This is an Accepted Manuscript of an article published in Environmental Modelling and Software in March 2013, available online: DOI: 10.1016/j.envsoft.2012.11.010

Models as web services using the Open Geospatial Consortium (OGC) Web Processing Service (WPS) standard

Anthony M. Castronova[a], Jonathan L. Goodall[b,*], Mostafa M. Elag[c]

[a] *Postdoctoral Researcher, Department of Civil and Environmental Engineering, University of South Carolina, 300 Main Street, Columbia, SC 29208*
[b] *Assistant Professor, Department of Civil and Environmental Engineering, University of South Carolina, 300 Main Street, Columbia, SC 29208*
[c] *Graduate Research Assistant, Department of Civil and Environmental Engineering, University of South Carolina, 300 Main Street, Columbia, SC 29208*

Abstract

Environmental modeling often requires the use of multiple data sources, models, and analysis routines coupled into a workflow to answer a research question. Coupling these computational resources can be accomplished using various tools, each requiring the developer to follow a specific protocol to ensure that components are linkable. Despite these coupling tools, it is not always straight forward to create a modeling workflow due to platform dependencies, computer architecture requirements, and programming language incompatibilities. A service-oriented approach that enables individual models to operate and interact with others using web services is one method for overcoming these challenges. This work advances the idea of service-oriented modeling by presenting a design for modeling service that builds from the Open Geospatial Consortium (OCG) Web Processing Service (WPS) protocol. We demonstrate how the WPS protocol can be used to create modeling services, and then demonstrate how these modeling services can be brought into workflow environments using generic client-side code. We implemented this approach within the HydroModeler environment, a model coupling tool built on the Open Modeling Interface standard (version 1.4), and show how a hydrology model can be hosted as a WPS web service and used within a client-side workflow. The primary advantage of this approach is that the server-side software follows an established standard that can be leveraged and reused within multiple workflow environments and decision support systems.

Keywords: Integrated Modeling, Web Services, Service-Oriented Computing, Component-based Modeling, OpenMI, OGC WPS

*Corresponding author
Email addresses: Castrona@email.sc.edu (Anthony M. Castronova), Goodall@cec.sc.edu (Jonathan L. Goodall), Elag@email.sc.edu (Mostafa M. Elag)

1. Introduction

Environmental systems are difficult to model in large part because they include dynamic physical, chemical, and biological processes that are coupled and can vary at different spatial and temporal scales. Added to this complication is the role of humans in altering natural systems through land use change and introducing management practices to mitigate environmental impacts. Creating accurate models of environmental systems requires designing software applications that are able to grow in scope and integrate models across disciplinary boundaries. In addition, there are social barriers in achieving model interoperability such as software and data ownership, ontological and conceptual modeling differences, and organizational cooperation (Argent, 2004; Lerner et al., 2011). Given these challenges, one of the goals of the environmental modeling community is achieving interdisciplinary modeling in a way that maintains state-of-the-art knowledge and modeling approaches, while also enabling interoperability of models across disciplinary and organizational boundaries. The technical approaches for achieving this level of model interoperability range from sharing model input and output files, to rewriting models into a single software system, to establishing software architecture principles that facilitate the coupling of otherwise independent models. In the latter case, models are linked to others within a workflow and data are passed between models at simulation run time. This approach where models are written in a modular way and utilize object oriented programming, enabling them to remain as flexible, extensible, and reusable as possible (Argent et al., 2006), is gaining momentum within the earth science community (Syvitski et al., 2004; Hill et al., 2004; Kralisch et al., 2005; Castronova et al., 2012).

Despite the recent progress made in technical solutions to model coupling, challenges still remain. These challenges can be expressed as the assumptions a modeler must make when linking models in workflow environments and include (i) all models are developed using the same programming language, (ii) all programming languages are compatible with the operating system that the workflow software targets, and (iii) all models require the same computer hardware architecture (i.e. desktop computer vs. high performance computing). In some cases, these assumptions can be overcome by clever software engineering or computational tools. For example, the Community Surface Dynamics Modeling System (CSDMS) uses the BABEL tool to provide support for multiple programming languages within its modeling system (Peckham and Goodall, 2012). However, general approaches that address one or more of these assumptions are not commonly available to the majority of the environmental modeling community, and therefore research is needed to provide additional ways for models to overcome these technical challenges.

In our prior work (Goodall et al., 2011) we proposed the use of web services as a means for coupling environmental models, in part because the service-oriented paradigm addresses the assumptions that were outlined above. A particular focus of our past work was investigating possible methods for exposing models as web services. We discussed how current standards including the Open Geospatial Consortium (OGC)

Web Processing Service (WPS) and the Open Modeling Interface (OpenMI) provide much of the needed technical specification for creating modeling services. However, the case study implementation was limited to a proof-of-concept example that largely ignored these standards. A motivation for the work presented here was to establish a more formal service oriented design pattern that is built from open source software. Therefore, through this research we advance our prior work by focusing on the OGC WPS standard as a mechanism for exposing models as web services and the OpenMI standard for consuming them. This research addresses the applicability of the WPS and OpenMI standards to be used in conjunction for service oriented environmental modeling. We offer a detailed implementation of a hydrologic model, TOPography-based MODEL (TOPMODEL), as a WPS service. We then demonstrate how client applications can use this web service by focusing on the HydroModeler application, which is built from version 1.4 of the OpenMI standard as described in Castronova et al. (2012). We then discuss insights gained through this work into many of the challenges that arise when implementing models as web services.

2. Background

Service-oriented computing is a concept in which a larger software system is decomposed into independent distributed components (Papazoglou and Georgakopoulos, 2003; Huhns and Singh, 2005). These components are deployed on remote servers and are designed to respond to individual requests from client applications. Service-oriented components often require input, which can be in a variety of languages and formats, for example eXtensive Markup Language (XML) or JavaScript Object Notation (JSON), and return output back to the client application. This client-server interaction follows a loose integration paradigm as depicted in Figure 1. First, the client makes a request to the web service, supplying any necessary input data. The service then interprets this input data and executes its processing sequence to produce the desired output. Finally, this output is sent back to the client application for further processing or visualization. Using this paradigm, the client and server operating systems, computer languages, and computational architectures can differ but remain interoperable because the data transferred between these systems is standardized.

In past work we have shown how a process-level model component can be written in the Python (version 2.5) programming language and be used in an OpenMI model configuration, implemented in C# (Microsoft .NET Framework 3.5) using the eXtensive Markup Language Remote Procedure Call (XML-RPC) specification (Goodall et al., 2011). Using this approach web service messages are "wrapped" into a standardized XML encoding and transfered between client and server side software. Other web service standards could be used in place of XML-RPC such as the Simple Object Access Protocol (SOAP) or the REpresentational State Transfer (REST) specifications. The SOAP specification is widely used and is very easy to consume in client applications using Microsoft .NET programming languages, but it is more complex than REST and as a result often requires the use of software toolkits. REST web services are generally easier and

faster to develop than SOAP web services, however, there can be difficulties in this approach. For instance, formulating Uniform Resource Identifier's (URI) to access resources is not always intuitive, and its simplistic nature makes it easy to misuse (Ray and Kulchenko, 2003). Despite these limitations of REST, we choose to use it here for its simplicity and flexible nature.

