Graduate School Form 30
Updated 1/15/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By SricharanLochan Gangaraju

Entitled
MACHINE-TO-MACHINE COMMUNICATION FOR AUTOMATIC RETRIEVAL OF SCIENTIFIC DATA

For the degree of Master of Science

Is approved by the final examining committee:

Yao Liang
Chair
Rajeev Raje
Xukai Zou

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Yao Liang

Approved by: Shiaofen Fang 3/21/2016
Head of the Departmental Graduate Program Date

MACHINE-TO-MACHINE COMMUNICATION FOR AUTOMATIC RETRIEVAL OF SCIENTIFIC DATA

A Thesis

Submitted to the Faculty

of

Purdue University

by

SricharanLochan Gangaraju

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2016

Purdue University

Indianapolis, Indiana

ACKNOWLEDGEMENTS

I am highly indebted to my thesis advisor Dr. Yao Liang. I feel fortunate to have been given the opportunity to have Dr. Liang as my mentor during my thesis work. His constant support, unwavering and limitless patience along with his perpetual guidance have helped make the process of my thesis a highly learning and a joyous process.

I would also like to thank Dr. Rajeev Raje and Dr. Xukai Zou for sparing their valuable time and providing me with insightful comments whenever I seek their advice. I would also like to thank Miguel Navarro and Felipe Hernandez for their good nature and assistance during the discussions held through the process of researching and writing my thesis.

My parents and friends have been the biggest source of motivation, my pillar of strength all through the process. I thank them for helping me stay focused and accomplishing this thesis.

This work was partially supported by the United States Department of Transportation through a sub-award to IUPUI under the #OASRTRS-14-H-PIT award.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

HDFR - Hydrological Disaster Forecasting and Response System

GES DISC NASA - Goddard Earth Sciences Data and Information Services Center

USGS - United States Geological Survey

NOAA - Nation Oceanic and Atmospheric Administration

NWS - National Weather Service

XML - Extensible Mark-up Language

DBF - Database File

DAS - Dataset Attribute Structure

DDS - Dataset Descriptor Structure

DataDDS - Data Dataset Descriptor Structure

DAP - Data Access Protocol

MIME - Multimedia Internet Mail Extension

FTP - File Transfer Protocol

HTTP - Hypertext Transfer Protocol

HTTPS - Hypertext Transfer Protocol secure

URI - Uniform Resource Identifier

URL - Uniform Resource Locator

DICT - Dictionary Server Protocol

FILE - Host-specific file names

FTPS - Secure FTP over SSL

GOPHER - The Gopher protocol

IMAP - Internet Message Access Protocol

IMAPS - IMAP over SSL

LDAP - Lightweight Directory Access Protocol

LDAPS - LDAP over SSL

POP3 - Post Office Protocol 3

POP3S - POP over SSL

RTMP - Real Time Messaging Protocol

RTSP - Real Time Streaming Protocol

SCP - Secure Copy

SFTP - Secure File Transfer Protocol

SMTP - Simple Mail Transfer Protocol

SMTPS - Simple Mail Transfer Protocol Secure

TFTP - Trivial File Transfer Protocol

ABSTRACT

Gangaraju, SricharanLochan. M.S., Purdue University, May 2016. Machine-to-Machine Communication for Automatic Retrieval of Scientific Data. Major Professor: Yao Liang.

With the increasing need for accurate weather predictions, we need large samples of data from different data sources for an accurate estimate. There are a number of data sources that keep publishing data periodically. These data sources have their own server protocols that a user needs to follow while writing client for retrieving data. This project aims at creating a generic semi-automatic client mechanism for retrieving scientific data from such sources. Also, with the increasing number of data sources there is also a need for a data model to accommodate data that is published in different formats.

We have come up with a data model that can be used across various applications in the domain of scientific data retrieval.

CHAPTER 1. INTRODUCTION

1.1 Research Background

The Hydrologic Disaster Forecasting and Response system (HDFR) is a software application designed to provide support to engineering teams that are in charge of the transportation infrastructure during severe weather conditions. The system is designed to access hydro-meteorological information and use it to perform hydrological model simulations that will help users determine the threat levels of the numerous vulnerable structures state-wide. This also includes bridge monitoring which helps in determining bridges that have been compromised and weather alarming system helping users with severe weather conditions relating to flooding, snow storms, icy roads and fog.

With the increasing need for accurate weather estimates, we need large samples of data from various data sources for an accurate estimate. The hydro-meteorological information is obtained from NASA Goddard Earth Sciences Data and Information Services Center (GES DISC), United States Geological Survey (USGS), Nation Oceanic and Atmospheric Administration (NOAA) and National Weather Service (NWS). These data sources keep publishing data periodically. To retrieve data from these servers, clients following the specific server protocol have to be developed.

The development of clients is a very protracted process, each response is of a different format, and designing a new schema for each response is not an option.

1.2 Research Goal

This thesis aims at creating a framework that can be used to significantly reduce the development time for writing machine-to-machine (M2M) communication clients for such servers. A software library is designed and developed that can be used by the client to retrieve scientific data from the data servers through the internet.

Once the data is available, we need to conduct some processing, depending on the format of the response, which will eventually be stored into the database.

The following are the steps involved through the process of requesting data from the server until performing the simulations:

1. Request for data

   When a request for data is sent to the server, specific server protocol has to be followed, so that the server can interpret the request and send back the response accordingly. The request contains the request URL in the format specific to the server.

2. Processing the response

   The format of the response is server specific, e.g. XML, Text File, ZIP, DBF etc. The raw data is of no use to the user unless it is processed. The extraction of all the useful information helps us store meaningful information required for simulations into the database.

3. Storing the data

   The DAO (Data Access Objects) components are responsible for storing and reading information to/from the database.

We designed our library in such a way that it supports *DAP*, *FTP, HTTP* and *HTTPS*. It also supports data responses of formats *XML, DBF and DAP grid response.*

With the research that has been put in, we figured out that the time consuming part in the development of a client is the processing of the response.

This thesis also aims at addressing the common problem of creating a new schema for every new retrieval data format that is to be processed. This adds an overhead of both designing the database and writing the necessary DAO for each new schema introduced and also the logic to build/generate graphs. To avoid this tedious process, we have come up with a new data model that can be used across the application for every data source that is to be added.

1.3 Conceptual Challenge

Every data source responds with data in a different data type, creating a need for handling heterogeneous data. Appropriate handling of heterogeneity in the response from the remote data servers is a challenge. There is heterogeneity possible in multiple ways:

1. The data type of the response.

   The response from the server is in different data types such as zipped file(.gz), string, an array etc. This would refer to the heterogeneity in the data type of the response.

2. Data type of the key fields that are to be extracted from within the response.

   The data to be extracted from the response is in different formats.

   Eg: When data retrieved is a string. Latitude and study area description are to be extracted and are in long and string data types respectively. This would refer to heterogeneity within the response.

At each of these levels of heterogeneity, data needs to be handled appropriately based on the data type.

1.4 Contribution

The framework provided has a unified GUI. Based on the user's input, it identifies the data source to which the request has to be submitted. Once it identifies the source, creates the request object, sends it over to the server and transforms the response into appropriate data model. This transformation into the appropriate data model involves dealing with the heterogeneity in the responses from different sources. Identification of the data type, extracting the necessary information, transforming into the provided data model and writing to the database are the steps involved in processing the response.

CHAPTER 2. RELATED WORK

2.1 Existing Systems

2.1.1 A Methodology for Management of Heterogeneous Site Data

A prototype [1] was developed mainly aiming at managing heterogeneous site data. With the large volume of heterogeneous data that is available, it is difficult to have a new schema to accommodate each data format. Having a unified data model where the user can access the data as a single coherent dataset is designed. All the views are always pre generated. A staging system is designed where all the data is fed, data in staging is then moved to archived database. Unified data views are then constructed from the archived database. Unified data views are instances of data being organized based on the access patterns.

2.1.2 Visualization platform for climate research using Google Earth

A system with an aim to maintain and visualize vast amounts of distributed and heterogeneous data has been proposed in [2]. Here the system architecture is designed into three layers: data layer, business layer and presentation layer. The data layer would be the central repository for all the information especially the data, data source and the attributes. Business layer would be responsible to access the data from the remote server, process and integrate into the uniform data model. The presentation layer would access the data and generate views. The generation of views is not done locally always, there are pre generated images stored in the database and also images readily available through the web servers at the remote data sources. All these images are stored as KML files which are later opened in google earth or other visual earth tools.

With the architecture and the data model layered out for the application, we have come up with the library that can be reused to significantly improve the productivity of the software production process.

We have come up with a unified data model as in [1]. We have identified the possible data types of the data being reported as grid and point. We designed the database to accommodate these two data types. The approach of having all the views pre generated would not be appropriate as it would need the application to store all the data into the local database which would result in wastage of space if the data is never used. Instead we fetch the data on demand as per the needs of the user and then generate views.

We have followed a similar architecture as layered out in [2], and designed our application is the same three tier architecture of having data, business and presentation layers.

2.2 Libraries for data retrieval

The data that is available for analysis from all the different data sources on the internet is available as REST friendly services, which means they are URL accessible [8]. The

HTTP/1.0 specification [9] defined the GET, POST and HEAD methods. Most of the data sources in this thesis, have API's defined that can be accessed using a HTTP GET.

"The GET method means retrieves whatever information (in the form of an entity) is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process [9]".

We need a client side URL transfer library to access such API's. We have chosen libcurl [10] [11], an easy to use client side URL transfer library. Libcurl is a highly portable and easy-to-use client-side URL transfer library, supporting DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS, TELNET and TFTP. libcurl also supports HTTPS certificates, HTTP POST, HTTP PUT, FTP uploading, kerberos, HTTP form based upload, proxies, cookies, user + password authentication, file transfer resume, http proxy tunneling and more. The study in [11] gives us a very brief overview on the key features of libcurl in comparison with its competitors.

With a large amount of data that is available from NASA using the OPeNDAP protocol [3], it was a very straight forward decision to use libdap [13], the SDK which contains an implementation of DAP 2.0 [3].

These libraries assist us in achieving the goal of retrieving the data through URI. When the process of retrieving data needs to be applied for each individual data source, there is every necessary reason to have this re-usable.

This would increase productivity, shorten the time to release the software and many more [14]. This has motivated us to design a library with needs of the application which can be reused.

The structure of the reuse can be broadly classified as Referenced and Forked reuse [7]. If there is a functionality that you would want to achieve, you look for existing module in the software that provides such functionality and you reuse it. This is referred to as Referenced reuse. Fork reuse is when you make a copy of the existing code and develop on top of it, essentially creating your own version of it. We have taken the approach of using both Referenced and Fork reuse in our application as needed.

Most of the organizations write code following few business rules called domains [6]. All the software in the organization would be variants of these domains. Similar need has been identified in Hydrological System for National Weather Service and the Radar Forecast Service (HSNWSRFS) and we have come up with domains and we have provided variations of these domains.

With a huge number of data sources from which data can be downloaded, we have faced the challenge of having to deal with heterogeneous data formats. This involves parsing of the response data and converting them into domain object. The theory of parsing in itself is a broad field of study in computer science [16].

In our application, one of the response formats we had to encounter is XML. The parsing of an XML document can become expensive if not done in the right way. Choosing a parsing model suitable to the application is necessary. DOM and SAX parsing models are most widely known and used.

