We recommend that the thesis prepared under our supervision by

MD MOINUL HOSSAIN

entitled

A SOFTWARE ENVIRONMENT FOR WATERSHED MODELING

be accepted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Sergiu Dascalu, Ph.D. – Advisor

Frederick C. Harris, Jr, Ph.D. – Committee Member

Yantao Shen, Ph.D. – Graduate School Representative

David W. Zeh, Ph.D. – Dean, Graduate School

August 2016
ABSTRACT

The proposed Software Environment for Watershed Modeling is a service-oriented framework developed to assist watershed modeling for the NSF EPSCoR-funded Western Consortium for Water Analysis, Visualization, and Exploration (WC-WAVE) project. Watershed scientists use different modeling tools to model physical phenomena across different basins. We propose a web-service centric framework to expose models as services. The framework will allow model execution in the cloud environment, submission of model data through the NetCDF standard data format and storage and access to the model resources through web services. The framework allows an easy way to publish models as linux container images through an image hub.

We present background on the WC-WAVE project and the different models used in the system, describe the design and implementation of the system, provide some feature based comparisons with other related works and finally conclude the thesis with remarks and directions for future works.
ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Sergiu Dascalu for being an excellent mentor throughout the course of my study at UNR. I would also like to thank Dr. Fredrick C Harris and Dr. Yantao Shen for their continuous support on my thesis work.

I would also like to thank all the members of the WC-WAVE team with whom I had a chance to work in close collaboration, including Matt Turner, Rui Wu, Chao Chen, Lisa Palathingal, Chase Carthen, Dr. Luke Sheneman, Dr. Karl Benedict, John Erickson, John Savickas and Hays Barrett.

Finally, I am thankful to the National Science Foundation for the support provided during my Masters studies. The material presented in this thesis is based upon work supported by the National Science Foundation under grant number IIA1329469. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.
# TABLE OF CONTENTS

Abstract ................................................................. i
Acknowledgements .................................................... ii
Table of Contents ....................................................... iii
List of Tables ............................................................ v
List of Figures ........................................................... vi

1 Introduction .......................................................... 1

2 Problem and Motivation ............................................. 3
  2.1 Watershed Modeling ............................................ 3
      2.1.1 Watershed .............................................. 3
      2.1.2 ISNOBAL Model ........................................ 4
      2.1.3 PRMS Model ........................................... 5

3 Background and Related Works .................................. 8
  3.1 Service Oriented Architecture (SOA) and Web Services .... 8
  3.2 RESTful Web Services ....................................... 8
      3.2.1 Characteristics of REST ............................. 9
      3.2.2 Components in REST Architecture ................. 10
  3.3 Software as a Service (SaaS) ................................ 12
  3.4 Microservices ................................................ 12
      3.4.1 Microservices Characteristics ...................... 14
      3.4.2 Benefits of Using Microservices ................... 15
  3.5 Related Works ................................................ 16
  3.6 Related Software Products .................................. 16
      3.6.1 CSDMS: The Community Surface Dynamics Modeling System 17
      3.6.2 HydroShare ........................................... 18
      3.6.3 Model as a Service (MaaS) .......................... 20
      3.6.4 Other Related Products ............................... 21

4 Overview of the Solution ........................................ 23
  4.1 Overview of the WC-WAVE Project .......................... 23
      4.1.1 Data Service .......................................... 23
      4.1.2 Modeling Service .................................... 24
      4.1.3 Virtual Watershed Clients ........................... 25
  4.2 Overview of Model API Design ................................ 26

5 Specification and Requirements ............................... 27
  5.1 Requirement Specification .................................. 27
      5.1.1 Functional Requirements ............................. 27
      5.1.2 Non Functional Requirements ......................... 27
  5.2 Use Case Modeling ............................................ 28
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.1</td>
<td>Use Case Diagram</td>
<td>29</td>
</tr>
<tr>
<td>5.3</td>
<td>Glossary</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>Design and Implementation</td>
<td>36</td>
</tr>
<tr>
<td>6.1</td>
<td>System Level Diagram</td>
<td>36</td>
</tr>
<tr>
<td>6.2</td>
<td>Detailed Design</td>
<td>38</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Authentication Service</td>
<td>38</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Model Web Service</td>
<td>40</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Model Adaptors</td>
<td>41</td>
</tr>
<tr>
<td>6.2.4</td>
<td>Model Worker</td>
<td>43</td>
</tr>
<tr>
<td>6.3</td>
<td>Deployment Workflow</td>
<td>44</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Model Registration</td>
<td>46</td>
</tr>
<tr>
<td>6.4</td>
<td>Prototype Details</td>
<td>46</td>
</tr>
<tr>
<td>6.5</td>
<td>Technologies Used and Software Characteristics</td>
<td>52</td>
</tr>
<tr>
<td>7</td>
<td>Comparison with Related Tools</td>
<td>54</td>
</tr>
<tr>
<td>8</td>
<td>Conclusion and Future Work</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>58</td>
</tr>
</tbody>
</table>
### LIST OF TABLES

3.1 HTTP Verbs and Their Meaning ........................................... 12

5.1 Functional Requirements of the System ............................ 28

5.2 Non Functional Requirements of the System ...................... 28

5.3 Glossary of the terms ..................................................... 32

7.1 Feature Comparison with Related Hydrologic and Environmental Modeling tools ................................................. 55
LIST OF FIGURES

2.1 Conceptual model of hydrologic process in PRMS [13] . . . . . . . . 6
3.1 The monolithic architecture ........................................ 13
3.2 The microservices architecture ..................................... 14
3.3 The graphical user interface of CSDMS Modeling Tool (CMT) where modelers can setup and run their models to experiment with minimal effort [17] ........................................... 18
3.4 The high level architecture of HydroShare [21] .................. 19
3.5 The general architecture of MaaS [11] ........................... 20
4.1 High-level diagram of the WC-WAVE project components ...... 24
5.1 The use case diagram of the system ............................... 29
6.1 The system level diagram of the system. .......................... 37
6.2 The workflow for accessing secure REST endpoint with JWT token . 39
6.3 The work flow from user’s perspective to run a model .......... 41
6.4 The class diagram of the components in model web service backend 42
6.5 The design of a model adaptor interface .......................... 43
6.6 The design of the worker through class diagram .................. 44
6.7 The workflow of a model worker depicted in a flow chart ....... 45
6.8 The workflow for registering a model in the system ............. 47
6.9 The registration page of the portal ................................. 48
6.10 The dashboard for user’s model runs ............................. 49
6.11 The dynamically generated upload form ......................... 49
6.12 Scenario creation interface for PRMS model that uses Modeling REST API to execute model .............................. 51
6.13 Comparing the results of the scenarios created using the modeling service ........................................... 52
6.14 Example of running a hundred model through an ipython notebook with different parameters ............................. 53
CHAPTER 1
INTRODUCTION

Modeling of physical processes is a core part of the scientific inquiries. Scientists in all domains including earth science build computer models to investigate physical phenomena. Softwares are becoming a critical part of the modern scientific research as a result. Quality, scalability, and maintainability are significant concerns for scientific software. Issues like data storage, retrieval, running and coupling models are hard problems and require extra care from the perspective of software engineering. Designing integrated systems that provide means to handle all these issues can be a challenging job.