REST is a specification that defines four mechanisms for communicating between client and web service: identification of resources (via URI), manipulation of resources through representations, self-descriptive messages, and interactions using hyperlinks (Fielding, 2000). A single RESTful web service can host a set of web resources, so URI's are used to help clients discover, identify, and consume a specific one. Once a connection is established with a resource, a set of generic methods that follow the HTTP protocol (GET, PUT, POST, DELETE, HEAD) are used to interact with it (Fielding, 2000). Individual client-server interactions are stateless, however there are methods for imposing state, if necessary (Pautasso et al., 2008). The REST specification, like SOAP and XML-RPC, is designed in a general way and is not associated with a specific type of application. Therefore there have been efforts within communities to standardize how specific types of data are described and communicated across the Internet. For example, the Open Geospatial Consortium (OGC) is a non-profit organization that aims to standardize the communication of geospatial data. Over the years this organization has released many data specification standards such as the Web Feature Service (WFS), Web Map Service (WMS), and Web Coverage Service (WCS) (Vretanos, 2005; La Beaujardière, 2002; Whiteside and Evans, 2008). These data specifications are commonly used in software systems including Geographic Information Systems (GIS) applications. Furthermore, they are designed to encapsulate data into a common format that can be understood by both client and server applications. For instance, the WFS, WMS, and WCS are designed to be consumed by client applications seeking to retrieve geospatial feature data, map images, or spatial coverage data, respectively.

When exposing models as web services, there is a need to consider how simulation data should be encoded for communications between the client and server. One solution is to leverage an existing data specification such as the OGC Web Processing Service (WPS). The WPS was designed to facilitate in the development web resources and the communication of their geospatial data calculations (Schut, 2007). Furthermore, it supports the communication of various types of data (i.e. complex, literal, and bounding box) that are useful for geospatial calculations (Schaeffer, 2008). WPS specification requires that web resources be implemented in a specific way. For example, when invoked by a client, they must accept input data, use this data to perform a computation, and finally encode the result into XML and send it back to the client. Geospatial calculations, unlike those of typical environmental models, are often time-independent and therefore do not require storing and referencing previous calculations. In addition, WPS resources are designed to perform the same computation every time they are invoked by a client application. In contrast, environmental models often consist of several phases of simulation (e.g. initialization, run, and finish) and must be capable of performing more than one type of calculation in a given simulation. Although there are distinct disconnects

between these two implementations, the goal of this work was to show it is possible to adapt the WPS design for deploying models as web resources and encoding model computations.

3. Design

We have organized the discussion of the modeling service design by first addressing the server-side software required for exposing a model as a web service and then the client-side code required for using the model service within a software system. While the design is meant to be modeling system agnostic, much of the discussion is centered on OpenMI (version 1.4) as an example modeling system used when designing both the server and client-side software. This means that the server and client-side software is designed specifically to work with this version of the OpenMI, but also to be extensible so that it can be used in future work to support other modeling frameworks as well.

3.1. Modeling Web Service using the OGC WPS Specification

The Open Geospatial Consortium's (OGC) Web Processing Service (WPS) was chosen as the specification to encapsulate data communications between client application and web resource. This decision was made because the WPS implementation shares several similarities with modeling frameworks such as OpenMI, CSDMS, and others. First, WPS includes an *initialization* phase during which objects can be created on the server to store data, and preliminary calculations can be performed to setup the model. Second, WPS includes a *run* phase that can be used to perform computations for each time step of the model run. A difference between the two approaches, however, is that component-based models are initialized only once per simulation whereas their computational phase may be called multiple times throughout a simulation (e.g. OpenMI version 1.4). In contrast, a WPS resource performs its initialization routine when it is instantiated (i.e. every time it is invoked), and its run phase is designed to perform time-independent computations. This difference can be overcome by designing WPS resources to follow a specific coding structure to enable support of time-step model computations, and thus mimic the run phase of component-based modeling frameworks.

3.1.1. WPS Implementation

WPS resources were created by leveraging the existing PyWPS[1] project. PyWPS (version 3.2.1) is an implementation of the OGC WPS specification that supports Python version 2.6, and enables the creation and seamless deployment of WPS resources. One major benefit of using this existing technology, as opposed to creating a new implementation of WPS, is that new resources can be rapidly prototyped by creating a single function Python module. It is in this manner that models can be transformed into web service

[1]PyWPS, `http://pywps.wald.intevation.org` (accessed Nov. 2011)

resources and utilized by client applications such as the OpenMI Configuration Editor (Gregersen et al., 2007) or the HydroModeler (Castronova et al., 2012). Unlike OpenMI-compliant models, WPS resources can only perform an initialization routine when they are instantiated. Furthermore, resources consist of a single simulation phase: execution. Upon invocation, a new instance of the resource is created and the initialization is performed. Once this is complete, it immediately begins its execute phase. After successful completion of the execute method, output results are encoded in XML and sent back to the client. A resource retains no data calculations in memory, nor information about which client invoked it. Therefore we can say that WPS resources do not maintain *state*. The lack of initialization can be overcome by including a mandatory data input that flags the resource to perform an initialization routine when required (i.e. at the beginning of an OpenMI simulation), as described in Section 3.2.1. If flagged, the initialization phase will occur after instantiation of the resource, but prior to resource execution. However, this is not useful if the resource cannot retain these calculations for subsequent invocations by the client. Therefore, the more serious disconnect is maintaining state on the server.

3.1.2. Maintaining State

State is required on the server to form a connection between a specific client and the resource it is invoking. It enables the resource to hold calculations in memory or on disk, and also recognize which client is calling it and therefore identify the simulation *session*. By definition RESTful services are stateless (Fielding, 2000), however state can be maintained artificially. This can be done by storing data objects on the server after a resource has been invoked and has performed its execute phase. By serializing Python data objects and storing them for later calculations, it is then possible to un-serialize them for subsequent calculations. It is important that these objects be associated with a specific client, so they are saved using a randomly generated key that both the client and server share. Using this approach, a resource no longer has to perform a costly initialization routine every time it is invoked, rather it can lookup previously calculated model parameters stored on disk. Furthermore, this enables models to save calculations from previous time steps if they are required by the model.

For maintaining state and storing data values on the server, two additional HTTP methods were implemented on the PyWPS web server: PUT and DELETE. The HTTP PUT method can be used to send and store data for a resource on the server (Fielding et al., 1999). For a model deployed as a WPS resource, input data can be stored on the server and accessed when the resource is invoked during an OpenMI model simulation. The PyWPS web service implementation does not contain a definition for the PUT method because traditional WPS services do not require it. However, models often require time-independent input data prior to simulation. The PUT method is implemented on the server in a very similar manner to the PyWPS implementation of POST. It is designed to extract key-value pairs from an XML text stream, reconstruct them as Python objects, and serialize them using a unique session key (Figure 2). This key is stored by

the client to establish a model session, and passed to the WPS resource upon subsequent invocations. This enables the resource to identify which client is calling it, and use data specific to that instance to perform its computation. Since each data serialization stored on the server represents an individual simulation, it is important to remove them when they are no longer required, i.e. when a model simulation is complete. This is done by implementing the HTTP DELETE method. Again, the PyWPS does not contain a definition for DELETE because the typical WPS usage does not require it. The DELETE method was designed to remove all files on the server that are associated with a given session key. Furthermore, it is called by the client once a resource is no longer needed (Figure 2). This functions to keep the server free of unnecessary data files. In this research we decided to implement these two additional REST methods within the existing PyWPS software package. However, this could have also be accomplished by creating additional WPS resources specifically to mimic the HTTP PUT and HTTP DELETE functionality. We chose the former approach because it is a more RESTful solution, i.e. it adheres to the REST architectural style proposed by Fielding (2000).