An extensive study conducted in [14] shows that DOM is most suitable for database applications, while SAX (Simple API for XML) is more appropriate for streaming applications.

2.3 Summary of This Chapter

In this chapter we discuss in detail a few existing systems that are similar to the application developed. A model that emphasizes on having a unified data model for all different data formats that are retrieved from the remote data sever and an approach of pre-generating the images for user's needs has been discussed. A model that leverages google earth for generating views is discussed. This model has three different ways of generating views – at local server on the fly on user's request, images directly retrieved from remote data server and a batch program that runs to generate and store views locally to retrieve on user needs. We provide the rationale on when this existing architecture would be appropriate to use. We need a client side URL transfer library to retrieve data that is available and we have chosen 'libcurl' for this. With having to retrieve the data continuously, we have decided to have reusable http component to retrieve data. We decided to follow DOM method of parsing XML, as we have a need to modify the xml retrieved.

CHAPTER 3. DATA TYPES, SERVERS, RESPONSE FORMATS AND PROTOCOLS

3.1 Data Types

The geographic information is mainly stored in two different formats.

3.1.1 Raster Data (Grid Data)

Representation of data in the form of matrix of cells which are organized into rows and columns. Each cell is attributed with a value representing information regarding the sampled variables (soil moisture, temperature, etc.). For most of the data, the value associated with the cell is considered to be the data for the whole area inscribed inside the cell. It is best suited to represent continuous data such as slope, elevation etc.

The size of each cell can be as small or as large as needed. As the cell size decreases, more detailed the data is and vice versa. With the decrease in cell size, the number of cells required to represent any given study area increases in return increasing the storage space required.

When data is stored using relational database management system, and properly indexed, retrieving the required data can be very quick but requires a very significant size of records to be stored.

3.1.2 Vector Data (Point data)

Vector data is used to best represent non-continuous data such as rivers, roads, sate boundaries etc. Vector data is classified into different types viz., points, lines and polygons based on the geometry with which we can represent the study area.

Points are zero-dimensional points that are used to represent geographical points that can represented with a single dot such as wells, peaks, point of interests etc. Lines are one-dimensional lines that best describe linear features such as rivers, roads, etc. Polygons are two-dimensional polygons that represent geographical features that cover a particular area such as parks, lakes, etc.

3.2 Data Sources and Servers

3.2.1 North America's Land Data Assimilation Systems (NLDAS-2)

NLDAS is hosted by NASA Goddard Earth Sciences Data and Information Services Center (GES DISC). NLDAS integrates a large quantity of observation-based and model reanalysis data to drive offline (not coupled to the atmosphere) land-surface models (LSMs), and executes at 1/8th-degree grid spacing over central North America, enabled by the Land Information System (LIS). Some of the data is taken form observations from Geostationary Operational Environmental Satellites (GOES).

NLDAS data is hosted on GrADS Data Server. GrADS Data Server is a secure and stable data server that provides sub-setting of large data and analyzing the data through the query itself. The core component of GrADS is OPeNDAP (formerly known as DODS).

OPeNDAP is a software framework that depends on the DAP 2.0(Data Access Protocol) protocol to provide access to the local data from remote location. The data is independent of the remote location storage format. OPeNDAP provides a number of API's to access data. The DAP 2.0 is endorsed as community standard by NASA in November 2005.

NLDAS data is gridded with a spatial resolution of 0.125°. The data is available since 1st January 1979. Data is published on the web with a time lag of approximately four days. Data for every hour is present on the server. When queried for a data sample, the response given to the client is a pointer to a one-dimensional array containing data.

Response format: Pointer to the data array from NLDAS.

Using libdap (C++ API for OPeNDAP) when requested for data, an Array class object with the data is returned from the server which can be converted into a standard array.

3.2.2 United States Geological Survey (USGS)

This service is hosted by the United States Geological Survey (USGS). USGS collect data at approximately 1.5 million sites in all the 50 states. The type of data collected can be broadly classified as surface water and ground water data. Most of the data collected is by automated equipment on the site and is transmitted to the NOAA's Geostationary Satellite Server (GOES). This data is then processed at the USGS Water Science Centers and then published to the web.

The USGS site services can be invoked with Representational State Transfer (REST) Web API. This means that the data available can be accessed through a URL. The response to the query is in Extensible Markup Language (XML) format. CURL or WGET can be used to retrieve data from the servers by a simple HTTP GET request which is defined to be safe [8].

Data reported through USGS is Vector data of type point. All other characteristics such as spatial resolution, time lag to publish data depends on the station publishing data. The time resolution at which the data is published is one hour. The response for any data request is a well-defined XML file.

Response Format**:** XML Response from USGS.

The schema of the XML response is attached in the appendix.

Missing data is reported as very significant negative values (in the order of -1E7). If there is no data, then the XML contains only the queryinfo tag.

3.2.3 National Weather Service Radar Forecast Centers (NWS)

The data published here is the water equivalent estimate of all the types of precipitation such as snow, rain, sleet, hail etc., from the NWS Radar Forecast Centers (RFC's). There are two sources from which the data is being sampled, one being the satellite precipitation estimates and the other being ground rainfall gauge reports. These two reports are compared and a correction factor is calculated and applied, which is later published.

The services can be invoked using a simple 'wget' which mimics the capabilities of an File Transfer Protocol (FTP). The original data is in XMRG format which is later projected into Hydrological Rainfall Analysis Project (HRAP) grid coordinate system, where the coordinates (401,1601) are placed at the north pole which will result in all positive coordinates within United States of America.

Data published is gridded with a spatial resolution of 0.05°. The time resolution at which data is available is one hour. Data is published for every hour with a time lag of one hour.

Response Format: DBF File.

The response from the server for any data request is a Database File (DBF). The DBF file can be assumed as a binary file and can be processed by any file reading library.

The DBF contains the following fields:

1. id - a unique value for each cell
2. hrapx - column number of the HRAP grid cell
3. hrapy - row number of the HRAP grid cell
4. latitude of the HRAP grid point
5. longitude of the HRAP grid point
6. Value - precipitation value in inches.

For any missing data the value is "-2" and for data over the oceans it is reported as "-1".

"The Hydrologic Rainfall Analysis Project (HRAP) grid system is used for precipitation estimations from the WSR-88D radars.

The grid is based on a polar stereo graphic map projection with a standard latitude of 60° North and standard longitude of 105° West.

The mesh length at 60° North latitude is 4.7625 KM.

Mesh lengths for other latitudes can be computed from:

zmesh = 4.7625/((1+SIN(60)/1+SIN(xlat)) (1) where zmesh is the mesh length in KM and xlat is the latitude.

The grid is positioned such that coordinates (401,1601) are at the North Pole, resulting in all positive coordinates within the United States. The coordinates of a point P(X,Y) are computed as follows:

RE = (EARTHR*(1+SIN(60)))/xmesh

R = (RE*COS(xlat))/(1+SIN(xlat))

WLONG = xlon+75

X = R*SIN(WLONG)+401

Y = R*COS(WLONG)+1601

where EARTHR is the radius of the earth (6371.2 KM),

xmesh is the mesh length at 60° latitude (4.7625 KM),

xlat is the latitude of point to be converted (decimal degrees),

xlon is the longitude of point to be converted (decimal degrees).

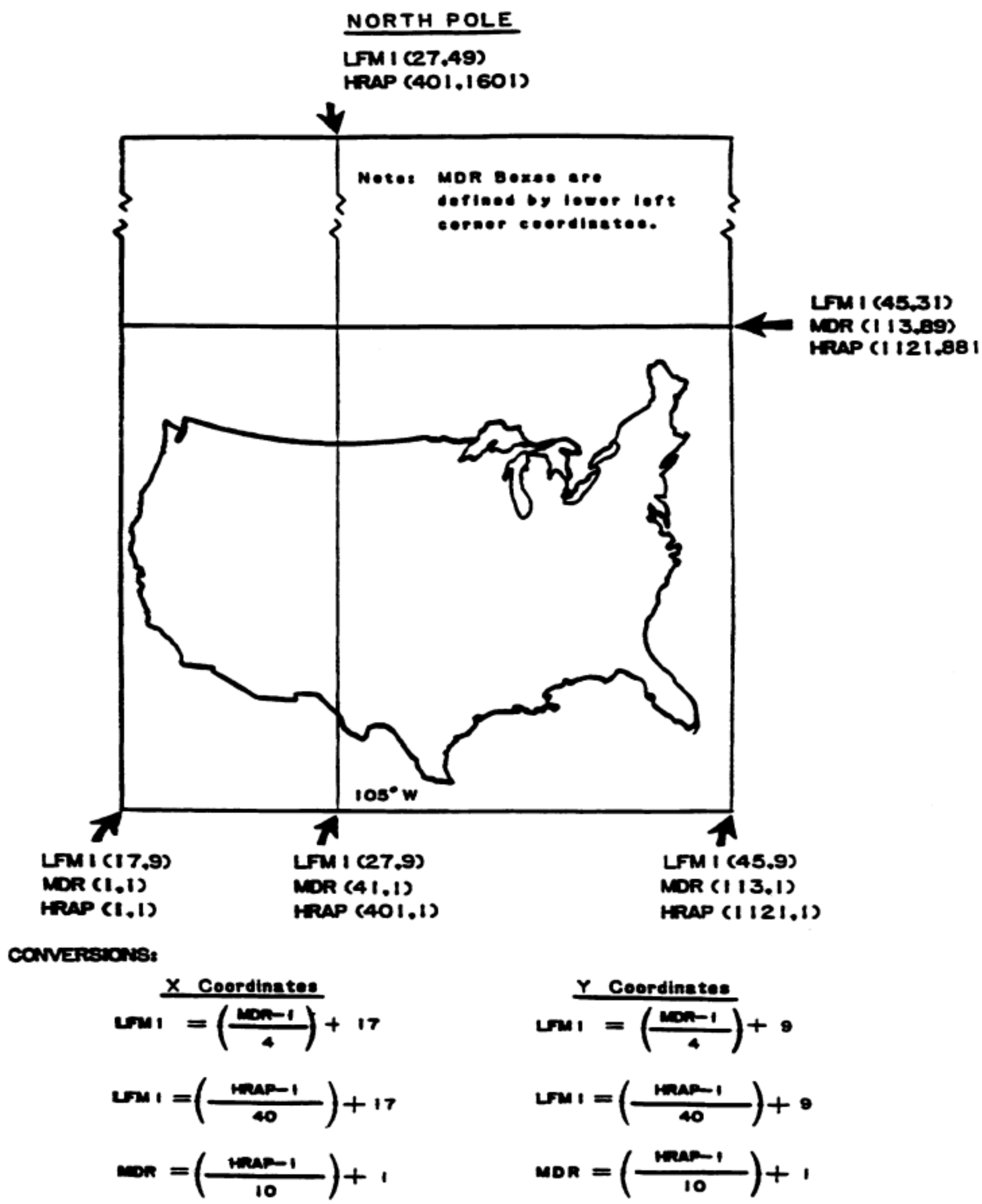

Figure 3.1 HRAP Coordinates [17]

The orientation and mesh length of the grid was selected such that it contains the National Meteorological Center (NMC) Limited Fine Mesh I (LFM I) and the NWS

Manually Digitized Radar (MDR) grids as subsets. The HRAP grid mesh length is 1/40 the size of the LFM I mesh length and 1/10 the size of the MDR mesh length" [17]. The coordinate systems are shown in the figure 3.1.

