental sensor as well as setting up a HydroServer to manage, view, and share their locally collected data.

Second, a connection between CUAHSI HIS and the HydroShare system was established with the development of the HIS Referenced Time Series resource type in HydroShare. As

HydroShare resources, HIS time series are social media objects providing hydrologists a tool that is expected to facilitate better understanding and more effective collaboration around hydrologic time series data. The utility of this new development is still yet to been seen. However, as social media has transformed many aspects of society including advertising, communication, and media consumption (e.g. photos and videos), it is expected to have a significant effect on the use and value of hydrologic time series data and related hydrologic research.

The contributions presented in this thesis provide hydrologists tools to assist in hydrologists' progression through the data life cycle. Specifically, addressed are challenges in the collection of, curation of, and collaboration around hydrologic time series datasets; data which are critical to the understanding and prediction of impactful hydrologic phenomena. These tools have the potential enhance the hydrologic science community's understanding and ability to address the water resource challenges that currently face society today.

REFERENCES

Adams, B., 2012. nethoncho/Arduino-DHT22. In: GitHub. https://github.com/nethoncho/Arduino-DHT22 Accessed February 2014.

Ames, D., J. Kadlec & J. Horsburgh, HydroDesktop: A Free and Open Source Platform for Hydrologic Data Discovery, Visualization, and Analysis. In: 2010 AWRA Spring Specialty Conference: Geographic Information Systems (GIS) and Water Resources VI American Water Resoruces Association, TPS-10-1, ISBN, 2010. p 1-882132.

Ames, D. P., J. S. Horsburgh, Y. Cao, J. Kadlec, T. Whiteaker & D. Valentine, 2012. HydroDesktop: Web services-based software for hydrologic data discovery, download, visualization, and analysis. Environmental Modelling & Software 37:146-156.

Anzlone, G. C., A. G. Glover & J. M. Pearce, 2013. Open-Source Colorimeter. Sensors 13(4):5338-5346.

Arduino, 2014. Arduino - Home. In. http://www.arduino.cc/ Accessed August 15 2014.

Assante, M., L. Candela, D. Castelli, F. Mangiacrapa & P. Pagano, 2014. A Social Networking Research Environment for Scientific Data Sharing: The D4Science Offering. The Grey Journal 10(2):65-71.

Baker, D. M., 1936. Basic Hydrologic Data. Transactions, American Geophysical Union 17(2):481-486.

Baker, M. E., 2014. Open source data logger for low-cost environmental monitoring. Biodiversity data journal(2).

Basha, E. & D. Rus, Design of early warning flood detection systems for developing countries. In: Information and Communication Technologies and Development, 2007 ICTD 2007 International Conference on, 2007. IEEE, p 1-10.

Bechhofer, S., I. Buchan, D. De Roure, P. Missier, J. Ainsworth, J. Bhagat, P. Couch, D. Cruickshank, M. Delderfield & I. Dunlop, 2013. Why linked data is not enough for scientists. Future Generation Computer Systems 29(2):599-611.

Beran, B. & M. Piasecki, 2009. Engineering new paths to water data. Computers & Geosciences 35(4):753-760.

Berz, G., 2000. Flood disasters: lessons from the past—worries for the future. Proceedings of the ICE-Water and Maritime Engineering 142(1):3-8.

Borgman, C. L., 2012. The conundrum of sharing research data. Journal of the American Society for Information Science and Technology 63(6):1059-1078.

Botts, M., G. Percivall, C. Reed & J. Davidson, 2008. OGC® Sensor Web Enablement: Overview and High Level Architecture. In Nittel, S., A. Labrinidis & A. Stefanidis (eds) GeoSensor Networks. Lecture Notes in Computer Science, vol 4540. Springer Berlin Heidelberg, 175-190.

Boyko, A., J. Kunze, J. Littman, L. Madden & B. Vargas, 2011. The BagIt File Packaging Format (V0. 97). Washington DC.

Butterfield, M. L., J. S. Pearlman & S. C. Vickroy, 2008. A system-of-systems engineering GEOSS: Architectural approach. Systems Journal, IEEE 2(3):321-332.

Buytaert, W., Z. Zulkafli, S. Grainger, L. Acosta, T. C. Alemie, J. Bastiaensen, B. De Bièvre, J. Bhusal, J. Clark & A. Dewulf, 2014. Citizen science in hydrology and water resources: opportunities for knowledge generation, ecosystem service management, and sustainable development. Hydrosphere 2:26.

Charvériat, C., 2000. Natural disasters in Latin America and the Caribbean: An overview of risk. Working Paper, Inter-American Development Bank, Research Department.

Clasen, T., D. Fabini, S. Boisson, J. Taneja, J. Song, E. Aichinger, A. Bui, S. Dadashi, W.-P. Schmidt, Z. Burt & K. L. Nelson, 2012. Making Sanitation Count: Developing and Testing a Device for Assessing Latrine Use in Low-Income Settings. Environmental Science & Technology 46(6):3295-3303.