3.1.3. Model Execution

The typical implementation of a WPS resource entails performing time-independent calculations upon invocation. To enable time-dependent calculations, we first established a means for maintaining session state on the server. This made it possible for a resource to identify the model session, load the relevant data, and finally perform its computation. To support models as web services, concepts from the OpenMI standard were implemented into the design, specifically the concept that models are incremented in time so that each invocation of the service advances the model a single time step. By developing a resource so that a single computation is performed during its execute phase, the client can simply call the resource once for each time step of the simulation. This call to the web resource fits within the *run* phase of the OpenMI. In this way, the client receives output generated by the web resource on every time step of simulation, and these data can then be utilized by other components within a workflow. Finally, at the end of the execute phase the simulation data and the time-independent calculations are re-serialized and saved to disk so that they can be accessed during the next invocation. Although time-step computations are used in this work, the same approach can be adopted for component-based modeling systems that are designed to perform on either time-steps or time spans (e.g. CSDMS or the Earth System Modeling Framework (ESMF)), by modifying WPS implementation to include start and end times for each invocation (Peckham et al., 2012).

3.2. Creating an OpenMI Client Component for WPS Models

A client application for the modeling service was created using the Microsoft C# .Net Framework 4.0 to bring the modeling service into the OpenMI-based HydroModeler environment. This was accomplished using a methodology similar to Castronova and Goodall (2010) that begins by creating a client-side wrapper class

called the Wps-Wrapper. This wrapper is a generic implementation of the OpenMI ILinkableComponent class that has been specifically designed to consume WPS resources at simulation run time. This is possible by deriving generic implementations for many of the OpenMI methods, thereby allowing their values to be automatically populated after a connection is made to the desired WPS resource. User input for the Wps-Wrapper is specified in the model's *.omi file (Table 1). Inputs are categorized as either *service*, *OpenMI*, or *resource*. Inputs such as the WPS URI (Uniform Resource Identifier), service name, version, and resource identifier, fall into the service inputs category. These are used to establish a connection with the desired web service. Similarly, OpenMI inputs are those that are required on the client machine, but not on the server, such as simulation start and end times, time step length, and spatial data. Finally, resource specific input parameters can also be defined within this file. The Wps-Wrapper has been designed to accept two different formats of resource input, (1) a single value or (2) a path to a file containing values stored in Python list form. Using this approach, WPS resources can be accessed and wrapped into an OpenMI-compliant components by using the information specified in this file. To conform to the OpenMI paradigm, the WPS resource must be accessed during different phases of simulation like all other OpenMI-compliant models. To do this, the client component can cache WPS metadata locally to reduce the number of web service calls. In the following subsections, the three main phases of OpenMI simulation are discussed along with the necessary communications with the web service during each phase.

3.2.1. Initialization Phase

The initialization phase of an OpenMI model simulation is when model components are loaded into memory. All OpenMI components are required to enter this phase immediately after they are added into a model configuration. In addition, components perform pre-processing routines and create the data structures that they require for model simulation. One disconnect between the OpenMI standard and WPS specification is that the WPS does not have a definition for model initialization, rather resources are initialized and subsequently executed every time they are invoked by the client. This presents a challenge when deploying models as WPS resources, because for models, the initialize phase is often used to prepare for simulation. Furthermore, this task requires performing calculations that may take a noticeable amount of time to complete, and it would be computationally inefficient to perform them on every simulation time step. To overcome this shortcoming, a mandatory input parameter is defined for any OpenMI-compliant model deployed as a web resource. This input parameter, named *initialize*, takes a boolean value that designates if a resource must perform its initialization routine. This input parameter enables the client to control when a web resource will initialize itself, and thereby eliminating redundant calculations. These calculations are serialized by the resource to maintain state for subsequent calculations, as discussed in Section 3.1. Unfortunately, since WPS services execute a computation every time they are invoked, the initialization routine will not be performed until the first time step of simulation, unlike other OpenMI-compliant models.

However, using this approach resources can now be initialized in a similar fashion to OpenMI-compliant models; once per simulation.

At the beginning of the initialization phase, the Wps-Wrapper retrieves information about the inputs and outputs of a web resource using the WPS DescribeProcess operation (Figure 3) (Schut, 2007). Using this approach, resource inputs and outputs are discovered and converted into OpenMI exchange items. Furthermore, resource name, description, and other metadata are extracted and used to specify component metadata. Unfortunately, there are several parameters that do not exist in the WPS specification which must be included to successfully translate resource inputs and outputs into OpenMI exchange items. Unlike the OpenMI, the WPS inputs and outputs contain a overly simplistic definition for units of measure (UOM). To accommodate the OpenMI standard, several additional descriptors must be included within WPS UOM field. These additional descriptors, such as variable dimension, unit id, and unit description, are included as key-value pairs that can be easily parsed by the Wps-Wrapper. This provides the Wps-Wrapper with all of the necessary information to translate resource inputs and outputs into OpenMI exchange items. Other simulation attributes, such as the simulation start time, end time, and time step, are defined within the model's *.omi file (see Table 1).

In addition to the quantity portion of an exchange item, the spatial (i.e. element set) portion must also be defined. Since the modeler initiates a simulation from the client, it is suitable for model files, such as spatial definition, to also exists on the client. Furthermore, by maintaining the spatial definition of exchange items on the client, we eliminate the burden of encoding this data and sending it to the server. Additionally, the OpenMI Software Development Kit (SDK) libraries can be leveraged to perform spatial interpolations during data transfers, if necessary, rather than handling these complex tasks on the server. The spatial definition for input and output exchange items are also specified in the *.omi file. In its current state, it is assumed that a model consists of one input element set and one output element set. This shortcoming is discussed in Section 5, however it is possible to expand our implementation such that unique element sets for each input and output exchange item can be defined. Exchange item element sets are created using the approach illustrated by Castronova and Goodall (2010) in which geospatial feature data stored in shapefile format are parsed into OpenMI element sets using an open-source geospatial application framework, called SharpMap[2].

Models also typically require input data that remains static throughout simulation, i.e. data that is not supplied by other components during simulation. Similar to other inputs, these are specified in the model's *.omi file. Since this file follows a simple XML schema, these inputs can be specified as key-value pairs and then parsed within the Wps-Wrapper. It is then possible to supply them to the web service resource using the HTTP PUT method. This method enables clients to add data to the server, for a specific resource. This

[2]SharpMap: "Geospatial Application Framework for the CLR," `http://www.codeplex.com/sharpmap`

functionality was added to the PyWPS specification specifically to accept key-value data encoded in XML, and save it on the server so that it can be accessed by resources during simulation. Using this approach, the client can seamlessly transfer simulation inputs to the server, prior to simulation run time. These data are saved on the server using a unique identifier, enabling the resource to locate data for a specific simulation instance when invoked.