3.3 Protocols

3.3.1 Libcurl (HTTP get/post and FTP)

Both the NWS Radar data and the USGS water data can be invoked with libcurl – The multiprotocol transfer library (HTTP get for USGS and a FTP for NWS).

3.3.2 Open-source Project for a Network Data Access Protocol (OPeNDAP)

The NASA NLDAS data can be retrieved using the Data Access Protocol (DAP) [3]. The DAP is a stateless protocol that follows the client-server architecture where a client requests for data and the server responds accordingly. The DAP Hypertext Transfer Protocol (HTTP) as a transport protocol.

3.3.2.1 The Data Access Protocol — DAP 2.0

Currently, this is the version of protocol being used and is NASA Earth Science Data Systems Recommended Standard.

Operationally, a DAP client sends a request to a server using HTTP. The request consists of a HTTP GET request method, a Uniform Resource Identifier (URI) that encodes information specific to the DAP and an HTTP protocol version number followed by a MIME-like message containing various headers that further describe the request. In practice, DAP clients typically use a third-party library implementation of HTTP/1.1 so the GET request, URI and HTTP version information are hidden from the client; it sees only the DAP Uniform Resource Locator (URL) and some of the request headers. The DAP server responds with a status line that includes the HTTP protocol version and an error or success code, followed by a MIME-like message containing information about the response and the response itself. The DAP response is the payload of the MIME-like HTTP response.

The various request and responses are listed below:

Request	Response
DDS	DDS or Error
DAS	DAS or Error
DataDDS	DataDDS or Error
Server version	Version information as text
Help	Help text describing all request-response pairs

The DAP uses three responses to represent a data source. Two of these responses, the Dataset Descriptor Structure (DDS) and Dataset Attribute Structure (DAS), characterize the variables, their datatypes, names and attributes. The third response, the Data Dataset Descriptor Structure (DataDDS), holds data values along with name and datatype information.

The DAP client sends a request to the server using HTTP. The request contains the DAP Uniform Resource Locator (URL) and some request parameters. The DAP server responds with a MIME-like message containing some information regarding the response and the response itself. The DAP response is the payload of the HTTP response. The DAP returns error information using an Error response. If a request for any of the three basic responses cannot be returned, an Error response is returned in its place.

The DAP characterizes a data source as a collection of variables. Each variable consists of a name, a type, a value, and a collection of Attributes. Attributes, in turn, are themselves composed of a name, a type, and a value.

Summarizing, OPeNDAP relies on HTTP as the protocol for transport of requests and responses, and relies on MIME Standards. It is independent of the server side storage format.

3.3.2.2 DAP Data Model

Every dataset is attributed to data and a corresponding data model. The data model can be considered as a data type that would represent the data. As already stated all the data access using DAP is Grid Data.

A Grid is an association of an N dimensional array with N named vectors (one-dimensional arrays), each of which has the same number of elements as the corresponding dimension of the array. Each data value in the grid is associated with the data values in the vectors associated with its dimensions.

For instance, let us consider an array representing precipitation values that has 4 columns and 5 rows. This can be considered as an array representing precipitation at 4 different locations at 5 different times.

The following description would accommodate a structure described above:

```
Grid {
    Float64 data[location = 4][time = 5];
    Float64 location[4];
    Float64 time[5];
 } precipitation;
```

Figure 3.2 Grid data structure [19]

In the above example, a vector called location contains four corresponding values to time of each row of the array, and another vector called time contains the time. In addition, a location array could contain four pairs of latitude and longitude corresponding to each row of the array.

```
Grid {
   Float64 data[location = 4][time = 5];
   Array Structure {
      Float64 latitude;
      Float64 longitude;
   } location[4];
   Float64 time[5];
} precipitation;
```

Figure 3.3 Grid data structure with two dimensional array in itself [19]

3.3.2.3 DAP Messages

3.3.2.3.1 Data Attribute Structure (DAS)

Data Attribute Structure is used to store attributes for variables in the dataset. Attributes are used to describe the variable such as range of values, description, resolution etc.

Every attribute is associated with an attribute name, type and value. From the image below, we can understand that the variable apcpsfc is reported in unit's kg/m^2, its filling value, missing value and the long name to describe the variable etc.

```
Attributes {
    NC_GLOBAL {
        String title "0.125 Degree Hourly Primary Forcing Data for NLDAS-2";
        String Conventions "COARDS", "GrADS";
        String dataType "Grid";
        String history "Thu Feb 26 07:10:14 GMT 2015 : imported by GrADS Data
Server 2.0";
    }
    apcpsfc {
        String units "kg/m^2";
        Float32 _FillValue 9.999E20;
        Float32 missing_value 9.999E20;
        String long_name "** precipitation hourly total [kg/m^2] ";
    }
lon {
        String grads_dim "x";
        String grads_mapping "linear";
        String grads_size "464";
        String units "degrees_east";
        String long_name "longitude";
        Float64 minimum -124.93750000000;
        Float64 maximum -67.06250000000;
        Float32 resolution 0.125;
    }
    lat {
        String grads_dim "y";
        String grads_mapping "linear";
        String grads_size "224";
        String units "degrees_north";
        String long_name "latitude";
        Float64 minimum 25.06250000000;
        Float64 maximum 52.93750000000;
        Float32 resolution 0.125;
    }
```

Figure 3.4 Data Attribute Structure

3.3.2.3.2 Dataset Descriptor Structure (DDS)

DDS is sent by the server to the client to describe the structure of the particular dataset. From the image below, apcpsfc is variable, which is a three dimensional. And the datatype of each dimension is also specified along with the maximum value accepted.

```
Dataset {
  Grid {
    ARRAY:
      Float32 apcpsfc[time = 316824][lat = 224][lon = 464];
    MAPS:
      Float64 time[time = 316824];
      Float64 lat[lat = 224];
      Float64 lon[lon = 464];
  } apcpsfc;
  Float64 time[time = 316824];
  Float64 lat[lat = 224];
  Float64 lon[lon = 464];
} /NLDAS_FORA0125_H.002;
```

Figure 3.5 Data Descriptor Structure

For a client this would help determining the appropriate data types to be used along with the maximum acceptable values.

3.4 Summary of This Chapter

In this chapter, we discuss about the classification of data being reported online, description of data sources the system is currently connected to and their respective protocols. Raster and Vector data are the types of data being reported. Raster data corresponds to data being recorded in the form of matrix, organized into rows and columns. Vector data is used to best represent non-continuous data such as rivers, roads, etc. Currently, NLDAS, USGS and NWS Radar are the three data sources that are connected. NLDAS follows the DAP 2.0 protocol and responds with a data array. The USGS and NWS have RESTful services that can be invoked and they respond with XML and .gz file respectively.

CHAPTER 4. ARCHITECTURE

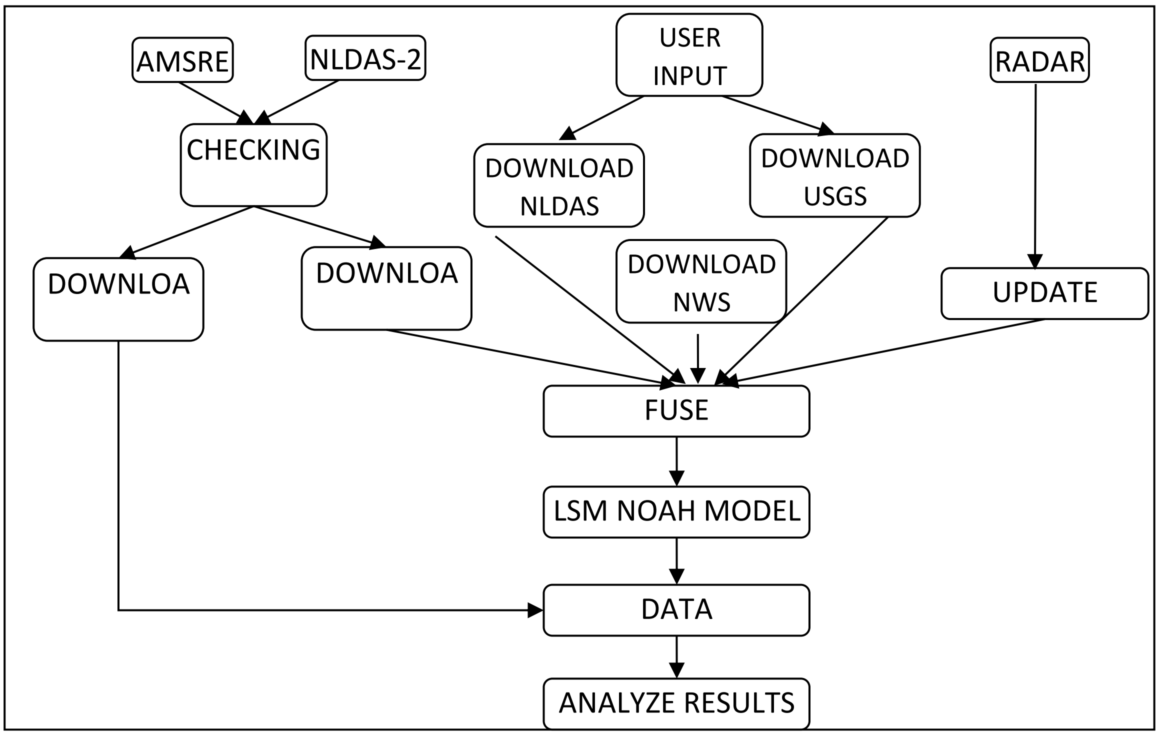

Figure 4.1 Workflow diagram of HSNWSRFS

This view represents the main activities and how they connect to complete the results. First, the AMSRE and NLDAS-2 information is obtained, preprocessed and published by NASA. Data from both NLADS and USGS can be downloaded and stored in the database for further analysis. The radar information is updated manually and also retrieved for NWS.

A module uses the downloaded precipitation information contained in the NLDAS-2 forcing data, and the radar information to perform a precipitation fusion process. At this point, the LSM NOAH model can be used. It will use the NLDAS-2 information (stored in database) to compute the state and output variables of the land-surface interaction. The output will be stored in files or database, depending on the user needs. This thesis addresses the designing the implementation of agents necessary to retrieve data.

4.1 Agent Architecture

Downloading NLDAS, Downloading USGS and Downloading NWS are three components implemented using the framework provided. The architecture of each individual agent has been kept the same. Every Agent has a GUI that takes the user inputs, and the GUI_Handler sending these values to the appropriate request handler to send out the request to the server, and the response handler processing the response and the DAO being responsible for storing the data into the database.