Conner, L. G., D. P. Ames & R. A. Gill, 2013. HydroServer Lite as an open source solution for archiving and sharing environmental data for independent university labs. Ecological Informatics 18(0):171-177 doi:http://dx.doi.org/10.1016/j.ecoinf.2013.08.006.

CUAHSI, 2014. CUAHSI Hydrologic Information System (CUAHSI-HIS). In. http://his.cuahsi.org/ Accessed August 1 2014.

Fedi, A., D. Ferrari, M. Lima, F. Pintus & C. Versace, Acronet Paradigm: An Open Hardware Project. In: 2nd Open Water Symposium, Brussels, 2013.

Giuliani, G., N. Ray, S. Schwarzer, A. De Bono, P. Peduzzi, Q.-H. Dao, J. Van Woerden, R. Witt, M. Beniston & A. Lehmann, 2011. Sharing environmental data through GEOSS. International Journal of Applied Geospatial Research 2(1):1-17.

Goble, C. A., J. Bhagat, S. Aleksejevs, D. Cruickshank, D. Michaelides, D. Newman, M. Borkum, S. Bechhofer, M. Roos & P. Li, 2010. myExperiment: a repository and social network for the sharing of bioinformatics workflows. Nucleic acids research 38(suppl 2):W677-W682.

Goff, S. A., M. Vaughn, S. McKay, E. Lyons, A. E. Stapleton, D. Gessler, N. Matasci, L. Wang, M. Hanlon & A. Lenards, 2011. The iPlant collaborative: cyberinfrastructure for plant biology. Frontiers in plant science 2.

Henson, C., P. Barnaghi & A. Sheth, 2013. From Data to Actionable Knowledge: Big Data Challenges in the Web of Things. IEEE Intelligent Systems 28(6):0006-11.

Hicks, S., S. Damiano, K. Smith, J. Olexy, J. Horsburgh, E. Mayorga & A. Aufdenkampe, An open-source wireless sensor stack: from Arduino to SDI-12 to Water One Flow. In: AGU Fall Meeting Abstracts, 2013. vol 1. p 1572.

Horsburgh, J. S., D. G. Tarboton, M. Piasecki, D. R. Maidment, I. Zaslavsky, D. Valentine & T. Whitenack, 2009. An integrated system for publishing environmental observations data. Environmental Modelling & Software 24(8):879-888.

Hut, R., 2013. New Observational Tools and Datasources for Hydrology: Hydrological data Unlocked by Tinkering.

Kadlec, J. & D. P. Ames, Development of a Lightweight Hydroserver and Hydrologic Data Hosting Website. In: Proceedings of the AWRA Spring Specialty Conference on GIS and Water Resources, New Orleans, 2012.

Kadlec, J., B. StClair, D. P. Ames & R. A. Gill, 2015. WaterML R Package for Managing Ecological Experiment Data on a CUAHSI HydroServer. Ecological Informatics In review.

Kaplan, A. M. & M. Haenlein, 2010. Users of the world, unite! The challenges and opportunities of Social Media. Business horizons 53(1):59-68.

Kean, J. W., D. M. Staley, R. J. Leeper, K. M. Schmidt & J. E. Gartner, 2012. A low-cost method to measure the timing of postfire flash floods and debris flows relative to rainfall. Water Resources Research 48(5).

Lagoze, C., H. Van de Sompel, M. L. Nelson, S. Warner, R. Sanderson & P. Johnston, 2008. Object re-use & exchange: A resource-centric approach. arXiv preprint arXiv:08042273.

Lundquist, J. D. & F. Lott, 2008. Using inexpensive temperature sensors to monitor the duration and heterogeneity of snow-covered areas. Water Resources Research 44(4).

Michaelis, C. D. & D. P. Ames, 2012. Considerations for Implementing OGC WMS and WFS Specifications in a Desktop GIS.

Michener, W. K., S. Allard, A. Budden, R. B. Cook, K. Douglass, M. Frame, S. Kelling, R. Koskela, C. Tenopir & D. A. Vieglais, 2012. Participatory design of DataONE—enabling cyberinfrastructure for the biological and environmental sciences. Ecological Informatics 11:5-15.

Michener, W. K. & M. B. Jones, 2012. Ecoinformatics: supporting ecology as a data-intensive science. Trends in ecology & evolution 27(2):85-93.

Mitchell, K. A., B. Chua & A. Son, 2014. Development of first generation in-situ pathogen detection system (Gen1-IPDS) based on NanoGene assay for near real time< i> E. coli</i> O157: H7 detection. Biosensors and Bioelectronics 54:229-236.

Morgenschweis, G., 2011. Datenerfassung und-fernübertragung Hydrometrie. Springer, 513-535.

Mousa, M. & C. Claudel, 2014. Energy Parameter Estimation in Solar Powered Wireless Sensor Networks Real-World Wireless Sensor Networks. Springer, 217-229.