Building software tools and frameworks for scientific research can be interesting for many reasons. With the advancement of computing power in recent decades, scientific research is creating more and more data and models independently built by scientific researchers. It is an exciting field where software engineering can assist this emergence by facilitating the creation of distributed software systems and frameworks to assist scientists to have collaboration on these data and models. Another significant aspect of this field is, being an interdisciplinary field it poses lots of challenges regarding barriers to communication and team building among different communities involved in the process.

The work presented in this thesis is part of the NSF EPSCoR-supported Watershed Analysis, Visualization, and Exploration (WC-WAVE) project, initiated by the Nevada, Idaho and New Mexico jurisdictions of EPSCoR. WC-WAVE is a collaborative project with three principal components, watershed science, cyberinfrastructure data and visualization. The goal of the project is to bring watershed scientists, hydrologists, and cyberinfrastructure teams together to build a platform
called Virtual Watershed. The overall plan of the project is to develop software tools for watershed scientists to allow data storage and sharing, on demand modeling and visualization through an integrated system.

Researchers in WC-WAVE project use different hydrologic models like ISNOBAL, PRMS, etc. to do modeling of hydrologic processes of various watersheds including Dry Creek and Reynolds Creek in Idaho, Jemez Creek in New Mexico and Lehman Creek in Nevada. We introduced a framework for representing these model data in the standard format called Network Common Data Format (NetCDF) and expose the models through web services.

This thesis, in its remaining parts, is arranged as follows: Chapter 2 presents the problem and its motivations; Chapter 3 provides background and related works; Chapter 4 gives an overview of the solution; Chapter 5 describes the specification and requirement analysis; Chapter 6 contains the detailed design and implementation; Chapter 7 contains a feature-based comparison with related tools and Chapter 8 concludes the thesis with possible future work and our final remarks.
CHAPTER 2
PROBLEM AND MOTIVATION

2.1 Watershed Modeling

As this work is tailored for watershed modeling, a branch of environmental science dedicated for understanding watersheds, this chapter gives an understanding of watershed and different watershed models used in the process of building the environment.

2.1.1 Watershed

A watershed is the area of land where all of the water that falls in it and drains off of it goes to a common outlet. A watershed can vary in size. It can be as small as a footprint and as big enough to encompass all the land that drains water into rivers that drain into the Chesapeake Bay, where it enters the Atlantic Ocean. Drainage basin or catchment are also synonymous terms used to denote watershed in general. A watershed is a complex system and a career could be built on trying to model a watershed water budget (correlating water coming into a watershed to water leaving a watershed). So the study of watersheds is very important to understand the variables related to the environment around it. There are many factors such as precipitation, infiltration, soil characteristics, evaporation etc that determine how much water flows in a stream and makes watershed modeling a hard task [23].
2.1.2 ISNOBAL Model

ISNOBAL is a model initially developed by Marks et al. to simulate the development and melting of the seasonal snow cover in several mountain basins in California, Idaho, and Utah. It is a DEM (Digital Elevation Model) grid-based model that uses the energy balance to calculate snowmelt, runoff, from snow properties, terrain and region characteristics, precipitation, and climate [12].

The model approximates the snow cover as a composition of two layers, a surface fixed-thickness active layer and a lower layer. The model calculates the temperature ($^\circ$C) and specific mass ($kg/m^2$) which is the mass per unit area (from $density \times depth (kg/m^3 \times m)$) for each layer, and then calculates the temperature and specific mass for the snow cover on top [12]. Melt is computed in both layers when the accumulated energy exceeds the cold content or when the cold content is $> 0.0$. Cold content is the energy required to bring the snow cover temperature up to freezing ($0^\circ$C). Runoff is estimated when the accumulated melt and liquid $H_2O$ content exceeds a user-defined threshold [12].

ISNOBAL comes as a pre-compiled binary with the IPW (Image Processing Workbench) software package. As ISNOBAL is a DEM based model input image data are required for initial snow cover and surface conditions, for input climate data at each time-step, and for precipitation events. Initial condition data are specified only at the beginning of the run. Precipitation data include an ASCII file determining the time since the run start for the event and the name of the precipitation file, which contains precipitation mass, % snow, snow density, and precipitation temperature [9].

ISNOBAL being an energy balance snowmelt model, it provides spatially dis-
tributed estimation of snow melt over large mountain basin or watersheds. The model simulates the topographic controls on snow deposition and melt, during the winter when the snow starts to develop and the during spring and early summer when snow begins to melt and get disposed [12].

2.1.3 PRMS Model

The Precipitation-Runoff Modeling System or PRMS is another widely used model for general watershed hydrology. It is a deterministic, distributed-parameter, physical process based modeling system that evaluates the response of various combinations of climate and land use on a watershed [13]. The model was first developed in 1983 as a single FORTRAN program composed of algorithms describing the physical processes as subroutines. The current version of the model is version 4 which has become more mature over the years of development. It has been used to model different hydrology application since its release including water and natural resource management, measurement of interaction of groundwater and surface water, interaction of climate and atmosphere with surface water and many more [13]. Fig.2.1 shows the conceptual model of the hydrologic processes in a watershed according to PRMS model.

The inputs to the PRMS model are daily time-series values of precipitation, minimum and maximum air temperature, and short-wave solar radiation. If not provided, short-wave solar radiation can be calculated by the model itself. Precipitation can be provided in the form of rain, snow, or a mixture of both. The precipitation is reduced by vegetative canopy interception. Air temperature and solar radiation energy inputs simulate the processes of evaporation, transpiration,
sublimation, and snow melt. PRMS calculates the hydrologic response of the watershed based on the parameters provided. PRMS divides the land into discrete Hydrologic Response Units (HRUs) based on hydrologic and physical characteristics and each HRU is assumed to be homogeneous on these hydrologic and physical properties. In PRMS Version 4 an HRU can be one of four types available which are land, lake, swale, or inactive [13].

Input data for a PRMS model run are specified in 3 textual files with PRMS specific format. The output files created by a model run are also in the same format as inputs that describes the output variables. The inputs files for PRMS are control file, data file, and the parameter file. Control parameters are specified in the control file. The parameters can be of 5 types [13]:

- Parameters related to model execution.
• Parameters related to model input.
• Parameters related to model output.
• those related to initial conditions.
• those related to the specification of the active modules.

The data file is used to provide time-series data which can include daily precipitation, maximum and minimum air temperatures, solar radiation, pan evaporation, stream flow, humidity, wind speed, and snow pack water equivalent. The parameter file contains the parameter values specific to each module that does not change during a model run [13].