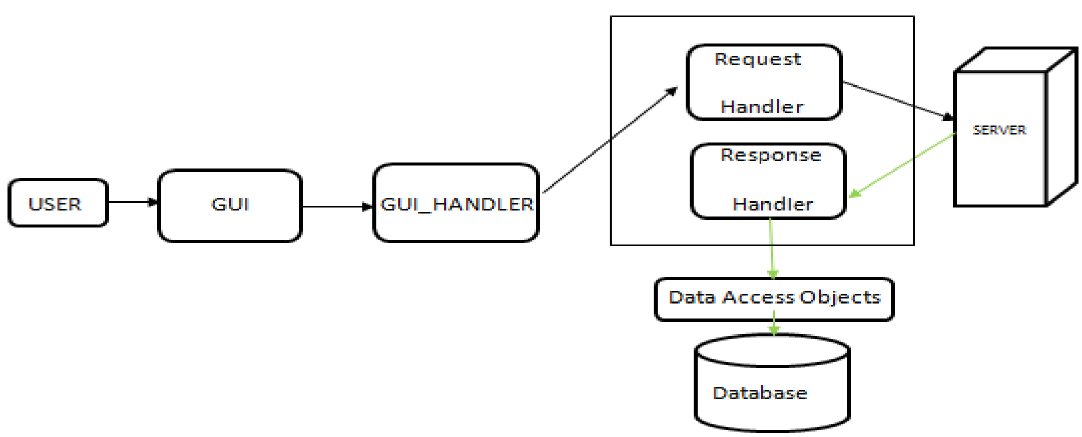

Figure 4.2 A general architecture of an individual Agent

Figure 4.2 explains all the different modules. Each module has its own significance.

GUI: User will have to input his requirements on what data needs to be collected i.e. the time for which the data needs to be retrieved, the coordinates of the study area and server from which data is to be retrieved.

GUI_HANDLER: This module has the role of carrying the data from input to the module responsible for sending it over to the server.

Request Handler: This module would be responsible for transforming the data input into a meaningful request object. The request object would be created in the way the server can interpret. This module is responsible for sending request to the server.

Response Handler: The response from the server would be handed over this module. This module transforms the data from server format into the data model that is lying underneath for storage.

Data Access Object: An object that provides an abstract interface to some type of database or other persistence mechanism. By mapping application calls to the persistence layer, DAO provide some specific data operations without exposing details of the database.

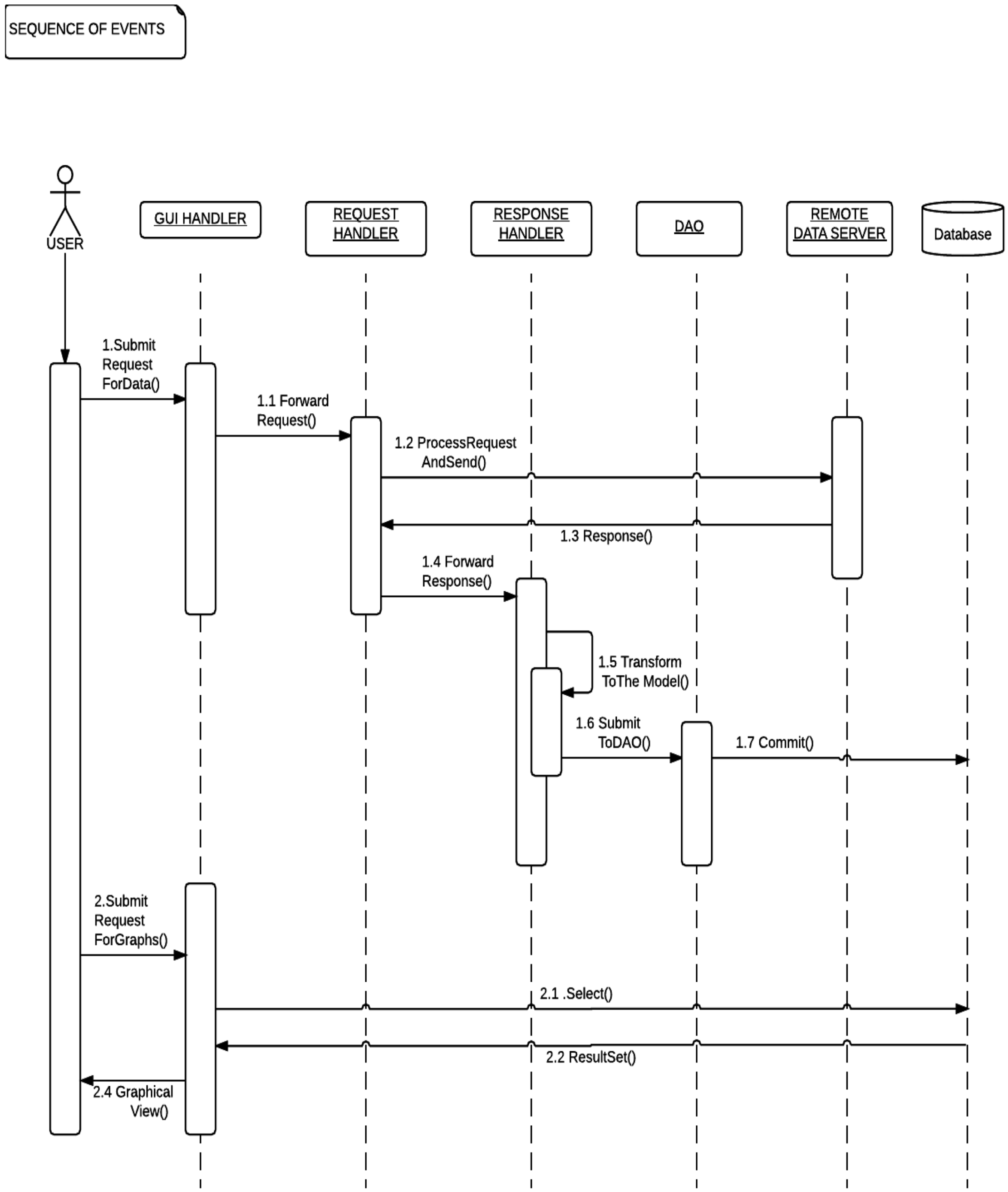

Figure 4.3 Sequence of events

4.1.1 Agent NLDAS

The NLDAS Agent has been developed using libdap, the C++ SDK to implement the DAP 2.0 protocol. The libdap library includes classes and methods that would simplify writing clients and servers for DAP 2.0.

NLDAS reports data for a number of ‘variables’ such as surface pressure, precipitation etc. The GrADS Data Server allows sub-setting the data to be retrieved instead of the whole data on the server. This sub-setting of data can be done by creating a constraint-expression in the format the server can interpret. Every constraint expression has five main parts, Server URL, variable, Index of grid coordinates i.e. Latitude (minimum and maximum) and Longitude (minimum and maximum)), Timestamp (Initial and Final).

All the input parameters required to build the constraint expression (as stated above) are taken from the user. The NLDAS has a unique way of accepting the coordinates and the timestamp. It accepts the Indices to them but not the original values itself. To compute the indices we need to extract the metadata from server. Meta data contains the initial acceptable value and the resolution for each step. When we have received the input parameters from the user, we compute the index for each attribute (subtracting the input from the minimum and dividing it with resolution) and then create the constraint expression. Once we have the constraint expression, we validate by checking all the indices if they are acceptable, using libdap client API we connect to the server and receive the response (using the classes and methods provided by libdap). The data received is always a pointer to the first element of the one-dimensional data array.

Appropriate algorithm to construct a grid out of the received array is applied. This gridded data is stored into the database.

The user interacts with the GUI and inputs that parameters he wants to query with. The GUI_HANDLER calls the Agent with the parameters passed from the user. The Request Handler processes the request and builds the request URL according to the specified server format, which is sent to the server. The Response Handler listens to the server response and processes it. With the help of Data Access Objects (DAO) components that are responsible for storing and reading information to/from the database, we write the retrieved data into the database.

4.1.2 Agent USGS

The USGS Agent has been developed using CURL C API, with which we can implement the REST web services. A HTTP GET request is issued with a unique URL using the client, and a well-structured document specific to the URL is sent back by the server (XML document in this case).

USGS data servers have a unique server URL that is always followed by 'query parameters' which will help us constrain the data to be retrieved. Query parameters can be anything from state code, a contiguous range of decimal latitude and longitude, begin or end date/time, USGS time-series parameter code etc.

List of all the query parameters to create the 'request URL' is requested from the user. USGS provides class to build the URL in the format that the server can interpret. We validate the URL to check if the values are in acceptable ranges. A HTTP GET request is posted to the server with CURL, and a XML file with a well-defined schema is obtained. A Document Object Model (DOM) XML parsing library is used to parse the received response. As the document is parsed sequentially, for every node in the tree, all the required data is collected and written in the database as we read the response.

The user interacts with the GUI and inputs that parameters he wants to query with. The GUI_HANDLER calls the Agent with the parameters passed from the user. The Request Handler processes the request and builds the request URL, which is sent to the server. The Response Handler listens to the server response and processes it. With the help of Data Access Objects (DAO) components that are responsible for storing and reading information to/from the database.

4.1.3 Agent NWS

The NWS Agent has been developed using CURL C API, with which we can implement the REST web services. A HTTP GET request is issued with a unique URL using the client, and a DBF file with the data for the request is given back as response. Radar data servers have a unique server URL that is always followed by 'query parameters' which will help us constrain the data to be retrieved. Query parameters can be begin or end date/time The timestamp required to create the 'request URL' is requested from the user. We validate the URL if it has values in acceptable ranges. A HTTP GET request is posted to the server with CURL, and a DBF file with already described structure is received. Standard C++ library is used to read the file and parse it with regular expressions. As the document is parsed sequentially, a grid model object with all the data is created and inserted into the database.

The user interacts with the GUI and inputs that time range he wants to query with. The GUI_HANDLER calls the Agent with the parameters passed from the user. The Request Handler processes the request and builds the request URL, which is sent to the server. The Response Handler listens to the server response and processes it. With the help of Data Access Objects (DAO) components that are responsible for storing and reading information to/from the database we write the data to the database.

4.2 Class Organization

Classes with required methods to download data have been provided in classes Rest and OPeNDAP. Once the data is downloaded it needs to be transformed into the data model and then the appropriate DAO (i.e. DAOGridData or DAOPointData) classes which have methods to write the data to the database can be used.