Newman, G., A. Wiggins, A. Crall, E. Graham, S. Newman & K. Crowston, 2012. The future of citizen science: emerging technologies and shifting paradigms. Frontiers in Ecology and the Environment 10(6):298-304.

Oliveira, L. M. & J. J. Rodrigues, 2011. Wireless sensor networks: a survey on environmental monitoring. Journal of communications 6(2):143-151.

Paskin, N., 2008. Digital object identifier (DOI) system. Encyclopedia of library and information sciences 3:1586-1592.

Pearce, J. M., 2012. Building Research Equipment with Free, Open-Source Hardware. Science 337(6100):1303-1304.

Piasecki, M., D. Ames, J. Goodall, R. Hooper, J. Horsburgh, D. Maidment, D. Tarboton & I. Zaslavsky, Development of an information system for the hydrologic community. In: Proceedings of the Ninth International Conference on Hydroinformatics, Tianjin, China, 2010.

Plale, B., R. H. McDonald, K. Chandrasekar, I. Kouper, S. Konkiel, M. L. Hedstrom, J. Myers & P. Kumar, 2013. SEAD Virtual Archive: Building a Federation of Institutional Repositories for Long-Term Data Preservation in Sustainability Science. International Journal of Digital Curation 8(2):172-180.

Pourmirza, Z. & J. M. Brooke, The Wireless Sensor Network and Local Computational Unit in the Neighbourhood Area Network of the Smart Grid. In: Proceedings of the 2nd International Conference on Sensor Networks (SENSORNETS 2013), 2013. SCITEPRESS? Science and Technology Publications, p 84-88.

Quinn, N. W., R. Ortega, P. J. Rahilly & C. W. Royer, 2010. Use of environmental sensors and sensor networks to develop water and salinity budgets for seasonal wetland real-time water quality management. Environmental Modelling & Software 25(9):1045-1058.

Sadler, J. M., 2014. jsadler2/arduino_to_hsl. In. https://github.com/jsadler2/arduino_to_hsl/releases/1.0 Accessed November 19 2014.

Sadler, J. M., D. P. Ames & R. Khattar, 2015. A Recipe for Standards-Based Data Sharing using Open Source Software and Low-Cost Electronics. The Journal of Hydroinformatics.

Stroud Research Center, 2014. StroudCenter/Arduino-SDI-12. In. https://github.com/StroudCenter/Arduino-SDI-12 Accessed June 15 2014.

Tarboton, D., R. Idaszak, J. Horsburgh, D. Ames, J. Goodall, L. Band, V. Merwade, A. Couch, J. Arrigo & R. Hooper, HydroShare: an online, collaborative environment for the sharing of hydrologic data and models. In: AGU Fall Meeting Abstracts, 2013. vol 1. p 1510.

Tarboton, D., R. Idaszak, J. Horsburgh, J. Heard, D. Ames, J. Goodall, L. Band, V. Merwade, A. Couch & J. Arrigo, HydroShare: Advancing Collaboration through Hydrologic Data and Model Sharing. In: International Environmental Modelling and Software Society (iEMSs) 7th International Congress on Environmental Modelling and Software San Diego, California, USA, DP Ames, N Quinn(Eds) http://www iemss org/society/index php/iemss-2014-proceedings, 2014.

Tarboton, D. G., J. S. Horsburgh, D. R. Maidment, T. Whiteaker, I. Zaslavky, M. Piasecki, J. Goodall, D. Valentine & T. Whitenack, 2009. Development of a Community Hydrologic Information. 18th World IMACS Congress and MODSIM09 International Congress on Modelling and Simulation, ed RS Anderssen, RD Braddock and LTH Newham, Modelling and Simulation Society of Australia and New Zealand and International Association for Mathematics and Comput:988-994.

Taylor, P., S. Cox, G. Walker, D. Valentine & P. Sheahan, 2013. WaterML2. 0: development of an open standard for hydrological time-series data exchange.

Taylor, P., G. Walker, D. Valentine & S. Cox, WaterML2. 0: Harmonising standards for water observations data. In: EGU General Assembly Conference Abstracts, 2010. vol 12. p 7680.

Tenopir, C., S. Allard, K. L. Douglass, A. U. Aydinoglu, L. Wu, E. Read, M. Manoff & M. Frame, 2011. Data sharing by scientists: practices and perceptions. PloS one 6(6).

Trevathan, J., R. Johnstone, T. Chiffings, I. Atkinson, N. Bergmann, W. Read, S. Theiss, T. Myers & T. Stevens, 2012. SEMAT — The Next Generation of Inexpensive Marine Environmental Monitoring and Measurement Systems. Sensors 12(7):9711-9748.