3.2.2. Run Phase

The run phase of an OpenMI simulation is comprised of multiple time step calculations, in which components communicate data amongst each other between subsequent calculations. This enables components to transfer values back and forth to resolve dynamic system interactions. During this phase, a method named PerformTimeStep (defined in the OpenMI Standard Development Kit, SDK) is called on every time-step of the simulation (Figure 4). This method is implemented in a manner consistent with Castronova and Goodall (2010) in which time-dependent input data are retrieved from incoming links, prior to computation, using the OpenMI GetValues method. Next, a Uniform Resource Identifier (URI) string is formulated in which these data values are encapsulated. This string is passed to the server, where the arguments are parsed into inputs for the WPS resource. The resource processes its calculation, encodes the output values in XML, and sends them back to the client as XML text. Once received by the Wps-Wrapper, the results are extracted from the XML string and exposed to other models within the OpenMI configuration. This procedure is repeated for every time step of simulation.

3.2.3. Finish Phase

This finish phase of an OpenMI model simulation is initiated once all components have completed their run phase. At this time, each component typically saves locally stored calculations, or performs a post processing routine. Once complete, all memory allocated during simulation is released and the component is officially finished with its simulation. For the Wps-Wrapper, this method follows a very simple implementation. Each data value that was calculated during the run phase of simulation is written to file. This enables the modeler to review calculations, as well as document a simulation run. Next, the HTTP DELETE method is called on the server, which signals the resource to delete all data that was serialized during simulation.

4. Case Study Implementation

The previous section described the design of a client-server architecture for environmental modeling. This section focuses on an implementation of this design for a specific case study. In this case study, the hydrologic Topography based MODEL (TOPMODEL) is deployed as a WPS resource. The Wps-Wrapper is the used to consume it within a larger modeling workflow application.

4.1. Model Description

The TOPMODEL is a hydrologic model designed to predict watershed runoff using readily available observation measurements (Beven, 1997). It is a continuous simulation model that has been extensively applied to a variety of catchments in various regions throughout the world (Beven, 2004). Furthermore, calculations are performed in a gridded fashion, in which the watershed is subdivided into a matrix of equal sized cells. From Hornberger (1998), the total outflow from the watershed is partitioned into overland and subsurface flows (Equation 1).

$$Q_{total} = Q_{subsurface} + Q_{overland} \tag{1}$$

Overland flow is produced only when the soil becomes saturated. It is calculated as function of fractional saturated area A_{sat}/A, the rate of throughfall P, and return flow Q_{return}.

$$Q_{overland} = \frac{A_{sat}}{A} P + Q_{return} \tag{2}$$

Furthermore, subsurface flow is calculated as a combination of the flow transmitted downslope and the flow coming into a given area at the current time. These concepts are used to derive Equation 3, where T_{max} is the saturated soil transmissivity, λ is the average topographic index for the watershed, $\bar{s}$ is the average saturation deficit, and m is a user defined scaling parameter (Hornberger, 1998).

$$Q_{subsurface} = T_{max} e^{-\lambda} e^{\frac{\bar{s}}{m}} \tag{3}$$

The topographic index parameter plays a significant role in this calculation because it is used to determine when saturation conditions are met. It is a precomputed model input parameter that is derived from the watershed elevation. Furthermore, TOPMODEL consists of several additional watershed specific input parameters that can be used for calibration. All of these are either predetermined or specified at run time by the user. TOPMODEL, however, still requires precipitation and evapotranspiration as time-dependent input for its calculation. Both of these variables are provided to the resource during simulation from client-side OpenMI components.

4.2. Setting up the Model as a Service

A model is deployed as a WPS resource by creating a python class that implements the PyWPS package. PyWPS effectively handles the communication to and from the resource, therefore development efforts can focus solely on the computational algorithm. For this study the TOPMODEL algorithm is developed as a web resource by implementing the derivation outlined by Hornberger (1998). To solve the relationships described in the previous paragraphs, several time-independent inputs must be supplied to the resource. These time-independent inputs, unlike the time-dependent ones, remain constant throughout the simulation.

For example, the TOPMODEL relationships rely on a precomputed watershed parameter, derived from surface topography (Beven, 2004), known as the topographic index. Furthermore, other watershed specific inputs such as saturated soil transmissivity, interception, recession coefficient, and a scaling parameter must be supplied to the web resource prior to model execution. The TOPMODEL resource also requires two time-dependent inputs: precipitation and evapotranspiration. These inputs are supplied by two pre-existing OpenMI-compliant components that are executed on the client.

Once the resource has been developed and deployed on a web server, it is consumed into a client-side OpenMI configuration using the Wps-Wrapper. To set up the wrapper for simulation, web service inputs such as uri, service name, version, and resource identifier are specified within *.omi file. These parameters provide the client with the necessary information to communicate with the desired resource. Similarly, OpenMI specific model input data such as start time, end time, time step, and element sets are included. These parameters enable the user to define OpenMI specific variables that are not required by the resource, but are fundamental to the OpenMI standard. Finally, resource specific inputs are added to the *.omi file. For this study a scaling parameter m, saturated soil transmissivity T_{max}, an interception parameter i , recession coefficient r, grid cell size, and topographic index ti are included. All of these input parameters are expressed as a single value except for topographic index. It consists of an array of values that are precomputed using a Digital Elevation Model (DEM) collected from the United State Geological Survey (USGS). Because of this, the ti entry in the *.omi file is a path to a file containing values stored in specific file structure (see Table 1), which mimics the Python definition for a list object. Once all of the necessary data are provided, the model can be loaded into an OpenMI-compliant editor such as the HydroModeler (Castronova et al., 2012).

4.2.1. Running the Model within a Workflow Composition

A model configuration is constructed using a plugin application that enables OpenMI-compliant modeling within the HydroDesktop environment, called the HydroModeler (Castronova et al., 2012). To create a model composition for this study, several additional model components must be added to the workflow. First, evapotranspiration is calculated using a pre-existing component that implements the Hargreaves and Samani algorithm (Hargreaves and Samani, 1982). It is a simplification of the complicated evapotranspiration process, and requires fewer inputs than other commonly employed methods (e.g. Monteith et al., 1965; Priestley and Taylor, 1972). It approximates the potential moisture loss by considering evaporation into the to the atmosphere, as well as transpiration by vegetation. Furthermore, it requires temperature data as input, which is readily available from various online sources. This component is used to supply predictions for evapotranspiration to the TOPMODEL resource during an OpenMI simulation. In addition to evapotranspiration, a component is required to supply precipitation data to the TOPMODEL resource. This is done using another component named the DbReader. It functions by reading and exposing data stored in

the Consortium of Universities for the Advancement of Hydrologic Science, Inc. (CUAHSI) HydroDesktop application, as described in Castronova et al. (2012). Similarly, the DbReader component also supplies the required temperature data to the Hargreaves-Samani evapotranspiration component. The flow of data amongst model components is illustrated in Figure 5.