There can be four output files generated by a PRMS model run which are:

• The Water-Budget File.
• The Statistic Variables File.
• The Animation File.
• The Map Results File.

The Water-Budget file contains a summary table of the water budget for a PRMS model run. The statistic variables file is a text file that provides selected variables as time-series output. Animation files are text files that provide selected variable output as time series of spatial arrays. Map results files contain selected variable output as time series of spatial arrays that have been mapped to alternative spatial and temporal resolutions [13].
CHAPTER 3
BACKGROUND AND RELATED WORKS

In this chapter, we introduce the high level idea of different software engineering techniques used throughout the work. We also overview different projects and tools that are related to this work.

3.1 Service Oriented Architecture (SOA) and Web Services

Service Oriented Architecture (SOA) is an architecture for software systems which has gained significant focus in the IT industry in recent years [5]. Service Oriented Architecture is an abstract architectural pattern that is inspired from real life societies. Erl used the analogy of an average cosmopolitan city to describe the central idea of service oriented approach for software design. In such an environment businesses are decomposed into standalone services offered to consumers by allowing to achieve a distributed model. So Service Oriented Architecture constitutes a model where business logic for software is decomposed into distinct units or services, where each unit is self-contained, and collectively they represent the aggregated business logic [5].

3.2 RESTful Web Services

REST stands for Representational State Transfer Protocol. It is primarily an architectural style for distributed hypermedia systems introduced by Fielding, Roy Thomas in his PhD dissertation [6]. REST is explained in details in this section as
it has been adopted for building the main architecture of the system. REST defines
a way for a client-server architecture on how a client and a server should interact.
REST defines a set of principles on how a client will consume resources from a
RESTful server. Though REST is widely used with the HTTP protocol, it can be
used as any application layer protocol that provides means for RESTfulness.

3.2.1 Characteristics of REST

There are several properties that makes a client-server architecture RESTful when
applied. The main properties include:

- Statelessness
- Uniform Interface
- Cache

The properties are described in details in the next sections:

**Statelessness** Statelessness is the most important property or constraint for a
client-server architecture to be RESTful. The communication between the client
and the server must be stateless, which means the server is not responsible for
keeping the state of the communication. It is the client’s responsibility. A request
from the client must contain all the necessary information for the server to under-
stand the request [6]. Two subsequent requests to the server will not have any in-
terdependence between each other. Introducing this property on the client-server
architecture presents several benefits regarding visibility, reliability, and scalability
For example, as the server is not responsible for keeping the state and two subsequent requests are not interrelated, multiple servers can be distributed across a load-balanced system where different servers can be responsible for responding to different requests by a client.

**Uniform Interface**  Another important property of a RESTful architecture is it provides a uniform interface for the client to interact with a server. Instead of an application’s particular implementation, it forces the system to follow a standardized form. For example HTTP 1.1 which is a RESTful protocol provides a set of verbs (e.g., GET, POST, PUT, DELETE, etc.) for the client to communicate with the server. The verbs work as an interface making the client-server communication generic. Table 3.1 provides the details of the verbs.

**Cache**  REST architecture introduces cache constraint to improve network efficiency [6]. A server can allow a client to reuse data by enabling explicitly for labeling some data cacheable or non-cacheable. A server can serve data that will not change in future as cached content allowing the client to eliminate partial interaction for those data in a series of requests.

### 3.2.2 Components in REST Architecture

REST is an abstract protocol. It describes the principal and components needed for a concrete protocol based on it. The main components of REST architecture include:

- Resources
• Representations

• Resource Identifiers

**Resources**  The resource is the main abstract representation of data in REST architecture [6]. Any piece of data in server can be represented as a resource to a client. A document, an image, data on today’s weather, a social profile, everything is considered a resource in the server. formally, a resource is a temporarily varying function of $M_R(t)$ that maps to a set of entities for time $t$ [6].

**Representations**  A resource is the abstract building block of the data in a web server. For a client to consume the resource it needs to be presented in a way the client can understand. This is called representation. A representation is a presentation format for representing the current state of a resource to a consumer. Some commonly used resource representation format in the current standard are HTML (Hypertext Markup Language), XML (Extensible Markup Language), JSON (JavaScript Object Notation), etc. A server can expose data content in different representations so that consumer can access the resources through resource identifiers (discussed in section 3.2.2) in the desired format.

**Resource Identifiers**  A resource is uniquely identified through a resource identifier in a RESTful architecture. For example, in HTTP Uniform Resource Identifier (URI) is used to identify a resource in a server. A URI can be thought as the address of a resource in the server [19]. A resource identifier is the key for a client to access and manipulate a resource in the server.
### 3.3 Software as a Service (SaaS)

With the widespread popularity and effective use of RESTful web services, a new kind of software delivery architecture has emerged which is termed as Software as a Service (SaaS). SaaS is a similar outcome of service oriented approaches like Infrastructure as a Service (IaaS) or Platform as a Service (PaaS). SaaS is essentially a software delivery method where the service is delivered to a customer through the Internet instead of the need for local installation. RESTful web service architecture has become the de facto standard for shipping software as a service. SaaS is currently regarded as an important IT trend as according to industry analysts, its increasing sales and continuing growth in the industry [3].

### 3.4 Microservices

Microservices is a software architecture in which a complex application is decomposed into small *components* or *services* that communicates with each other through well-defined language-agnostic APIs [7]. The Microservices is a relatively new buzzword in the world of software architecture. The architecture emerged as a solution to the numerous complexities attached with traditional monolithic archi-

---

**Table 3.1: HTTP Verbs and Their Meaning**

<table>
<thead>
<tr>
<th>VERB</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Read a Resource from server</td>
</tr>
<tr>
<td>POST</td>
<td>Create a Resource in server</td>
</tr>
<tr>
<td>PUT</td>
<td>Update a Resource in server</td>
</tr>
<tr>
<td>DELETE</td>
<td>Remove a Resource from server</td>
</tr>
<tr>
<td>HEAD</td>
<td>Return response headers (metadata on resource)</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>List allowed operations on a Resource</td>
</tr>
</tbody>
</table>
Traditional monolithic applications are built as a single unit using a single language stack and often composed of three parts, a front end client, a backend database and an application server sitting in the middle that contains the business logic. Here the application server is a monolith that serves as a single executable. A monolithic application can be scaled horizontally by replication the application server behind a load balancer to serve the clients in scale. The biggest issue with monolithic architecture is that, as the application grows, the deployment cycle becomes longer as a small change in the codebase requires the whole monolith to be rebuilt and deployed [7]. It leads to higher risk for maintenance as the application grows. These pitfalls have lead to the idea of decomposing the business capabilities of an application into self-contained services.
3.4.1 Microservices Characteristics

Being a relatively new idea, researchers have attempted to formalize a definition and characteristics of Microservices. Fowler and Lewis [7] has put together a few essential characteristics of a microservice architecture that are described in brief in the following sections.