U.S. Geologic Survey, 2014. USGS Surface Water Information. In: USGS. http://streamstatsags.cr.usgs.gov/ThreatenedGages/ThreatenedGages.html Accessed August 1 2014.

van de Giesen, N., R. Hut & J. Selker, 2014. The Trans-African Hydro-Meteorological Observatory (TAHMO). Wiley Interdisciplinary Reviews: Water 1(4):341-348.

Wickert, A. D., 2014. The ALog: Inexpensive, Open-Source, Automated Data Collection in the Field. Bulletin of the Ecological Society of America 95(2):166-176.

Winter, L., 2014. PLOS ONE To Require Public Access for Data IFL Science. vol 2015.

Younis, M. & K. Akkaya, 2008. Strategies and techniques for node placement in wireless sensor networks: A survey. Ad Hoc Networks 6(4):621-655.

Zaslavsky, A., C. Perera & D. Georgakopoulos, 2013. Sensing as a service and big data. arXiv preprint arXiv:13010159.

APPENDIX A. SOURCE CODE

This section contains source code relevant to Chapter 2. Code Block 1 is the source code uploaded to the Arduino used to interact with the DHT22 temperature and humidity sensor. Code Block 2 is similar, but interacts with an SDI-12 sensor. Finally Code Block 3 is the PHP script which is part of HydroServer Lite. This script receives the data transmitted by the sensor system, parses them, and inserts them into the HydroServer Lite database.

