Web-Service Framework For Environmental Models

Md Moinul Hossain[^] Rui Wu[*] Jose T. Painumkal[^] Mohamed Kettouch[#]
Cristina Luca[#] Sergiu M. Dascalu[*] Frederick C. Harris, Jr[*]

[^*]Department of Computer Science and Engineering
University of Nevada, Reno
Reno, NV, USA

[#]Computing and Technology
Anglia Ruskin University
Cambridge, UK

[^]{hossain, josepainumkal}@nevada.unr.edu [#]{mohamed.kettouch, cristina.luca}@anglia.ac.uk
[*]{rui, fred.harris, dascalus}@cse.unr.edu

***Abstract*—Environmental scientists always use different models to simulate the real world with their own devices. Because of the limited computing power, the simulation can consume very long time. Also, it is hard to share their models with others and they may need to rebuild the same model for similar problems. To solve these problems, we propose a web-service centric framework to expose models as services in this paper. The framework allows 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. A prototype is introduced and implemented to prove the idea works.**

Keywords—model as service; web-based application; environmental model; hydrological model; cloud-based application

I. 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. Software is 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 its interdisciplinary nature. It poses a lot of challenges regarding barriers to communication and team building among different communities involved in the process.

The work presented in this paper 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 [1]. 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 the 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. Because of the flexible design (blueprint and templates), the framework can be applied for the general use. To clarify these ideas, ISNOBAL and PRMS are used in this paper to demonstrate the proposed framework. In the rest of the paper, the prototype system is named Virtual Watershed System (VWS).

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 [2].

The Precipitation-Runoff Modeling System (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 [3]. 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 the interaction of groundwater and surface water, the interaction of climate and atmosphere with surface water and much more [3].

This paper, in its remaining parts, is arranged as follows: Section II introduces related work that has been done; Section III proposed a design to build the system; Section IV describes the prototype system and how to build software using RESTful APIs; and Section V contains the paper's conclusions and outlines planned future work.

II. Background

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.

Community Surface Dynamics Modeling System (CSDMS) was a project started in 1999 to facilitate earth surface modelers by creating a community driven software platform. CSDMS applies a component-based software engineering approach in the integration of plug-and-play components, as the development of complex scientific modeling system requires the coupling of multiple independently developed models [5]. CSDM allows users to write their components in any of the popular languages. Also they can use components created by others in the community for their simulations. CSDMS treats components as pre-compiled units which can be replaced, added to, or deleted from an application at runtime via dynamic linking. The key design criteria that drove the design of CSDMS includes the 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 to integrate parallel tools and models into the ecosystem.

The Consortium of Universities for the Advancement of Hydrologic Science Inc. (CUAHSI), one of the leading research organizations representing universities and international water science-related organizations, has several software projects such as HydroShare 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 access data, and models, retrieve them to their desktop or perform analyses in a distributed computing environment that may include grid, cloud or high-performance computing model instances [6]. 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 analyses with other modelers. The architecture of HydroShare separates the web application interface layer from the service layer, exposing the functionality through an application programming interface (API) to enable direct client access and interoperability with other systems [6].

Li et al. [7] 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 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.

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.

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 an extensible, 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 [9].

The following paragraphs will briefly introduce the high-level idea of different software engineering techniques used in the development of the proposed system.

Service Oriented Architecture (SOA) is an architecture for software systems which has gained significant focus in the IT industry in recent years [10]. Service Oriented Architecture constitutes a model where business logic for software is decomposed into distinct units or services. Each unit is self-contained, and they collectively represent the aggregated business logic [10].

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 essentially a software delivery method where the service is delivered to a customer through the internet instead of the need for local installation. SaaS is currently regarded as an important IT trend as according to industry analysts, considering the increasing sales and continuing growth in the industry [11].

Micro-services is a software architecture in which a complex application is decomposed into small components or services that communicate with each other through well-defined language-agnostic APIs [12]. The Micro-services 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 architecture.

In micro service architecture, the capabilities of an application are decomposed into self-contained services. The usage of micro-service architecture provides the liberty to use different technologies that best suit the needs, enhances the scalability of the application and facilitates low-risk deployments without interrupting the rest of the services.

III. The Proposed Method

The detailed design of the system is introduced in this section. The design is presented through different standard software design tools such as system level diagram and workflow diagram.