**Componentization via Services** The most important characteristic of a microservice architecture is that the functionalities need to be componentized regarding services. Instead of thinking components as libraries that use in-memory function calls for inter-component communication, we can think of components in terms out-of-process services that communicate over the network, quite often through a web service or a remote procedure call. A service has to be atomic, doing one thing
and doing one thing well [16].

**Organized Around Business Capabilities**  Monolithic applications are typically organized around technology layers. For example, a typical multi-tiered application might be split logically into persistence layer, application layer and UI layer and teams are also organized around the technologies. This logical separation creates the need for inter-team communication even for a simple change. In microservices, the product is organized around business capabilities where a service is concentrated on one single business need and owned by a small team of cross-domain members.

**Decentralized Governance**  In a monolithic application, the product is governed in a centralized manner, meaning, it restricts the product to a specific platform or language stack. But in microservices, as each service is responsible implementing an independent business capability, it allows for building different services with different technologies. As a result, the team gets to choose the tools that are best suited for each of the services.

### 3.4.2 Benefits of Using Microservices

Newman, in his book Building Microservices [16], has discussed several concrete benefits of using microservices over a monolith. Several important benefits are discussed below in brief:
**Technology Heterogeneity**  When independent services are built separately, it allows adoption of different technology stacks for different services. This gives a team multiple advantages: liberty to choose a technology that best suits the need and adopt technology quickly for the needs.

**Scaling**  Monolithic applications are hard to deal with when it comes to scaling. One big problem with monolithic applications is that everything needs to be scaled together as a piece. But microservice allows having control over the scaling application by allowing to scale the services independently.

**Ease of Deployment**  Applying changes for a monolithic application requires the whole application to be re-deployed even for a minor change in the codebase. It poses a high risk as unsuccessful re-deployments even with minor changes can take down the software. Microservices, on the other hand, allow low-risk deployments without interrupting the rest of the services. They also allow having faster development process with small and incremental re-deployments.

### 3.5 Related Works

### 3.6 Related Software Products

There has been numerous research on creating software frameworks and environments to facilitate scientific research by different interdisciplinary research groups. Several successful collaborative research work on software frameworks and environments in the fields related to earth science are discussed in brief in this section.
3.6.1 CSDMS: The Community Surface Dynamics Modeling System

The Community Surface Dynamics Modeling System (CSDMS) project started in 1999 to facilitate earth surface modelers by creating a community driven software platform. CSDMS applies a component-based software engineering approach to the integration of plug-and-play components as the development of complex scientific modeling system requires the coupling of multiple, independently developed models [17]. There are several benefits CSDMS brings to the community of modelers. Firstly, it provides means for the modeling from different backgrounds to write their components in any of the popular languages. CSDMS achieves this via language interoperability using Babel language interoperability tool. CSDMS treats components as pre-compiled units which can be replaced, added to, or deleted from an application at runtime via dynamic linking. This gives modelers the option to use components created by others in the community to use in their simulation easily. The key design criteria that drove the design of CSDMS includes support for multiple operating systems, language interoperability across both procedural and object-oriented programming languages, platform independent graphical user interfaces, use of established software standards, interoperability with other coupling frameworks and use of HPC tools integrate parallel tools and models into the ecosystem.
3.6.2 HydroShare

The Consortium of Universities for the Advancement of Hydrologic Science, Inc (CUAHSI) is one of the leading research organizations representing universities and international water science-related organizations to develop software infrastructure and services for advancing water science. CUAHSI has several software projects such as HydroShare and CUAHSI HIS to provide infrastructure for water science research.

HydroShare is an online, collaborative software system for sharing hydrologic data and models. The goal of HydroShare is to help scientists to discover and ac-
cess data, and models, retrieve them to their desktop or perform analyzes in a distributed computing environment that may include grid, cloud or high-performance computing model instances [21]. Scientists can also publish outcomes of their research whether its data or model into HydroShare, using the system as a collaboration platform for sharing data, models and analyzes with other modelers. HydroShare is built using a python based technologies. The main components are built on top of Django Web Application Framework. Django is an application framework to facilitate web oriented architectures. It has several components designed for the end users to communicate with the system. HydroDesktop, a web service based desktop client is designed for the end users for hydrologic data discovery, download, visualization, and analysis.

![Figure 3.4: The high level architecture of HydroShare [21]](image)

The architecture of HydroShare separates the web application interface layer from the service layer, exposing the functionality through an application program-
ming interface (API) to enable direct client access and interoperability with other systems [21].

### 3.6.3 Model as a Service (MaaS)

Li et al. [11] proposed a cloud based solution called Model As A Service (MaaS) to support Geoscience Modeling. The authors have provided the solution as a proof of concept to allow remote execution of complex cpu and memory consuming models by exposing them as a service on top of a cloud provider like Amazon AWS.

![The general architecture of MaaS](image.png)

Figure 3.5: The general architecture of MaaS [11]

The central idea of MaaS is to allow users to upload input data, run a model and access and manipulate the output data through a web interface. The MaaS backend runs on top of a cloud provider like Amazon EC2 and takes care of provisioning
computing resources on the provider and running the model. The framework allows model registration through a virtual machine image repository, ensemble of model runs through on demand virtual machine provisioning and input/output data persistence through a common data backend.

### 3.6.4 Other Related Products

McGuire and Roberge designed a social network for assisting collaborations between watershed scientists. Being highly available, hydrologic data has not been integrated into a single system, and no system exists to facilitate collaboration between scientists, citizen scientists, and the general public. This work presents the design of a collaborative social network aimed at multiple user groups who are focused on hydrology and watershed science [14].

The Demeter Framework by Fritzinger et al. [8] is another attempt to bring software framework for assisting scientists in the area of climate change research. This work presents an overview of a software framework named the Demeter Framework that proposes a new solution to the model coupling problem by taking a component-based approach that allows almost any standard or type of component to be integrated into the system.

Walker and Chapra proposed a web-based client-server approach for solving the problem of environmental modeling compared to the traditional desktop-based approach. With the improvement in modern day web browsers, client-side approaches allow for improved user interfaces that are better than traditional desktop software, as well as the ability to perform simulations and visualizations within the browser [24].
The Geographic Storage, Transformation and Retrieval Engine (GSToRE) is a project initiated by the Earth Data Analysis Center at the University of New Mexico which provides a data framework for data discovery, delivery, and documentation for scientific research specializing in earth science. It has been developed as a flexible, scalable data management, discovery and delivery platform that supports a combination of open and community standards. It is built upon the principle of a services oriented architecture that provides a layer of abstraction between data and metadata management technologies [4].
CHAPTER 4
OVERVIEW OF THE SOLUTION

This chapter explains the high-level idea of the proposed software system for watershed modeling. The overall project goals and how the work fits into the project are described in this chapter.