Code Block 1. Arduino code which interacts with DHT22 temperature and humidity sensor

```
///Sketch written by Jeff Sadler ///

#include <SD.h>
const int chipSelect = 10;
#include <DHT22.h>
#define DHT22_PIN 9


/*Note:This code is used for Arduino 1.0 or later*/
//serial communication to Sim900
#include <SoftwareSerial.h>
SoftwareSerial Sim900Serial(2, 3);


//define variables for HydroServer Lite
byte SourceID = 6;
byte SiteID = 84;
byte VarID1 = 10;
byte VarID2 = 11;
String url = 'http://worldwater.byu.edu/interactive/sandbox/parsetest.php';

//define variables
char character;

int Var1 = 2;
byte Var2 = 0;

byte minInt = 0;
byte secInt = 0;
byte hourInt = 0;
byte yearInt = 0;
```

```
byte monthInt=0;
byte dayInt=0;

char tempBuf[3];

byte recTime = 30;
byte recIntv = 15;

int bufVar1[8];
byte bufVar2[8];
//char dateTimeBuf[25];

byte bufYear[8];
byte bufMonth[8];
byte bufDay[8];
byte bufHour [8];
byte bufMin [8];

int i=0;


// Setup a DHT22 instance
DHT22 myDHT22(DHT22_PIN);




void setup()
{
  Sim900Serial.begin(115200);               // the GPRS baud rate
  delay(500);
  Sim900Serial.println("AT+IPR=19200");
  delay(500);
  Sim900Serial.begin(19200);               // the GPRS baud rate
  delay(1000);
  Serial.begin(9600);               // the Hardware serial rate
  pinMode(10, OUTPUT);                  //set pinmode 10 as output for SD card
  Serial.println("Please type 's' to send  SMS");
  Serial.println("'d' for timestamp");
  Serial.println("'w' to write to SD");
  Serial.println("'t' to write sensor data");
  Serial.println("'tcp' to send data to server");
  Serial.println("or 'h' to submit http request");



  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
    // don't do anything more:
    }
  else {Serial.println("card initialized.");}
 timestamp();
 timestamp();
 powerDown();
}
void loop()
{
  //begin each loop with timestamp
  timestamp();
  Serial.println(secInt);

  //take measurement when minutes is recTime (e.g. 15)
 if (minInt==recTime){
    //get readings and write to sd card
    sensors();
    sd();

    //add timestamp and reading data to arrays
    bufYear[i] = yearInt;
    bufMonth[i] = monthInt;
    bufDay[i] = dayInt;
```

```
    bufHour[i] = hourInt;
    bufMin[i] = minInt;
    bufVar2[i] = Var2;
    bufVar1[i] = Var1;

    //scaffolding
    Serial.print("i= ");
    Serial.println(i);

    recTime += recIntv;
    i++;
    //on fourth measurement submit the http request and push measurements along in arrays
    if(i>3){
      powerUp();
      delay (5000);
      SubmitHttpRequest();
      //SendTextMessage();
      powerDown();

      for(i=0;i<4; i++){

        bufYear[i+4] = bufYear[i];
        bufMonth[i+4] = bufMonth[i];
        bufDay[i+4] = bufDay[i];
        bufHour[i+4] = bufHour[i];
        bufMin[i+4] = bufMin[i];
        bufVar2[i+4] = bufVar2[i];
        bufVar1[i+4] = bufVar1[i];
      }
      i = 0;
    }

  if(recTime  >=  60){
      recTime = recTime-60;
      }
}



  //scaffolding
  if (Serial.available())
    switch(Serial.read())
   {
      case 'c':
      Clock();
      break;
      case 's':
      SendTextMessage();
      break;
      case 'd':
      powerDown();
      break;
      case'u':
      powerUp();
      break;
      case 'p':
      powerStatus();
      break;
      case 't':
      sensors();
      break;
      case 'w':
      sd();
      break;
      case 'h':
      SubmitHttpRequest();
      break;
      //case 'tcp':
      //GPRS_SendText();
      //break;
    }
```

```
 delay (300);

}

//used to adjust time and date for realtime clock on sim900
void Clock(){
  Sim900Serial.println("AT");
  delay(100);
  ShowSerialData();
  Sim900Serial.println("AT+CCLK=\"14/03/25,10:04:30-24\"");
  delay(100);
  ShowSerialData();
}

//powers down sim900
void powerDown(){
  Sim900Serial.println("AT");
  delay(100);
  ShowSerialData();
  Sim900Serial.println("AT+CFUN=0");
  delay(100);
  ShowSerialData();
}

//powers up sim900
void powerUp(){
  Sim900Serial.println("AT");
  delay(100);
  ShowSerialData();
  Sim900Serial.println("AT+CFUN=1");
  delay(100);
  ShowSerialData();
}

//reads whether sim900 is powered up or down
void powerStatus(){
  Sim900Serial.println("AT");
  delay(100);
  ShowSerialData();
  Sim900Serial.println("AT+CFUN?");
  delay(100);
  ShowSerialData();
}

//sends text message
void SendTextMessage()
{
  Sim900Serial.print("AT+CMGF=1\r");    //Sending the SMS in text mode
  delay(100);
  Sim900Serial.println("AT + CMGS = \"18012315581\"");//The target phone number
  delay(100);
  //Sim900Serial.println(dateTimeBuf);
  Sim900Serial.print(",");
  Sim900Serial.print(Var1);
  Sim900Serial.print("C");
  Sim900Serial.print(",");
  Sim900Serial.print(Var2);
  Sim900Serial.println("%"); //the content of the message
  delay(100);
  Sim900Serial.println((char)26);//the ASCII code of the ctrl+z is 26