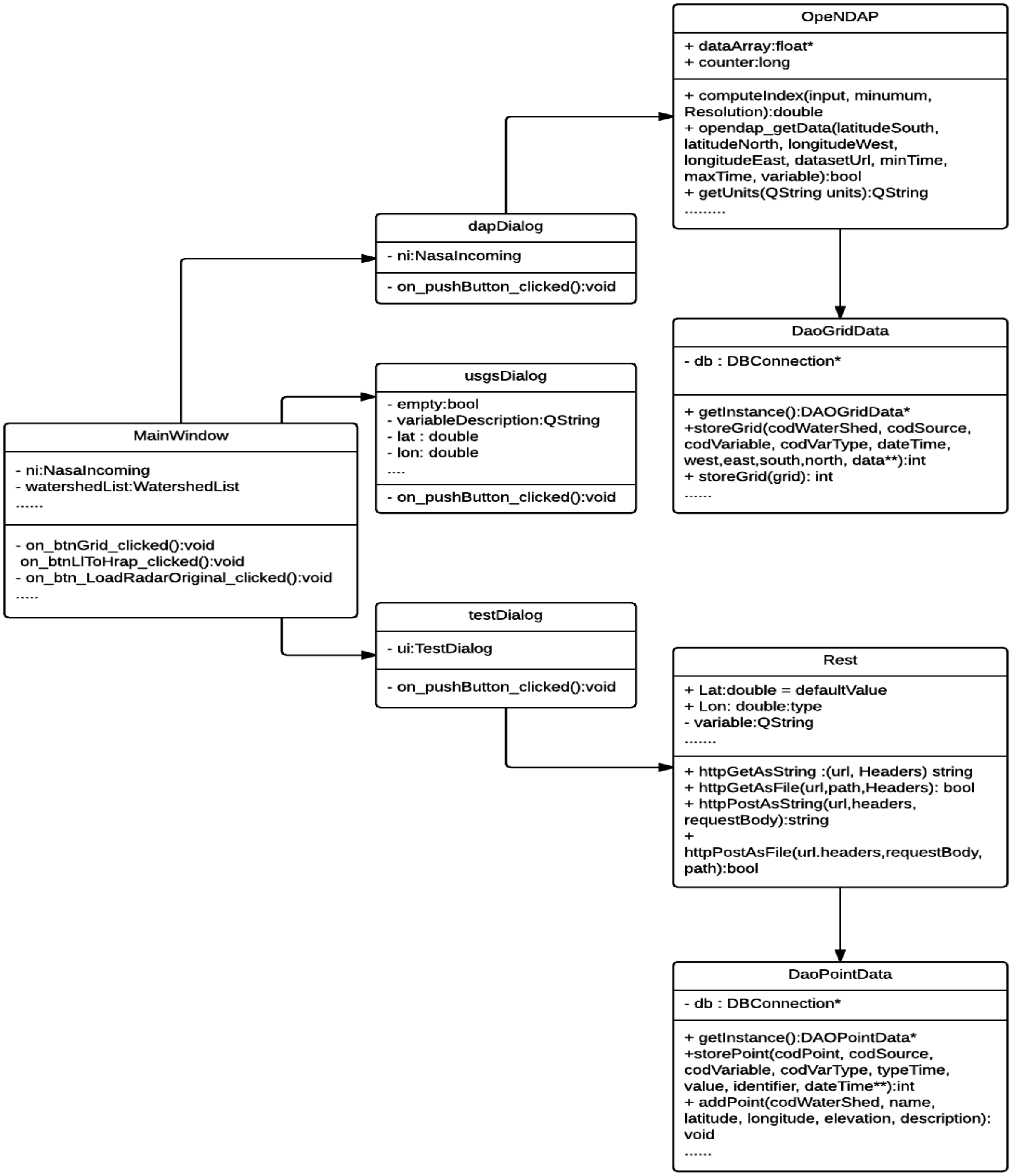

Figure 4.4 Class diagram

4.3 Data Modeling

With the increasing number of servers having different response formats it is difficult to design a new database for each response format. For the graphical user interface to build graph on top of the data retrieved, it would be optimal to have a single data model. As we have already categorized the data reported can only be in two basic formats, Grid and Point, we designed the data model to accommodate these two formats and the user interface to build the graph is also categorized into two categories as grid and point. This not only allows data in heterogeneous sources having different formats to be stored in a uniform format, but also helps in generation of the graphs for analysis. We have come up with these two data models that we believe work well.

1. Griddata: This table stores all the gridded data in 2D arrays. Although each matrix is supposed to correspond to a watershed, the enclosing rectangle should be specified for the cases when the available observations do not exactly match the limits of the watersheds.

2. Pointdata: Stores the locations where point data for a certain variable is available (for example stream stage measured with a USGS gauge).

3. Pointvalues: Stores the point values with their corresponding time for the locations in the pointdata table.

4. Dimensions: Stores the available measuring dimensions with their symbols, reference units, and preferred units.

5. Units: Stores measuring units with their symbols and conversion factors to a reference unit.

6. Variables: Dictionary of variables that can be expected to be stored.

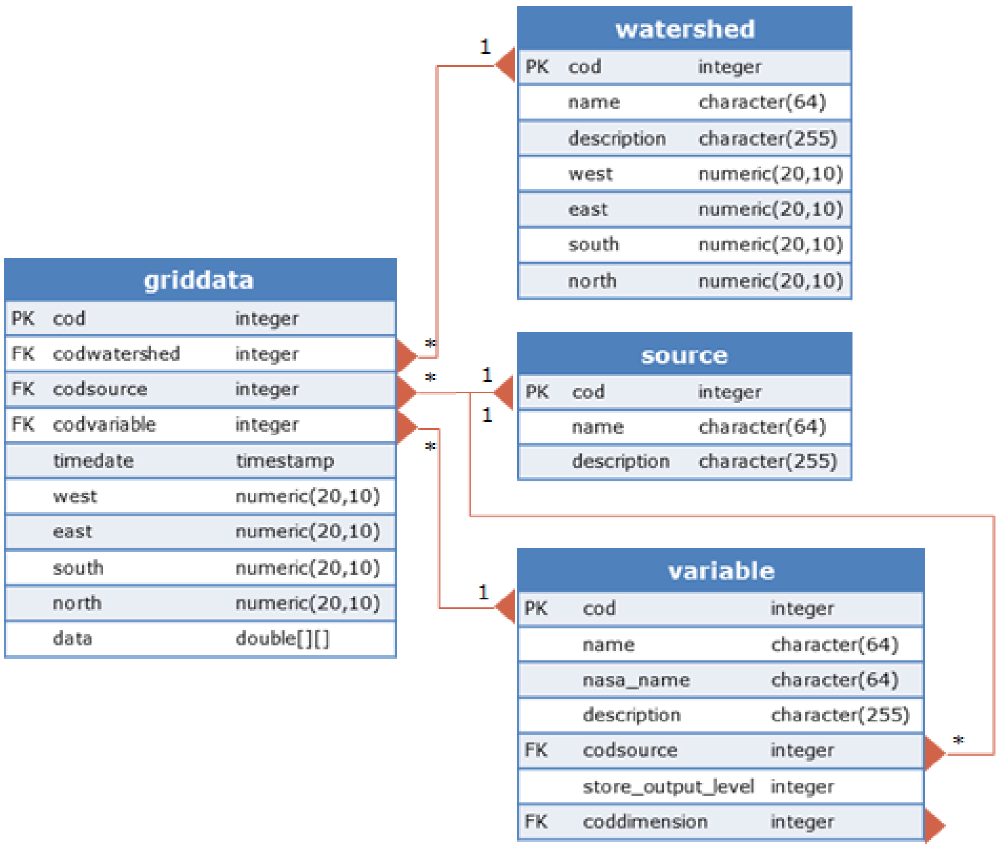

Figure 4.5 Grid Data Design

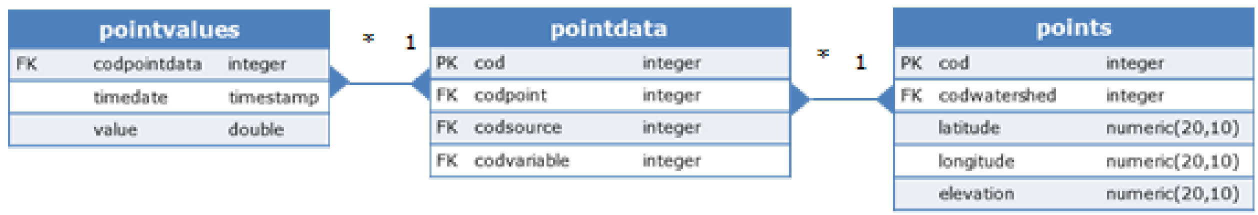

Figure 4.6 Point data design

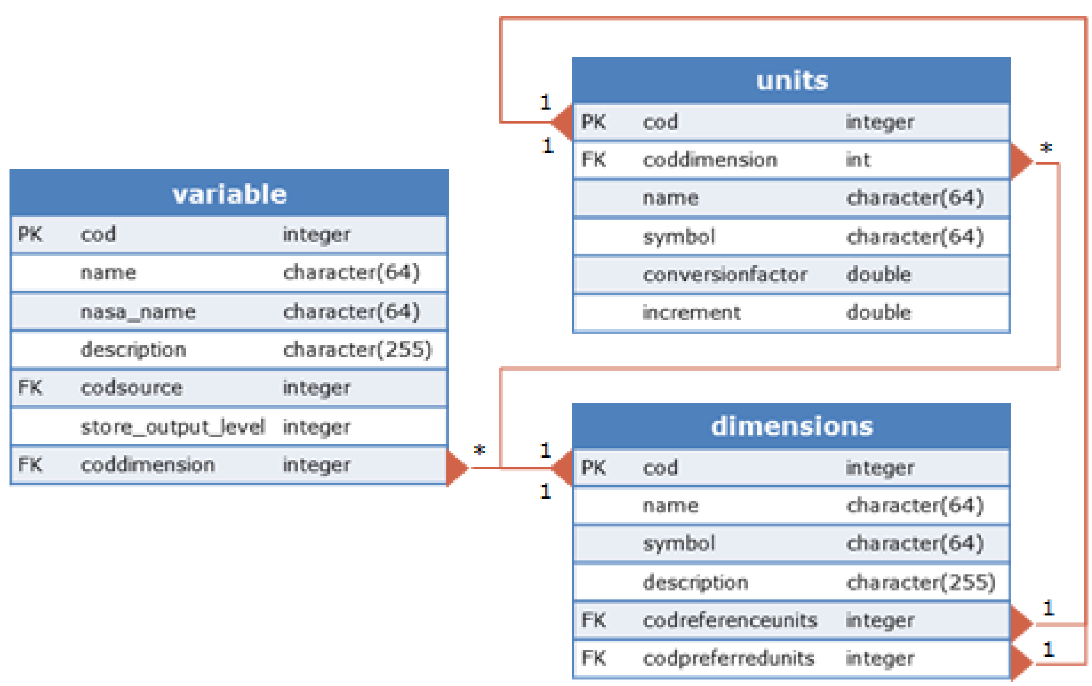

Figure 4.7 Dimensions design

The development of a flexible framework to bring multiple data sources with different formats into a heterogeneous data management system is a challenge. Transforming each data format into the underlying data model is expensive, at the same time having multiple data models for each data format is not an option. We have come up with design to have the development of the modules to render the graphs from database in the same way for data from different sources.

4.4 Exception Handling

The list of possible exceptions and how they are handled are listed in the Table 4.1.

Table 4.1 Exception Handling

Exception	Handling – Message shown to the user
Invalid Input parameters	Input valid parameter(s).
Any exception while invoking the web service to retrieve data	Run time exception. Please retry.
Web service call time out	Request Read Timed out. Reduce the query size and try again.
Server responds back with no data	No data available at the server for the requested parameters.
Exception while parsing the response	Server responded back in an unknown/unexpected format.

Table 4.1 Continued

Table 4.1 Continued

Exception while writing to the data base	Run time exception. Please retry.
Exception while reading from the data base	Run time exception. Please retry.
Any Unknown/Unexpected/Unhandled Run time exception	Run time exception. Please retry.

4.5 Performance metrics

We have done performance evaluation of each agent that is connected in the system. The results in Figure 4.2 show the average time taken with a minimum of three transactions for each. All the metrics are captured in milliseconds.