4.1 Overview of the WC-WAVE Project

The Western Consortium for Watershed Analysis, Visualization, and Exploration (WC-WAVE) project started as a collaboration between three research areas, watershed science, cyberinfrastructure and visualization to accelerate the hydrology studies in the western US. The project aims to create a software ecosystem namely Virtual Watershed by integrating cyberinfrastructure and visualization tools to advance watershed science research. Fig. 4.1 shows a general high-level diagram of components of the ecosystem. The envisioned system is centered around services comprised of data, modeling, and visualization components. A brief high-level description of each of the components are provided below:

4.1.1 Data Service

To create a scalable and maintainable ecosystem of services the most important component we need is a robust data backend. The envisioned data service is a data management backend that exposes a RESTful web service to allow easy storage and management of watershed modeling data. It allows retrieval of data in various
OGC standards like WCS, WMS to allow OGC compliant clients to retrieve data automatically.

### 4.1.2 Modeling Service

Watershed researchers often use different modeling tools to simulate and investigate the change of different hydrologic variables around watersheds. The modeling tools are often complex to setup in local environments and takes up a good amount of time setting up [11]. Besides these modeling tools may require high
computational and storage resources that make them hard to run in local environments. The proposed modeling service aims to solve these issues regarding running models and managing the produced outputs by allowing users to submit model execution tasks through simple RESTful web service API. This approach of allowing model-runs through a generic API solves multiple problems:

- It allows the users to run models on demand without having to worry about setting up environments.
- It allows modelers to accelerate the process of running models with different input parameters by submitting multiple model-runs to be run in parallel which might not be feasible in local environment due to lack of computational and storage resources.
- It opens the door for other services and clients to take advantage of the API to automate the process of running models in their workflow.

This work concentrates on the details of modeling API backend. The detailed implementation of the system is provided in Chapter 5 and 6.

### 4.1.3 Virtual Watershed Clients

The system comes with different client tools to accommodate researchers with automated modeling workflow. For example, the visualization client allows visualization of different model-run variables in a 3D virtual environment. It lets users load data from the data service and visualize different hydrologic variables from that. The system also ships a web client enabling users to search, create, manage
and share their datasets. It also provides a simple interface for model-run submission, management and progress tracking.

4.2 Overview of Model API Design

The primary goal of this work is to allow watershed modelers to share their data resources and execute relevant models through web services without having to install the models locally. The system is designed as a collection of RESTful web services following microservice architecture that communicates internally. The web service sits on top of an extensible backend that allows easy integration of models and scalability over model runs and number of users connected to the service.

The system provides a mechanism for integrating new models by conforming with a simple event driven architecture that allows registering a model in the system by wrapping it with a schema driven adaptor.

As model execution is CPU intensive process scaling the server with growing number of parallel model execution is also an important issue. The architecture aims to solve this problem by introducing a simple database oriented job queue that allows options for adding more machines as the system grows.
CHAPTER 5
SPECIFICATION AND REQUIREMENTS

This chapter provides details on the formal design of the system. It contains requirement specification, use case diagrams and system level activity diagrams.

5.1 Requirement Specification

This section describes the functional and non-functional requirements of the system. Functional and non-functional requirements are helpful to understand the overall design of the system.

5.1.1 Functional Requirements

Table 5.1 shows the functional requirements with a priority set for each of the requirements where the priority indicates how crucial the implementation of the requirement is for the design of the system. These functional requirements with a brief description of what each requirement entails are provided to help the reader understand the system.

5.1.2 Non Functional Requirements

Table 5.2 shows the non functional requirements of the system. Each entry provides a brief description of each requirement.
Table 5.1: Functional Requirements of the System

<table>
<thead>
<tr>
<th>Id</th>
<th>Priority</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R 01</td>
<td>1</td>
<td>The system should allow clients to submit model execution job in the system.</td>
</tr>
<tr>
<td>R 02</td>
<td>1</td>
<td>The system should allow clients to upload model resources to the system.</td>
</tr>
<tr>
<td>R 03</td>
<td>3</td>
<td>The system should allow clients to track the progress of the submitted model executions.</td>
</tr>
<tr>
<td>R 04</td>
<td>1</td>
<td>The system should allow clients to access the outputs of the submitted model executions.</td>
</tr>
<tr>
<td>R 05</td>
<td>1</td>
<td>The system should allow workflow for new model registration.</td>
</tr>
<tr>
<td>R 06</td>
<td>1</td>
<td>The system should allow workflow for scaling model execution resources.</td>
</tr>
<tr>
<td>R 07</td>
<td>1</td>
<td>The system should allow client authentication and authorization against API endpoints.</td>
</tr>
<tr>
<td>R 08</td>
<td>2</td>
<td>The system should allow clients to initiate model execution with data from external source.</td>
</tr>
<tr>
<td>R 09</td>
<td>2</td>
<td>The system should allow users to monitor their model runs through an interface.</td>
</tr>
<tr>
<td>R 10</td>
<td>3</td>
<td>The system should allow users to share input/output data resources with other external sources.</td>
</tr>
</tbody>
</table>

Table 5.2: Non Functional Requirements of the System

<table>
<thead>
<tr>
<th>Id</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NR 01</td>
<td>The system should be implemented in microservice architecture.</td>
</tr>
<tr>
<td>NR 02</td>
<td>The system should be implemented as a collection of RESTful web services independent of platform.</td>
</tr>
<tr>
<td>NR 03</td>
<td>The system should support multiple database backends.</td>
</tr>
<tr>
<td>NR 04</td>
<td>The system should support multiple storage backends.</td>
</tr>
<tr>
<td>NR 05</td>
<td>The system REST services should be implemented in Python Flask microframework.</td>
</tr>
</tbody>
</table>

5.2 Use Case Modeling

Use case modeling is a way of engineering requirements of a system. It is a different way of looking at the requirements from the user’s perspective. It works as
a complementary way of eliciting requirements besides gathering functional and non-functional requirements through traditional model [2]. We described a use case diagram and a detailed description of the use cases in this section.

### 5.2.1 Use Case Diagram

![Use Case Diagram](image)

**Figure 5.1: The use case diagram of the system**

Figure 5.1 shows the use case diagram for the system. It gives a useful perspec-
tive to see the system from the user or client’s perspective. The ellipses represent the functionalities from the users point of view. A short description of the use cases is provided below.

**UC01: Create Account**  A user can go to the authentication portal and create an account in the system. The user needs to provide his/her email address and other information to sign up. The account needs to be activated by following a confirmation link sent to the email. This will prevent the system from spammers and automated bots.

**UC02: Login**  A user can log in through the authentication portal to access the system through the web browser. The authentication server is a micro service that is used by all the components of the umbrella virtual watershed platform to allow common authentication/authorization endpoint.

**UC03: Get API Token**  A user can go to the authentication portal and get an API access token to be used by a REST client to communicate with the different components of the virtual watershed platform. An API access token allows a user to access the virtual watershed API endpoints through REST clients without having to expose the password. The API access token can also be used by the other components of the system to communicate with each other for user data.