To complete the model configuration, links are established between components to define how data will flow during simulation run time. For this study, a link is added between the DbReader and the Hargreaves-Samani components to supply the temperature data required for the potential evaportation computation. Similarly, a link is added between the DbReader and the Wps-Wrapper components to the supply precipitation data for the watershed outflow calculation. Next, a link is established to send the potential evapotranspiration calculation from the Hargreaves-Samani component to the Wps-Wrapper component. Collectively, these links define how data will flow during model simulation (Figure 5).

To drive model simulation, observation data must be collected and stored on the client. This is done using the CUAHSI HydroDesktop application (Ames et al., 2009), to discover, search, and then download the desired data series. For this simulation, the HydroDesktop application is used to retrieve data for the Coweeta #18 Watershed, located in western North Carolina (Figure 6). This is a small (0.12 km^2) experimental catchment maintained by the U.S. Department of Agriculture (USDA) Forest Service that is part of the larger Coweeta watershed (16.26 km^2). The Coweeta watershed is part of the Long Term Ecological Research (LTER) project and is maintained by the USDA specifically for research studies. It contains a climate station located approximately 1.7 km from the Coweeta #18 watershed boundary, that is used to supply the required precipitation and temperature data for this model simulation. These data were downloading using the HydroDesktop and stored in a local database repository. The DbReader component seamlessly loads these data and converts them into exchange items that can be used during OpenMI model simulation.

Once loaded into the HydroModeler, each component enters its initialization phase of simulation. During this phase, components perform a series calculations to prepare themselves for simulation. For the Wps-Wrapper component, all inputs are first read from the *.omi file. The web service related inputs are used to establish a connection with the specified WPS resource. Once a connection has been established, the resource is queried using the WPS DescribeProcess operation (Schut, 2007) to discover the inputs and outputs. These are then used by the Wps-Wrapper to construct OpenMI exchange items. Next, the OpenMI-specific *.omi inputs are used to populate client-side parameters that will be used during model simulation such as start time, end time, time step, and element sets. Finally, resource specific inputs are encapsulated within an eXtensive Markup Language (XML) encoding and sent to server using the HTTP PUT method. This procedure loads resource inputs (i.e. ti, r, m, T_{max}, and grid cell size) onto the server. Once these operations have been performed, the Wps-Wrapper component is ready for simulation.

During the model simulation, calculations are performed and data are transferred between components

on each time step of the model simulation. For each time step, the DbReader provides output temperature data that serves as input to the Hargreaves-Samani component. This Hargreaves-Samani component then uses the temperature values to compute potential evapotranspiration. Next, the calculated potential evapotranspiration along with the observed precipitation are sent to the Wps-Wrapper client component. These data values are encoded into a Uniform Resource Identifier (URI) that is used to query the WPS resource. On the server, data inputs are extracted and converted into Python data objects. On the first time step of the simulation, the "initialize" parameter (described in Section 3.1) is flagged, which triggers the resource to perform its initialization routine. For all other time steps the Wps-Wrapper this parameter will not be flagged, so the resource will proceed to reading the serialized inputs that were loaded during the client-side initialization phase. Using the supplied values and the serialized data, the resource calculates watershed outflow using the TOPMODEL algorithm as described in Section 4.1. After the calculation is complete, the result is encoded into XML and sent back to the client. The Wps-Wrapper component reads the XML response and parses the output back into OpenMI objects. These are then exposed to other components within the simulation.

Once all components have completed their model simulations, each enters the finish phase. The Hargreaves-Samani and Wps-Wrapper components use this opportunity to save data calculations to a local file for post-processing and documentation purposes. In addition, the Wps-Topmodel component invokes the server using the HTTP DELETE operation to remove data serializations that are no longer needed. The streamflow predictions computed by the TOPMODEL resource (Figure 7) are available to be viewed on the client machine at this point as well. At this stage the TOPMODEL parameters (e.g. T_{max}, r, m, and i) can also be adjusted and the model simulation re-run in order to understand the sensitivity of model output to parameter values, as well as to calibrate the model to observed streamflow. Because these model parameters are specified in the Wps-Wrapper's *.omi file, changing the parameter values and then re-running the simulation is straight forward and can be automated through scripting.

5. Discussion

The method used in this study to deploy time-dependent models as web resources has been effective in creating a loosely integrated simulation composed of both client-side and server-side computations. This work utilizes existing technology where possible, such as the Web Processing Service (WPS) specification for data encapsulation and the PyWPS software for creating and deploying web resources. Using the WPS specification has some advantages including standardization of the service interface that enables client applications to more easily utilize resources. This standardization of model web services was the driving motivation for our work. We found the PyWPS implementation to be robust, but it required some extension to handle time-dependent modeling, primarily due to the need to maintain state of the server. In general,

it can be said that we are extending these technologies beyond their typical use cases, and as a result, have needed to overcome challenges. While we have addressed some of these, it is important to understand what challenges remain so that they can be taken into account when designing future software systems that require web service computations. One challenge that cannot be easily overcome is that as software systems begin to rely on remote resources for their computation, they become more fragile. Such software applications will not function correctly without an Internet or network connection, because they will not be able to access online resources. In addition, server downtime will break any client-side software that utilizes the resources they host. However, this limitation of the approach can be minimized through robust software engineering and hardware resources.

To design time-dependent models as web services, we employed an existing data specification standard known as the OGC WPS. It defines an interface specification that was originally designed for time-independent geospatial computations. Because of this, several additional design requirements were established so that it could be adopted for time-dependent calculations. First, time-dependent model resources using the WPS specification must accept a boolean *initialize* input parameter and perform initialization routines when it is set to true. This is required to impose an initialize phase for preparing a model resource for computation. This technique was effective, however it would be advantageous to modelers if the interface specification contained a standardized method for model initialization. By defining an initialization mechanism which can be easily adopted, it would be possible for modelers to initialize resources in a standardized manner, thus eliminating a possible source for design error. Second, output data were encoded in eXtensive Markup Language (XML) using the basic output data structure defined by Schut (2007). Additional descriptors are required to encapsulate the resource metadata that is necessary to construct OpenMI exchange items. In this work, these descriptors were specified in the WPS Units Of Measure (UOM) field as simple key-value pairs. Using this technique all of the necessary data for creating OpenMI exchange items can be supplied to the client. However, this basic data encapsulation lacks detail in describing the metadata for all resource inputs and outputs. An alternative method is to build from existing XML-based standards such as the Geography Markup Language (GML) or Water Markup Language (WaterML) (Portele, 2012; Taylor, 2012). Both of these standards are designed to describe more complex data, i.e. geospatial features and time-series data, respectively. In the future, a formal data standard should be designed to encapsulate time-dependent and time-independent model simulation data which may leverage both GML and WaterML concepts. This, more robust solution, could be designed to include all input, output, and simulation parameters as fields within an XML encoding.