Table 4.2 Performance Metrics

Source	Time range of the query	Server response time (milliseconds)	Process and persist time (milliseconds)	Response data size (Bytes)	Study Area
NLDAS	1 hour	10	10	7680	Pennsylvania
NLDAS	2 hours	10	23.33	11520	Pennsylvania
NLDAS	1 Day	10	113.33	96000	Pennsylvania
NLDAS	1 Month	116.66	2706	2856960	Pennsylvania
USGS	1 hour	36.66	1857	637615	Pennsylvania
USGS	2 hours	43.33	3046	709560	Pennsylvania
USGS	1 Day	110	39210	6856940	Pennsylvania
NWS Radar	1 Hour	1 X 80.6	1 X 11590	1X144179	North America
NWS Radar	2 Hours	2 X 82	2 X 10391	2X144179	North America
NWS Radar	1 Day	24 X 81	24 X 11342	24X144179	North America

NWS Radar server can only be queried per hour. The multipliers in front of all the metrics corresponding to NWS Radar data denote the number of hours in the time range of the query. After reviewing the average time taken for each time range, we have set the socket read time out to be 30 seconds for every http call.

After the performance evaluation, we noticed that processing the response and writing to the data base is time consuming. With the increase in the response size, the processing time does not increase proportionally but exponentially. The exponential factor of this increase is specific per data source.

In case of USGS water data, we write the data to the database as we parse through the XML response. This increases the number of writes to the data base. Instead, we can maintain a data structure that holds these values and writes to the data base one time, post parsing of the response.

In case of NWS Radar data, if the query size is more than one hour, we send multiple http calls requesting data per hour and all these calls are done synchronously one after the other. We can have these calls done asynchronously to significantly improve the performance.

4.6 Summary of This Chapter

Every module responsible for connecting to the remote data server, retrieving the data and processing the data is called an Agent. Every agent in the application follows the same architecture. The system architecture has been layered out into three layers. Data layer dealing with the actual data, business layer having to retrieve the data from online, and the presentation layer to generate view. Each agent has a GUI to request for user's input, GUI Handler to relay the information to the request handler. The request handler understands the user inputs and creates the server specific request object and sends it over to the server. Once the server responds, relays the information to the response handler, where the key information is extracted and transformed into the data model. Using the DAO provided, we write the data to the database. As we have already identified that the data reported can only be of Grid or Point data type, we have designed a data model to accommodate the two. We have also classified the possible exception and listed out the action after each one of them is thrown. We did a system performance evaluation by requesting the data for the most frequently used time periods.

CHAPTER 5. HOW TO ADD NEW DATA SOURCE, A WALK THROUGH

List of data sources that are currently connected

1. Precipitation, land-surface states (e.g., soil moisture and surface temperature), and fluxes (e.g., radiation and latent and sensible heat fluxes) generated by the North American Land Data Assimilation System (NLDAS).
2. Surface water and ground water data published by United States Geological Survey
3. Water equivalent estimate of all the types of precipitation such as snow, rain, sleet, hail etc., from the NWS Radar Forecast Centers (RFC's).
4. National Oceanic and Atmospheric Administration's (NOAA) Global Forecast System (GFS) data.
5. National Oceanic and Atmospheric Administration's (NOAA) Meteorological Assimilation Data Ingest System (MADIS).

5.1 A walkthrough on how to add a data source

There are 5 layers in the development involved

1. GUI**:** This module is needed for asking the user input. Every data source has its own minimum parameters to be configured. This module should be designed in a way that all the necessary information is input by the user.

2. UI Logic**:** On click of submit by the user, this layer takes all the information from the GUI and passes it over to the corresponding integration layer.
3. Integration Layer**:** In this module, we send a request to the server and get the response back and transform it into the data model.
4. Math Logic**:** Once the data is retrieved, there might be a need to transform the data received. This Logic may be doing some data interpolation that is necessary, or converting the data received into appropriate units etc..
5. DAO**:** This module is responsible for storing and reading information to/from the database.

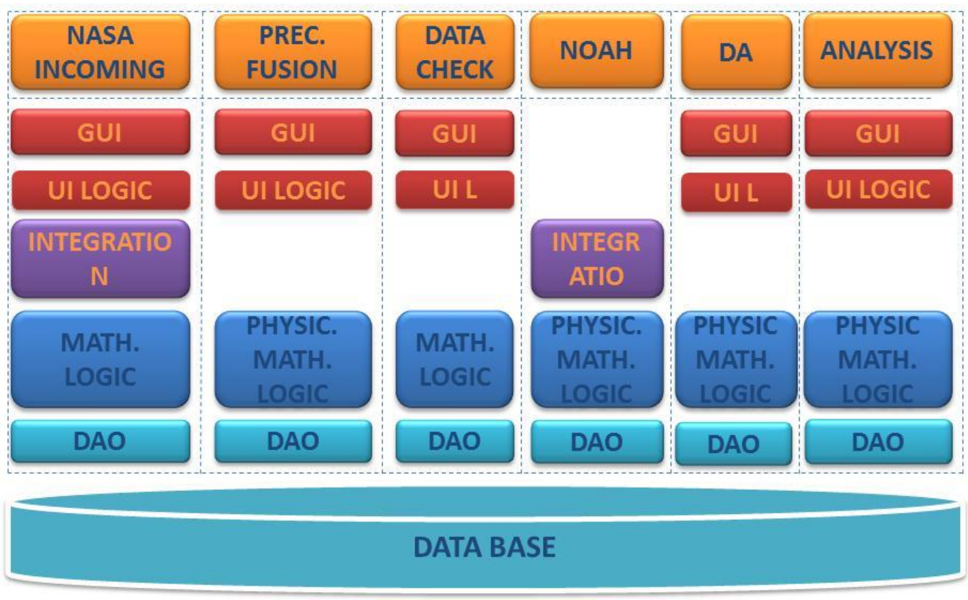

Figure 5.1 Development View

Every data source that is added to the system will have to follow the architecture in Figure 5.1. GUI can be built to get all the configuration input data that is required from the user to help fetch the data from servers.

Classes with required methods to download data have been provided in classes Rest and OPeNDAP.

Two standard HTTP methods are supported in the Rest class. HTTP GET and POST.

Each of the HTTP methods has two corresponding methods in the class. Each of them has the option of either downloading the response into a file or a standard string.

The GET method accepts a URL and 'Accept' header parameter. This 'Accept' header parameter tells the server the content-type acceptable at the client end.

The POST method accepts a URL, Request Headers list and Request Body. There are a number of request headers that are supported by HTTP that are listed in [9]. Both POST and GET are implemented using CURL.

```
rest.h    Rest    Line: 84, Col: 86

    /**
     * Downloads a remote file to a string using CURL (PROTOCOL:HTTP GET)
     * @param url             The remote location of the file as an URL address
     * @param AcceptHeader    tells the server the content-type acceptable at the client end (Optional)
     */

    std::string HttpGetAsString(const std::string & url,std::string AcceptHeader = NULL);

    /**
     * Downloads a remote file to the local hard drive using CURL (PROTOCOL:HTTP GET)
     * @param url       The remote location of the file as an URL address
     * @param path      The local route and file name to save the downloaded file to
     * @param AcceptHeader    tells the server the content-type acceptable at the client end (Optional)
     */

    void HttpGetAsFile(const std::string &url,const char* path, std::string AcceptHeader = NULL);

    /**
     * Downloads a remote file to a string using CURL (PROTOCOL:HTTP POST)
     * @param url              The remote location of the file as an URL address
     * @param Headers          All the header parameters with a coma separated string
     * @param requestBody      The request body that needs to sent to the server
     */

    std::string HttpPostAsString(const std::string & url, std::string headers, std::string requestBody);

    /**
     * Downloads a remote file to the local hard drive using CURL (PROTOCOL:HTTP POST)
     * @param url       The remote location of the file as an URL address
     * @param path      The local route and file name to save the downloaded file to
     * @param Headers          All the header parameters with a coma separated string
     * @param requestBody      The request body that needs to sent to the server
     */

    void HttpPostAsFile(const std::string &url, std::string headers,std::string requestBody,const char* path);
```

Figure 5.2 Methods to download data with libcurl (GET and POST)

```
    /**
     * Downloads a remote file to the local Database/ local hard drive using CURL (PROTOCOL:HTTP GET/POST)
     * @param dataseturl        The remote location of the file as an URL address
     * @param initialtime       The initial time from which data needs to be retrieved (Required for USGS & RADAR)
     * @param finalTime         The final time upto which data needs to be retrieved (Required for USGS & RADAR)
     * @param latitudesouth     southern boundary of the study area (Required for USGS)
     * @param latitudenorth     northern boundary of the study area (Required for USGS)
     * @param longitudewest     western  boundary of the study area (Required for USGS)
     * @param longitudeeast     eastern  boundary of the study area (Required for USGS)
     * @param parametercode     The variable that needs to measured (Required for USGS)
     * @param protocol          Protocol to be used (GET/POST) (Required for all datasets (except: USGS/RADAR))
     * @param headers           All the header parameters with a comma separated string (Optional for all datasets)
     * @param requestBody       The request body that needs to sent to the server (Ooptional for all datasets)
     */

    bool getDataAndStore(QString datasetUrl, QDateTime* initialtime = NULL,QDateTime* finaltime = NULL,
                        double latitudesouth = 0, double latitudenorth = 0, double longitudewest = 0, double longitudeeast = 0,
                         QString* paramtercode = NULL,QString* protocol = NULL,QString* headers = NULL, QString* requestBody = NULL);
```

Figure 5.3 Method to download data and persist

The method getDataAndStore can be used to download data specifying the protocol. It has two acceptable values: GET or POST. This method tries to identify if the request is to download USGS or Radar data based on the domain in the dataset url. If it is of any of the two sources it downloads the data and processes it and stores it in the database for analysis. The OPeNDAP class has methods to get data and store data following opendap protocol. The method is very specific to the dataset and the attributes that are reported in the hourly primary forcing data reported by NASA.

5.2 Results

Sample 2-m above ground temperature retrieved from NLDAS for one hour on Jan 1st 2005 from 00:00 AM to 1:00 AM.