**UC04: Initiate Model Run**  A user can initiate a model run in the modeling server by logging in to the web frontend or through a REST client providing personal the access token. A model run creation will require the user to submit the model name,
a title, and description.

**UC05: Upload Resources**  Once a model run is created on the system, the user can upload the resources necessary for the model to run. A resource is usually a file that describes the parameters and data of the model to be run.

**UC06: Start Model Run**  After successful creation of the model run with all necessary resources of the model run, a user can instruct the server to start the run. The modeling server will take care of the model run through a messaging queue and event-driven API for the progress update, which is described in details in Chapter 6.

**UC07: Track Model Run Progress**  The system allows a user to track progress by polling an API endpoint for a model run. A user can track the progress of the model run from the web front end or by accessing the API endpoint through a REST client providing the API access token.

**UC08: Download Model Run Resources**  Once a model run is finished the system provides endpoint URLs to download the resources generated by the model run. The system uses a generic storage backend to persist the data. A user can download the resources by following the storage URL generated by the system.

**UC09: View ModelRun**  A user can access his/her previous model runs, and the details of the model run through the web client or by accessing the API. The user can view the resources and progress events and their descriptions.
**UC10: Delete Model Run** A user can delete a model run from the persistent storage through either the web front end or the API. This action will erase the model run data, the resources and the progress events associated with the model run.

### 5.3 Glossary

Table 5.3 gives a description of the terminologies used in this thesis.

Table 5.3: Glossary of the terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Watershed</td>
<td>A watershed is the area of land where all of the water that falls in it and drains off of it goes to a common outlet. A watershed can vary in size. It can be as small as a footprint and as big enough to encompass all the land that drains water into rivers that drain into the Chesapeake Bay, where it enters the Atlantic Ocean. [23]</td>
</tr>
<tr>
<td>WC-WAVE</td>
<td>WC-WAVE stands for Western Consortium for Watershed Analysis, Visualization, and Exploration. The goal of the WC-WAVE project is to create a software platform called The Virtual Watershed (VW) to allow hydrologic modeling, data sharing, and visualization through a common integrated platform. [25]</td>
</tr>
<tr>
<td>NetCDF</td>
<td>NetCDF stands for Network Common Data Format. It is a data format designed to support the creation, access, and sharing of array-oriented scientific data. [15]</td>
</tr>
<tr>
<td>Flask</td>
<td>Flask is a cross-platform micro web application framework written in Python. [18]</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SqlAlchemy</td>
<td>SqlAlchemy is a Python SQL toolkit and Object Relational Mapper (ORM). It provides means for efficient and high-performing database access[20].</td>
</tr>
<tr>
<td>Docker</td>
<td>Docker is a platform for developing and deploying distributed applications. Docker uses operation system level virtualization technique called container virtualization. Docker creates a deployment engine for applications on a virtualized container execution environment [22]. It allows developers to compose applications as microservices, creates smooth development workflow centered around containerization and, deploy and scale applications.</td>
</tr>
<tr>
<td>GSToRE</td>
<td>Geographic Storage, Transformation and Retrieval Engine (GSToRE) is a data management framework for managing, discovering and sharing scientific and geographic data, developed at the Earth Data Analysis Center, University of New Mexico. [4]</td>
</tr>
<tr>
<td>Model</td>
<td>In the context of the Virtual Watershed platform, the term model is used to define a computer program that simulates any physical processes. A model essentially is a CPU consuming process that takes some inputs and produces some output as the result of the simulation.</td>
</tr>
<tr>
<td>Model Run</td>
<td>In the context of the Virtual Watershed platform, a model run denotes the set of data and metadata when a model is run for a specific set of input data.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Model Resource</td>
<td>A model resource is an abstraction to represent the inputs and outputs of a model. A resource is a generic wrapper over the inputs and outputs of a model. This abstraction allows a model adapter developer to have control over defining the resources of a model wrapper in a generic way.</td>
</tr>
<tr>
<td>REST</td>
<td>REST stands for Representational State Transfer Protocol. It is an architectural style for implementing server-client hypermedia systems by imposing a set of principals described in the REST architecture.</td>
</tr>
<tr>
<td>iSNOBAL</td>
<td>iSNOBAL is a coupled energy-balanced snowmelt model. It provides spatially distributed estimation of snow melt over large mountain basin or watersheds [12].</td>
</tr>
<tr>
<td>PRMS</td>
<td>PRMS Stands for The Precipitation-Runoff Modeling System. It is a widely used model for general watershed hydrology. It is a deterministic, distributed-parameter, physical process based modeling system that evaluates the response of various combinations of climate and land use on a watershed. [13]</td>
</tr>
<tr>
<td>Microservice</td>
<td>Microservices is a software architecture in which a complex application is decomposed into small components that communicate with each other through well-defined language-agnostic APIs [7]</td>
</tr>
<tr>
<td>JWT</td>
<td>JSON Web Token (JWT) is a standard method for secure data transfer between two parties [10]</td>
</tr>
</tbody>
</table>
CHAPTER 6
DESIGN AND IMPLEMENTATION

In this chapter, we describe the detailed design of the system. The design is presented through different standard software design tools such as system level diagram, activity diagram, class diagram and entity-relationship diagram. We describe the implementation details of the prototype application in the later portion of the chapter.

6.1 System Level Diagram

The Virtual Watershed system comprises of several different submodules where each of the modules provides different functionalities of the entire framework. Figure 6.1 provides the high-level system diagram of the system.

**VW-PY**  The VW-PY sub-system provides a module to define python adaptors over the different models that are available through the Virtual Watershed platform. An adaptor is essentially a Python wrapper over a model to allow running the model programmatically. It provides an interface to wrap a model with a light python wrapper that takes care of data format conversion, model execution and event triggering on progress of the model. The event driven system will allows a model-wrapper developer to easily signal on progress as the model execution happens.
VW-MODEL  The VW-MODEL submodule is the web service front end to the modeling system. It exposes a REST API endpoint to the user/client through which a user can submit, query and download a model run and its resources.

VW-WORKER  The VW-WORKER module is a messaging queue driven worker service that encapsulates a model adaptor in a messaging queue worker. It is loosely coupled with the VW-MODEL component through a common redis data-backend.

VW-STORAGE  VW-STORAGE component works as the storage backend for the VW-MODEL module. The VW-STORAGE is a generic wrapper for the object storage which can be configured for different storage provider, either in the cloud or locally.
**VW-AUTH**  The VW-AUTH module works as the common security gateway for the system. It provides authentication and authorization level access that can be used by other services to authenticate/authorize a user/client against a resource.

**VW-SESSION**  The VW-SESSION is a common session backend that can be used by different components inside Virtual Watershed that requires user session management. The session backend is managed with a key-value Redis data store that is shared across the services under Virtual Watershed.