Overall our approach for developing models as web services was proven effective in the TOPMODEL case study (Section 4), however we have yet to investigate more complicated workflows. Our study assumed a unidirectional communication stream in which inputs are sent to a web resource and calculations are sent back to the client. More complicated water resource processes, such as surface-groundwater interactions,

require bidirectional communication of data since these physical processes are fully coupled. Our approach can be adopted to accommodate this requirement. For instance, the data calculated by web resources can be supplied as input to client-side models. OpenMI links would then be used to define the flow of data between client-side model components, even bidirectional dependencies. Therefore, no modification is required to the web resource or the WPS Wrapper. This is also true in the case that two web resources require feedback from each other. Since each resource is "wrapped" into an OpenMI configuration with its own instance of the WPS Wrapper, a bidirectional link is simply established between the various WPS Wrapper instances on the client. In this scenario, data would flow from one web resource to its corresponding client-side WPS-Wrapper component, this data is then sent across an OpenMI link to the another WPS-Wrapper component which finally supplies the data to its corresponding web resource. Similarly, data can be sent back in the reverse order. The key is that the OpenMI handles the communication of data across all links, including bidirectional, within a client-side model configuration and therefore our web resource implementation and client-side WPS Wrapper do not require modification. However, both of these scenarios require more web based communication than was investigated in this work, which will most likely amplify any latencies in our service oriented modeling approach.

One important consideration that must be acknowledged is that simulation delays, or performance lags are inherent to service oriented modeling (Friis-Christensen et al., 2007; Goodall et al., 2011). The same features that make it possible to integrate models across computing platforms, computer architectures, and programming languages, are also the cause of performance lag. This is a result of the communication of data between computation resources, i.e. passing data between client software and web services. This method of exchanging data between computational resources is more time consuming than traditional methods in which models and data all exist on the same machine. For the case study presented in Section 4, it takes approximately 0.767 seconds to perform a single time step computation. This includes the communication of 219 bytes of data from the client to the web service, the model computation, and the subsequent communication of the results (3457 bytes of data) from the web service back to the client.

In addition, it takes approximately 1.083 seconds using a 202 byte data packet to invoke the WPS DescribeProcess method, which is required for preparing the client-side software for model simulation. Furthermore, it took approximately 0.248 seconds to upload model inputs for the Coweeta watershed study (83176 bytes of data) to the web resource, and 0.215 seconds to remove resource data (138 bytes of data) when the simulation was complete. In all, the service oriented model is approximately 20 times slower than a similar (although not identical) locally hosted model implementation. An important difference to understand when interpreting this finding, is that the model used for comparison was implemented entirely in C# (Microsoft .NET Framework 4.0), whereas the service oriented model utilized C# on the client and Python (version 2.6) for the web service. Therefore, this performance lag is not entirely due to web based communications, instead programming language speed is another potential source of performance lag.

Furthermore, little work was done to optimize the performance of the service oriented model. Finally, in this example data communication between client and server, and not model computation, dominated the total execution time. The service oriented approach will become more appealing as model computation time increases relative to the data communication time. For instance, web resources that employ large scale computations can be executed asynchronously, thus enabling the client to perform multiple tasks simultaneously (Friis-Christensen et al., 2007). In addition, models that require large amounts of input data could also greatly benefit from this approach because their data can be hosted on the same machine as the service, thus reducing the data transfers needed to "set up" the model. In the end, the modeler must be aware of these limitations when applying a service orientated approach for a given application in order to ensure acceptable performance.

Our approach for deploying time-dependent models as web services requires modification to the WPS standard implemented by the PyWPS. For instance, it is common for models to have time-independent input data that is supplied before simulation. This input data is used for, amongst other things, preparing the model and defining application specific parameters. In our approach, this is supplied to the web resource by the user using the HTTP PUT operation. Once this time-independent simulation data has been supplied to a resource it is serialized so that it can be accessed upon invocation. Each time a model resource is executed, it first reads the serialized data into memory, then performs a computation, and finally it re-serializes any computations or parameters that are required for the next calculation. This process of reading and re-serializing data is required so that resources can artificially maintain state. Unfortunately, it can be time consuming to perform on large data sets or across entire simulation durations and this is one reason for the slower performance in the case study. The implementation used in the study (Section 4) involved consuming web resources within an OpenMI (version 1.4) environment. Continuous serialization of data may have been an unavoidable overhead for this particular application due to the OpenMI communication protocol, however for other client applications that may not require extensive communication, this will not be a costly overhead. The latest release the OpenMI standard (version 2.0) may overcome some of these issues. The OpenMI 2.0 introduces several new concepts in addition to modifications of existing ones, to encourage use within a broader range of modeling domains. One significant change is that simulation control flow has been extended to enable "looping," in which models wait until all of their input data is available before beginning their computation (Gijsbers et al., 2010). Using this concept, models can now execute entire simulations, rather than time-step calculations, when data is requested from them by other components. This new simulation option enables OpenMI models to be constructed and executed in a sequential manner. Using this approach, an OpenMI 2.0 compliant model that does not require time-step level control could be developed directly as a WPS web service the executes an entire simulation upon invocation, and as a result it does not require artificial "state" to be imposed on the server (discussed in Section 3.1.2). Furthermore, the OpenMI 2.0 supports non-numerical data which would enable the passing of geospatial data between model

components (Gijsbers et al., 2010). The ability to pass geospatial data in combination with WPS-Wrapper would enable WPS spatial calculations to be supplied as input to client-side OpenMI components.

To eliminate the shortcomings presented in this work, additional research must be undertaken to provide a formal mechanism for deploying time-dependant models as web resources. This would likely require a new data specification that is able to encapsulate the complex interactions between client and service-oriented models. This proposed modeling specification should include a more detailed description of the data that is sent in each message. This would solve many of the discrepancies that arise when converting resource calculations into domain specific objects on the client. In addition, the web resource execution procedure must be expanded to include the other two phases of a model simulation: initialize, and finish. The initialization phase, would enable model resources to prepare themselves for simulation. It is required to implement the workaround presented in Section 3.2.1 in a standardized way by establishing a function to perform such procedures. Furthermore, this would require that state be maintained by the specification. This can be implemented in a manner similar to that presented in Section 3.1.2. Lastly, the model specification should also include a finish phase in which the resource can remove all simulation data that was stored on the server. In addition to this functionality, a method for sharing spatial data between the client and server during a model simulation may also be necessary for spatially dependent resource computations.

6. Summary and Conclusions

An approach for deploying time-dependent models as web services was established. Web service models were designed to communicate with clients using the REpresentation State Transfer (REST) web service standard in conjunction with the Open Geospatial Consortium's (OGC) Web Processing Service (WPS) data specification. A hydrologic model, TOPMODEL, was deployed as a WPS using our approach. It was demonstrated how these resources can be consumed by client applications by utilizing the TOPMODEL service within a loosely integrated modeling architecture that follows the OpenMI standard. To do this, a client-side Wps-Wrapper was developed specifically to consume models deployed as WPS services and convert them into OpenMI-compliant components.