A. System Level Design

The Virtual Watershed system comprises of several different submodules where each of the modules provides different functionalities of the entire framework. Figure 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 the progress of the model. The event driven system will allow a model-wrapper developer to easily signal on progress as the model execution happens.

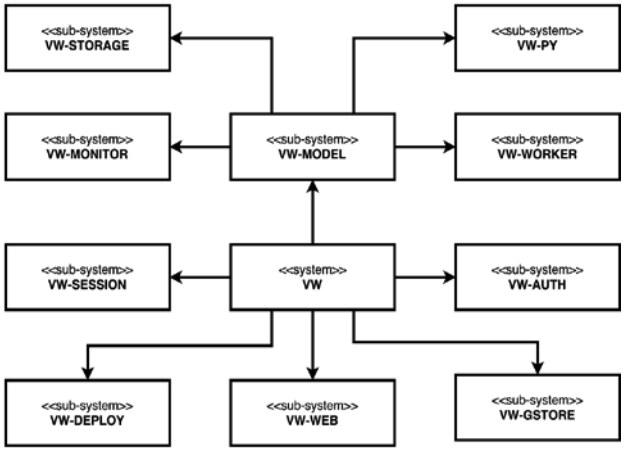

Figure 1. The system level diagram of the system

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: The 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: The VW-WEB is the common web frontend module that is exposed to the end users. Users can log into 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.

B. Detailed Design

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 micro service developed for the purpose. It provides a one-stop registration, authentication, and authorization point for the users of the system.

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 2 depicts the workflow for a user to be able to gain access to a secure REST endpoint following token based authentication.

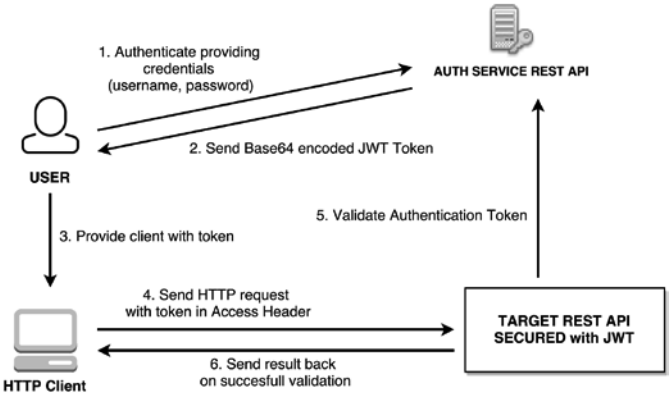

Figure 2. The workflow for accessing secure REST endpoint with JWT token

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 resources are needed to be uploaded to run the model. The basic steps from the client side perspective to run a model are: get the model schema stored in the server; create a model run in the server side; upload the model inputs; start the model run; track the model run progress until it is finished; download the model outputs. There are several advantages of using this workflow and the web-client architecture: 1) 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. 2) Users don't need to worry about installing the dependencies of a model. 3) Users can initiate an ensemble of model runs that can run in parallel on the server and get the results back altogether. 4) Users can persist the data in the server and access it from anywhere through the REST API.

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 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. 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.

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. 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.

C. *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 [4]. Docker containers are better than normal virtual machines because they use fewer resources. This workflow allows doing iterative deployment and scaling of the components, and a strategy to register new models in the system.

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 webhooks.

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. The basic steps for registering a model in the system through the creation of Docker image are: create model repository; implement the model wrapper; provide dependencies through dockerfile; test and publish code; create a Docker image in the image repository.

IV. SYSTEM PROTOTYPE AND TESTING

The system was built using different tools based on the need for the components. The project used some our previous codes and similar structures introduced in [14] and [15]. The web service frontends are built using a Python micro-framework called Flask with the usage of various extensions [13]. Some of the used libraries were Flask-Restless for implementing the REST API endpoints, SqlAlchemy for mapping the data models with a database back-end [16], PostgreSQL as the database [17], Flask-Security, and Flask-JWT for authentication and authorization. For the web front ends HTML5, CSS, Bootstrap, Javascript and ReactsJs library

were used. Celery, a python based task queue that supports multiple broker backend was used to implement the task queue for model workers [18]. Redis was used as the broker and result backend for Celery. A REST spec library called Swagger was used to create the spec for the REST APIs. This spec allows the creation of REST clients in numerous languages. To structure the code, MVC design pattern was used. GitHub was used as the central repository for code and issue management and the codes are publically made available through Virtual Watershed's GitHub repository [19].