  delay(100);
  Sim900Serial.println();
  timestamp();
}

//function returns timestamp
void timestamp(){
  String content = "";
```

```
  Sim900Serial.print("AT + CCLK?\r");
  delay(100);
  while (Sim900Serial.available()){
  character = Sim900Serial.read();
  content.concat(character);
  content.trim();}

  int slashPosition = content.lastIndexOf("/");
  String dateTime = content.substring(slashPosition -5,slashPosition +12);
  String temp = content.substring(slashPosition -5,slashPosition -3 );
  temp.toCharArray(tempBuf,3);
  yearInt = atoi (tempBuf);

  temp = content.substring(slashPosition -2,slashPosition -0);
  temp.toCharArray(tempBuf,3);
  monthInt = atoi (tempBuf);

  temp = content.substring(slashPosition +1, slashPosition +3);
  temp.toCharArray(tempBuf,3);
  dayInt = atoi (tempBuf);

  temp = content.substring(slashPosition +4,slashPosition +6);
  temp.toCharArray(tempBuf,3);
  hourInt = atoi (tempBuf);

  temp = content.substring(slashPosition +7,slashPosition +9);
  temp.toCharArray(tempBuf,3);
  minInt = atoi (tempBuf);

  temp = content.substring(slashPosition +10,slashPosition +12);
  temp.toCharArray(tempBuf,3);
  secInt = atoi (tempBuf);

  //dateTime.toCharArray(dateTimeBuf, 25);

}

//reads dht22 temperature and humidity values
void sensors(){
 DHT22_ERROR_t errorCode;

  // The sensor can only be read from every 1-2s, and requires a minimum
  // 2s warm-up after power-on.
  delay(2000);

  //Serial.print("Requesting data...");
  errorCode = myDHT22.readData();
  switch(errorCode)
  {
  case DHT_ERROR_NONE:
  Var1 = myDHT22.getTemperatureC();
  Var2 = myDHT22.getHumidity();
  Serial.print ("Sensor Success");
  Serial.print (Var1);
  Serial.print(",");
  Serial.print(Var2);
  }

}



//writes all values to sd card
void sd(){
 File dataFile = SD.open("datalog1.txt", FILE_WRITE);
  if (dataFile){
    dataFile.print(yearInt);
    dataFile.print("/");
    dataFile.print(monthInt);
    dataFile.print("/");
    dataFile.print(dayInt);
```

```
    dataFile.print(",");
    dataFile.print(hourInt);
    dataFile.print(":");
    dataFile.print(minInt);
    dataFile.print(",");
    dataFile.print(Var2);
    dataFile.print(",");
    dataFile.println(Var1);
    dataFile.close();
    Serial.println ("SD Success");
}
 else{
 Serial.println("error with SD card");
 }
}

///SubmitHttpRequest()
///this function is submit a http request
///attention:the time of delay is very important, it must be set enough
void SubmitHttpRequest()
{
  Sim900Serial.println("AT+CSQ");
  delay(100);

  ShowSerialData();// this code is to show the data from gprs shield, in order to easily see
the process of how the gprs shield submit a http request, and the following is for this purpose
too.

  Sim900Serial.println("AT+CGATT?");
  delay(100);

  ShowSerialData();

  Sim900Serial.println("AT+SAPBR=3,1,\"CONTYPE\",\"GPRS\"");//setting the SAPBR, the connection
type is using gprs
  delay(1000);

  ShowSerialData();

  Sim900Serial.println("AT+SAPBR=3,1,\"APN\",\"CMNET\"");//setting the APN, the second need you
fill in your local apn server
  delay(4000);

  ShowSerialData();

  Sim900Serial.println("AT+SAPBR=1,1");//setting the SAPBR, for detail you can refer to the AT
slashnd mamual
  delay(2000);

  ShowSerialData();

  Sim900Serial.println("AT+HTTPINIT"); //init the HTTP request

  delay(2000);
  ShowSerialData();

  Sim900Serial.print("AT+HTTPPARA=\"URL\",\""+url+"?info=");// setting the httppara, the second
parameter is the website you want to access
  Sim900Serial.print(SourceID);
  Sim900Serial.print(",");
  Sim900Serial.print(SiteID);
  Sim900Serial.print(",");
  Sim900Serial.print(VarID1);
  Sim900Serial.print(",");
  Sim900Serial.print(VarID2);

  Sim900Serial.print("&readings=");
  for (int j=0; j<8; j++){
  Sim900Serial.print(bufYear[j]);
  Sim900Serial.print("/");
  Sim900Serial.print(bufMonth[j]);
```

```
  Sim900Serial.print("/");
  Sim900Serial.print(bufDay[j]);
  Sim900Serial.print(",");
  Sim900Serial.print(bufHour[j]);
  Sim900Serial.print(":");
  Sim900Serial.print(bufMin[j]);
  Sim900Serial.print(",");
  Sim900Serial.print(bufVar1[j]);
  Sim900Serial.print(",");
  Sim900Serial.print(bufVar2[j]);
  Sim900Serial.print(";");
  }
  Sim900Serial.println("\"");

  delay(1000);
  ShowSerialData();

  Sim900Serial.println("AT+HTTPACTION=0");//submit the request
  delay(10000);//the delay is very important, the delay time is base on the return from the
website, if the return datas are very large, the time required longer.
  //while(!Sim900Serial.available());

  ShowSerialData();

  Sim900Serial.println("AT+HTTPREAD");// read the data from the website you access
  delay(300);

  ShowSerialData();

  Sim900Serial.println("");
  delay(100);
}



void ShowSerialData()
{
  while(Sim900Serial.available()!=0)
    Serial.write(Sim900Serial.read());
}
```