**VW-WEB**  VW-WEB is the common web frontend module that is exposed to the end users. Users can log in to the system through the vw-auth module that sets a shared session across the system and access resources, run models, track progress and upload/download resources of model runs.

### 6.2 Detailed Design

We described the components in details in the previous section. Each of the services are described in details in the following sections.

#### 6.2.1 Authentication Service

The entire Virtual Watershed system is built as an aggregation of different web services and web applications that interact with each other. A common authentication gateway is necessary to make the communications secure and centralized.
The VW-AUTH authentication module is a microservice developed for the purpose. It provides a one-stop registration, authentication and authorization point for the users of the system.

Figure 6.2: The workflow for accessing secure REST endpoint with JWT token

The component itself is developed as a web service that exposes RESTful endpoints for the users to be able to gain access to different service endpoints. It exposes API endpoints for registration and authentication. The service also implements a JSON Web Token (JWT) based authorization scheme for allowing secure access to the REST endpoints of different components. JWT is an RFC standard for exchanging information securely between a client and a server. Figure 6.2 depicts the workflow for a user to be able to gain access to a secure REST endpoint following token based authentication.
6.2.2 Model Web Service

The model web service component is a RESTful API that is exposed to a user. Upon authentication, a user can create a request for a model run, upload necessary input files needed for the model to run and instruct the server to execute the model. The API backend stores the model run data in a small database and uses a storage backend to store the files uploaded by the users.

Each available model in the system provides a schema to describe the necessary input files and their formats, execution policy of the model and a mapping between the parameters presented to the user and the parameters available in the model adaptor. The user or client can essentially extract the mapping and know which are the resources that need to be uploaded to run the model. Figure 6.3 shows a simple workflow chart from a user or client’s perspective.

The web service approach enables different advantages to the users.

- Users do not need to have the internals of the model. How the model is setup, what are the dependencies essentially get hidden from the user.
- Users don’t need to worry about installing the dependencies of a model.
- Users can initiate an ensemble of model runs that can run in parallel on the server and get the results back altogether.
- Users can persist the data in the server and access it from anywhere through the REST API.

Figure 6.4 shows the class diagram for the models in web service backend that handle data persistence.
6.2.3 Model Adaptors

The first step in creating the architecture for exposing models as services is to be able to run a model programmatically. Besides, a model can have dependencies it needs to meet before it can be run. We achieve this by allowing model adaptor developer to create a thin python wrapper around a model and expose all the depen-
Figure 6.4: The class diagram of the components in model web service backend dependencies through a Linux container image. Another important issue to tackle here is the data format heterogeneity of the model inputs and outputs. Different models have inputs and outputs in different formats. To achieve automated modification of the model inputs, we introduced an option to write NetCDF data adapters for each of the models. NetCDF is a data exchange format that allows easy storage, extraction, and modification of gridded scientific data. So in a nutshell, a model adaptor is essentially a Python program that takes care of data format conversion, running the actual model and emitting the progress.

Figure 6.5 shows the design of the interface to allow adaptor around a model. A wrapper is a simple python interface. Model adaptor developer can implement the interface to expose the model programmatically. Adaptor developer needs to provide a set of converters to allow conversion and deconversion of the native resources of the model to netCDF and its original format. The developer also needs to implement an execution method where the resource conversion and execution
of the model happens. The wrapper has access to an event emitter that can be used by the developer to emit events as the model progress on execution; the events can be caught by an event listener, which is responsible for persisting the progress to be sent back to the user through a REST endpoint.

6.2.4 Model Worker

Through a model adaptor, we can encapsulate a model to be executed programmatically. To set the bridge between the web service frontend and the actual model execution we need a process. The model worker module comes into play to facilitate that. We used a producer-consumer style messaging queue to accomplish the
process. A messaging queue or task queue is a lightweight middleware that creates a bridge between user frontend and worker backend. When a user submits a model run task through the web service frontend, it is placed into the queue by the web service through a unique id. The consumer/worker process listens to the queue through a common protocol for new jobs. The worker process is a python process that runs on a server where it has access to the model execution code, and the dependencies of the model are installed.

![Class Diagram](image)

**Figure 6.6: The design of the worker through class diagram**

The worker resides in an isolated server instance that has the dependencies and libraries of the model installed. It is ensured through the deployment workflow using Linux containerization, which is described in details in the Section 6.3.

### 6.3 Deployment Workflow

We have devised a Linux container based deployment workflow for the different components of the system. We used a technology called Docker to develop the workflow. This workflow allows doing iterative deployment and scaling of the components, and a strategy to register new models in the system.
Figure 6.7: The workflow of a model worker depicted in a flow chart.

1A. Download resources from the data and storage backend
1B. Initiate Model Run by updating state to RUNNING in Databackend
1C. Get execution policy from the ModelSchema

2A. Register event listener
2B. Invoke the process following execution policy

3A. Upload the results back to the storage backend
3B. Clean up temporary residue files
3C. Update model run state in data backend as FINISHED

Finish
Each of the components of the virtual watershed platform is dockerized. We have a central docker image repository set up that contains images for the different components. A docker image is essentially a template that encapsulates the os, dependencies of an application and the application itself. An image can be used to provision containers that run the application on top of Docker engine. Each of the repositories in virtual watershed contains a Dockerfile that describes how the image for the component should be built and how the application is served when deployed as a container. A typical Dockerfile usually contains instructions to install libraries and dependencies for the component to run when provisioned. The repositories are set up with automated build in the image hub through web hooks.

6.3.1 Model Registration

The dockerized workflow opens up an easy way to register model in the system. Different models have different requirements for setting up the environment. Our system allows a developer to register a model through the creation of docker image for the model. It allows a developer to specify the os, libraries and other dependencies for the model. Figure 6.8 shows the workflow for registering a model in the system through the creation of Docker image.

6.4 Prototype Details

We present the resulting prototype after implementation and usage details in this section. We present the usage from the user’s point of view as a demonstration of
Figure 6.8: The workflow for registering a model in the system
the system.

The system provides a registration portal, through which a user can register, login, reset the password and generate JWT authentication tokens to access the REST APIs. Figure 6.9 shows the registration page of the authentication portal.

![Figure 6.9: The registration page of the portal](image)

The portal also exposes a REST API to register and authenticate user credentials programmatically. This feature can be useful for clients (e.g., VW-WEB client) in virtual watershed that wants to use the same authentication endpoint.

The VW-WEB client provides a graphical interface like the MAAS software to allow users to create, upload, run, track progress and delete model runs by invoking the REST API exposed by the VW-MODEL component. So, the VW-WEB application works as a client for the Modeling REST API.

Figure 6.10 shows a screenshot of the dashboard where a user is trying to run ten PRMS models in parallel by uploading the three input resources needed for PRMS model. The upload interface as shown in figure 6.11, is generated dynamically from the model schema defined for each of the models.