The two main components of the prototype system are 1) Authentication module and 2) Modeling module.

The authentication module takes care of all the activities related to securing the Virtual Watershed system. It handles the registration of new users, login of existing users, verification of the user, resetting the user password, and generating JWT authentication tokens to access the REST APIs. The VW system also supports the registration and authentication of users through a REST API. This feature helps VW system to have multiple applications using a single authentication endpoint.

The modeling module contains an intuitive user interface to allow users to create, upload, run and delete models using the REST API service. The UI includes a progress bar to inform users about the percentage of the model run finished. The module also includes a dashboard where users can view the models being run, the finished model runs, and also download the model run files. Figure 3 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.

Figure 3. The dashboard for user's model runs

The web interface works as a proof of concept client for the modeling REST API. 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.

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. The virtual watershed system includes 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 4 shows the interface that allows manipulation of input data visually compared to a manual edition of the input 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. The system can also visualize the outputs after a model has completed run through the modeling web service.

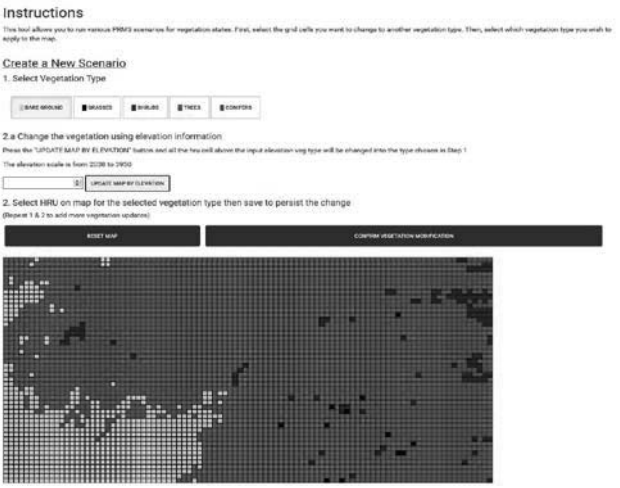

Figure 4. Scenario creation interface for PRMS model that uses Modeling REST API to execute model

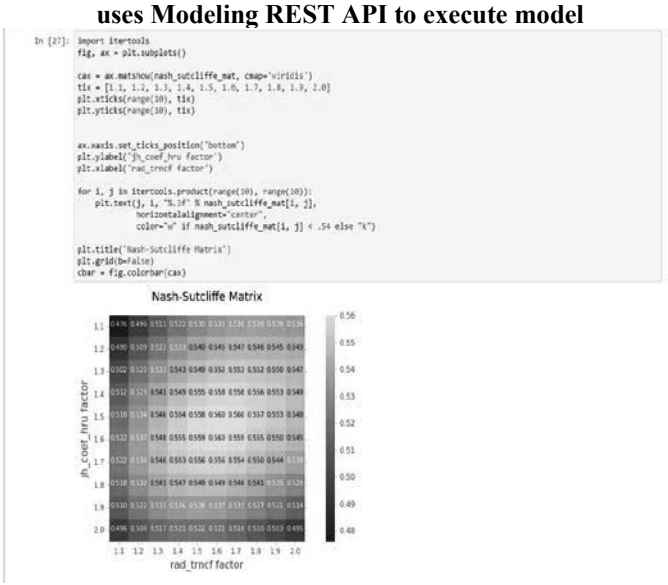

Figure 5. Example of tuning a hundred model through an IPython notebook with different parameters

A set of IPython notebooks were created to demonstrate the programmatic way of running an ensemble of models in parallel by using the modeling API [20]. The particular

example is shown in the screenshot in Figure 5, shows an important step of modeling with PRMS model. By using the modeling API, the user can achieve this capability programmatically writing a few lines of code. It also enables the user to leverage the computing power of the server to run the model as many time a user wants without having to worry about system configuration.