Code Block 2. Arduino code which interacts with SDI-12 sensor

```
        ///Sketch written by Jeff Sadler ///

        #include <SD.h>
        const int chipSelect = 10;

        /*Note:This code is used for Arduino 1.0 or later*/
        //serial communication to Sim900
        #include <SoftwareSerial.h>
        SoftwareSerial Sim900Serial(2, 3);

        #include <SDI12.h>
        #define DATAPIN 9         // change to the proper pin
        SDI12 mySDI12(DATAPIN);

        //define variables for HydroServer Lite
        byte SourceID = 6;
        byte SiteID = 84;
        byte VarID1 = 10;
        byte VarID2 = 11;
```

```
String url = "http://worldwater.byu.edu/jefftest.php";

//define variables
char character;

//String Var1 = "55.55";
//String Var2 = "66.66";

byte minInt = 0;
byte secInt = 0;
byte hourInt = 0;
/*byte yearInt = 0;
byte monthInt=0;
byte dayInt=0;*/

char tempBuf[3];

byte recTime = 30;
byte recIntv = 45;

char dateTimeBuf[25];

int count = 0;

void setup()
{
  Sim900Serial.begin(115200);               // the GPRS baud rate
  delay(500);
  Sim900Serial.println("AT+IPR=19200");
  delay(500);
  Sim900Serial.begin(19200);               // the GPRS baud rate
  delay(1000);
  Serial.begin(9600);               // the Hardware serial rate
  pinMode(10, OUTPUT);                  //set pinmode 10 as output for SD card
  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
     //don't do anything more:
    }
  else {Serial.println("card initialized.");}

 mySDI12.begin();

 timestamp();
 timestamp();
 powerUp();
 recTime = secInt + 5;
   if(recTime  >=  60){
      recTime = recTime-60;
      }
}
void loop()
{
  //begin each loop with timestamp
  timestamp();
  Serial.println(dateTimeBuf);

  //take measurement when minutes is recTime (e.g. 15)
 if (secInt==recTime){
   recTime += recIntv;
   sd();

  if(recTime  >=  60){
      recTime = recTime-60;
      }
 }
 if (Serial.available())
    switch(Serial.read())
   {
      case 'm':
      sd();
      break;
```

```
    }



 delay (300);

}

//used to adjust time and date for realtime clock on sim900
/*void Clock(){
  Sim900Serial.println("AT");
  delay(100);
  ShowSerialData();
  Sim900Serial.println("AT+CCLK=\"14/03/25,10:04:30-24\"");
  delay(100);
  ShowSerialData();
}*/

//powers down sim900
void powerDown(){
  Sim900Serial.println("AT");
  delay(100);
  ShowSerialData();
  Sim900Serial.println("AT+CFUN=0");
  delay(100);
  ShowSerialData();
}

//powers up sim900
void powerUp(){
  Sim900Serial.println("AT");
  delay(100);
  ShowSerialData();
  Sim900Serial.println("AT+CFUN=1");
  delay(100);
  ShowSerialData();
}
void test(){
  Serial.println("test");
}
//function returns timestamp
void timestamp(){
  String content = "";
  Sim900Serial.print("AT + CCLK?\r");
  delay(100);
  while (Sim900Serial.available()){
  character = Sim900Serial.read();
  content.concat(character);
  content.trim();}

  int slashPosition = content.lastIndexOf("/");
  String dateTime = content.substring(slashPosition -5,slashPosition +12);
  String temp = "";

  temp = content.substring(slashPosition +4,slashPosition +6);
  temp.toCharArray(tempBuf,3);
  hourInt = atoi (tempBuf);

  temp = content.substring(slashPosition +7,slashPosition +9);
  temp.toCharArray(tempBuf,3);
  minInt = atoi (tempBuf);

  temp = content.substring(slashPosition +10,slashPosition +12);
  temp.toCharArray(tempBuf,3);
  secInt = atoi (tempBuf);

  dateTime.toCharArray(dateTimeBuf, 25);

}
```

```
        //writes all values to sd card
        void sd(){
          String str = sensors();

          int comPos = str.indexOf(",");
          int comPos1 = str.indexOf(",",comPos+1);
          String Var1 = str.substring(comPos+1, comPos1);

          int comPos2 = str.indexOf(",",comPos1+1);
          int comPos3 = str.indexOf(",",comPos2+1);
          String Var2 = str.substring(comPos2+1, comPos3);

          Serial.println(Var1);
          Serial.println(Var2);

         File dataFile = SD.open("test.txt", FILE_WRITE);
          if (dataFile){                                          //do this part each time
            dataFile.print(dateTimeBuf);
            dataFile.print(",");
            dataFile.print(Var1);
            dataFile.print(",");
            dataFile.print(Var2);
            dataFile.print(";");
            dataFile.close();
            Serial.println ("SD Success");
            count ++;
            Serial.print("count: ");
            Serial.println(count);
        }
         else{
         Serial.println("error with SD card");
         }
          if(count == 4){
            String dataString = "";
            File dataFile = SD.open("test.txt", FILE_READ);
            if (dataFile) {
              int i=0;

              dataFile.seek(dataFile.size()-264);
              Serial.println(dataFile.position());
              while (dataFile.available()&& dataFile.position()<dataFile.size()) {
                 char c = dataFile.read();
                 dataString += c;
               }
              dataFile.close();
            }
            else {
                // if the file didn't open, print an error:
               Serial.println("error opening test.txt");
            }
            Serial.println(dataString);
            SubmitHttpRequest(dataString);
            count = 0;
         }
        }

        ///SubmitHttpRequest()
        ///this function is submit a http request
        ///attention:the time of delay is very important, it must be set enough
        void SubmitHttpRequest(String dataString)
        {
          Sim900Serial.println("AT+CSQ");
          delay(100);

          ShowSerialData();// this code is to show the data from gprs shield, in order to easily
see the process of how the gprs shield submit a http request, and the following is for this
purpose too.

          Sim900Serial.println("AT+CGATT?");
```

```
          delay(100);

          ShowSerialData();

          Sim900Serial.println("AT+SAPBR=3,1,\"CONTYPE\",\"GPRS\"");//setting  the  SAPBR,  the
connection type is using gprs
          delay(1000);

          ShowSerialData();

          Sim900Serial.println("AT+SAPBR=3,1,\"APN\",\"CMNET\"");//setting the APN, the second
need you fill in your local apn server
          delay(4000);

          ShowSerialData();

          Sim900Serial.println("AT+SAPBR=1,1");//setting the SAPBR, for detail you can refer to
the AT slashnd mamual
          delay(2000);

          ShowSerialData();

          Sim900Serial.println("AT+HTTPINIT"); //init the HTTP request

          delay(2000);
          ShowSerialData();


Sim900Serial.print("AT+HTTPPARA=\"URL\",\"http://worldwater.byu.edu/jefftest.php?info=");//
setting the httppara, the second parameter is the website you want to access
          Sim900Serial.print(SourceID);
          Sim900Serial.print(",");
          Sim900Serial.print(SiteID);
          Sim900Serial.print(",");
          Sim900Serial.print(VarID1);
          Sim900Serial.print(",");
          Sim900Serial.print(VarID2);

          Sim900Serial.print("&readings=");
          Sim900Serial.print(dataString);
          Sim900Serial.println("\"");

          delay(1000);
          ShowSerialData();

          Sim900Serial.println("AT+HTTPACTION=0");//submit the request
          delay(10000);//the delay is very important, the delay time is base on the return from
the website, if the return datas are very large, the time required longer.
          //while(!Sim900Serial.available());

          ShowSerialData();

          Sim900Serial.println("AT+HTTPREAD");// read the data from the website you access
          delay(300);

          ShowSerialData();

          Sim900Serial.println("");
          delay(100);
        }



        void ShowSerialData()
        {
          while(Sim900Serial.available()!=0)
            Serial.write(Sim900Serial.read());
        }

        String sensors(){
          mySDI12.sendCommand("0M7!");
```

```
          delay(300);                     // wait a while for a response
          String waitStr = bufferToStr();
          String waitTrim = waitStr.substring(0,3);
          int waitInt = waitTrim.toInt()*1000;
          delay(waitInt); // print again in three seconds
          mySDI12.sendCommand("0D0!");
          delay(300);                     // wait a while for a response
          String mStr = bufferToStr();
          return(mStr);
        }

        String bufferToStr(){
          String buffer = "";
          mySDI12.read(); // consume address

          while(mySDI12.available()){
            char c = mySDI12.read();
            if(c == '+' || c == '-'){
              buffer += ',';
              if(c == '-') buffer += '-';
            }
            else {
              buffer += c;
            }
            delay(100);
          }
         return(buffer);
}
```