The web interface works as a proof of concept client for the modeling REST API.
Figure 6.10: The dashboard for user’s model runs

Figure 6.11: The dynamically generated upload form
The real power of the modeling interface comes when it is used programmatically to run an ensemble of models in parallel without having to worry about the need of computing power. Applications can be built on top of the modeling web service to provide an extended capability to manipulate model input parameters and run the model.

Rui Wu, another contributor of the Virtual Watershed project, has built a web-based scenario design tool to manipulate different input variables of the PRMS model, run it through the modeling web service, and visualize and compare the output data of different model runs.

Figure 6.12 shows the interface that allows manipulation of input data visually compared to a manual edition of the input files by opening them in an editor. It allows the users to do modeling without having the deeper knowledge of how internally model data is represented, and it can be useful for public teaching for modeling. Figure 6.13 shows the output visualization after a model has completed run through the modeling web service.

A set of ipython notebooks created by another contributor of the Virtual Watershed project, Matthew Turner, demonstrates the programmatic way of running an ensemble of models in parallel by using the modeling API. The particular example is shown in the screenshot in Figure 6.14, shows an important step of modeling with PRMS model. Model calibration is a tedious process for the PRMS modelers since it requires running the model again and again with varied parameter values. Modelers often end up using a manual process to edit and manipulate the input files and running the model manually on a local machine. It requires an enormous amount of time to do the calibration.
Figure 6.12: Scenario creation interface for PRMS model that uses Modeling REST API to execute model
Figure 6.13: Comparing the results of the scenarios created using the modeling service

By using the modeling API user can achieve this capability programmatically writing a few lines of code. It also gives the user to use the computing power of the server to run the model as many time a user wants without having to worry about time and CPU resources.

6.5 Technologies Used and Software Characteristics

The system is built using different tools based on the need for the components. The web service frontends are built using a Python micro-framework called Flask with the usage of various extensions. The MVC architecture is followed to structure the code. Some of the used libraries are Flask-Restless for implementing the REST API endpoints, SqlAlchemy for mapping the data models with a database back-end, Postgresql as the database, Flask-Security, and Flask-JWT for authentication
and authorization. For the web front ends HTML5, CSS, Bootstrap, Javascript and ReactsJs library are used. Celery, a python based task queue that supports multiple broker backend is used to implement the task queue for model workers. Redis is used as the broker and result backend for Celery. A REST spec library called Swagger is used to create the spec for the REST APIs. This spec allows the creation of REST clients in numerous languages. Since its a collaborative project, Github is used as the central repository for code and issue management. The modules combined contain approximately 6000 lines of codes. The codes are published through virtualWatershed’s Github repository [26].
CHAPTER 7

COMPARISON WITH RELATED TOOLS

In this chapter, we aimed to make a feature based comparison between similar software tools focused on hydrologic and environmental modeling. The tools we discussed in Chapter 3 are CSDMS [17], Hydroshare [21] and MAAS [11]. Though each of these tools is designed and developed keeping different goals in mind, all of them are a demonstration of work to assist environmental modeling in general. Each of these tools has their pros and cons from different user perspective. The tools as a whole are not comparable to each other since each of them try to solve different problems when it comes to implementation details. Hence, we followed a feature based comparison approach to get a birds eye view on them and to get more insights on the features and how Virtual Watershed modeling tool can supplement the building of cyberinfrastructure for hydrologic and environmental modeling.

CSDMS is a community driver system where modelers can submit their models to CSDMS by implementing model interface provided by them. The model gets evaluated and added into the system by the CSDMS committee. CSDMS uses a language interoperability tool called Babel to handle language heterogeneity. From the user’s perspective, CSDMS provides a web and desktop based client to the users to get access to the models and run them in a different configuration from within the client. On the server side, CSDMS maintains an HPC cluster where they have all the models and relevant dependencies installed.

Hydroshare [21], on the other hand, is a project started to accommodate data sharing and modeling for the hydrologists. The platform is in active development and adding new feature till date. The current version of Hydroshare is more con-
centrated towards data sharing and discovery for the hydrology researchers [1]. Hydroshare also provides programmatic access to its REST API which makes it a good candidate for a data discovery backend for any other similar system.

Table 7.1: Feature Comparison with Related Hydrologic and Environmental Modeling tools

<table>
<thead>
<tr>
<th>Feature</th>
<th>CSDMS</th>
<th>HYDROSHARE</th>
<th>MAAS</th>
<th>VW-MODEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open source</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Provides REST API</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Provides interface for model implementaion</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Allows model coupling</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Allows model registration</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Data storage</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Data sharing and discovery</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

The MaaS [11] is the closest product which is also trying to achieve a similar outcome as Virtual Watershed modeling tool. The project aims to publish models as service. Though the approach and implementation of this work differ in various ways, the end goal is similar to what we are trying to achieve. MaaS introduced models as services by creating an infrastructure in cloud platforms. The framework provides users with a web application to submit their jobs and has a platform backend that takes care of on-demand provisioning of virtual machines that contains model execution environment setup. Maas achieve model registration by encapsulating a model as a virtual machine image in an image hub. It provides an FTP based database backend to store the model results.
CHAPTER 8
CONCLUSION AND FUTURE WORK

In this work, we introduced a web-based platform for assisting watershed modeling. We proposed a web-service platform for executing hydrologic models. We devised a strategy for exposing a model through thin python wrapper that allows representing the model resources through a common data format (e.g., NetCDF) and execution of the model in the server by submitting the resources in NetCDF format.

Furthermore, we implemented a common authentication/authorization framework as part of the development process. This framework allows a common security endpoint for the users to gain access to different applications under a virtual watershed platform.

Also, we have developed an extendable approach for allowing registration of new models in the system. A developer can integrate a new model by implementing the model wrapper interface and proving a schema for the model. We have implemented model wrappers for two hydrologic model called iSNOBAL and PRMS as a proof of concept. We devised a replicable deployment strategy for the entire system using a dockerized workflow which allows a process for progressive development and deployment of the components.

Exposing models as service will allow users to run models without having to worry about computational resources. It will allow building of model specific applications on top of the service and furthermore running ensemble of model in parallel is made easy through the service.
The work can be further extended by enabling more features. The system currently has the option to integrate with GSToRE data backend. Enabling a generic data backend with an option to integrate with other existing data providers like DataONE, Hydroshare, etc. will enable better access to data for modeling automation. Allowing provisioning of on-demand model containers in cloud providers to make cost-effective deployment can be an important aspect to pursue. Allowing model tuning directly through web service will also be interesting to consider for further development. Developing a pricing model for service usage would be another interesting aspect to look at. User centric computational resource management can also be a potential candidate for future development. Allowing users to request for resources and managing them on demand can be a challenging task. Yet another challenging feature to explore would be allowing model coupling as a service. Models are hard to couple. Providing coupling through service can be a good albeit very challenging extension of the current work.
BIBLIOGRAPHY


[23] USGS. What is a watershed?, 2015.