We concluded that it is possible to expose a model as a web service using the WPS specification and to utilize the service in the OpenMI-based HydroModeler environment by developing a client-side, OpenMI-compliant model wrapper component. The client-side component is designed such that it can connect and interact with the web resource during a model simulation. The Wps-Wrapper component offers a mechanism for integrating WPS resources into an OpenMI model configuration by simply changing properties within the *.omi file. This file consists of three types of inputs: web service, OpenMI specific, and resource. Together they are used by the Wps-Wrapper to connect to the web resource, populate client-side OpenMI parameters, and upload resource input data to the server. Because these properties are specified in an input file, there is

no need to recompile source code when using the Wps-Wrapper with different web resources. By providing this mechanism for wrapping service-oriented models, it was possible to establish a generic solution to the common model integration problem. This use case demonstrates how our approach for deploying time-dependent models as web services can be adopted by the scientific community to solve integrated modeling problems. Furthermore, this technique for facilitating communication between web services and clients illustrates how time-dependent models can be designed for a broader spectrum modeling applications.

The benefit of a service-oriented approach is that it overcomes the assumptions stated in the introduction of this paper that modelers must often make when integrating models into a workflow: model resources must (1) be written in a single computer language, (2) be compiled for a specific operating system, and (3) utilize the same hardware architecture. By allowing models to be implemented on different systems and serve as distributed resources using a loosely integrated into a modeling system, these three assumptions are no longer necessary. This increased flexibility comes at a cost, however, including decreases in model performance due to serialization and transfer of objects across the Internet, security challenges associated with having models available to client applications, and increased fragility of models because they are dependent on remote resources. As with many of the decisions that modelers face when designing and utilize a model, care should be taken to understand the benefits and costs of a decision, including the underlying architecture choice for the model, and to choose the option where the benefits outweigh the costs.

Acknowledgments

This work was supported by the National Science Foundation under the grant NSF EAR 0622374 "Geoinformatics: Consortium of Universities for the Advancement of Hydrologic Sciences, Inc. (CUAHSI) Hydrologic Information Systems."

References

References

Allman, M., 2003. An evaluation of XML-RPC. ACM SIGMETRICS Performance Evaluation Review 30 (4), 2–11.

Ames, D. P., Horsburgh, J. S., Goodall, J. L., Witeaker, T. L., Tarboton, D. G., Maidment, D. R., 2009. Introducing the open source CUAHSI hydrologic information system desktop application (HIS Desktop). In: 18th World IMACS / MODSIM Congress. Cairns, Australia, pp. 4353–4359.

Argent, R.M., 2004. An overview of model integration for environmental applications–components, frameworks and semantics. Environmental Modelling & Software 19 (3), 219–234.

Argent, R. M., Voinov, A., Maxwell, T., Cuddy, S. M., Rahman, J. M., Seaton, S., Vertessy, R. A., Braddock, R. D., 2006. Comparing modelling frameworks - a workshop approach. Environmental Modelling & Software 21 (7), 895–910.

Beven, K., 1997. Topmodel: a critique. Hydrological Processes 11 (9), 1069–1085.

Beven, K., 2004. Rainfall-runoff modelling: the primer. Wiley.

Castronova, A. M., Goodall, J. L., 2010. A generic approach for developing process-level hydrologic modeling components. Environmental Modelling & Software 25 (7), 819–825.

Castronova, A. M., Goodall, J. L., Ercan, M. B., 2012. Integrated modeling within a hydrologic information system: An OpenMI based approach. Environmental Modelling & Software (In press).

Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., 2001. Web services description language (WSDL) 1.1. Technical report, World Wide Web Consortium.

La Beaujardière, J., 2002. Web map service implementation specification. Open GIS Consortium 82.

Dong, S., Karniadakes, G., Karonis, N., 2005. Cross-site computations on the TeraGrid. Computing in Science & Engineering 7 (5), 14–23.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., 1999. Hypertext transfer protocol–HTTP/1.1. RFC 2616.

Fielding, R. T., 2000. Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California, Irvine, Ca.

Friis-Christensen, A., Ostländer, N., Lutz, M., Bernard, L., 2007. Designing service architectures for distributed geoprocessing: Challenges and future directions. Transactions in GIS, 11 (6), 799–818.

Gijsbers, P., Hummel, S., Vaneçek, S., Groos, J., Harper, A., Knapen, R., Gregersen, J., Schade, P., Antonello, A., Donchyts, G., 2010. From OpenMI 1.4 to 2.0. International Congress on Environmental Modelling and Software.

Goodall, J. L., Robinson, B. F., Castronova, A. M., 2011. Modeling water resource systems using a service-oriented computing paradigm. Environmental Modelling & Software.

Gregersen, J. B., Gijsbers, P. J. A., Westen, S. J. P., 2007. OpenMI: Open modelling interface. Journal of Hydroinformatics 9 (3), 175–191.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J., Nielsen, H. F., Karmarkar, A., Lafon, Y., 2007. SOAP version 1.2 part 1: Messaging framework. Technical report, World Wide Web Consortium.

Hargreaves, G., Samani, Z., 1982. Estimating potential evapotranspiration. Journal of the Irrigation and Drainage Division 108 (3), 225–230.

Hill, C., DeLuca, C., Balaji, Suarez, M., da Silva, A., 2004. The architecture of the earth system modeling framework. Computing in Science & Engineering 6 (1), 18–28.

Hornberger, G., 1998. Elements of physical hydrology. Johns Hopkins University Press.

Huhns, M., Singh, M., 2005. Service-oriented computing: Key concepts and principles. Internet Computing, IEEE 9 (1), 75–81.

Kralisch, S., Krause, P., David, O., 2005. Using the object modeling system for hydrological model development and application. Advances in Geosciences 4, 75–81.

Laurent, S., Johnston, J., Dumbill, E., 2001. Programming web services with XML-RPC. O'Reilly Media.

Lerner, D.N., Kumar, V., Holzkämper, A., Surridge, B.W.J., Harris, B., 2011. Challenges in developing an integrated catchment management model. Water and Environment Journal 25 (3), 345–354.

Mitra, N., Lafon, Y., 2007. SOAP version 1.2 part 0: Primer. Technical report, World Wide Web Consortium.

Monteith, J., et al., 1965. Evaporation and environment. Symposium of the Society for Experimental Biology 19, 205–234.

Papazoglou, M., Georgakopoulos, D., 2003. Service-oriented computing. Communications of the ACM 46 (10), 25–28.

Pautasso, C., Zimmermann, O., Leymann, F., 2008. Restful web services vs. "big" web services. In: Proceeding of the 17th international conference on World Wide Web. Beijing, China, p. 805.

Peckham, S. D., Goodall, J. L., 2012. Driving plug-and-play models with data from web services: A demonstration of interoperability between CSDMS and CUAHSI-HIS. Computers & Geosciences. (In press)

Peckham, S. D., Hutton, E. W.H., Norris, B., 2012. A component-based approach to integrated modeling in the geosciences: The design of CSDMS. Computers & Geosciences. (In press)

Portele, C., 2012. OpenGIS Geography Markup Language (GML) Encoding Standard. Open Geospatial Consortium.

Priestley, C., Taylor, R., 1972. On the assessment of surface heat flux and evaporation using large-scale parameters. Monthly Weather Review 100 (2), 81–92.

Ray, R., Kulchenko, P., 2003. Programming web services with Perl. O'Reilly Media.