Code Block 3. PHP script used to parse incoming data and insert them into HydroServer Lite

```
        <?php
        require_once 'AL_hidden_values.php';
        require_once 'database_connection.php';
        $info = $_GET["info"];
        $data = $_GET["readings"];
        $dateTimeArray= array();
        $UTCArray = array ();
        $MethodID = "1";
        $QualityControlLevelID = "0";
        $CensorCode = "nc";
        $UTCOffset = "-7";
        $parsedInfo = explode(",",$info);
        $SourceID = $parsedInfo[0];
        $SiteID= $parsedInfo[1];
        $VariableID1 = $parsedInfo[2];
        $VariableID2= $parsedInfo[3];
        $PastTimeStamps = file_get_contents("DateRepository.txt");
        $parsedData = explode(";",$data);
        $sql7    ="INSERT    INTO    `datavalues`(`DataValue`,    `LocalDateTime`,    `UTCOffset`,
`DateTimeUTC`,     `SiteID`,     `VariableID`,     `CensorCode`,     `MethodID`,     `SourceID`,
`QualityControlLevelID`) VALUES ";
        for($i=0, $size=count($parsedData); $i<$size; ++$i){
                $parsedMoreData  =  explode(",",$parsedData[$i]);  //parses  the  individual  data
records

                $mydate="20".$parsedMoreData[0].",".$parsedMoreData[1];  //formats  the  date  so
that it is 2014 instead of 14 and adds time value
                $timestamp=strtotime($mydate);
                $mydate = date('Y-m-d H:i:s', $timestamp);
                array_push($dateTimeArray,$mydate);


                if (strpos($PastTimeStamps, $mydate)!== false){
```

```
                        continue;
                }


                $ms = $UTCOffset * 3600;
                $utctimestamp = $timestamp - ($ms);
                $DateTimeUTC = date("Y-m-d H:i:s", $utctimestamp);

                $sql7.="('$parsedMoreData[2]',    '$mydate',    '$UTCOffset',    '$DateTimeUTC',
'$SiteID', '$VariableID1', '$CensorCode', '$MethodID', '$SourceID', '$QualityControlLevelID'),
";
                $sql7.="('$parsedMoreData[3]',    '$mydate',    '$UTCOffset',    '$DateTimeUTC',
'$SiteID', '$VariableID2', '$CensorCode', '$MethodID', '$SourceID', '$QualityControlLevelID'),
";


        }
        file_put_contents("DateRepository.txt", $dateTimeArray);
        $sql7=substr($sql7,0,(strlen($sql7)-2));
        echo ($sql7);
        $result7 = @mysql_query($sql7,$connection)or die(mysql_error());
        echo($result7);
        echo ("tests");
        require_once 'update_series_catalog_function.php';
        update_series_catalog($SiteID,        $VariableID1,        $MethodID,        $SourceID,
$QualityControlLevelID);
        update_series_catalog($SiteID,        $VariableID2,        $MethodID,        $SourceID,
$QualityControlLevelID);



?>
```