V. Conclusion and Future work

In this paper, a web-service platform is proposed 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. A common authentication/authorization framework as part of the development process is also implemented. This framework allows a common security endpoint for the users to gain access to different applications under a virtual watershed platform. Two hydrologic model called iSNOBAL and PRMS are implemented in the prototype system as model wrappers to prove the concept. We also 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.

The project 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 of pursuing. 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. Another challenging and useful feature would be allowing model coupling as a service. Models are hard to the couple. Providing coupling through service can be a good albeit very challenging extension of the current work.

Acknowledgment

This material is based upon work supported by the National Science Foundation under grant numbers IIA-1329469 and IIA-1301726.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Nmepscor.org. (2017). *The Western Consortium for Water Analysis, Visualization and Exploration | New Mexico EPSCoR*. [online] Available at: https://www.nmepscor.org/science/western-consortium-water-analysis-visualization-and-exploration [Accessed 4 Feb. 2017].

[2] Marks, D., Domingo, J., Susong, D., Link, T. and Garen, D., 1999. A spatially distributed energy balance snowmelt model for application in mountain basins. *Hydrological Processes*, *13*(12-13), pp.1935-1959.

[3] Steven L Markstrom, R Steven Regan, Lauren E Hay, Roland J Viger, Richard MT Webb, Robert A Payn, and Jacob H LaFontaine. PRMS-IV, the precipitation-runo modeling system, version 4. Tech. rep. US Geological Survey, 2015.

[4] Merkel, D., 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, *2014*(239), p.2.

[5] Peckham, S., Hutton, E. and Norris, B. (2013). A component-based approach to integrated modeling in the geosciences: The design of CSDMS. *Computers & Geosciences*, 53, pp.3-12.

[6] Tarboton, D.G., Idaszak, R., Horsburgh, J.S., Ames, D., Goodall, J.L., Band, L.E., Merwade, V., Couch, A., Arrigo, J., Hooper, R.P. and Valentine, D.W., 2013, December. HydroShare: an online, collaborative environment for the sharing of hydrologic data and models. In *AGU Fall Meeting Abstracts* (Vol. 1, p. 1510).

[7] Li, Z., Yang, C., Huang, Q., Liu, K., Sun, M. and Xia, J. (2017). Building Model as a Service to support geosciences. *Computers, Environment and Urban Systems*, 61, pp.141-152.

[8] Fritzinger, E., Dascalua, S.M., Ames, D.P., Benedict, K., Gibbs, I., McMahon, M.J. and Harris, F.C., 2012. *The Demeter framework for model and data interoperability* (Doctoral dissertation, International Environmental Modelling and Software Society (iEMSs)).

[9] Wheeler, J., 2017. Extending Data Curation Service Models for Academic Library and Institutional Repositories. *Curating Research Data*, p.171.

[10] Erl, T., 2005. Service-oriented architecture (SOA): concepts, technology, and design.

[11] Buxmann, P., Hess, T. and Lehmann, S., 2008. Software as a Service. *Wirtschaftsinformatik*, *50*(6), pp.500-503.

[12] Lewis, J. and Fowler, M., 2014. Microservices: a definition of this new architectural term. *Mars*.

[13] Grinberg, M., 2014. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.".

[14] Hossain, M., Dascalu, S. and Harris Jr, F.C., A Software Environment for Watershed Modelling. In Proceedings of the 24th International Conference on Software Engineering and Data Engineering (SEDE 2015), October 12-14, San Diego, CA.

[15] Wu, R., 2015. *Environment for Large Data Processing and Visualization Using MongoDB* (Doctoral dissertation, University of Nevada, Reno).

[16] Copeland, R., 2008. *Essential sqlalchemy*. " O'Reilly Media, Inc.".

[17] Momjian, B., 2001. *PostgreSQL: introduction and concepts* (Vol. 192). New York: Addison-Wesley.

[18] Docs.celeryproject.org. (2017). *Celery - Distributed Task Queue — Celery 4.0.2 documentation*. [online] Available at: http://docs.celeryproject.org/en/latest/. [Accessed 3 Feb. 2017].

[19] GitHub. (2017). *VirtualWatershed*. [online] Available at: https://github.com/VirtualWatershed/ [Accessed 3 Feb. 2017].

[20] Pérez, F. and Granger, B.E., 2007. IPython: a system for interactive scientific computing. *Computing in Science & Engineering*, *9*(3).