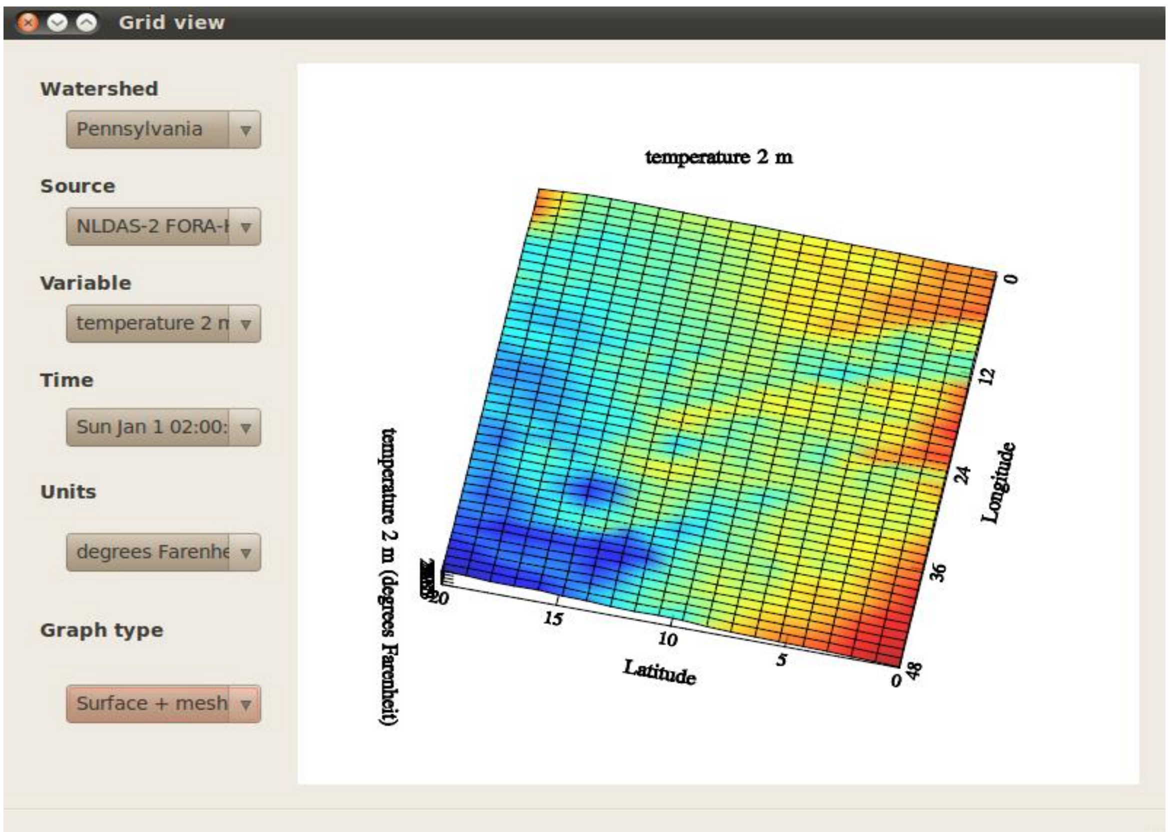

Figure 5.4 Grid data as graph

There are multiple ways in which the retrieved data can be drawn into graphs. Dropdown for Watershed (Study area), Data Source, variable, time, units and graph types which include boxes, surface, mesh, surface + mesh, contours, area etc..

A very easy to use interface has been provided. The user need not worry about the Server url, response format or how the data is stored in the database. Graphs can be drawn out of the data available in the database.

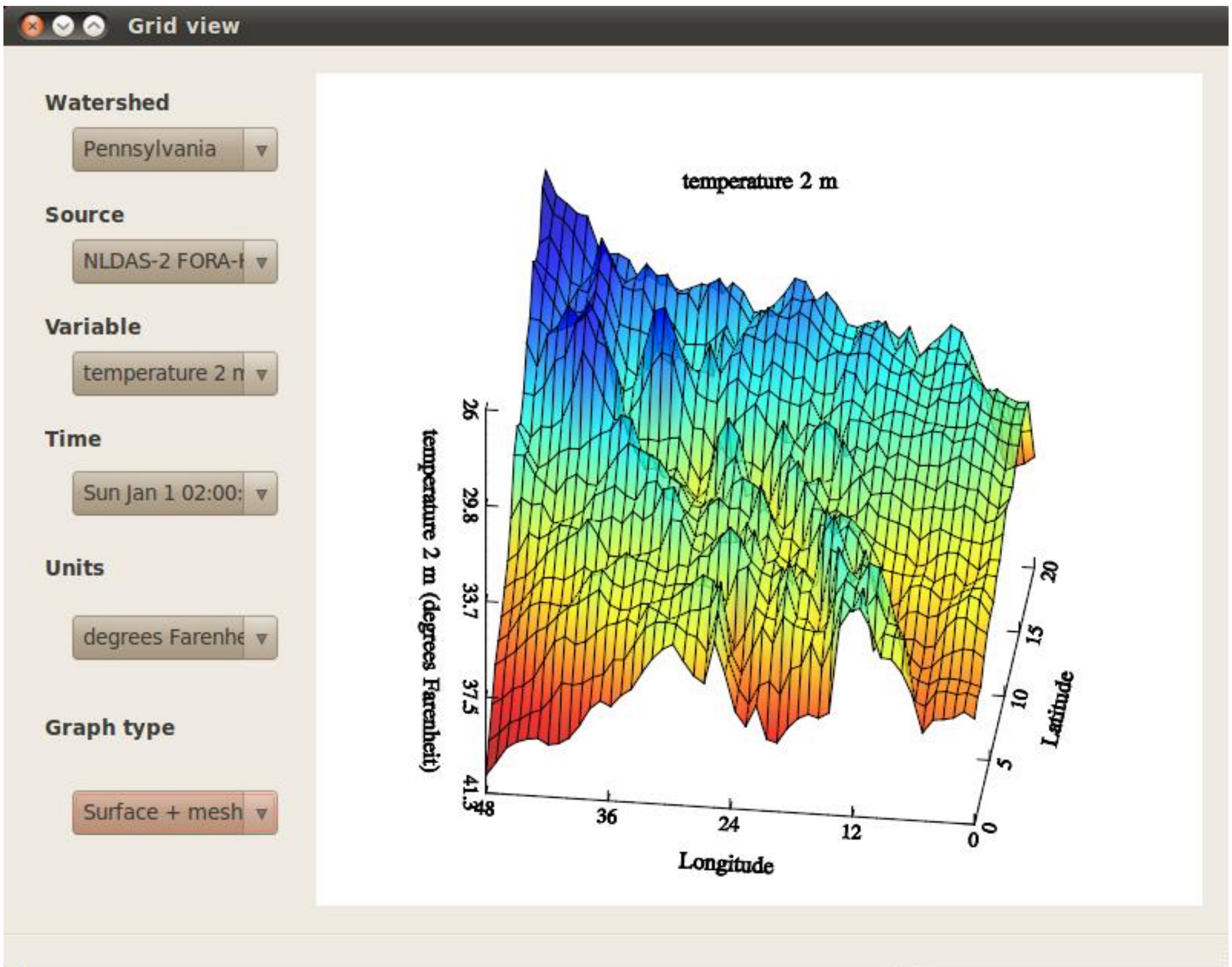

Figure 5.5 Grid data as graph 3d

5.3 Developers Guide for adding clients

To add a client using the architecture provided, the steps that are required to be followed are:

Identify the protocol followed by the server (Please note that the code written is written based on http GET/POST and OPeNDAP). Every GET request URL has a domain Url appended with query parameters.

Eg: http://waterservices.usgs.gov/nwis/iv/?bBox=-81.500000,36.000000,-81.000000,37.000000&startDT=2013-02-19&endDT=2013-02-19&parameterCd=00060,72131 - "Date Last Accessed: March 21, 2016".

http://waterservices.usgs.gov/nwis/iv/? : Domain URL + context path

bBox=-81.500000,36.000000,-81.000000,37.000000&startDT=2013-02-19&endDT=2013-02-19&parameterCd=00060,72131 : Query parameters

Depending on the specification provided by data source, the URL needs to be built by taking the user inputs. Forming the request URL according to the server specifications is vital.

Timestamp is mandatory, variable (if multiple measurements are reported this would be mandatory, for eg: USGS reports precipitation, water gauge etc., in case of NWS Radar data it is always precipitation, so no variable needs to be explicitly passed) and the coordinates (Location where we need to query the data) are required.

Sometimes the server gives back the data for the whole nation (in that case the coordinates would not be necessary, as in case of radar data, the precipitation is always reported for the country).

Once the server URL is ready, we need to send out the request depending on the protocol.

The two classes that will be useful here would be:

1. Opendap.h – Library for downloading and saving data from server following opendap.
2. Rest.h – Library for downloading and saving data from server on Rest GET/POST protocol.

In case of OPeNDAP : The method OPeNDAP_getData would retrieve the data and store it in the private variable dataArray. This array would contain the required data. The OPeNDAP has a very peculiar way of reporting the data. The data array is a one dimensional array. Here raises the question of how do we process/interpret it and store it in the DB. Please find the example below for better understanding:

Eg: data array: 0.1 0.0 0.2 0.4

If you have queried for data across two steps of latitude and longitude and one time stamp then the data would look similar as below:

For one time slice [0.1 0.0]

[0.2 0.4]

The array would be reported as a 1-d array but which actually is a 2X2. We need to have logic to transform this into the required dimensional array.

In case of Rest GET: HttpGetAsString or HttpGetAsFile can be used. This would retrieve the data from the servers. The method getDataAndStore would identify if the request is for USGS data or Radar data. Once identified, the method would retrieve the data, and transform it on the fly and save it in the database. In case of new file formats or new data sources, the data would be downloaded to a file automatically and the location would be specified. The file can be used for data transformation and storing. Few methods for XML parsing and reading DBF file are very specific to the schema. But can be reused by providing the tag names in case of xml parsing.

For any data source to be added that is not supported by the framework provided, clients and the libraries needed should be added and the response from the server needs to be transformed into the data model provided. The interface built to generate graphs can be used to generate the graphs.

The path to the header files and the libraries have to be added to the dataincoming.pro file, this file takes care of loading the libraries during run time.

5.4 Summary of This Chapter

There are five modules to be coded every time there is a new data source that is to be added. The GUI to take user inputs, UI Logic to pass on the information from GUI to the integration layer, where the request for data and the response is handled, Math logic layer to transform the retrieved data if needed and the DAO to persist the data.

OPeNDAP and Rest are the two classes provided that can be used in the integration layer to retrieve data from the server. DAOGridData and DAOPointData are the classes with necessary methods to help write data to the database.

CHAPTER 6. SUMMARY AND FUTURE WORK

6.1 Summary

With the framework provided it is easy to download all the data provided on the web that follow the OPeNDAP and REST (GET and POST) protocol with the DAO required to save the data. User that is going to use the framework has to transform the data into the provided data model. The data model is always going to be grid or point data. Depending on the type of data reported this appropriate DAO can be used to handle it. Once the data is downloaded, the interface to generate graphs can be used as needed.

6.2 Future Work

With different formats being reported, a framework that would transform the response into the required Grid or point data would be really helpful. XML and JSON are the widely used formats for a server to respond. There are wide varieties of libraries that can do that in JAVA, we can build libraries that can parse JSON/XML and convert it into the Plain old C++ Object which can be later transformed into the data model with ease. This effort would also involve in increasing the efficiency of processing the data.

Also, with the data sources publishing periodically, it would be appropriate to have a module that can continuously get the data and store it in the local database.

This would be really good for the end user where he does not have to wait till the data is downloaded and processed and stored into the database to use. It is also necessary to monitor the services at the remote data servers regularly to obtain information on the data version updated to have non-broken software. With huge data available, there is every chance that the database is going to run out of memory eventually. There is a definite need for purging data. Necessary rules that define what data needs to be purged have to put into place. We can incorporate asynchronous http calls whenever applicable. This would highly increase the performance of the system in case multiple http calls are involved for a single user request.

LIST OF REFERENCES

LIST OF REFERENCES

1. Agarwal, Deb, et al. "A methodology for management of heterogeneous site characterization and modeling data." *The XIX International Conference on Computational Methods in Water Resources (CMWR 2012). Urbana-Champaign, IL*. 2012.
2. Sun, Xiaojuan, et al. "Development of a Web-based visualization platform for climate research using Google Earth." *Computers & Geosciences* 47 (2012): 160-168.
3. Gallagher, James, et al. "The data access protocol—DAP 2.0." (2004).
4. Cornillon, Peter, James Gallagher, and Tom Sgouros. "OPeNDAP: Accessing data in a distributed, heterogeneous environment." *Data Science Journal* 2 (2003): 164-174.
5. Date, C. J. "Database systems." *Vols. I & II, Narosa Pub* (1986).
6. Frakes, William B., and Kyo Kang. "Software reuse research: Status and future." *IEEE transactions on Software Engineering* 7 (2005): 529-536.
7. Colombo, F. (2011), "It's not just reuse" - http://sharednow.blogspot.com/2011/05/its-not-just-reuse.html - "Date Last Accessed: March 21, 2016".
8. https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm - "Date Last Accessed: March 21, 2016".
9. https://tools.ietf.org/html/rfc1945#section-8 "Date Last Accessed: March 21, 2016"
10. https://curl.haxx.se/libcurl/ - "Date Last Accessed: March 21, 2016".
11. https://curl.haxx.se/docs/features.html - "Date Last Accessed: March 21, 2016".
12. https://curl.haxx.se/libcurl/competitors.html - "Date Last Accessed: March 21, 2016".
13. http://www.opendap.org/download/libdap - "Date Last Accessed: March 21, 2016".
14. http://lombardhill.com/what_reuse.htm - "Date Last Accessed: March 21, 2016".
15. Lam, Tak, Jianxun Jason Ding, and Jyh-Charn Liu. "XML document parsing: Operational and performance characteristics." *Computer* 9 (2008): 30-37.
16. Aho, Alfred V., and Jeffrey D. Ullman. *The theory of parsing, translation, and compiling*. Prentice-Hall, Inc., 1972.
17. http://www.nws.noaa.gov/oh/hrl/nwsrfs/users_manual/part2/_pdf/21hrapgrid.pdf - Date Last Accessed: March 21, 2016".

18. Peckham, Scott D., and Jonathan L. Goodall. "Driving plug-and-play models with data from web services: A demonstration of interoperability between CSDMS and CUAHSI-HIS." *Computers & Geosciences* 53 (2013): 154-161.
19. http://docs.opendap.org/index.php/UserGuideDataModel#Grid - "Date Last Accessed: March 21, 2016".

APPENDICES

Appendix A Development Environment and Configuration

Repository: SVN Subversion

Development IDE: Qt libraries and Qt Creator

External Libraries : libdap, libxml, zlib and libcurl.

Database: PostgreSQL 8.4

Installation: The documentation includes a procedure to install the environment for development or testing. It includes the pre-requisites checking, compilation of external libraries and installation of database engine, its configuration, and development tools.

Appendix B Table Definitions

Table B 1 Grid Data

Column Name	Type	Size	Key
cod	Integer		Primary
codwatershed	Integer		Foreign
codsource	Integer		Foreign
codvariable	Integer		Foreign
codvartype	Integer		Foreign
typetime	Bigint		
obsid	Varchar	64	
datetime	Timestamp		
west	Numeric	(20,10)	
east	Numeric	(20,10)	
south	Numeric	(20,10)	
north	Numeric	(20,10)	
data	double	Precision[]	

UNIQUE KEY (codwatershed, codsource, codvariable, obsid, datetime)

Table B 2 Watershed

Column Name	Type	Size	Key
Cod	Integer	64	Primary
codparent	Integer		
Name	varchar	64	
description	varchar	64	
West	numeric	(20,10)	
East	numeric	(20,10)	
South	numeric	(20,10)	
north	numeric	(20,10)	

UNIQUE KEY (name)

Table B 3 Source

Column Name	Type	Size	Key
cod	Integer	64	Primary
groupname	varchar	64	
name	varchar	64	
fullname	varchar	64	
description	varchar	64	
url	varchar	255	
since	timestamp		
until	timestamp		

UNIQUE KEY (name)

Table B 4 Variable

Column Name	Type	Size	Key
Cod	Integer		Primary
Name	varchar	64	
description	varchar	255	
coddimension	Integer		Foreign

UNIQUE (name)

Table B 5 Pointvalues

Column Name	Type	Size	Key
codpointdata	Integer		Foreign
datetime	timestamp		
Value	double		

Table B 6 Points

Column Name	Type	Size	Key
Cod	Integer		Primary
codwatershed	Integer		Foreign
name	varchar	64	
description	varchar	255	
latitude	numeric	(20,10)	
longitude	numeric	(20,10)	
elevation	numeric	(20,10)	

UNIQUE (codwatershed, name)

Table B7 Point Data

Column Name	Type	Size	Key
Cod	Integer		Primary
codpoint	Integer		Foreign
codsource	Integer		Foreign
codvariable	Integer		Foreign
codvartype	Integer		Foreign
typetime	bigint		
Obsid	varchar	64	

UNIQUE (codpoint, codsource, codvariable, obsid)

Table B 8 Units

Column Name	Type	Size	Key
Cod	Integer		Primary
Value	varchar	1024	
description	varchar	1024	

Table B 9 Dimensions

Column Name	Type	Size	Key
Cod	Integer		Primary
Name	varchar	64	
Symbol	varchar	64	
Description	varchar	255	
Cumulative	boolean		
Codreferenceunits	Integer		Foreign
Codpreferredunits	Integer		Foreign

UNIQUE (name)

Appendix C Sample Request Response

Table C 1 OPeNDAP Request/Response

Func	Format
Request	http://hydro1.sci.gsfc.nasa.gov/dods/NLDAS_FOR0125_H.001.asc?vgrd10m[0:1][121:122][121:122] - "Date Last Accessed: March 21, 2016".
Response	One dimensional Array -2.0, -2.0, -2.0, -2.0,-1.6, -1.65,-1.55,-1.6
Notes	**index = (input - minimum) / Resolution** [time][lat][lon] in query translates to [01/08/1996 00:00:00, 01/08/1996 01:00:00][40.1875: 40.3125][-109.8125: -109.6875] To be interpreted as [0][0], -2.0, -2.0 - 1/8/1996 00:00:00 at [40.1875,-109.8125] and [[40.1875,-109.6875]] [0][1], -2.0, -2.0 - 1/8/1996 00:00:00 at [40.3125,-109.8125] and [[40.3125,-109.6875]] [1][0], -1.6, -1.65 - 1/8/1996 01:00:00 at [40.1875,-109.8125] and [[40.1875,-109.6875]] [1][1], -1.55, -1.6 - 1/8/1996 01:00:00 at [40.1875,-109.8125] and [[40.1875,-109.6875]]

Table C 2 USGS Request/Response

Func	Format
Request	http://nwis.waterservices.usgs.gov/nwis/iv/?bBox=-81.250000,36.000000,-81.000000,36.250000&startDT=2013-02-19T00:00:00&endDT=2013-02-19T00:00:15&parameterCd=00060 - "Date Last Accessed: March 21, 2016".
Response	<ns1:timeSeriesResponse ….. > <ns1:queryInfo xmlns:ns2="http://www.cuahsi.org/waterML/1.1/">...</ns1:queryInfo> <ns1:timeSeries xmlns:ns1="http://www.cuahsi.org/waterML/1.1/" name="USGS:0211139110:00060:00011"> ………… <ns1:geogLocation xsi:type="ns1:LatLonPointType" srs="EPSG:4326"> <ns1:latitude>36.1361111</ns1:latitude> <ns1:longitude>-81.2233333</ns1:longitude> </ns1:geogLocation> ……….. <ns1:variable ns1:oid="45807197"> <ns1:variableCode network="NWIS" vocabulary="NWIS:UnitValues" default="true" variableID="45807197">00060</ns1:variableCode> <ns1:variableName>Streamflow, ft³/s</ns1:variableName> <ns1:variableDescription>Discharge, cubic feet per second</ns1:variableDescription> <ns1:valueType>Derived Value</ns1:valueType> <ns1:unit> <ns1:unitCode>ft3/s</ns1:unitCode> ………… <ns1:value qualifiers="A" dateTime="2013-02-19T00:00:00.000-

Table C 2 Continued

Table C 2 Continued

	05:00">520</ns1:value> </ns1:timeSeries> <ns1:timeSeries xmlns:ns1="http://www.cuahsi.org/waterML/1.1/" name=" USGS:02111500:00060:00011">...</ns1:timeSeries> <ns1:timeSeries xmlns:ns1="http://www.cuahsi.org/waterML/1.1/" name=" USGS:02112000:00060:00011">...</ns1:timeSeries> </ns1:timeSeriesResponse>
Note s	Request : http://waterservices.usgs.gov/nwis/iv/? bBox=-81.250000,36.000000,-81.000000,36.250000& startDT=2013-02-19T00:00:00& endDT=2013-02-19T00:00:15& parameterCd=00060. "Date Last Accessed: March 21, 2016". Readable requests and response formats.

Table C 3 NWS Radar data Request/Response

<table>
<tr><th>Func</th><th>Format</th></tr>
<tr><td>Request</td><td>http://www.srh.noaa.gov/ridge2/Precip/qpehourlyshape/2015/201512/20151210/nws_precip_2015121004.tar.gz - "Date Last Accessed: March 21, 2016".</td></tr>
<tr><td>Response</td><td>
<table>
<tr><th>Id</th><th>Hrap x</th><th>Hrap y</th><th>Lat</th><th>Lon</th><th>Globvalue</th></tr>
<tr><td>297818</td><td>1544</td><td>198</td><td>18.1236</td><td>-65.8086</td><td>0.01</td></tr>
<tr><td>415461</td><td>987</td><td>273</td><td>29.6554</td><td>-81.1638</td><td>0.02</td></tr>
<tr><td>492657</td><td>959</td><td>322</td><td>31.5997</td><td>-81.4024</td><td>0.05</td></tr>
<tr><td>494234</td><td>960</td><td>323</td><td>31.6178</td><td>-81.3483</td><td>0.02</td></tr>
<tr><td>495811</td><td>961</td><td>324</td><td>31.6358</td><td>-81.2941</td><td>0.02</td></tr>
<tr><td>631385</td><td>999</td><td>410</td><td>33.8159</td><td>-78.3100</td><td>0.03</td></tr>
<tr><td>631386</td><td>1000</td><td>410</td><td>33.7999</td><td>-78.2715</td><td>0.12</td></tr>
<tr><td>632961</td><td>999</td><td>411</td><td>33.8478</td><td>-78.2906</td><td>0.04</td></tr>
<tr><td>632962</td><td>1000</td><td>411</td><td>33.8318</td><td>-78.2522</td><td>0.14</td></tr>
<tr><td>632963</td><td>1001</td><td>411</td><td>33.8157</td><td>-78.2138</td><td>0.27</td></tr>
<tr><td>632964</td><td>1002</td><td>411</td><td>33.7996</td><td>-78.1754</td><td>0.22</td></tr>
<tr><td>632965</td><td>1003</td><td>411</td><td>33.7834</td><td>-78.1371</td><td>0.14</td></tr>
</table>
</td></tr>
<tr><td>Notes</td><td>Request :http://www.srh.noaa.gov/ridge2/Precip/qpehourlyshape/2015/201512/20151210/nws_precip_2015121004.tar.gz - "Date Last Accessed: March 21, 2016".
year/yearmonth/yearmonthdate/yearmonthdatehour Response is in readable format.</td></tr>
</table>