Schaeffer, B., 2008. Towards a transactional web processing service. Proceedings of the Sixth Geographic Information Days, Münster.

Schut, P., 2007. Open geospatial consortium inc. OpenGIS® web processing service. Open Geospatial Consortium, 1–87.

Syvitski, J., Paola, C., Slingerland, R., Furbish, D., Wiberg, P., Tucker, G., 2004. Building a community surface dynamics modeling system: Rationale and strategy. http://www.nsf-margins.org/S2S/S2SWhitePaper.pdf.

Taylor, P., 2012. OGC WaterML 2.0: Part 1–Timeseries. Open Geospatial Consortium.

Vretanos, P., 2005. Web feature service implementation specification. Open Geospatial Consortium Specification, 04–094.

Whiteside, A., Evans, J., 2008. Web coverage service (WCS) implementation standard. Open Geospatial Consortium.

Winer, D., 1999. XML-RPC specification. *http://www.XML-RPC.com/spec*.

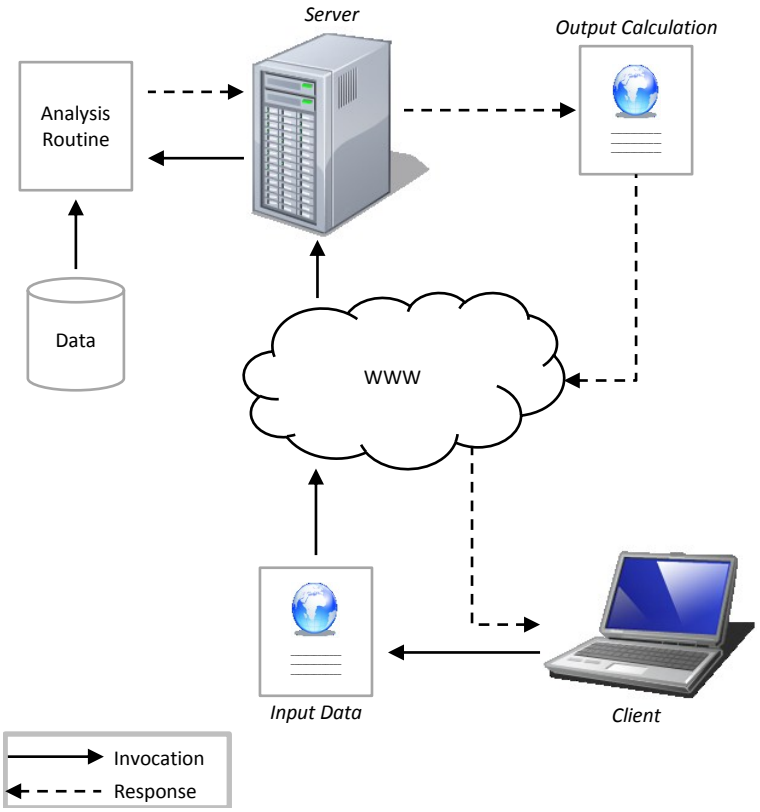

Figure 1: Communication between the client and server where an analysis routine is run on a server and called by a client application using a web service interface. Data can be transfered from the client to the server as input, and from the server and the client as output.

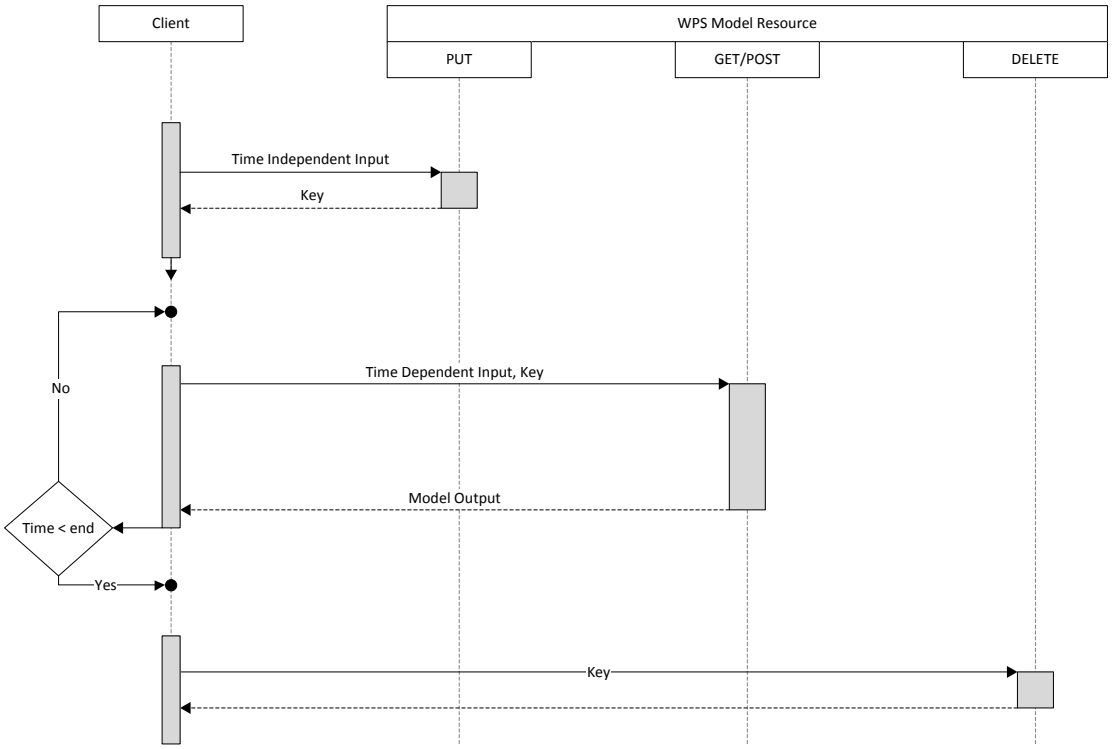

Figure 2: The PUT method is used to load time-independent input data onto the server to prepare a resource for simulation. During a model simulation, resources use time-dependent data values and the stored time-independent data to compute output. Once simulation is complete, unnecessary client session data are deleted the server using the REMOVE method.

Input Type	Key	Example Value	Description
Service	URI	http://my_server_location	URL of the web server
Service	service	my_wps	Name of the web service
Service	version	1.0.0	Version of the web service
Service	identifier	my_resource_name	WPS resource name
OpenMI	starttime	1/1/2010 10:00:00	Desired simulation start time
OpenMI	endtime	1/1/2011 10:00:00	Desired simulation end time
OpenMI	timestepinseconds	86400	Length of each time step in seconds
OpenMI	inputelementset	/location/of/elementset.shp	Shapefile that defines where input data must exist
OpenMI	outputelementset	/location/of/elementset.shp	Shapefile that defines where output data will be calculated
Resource	parameter_1	10	Single value resource parameter
Resource	...	...	Optional parameters
Resource	parameter_n	variable_x.txt	List of parameter values, e.g. $var\_x = [v_1, v_2, \ldots, v_n]$

Table 1: A graphical representation of an *.omi file for the client-side Wps-Wrapper. WPS parameters are specified as key-value pairs, and are categorized as either *service*, *OpenMI*, or *resource* inputs.

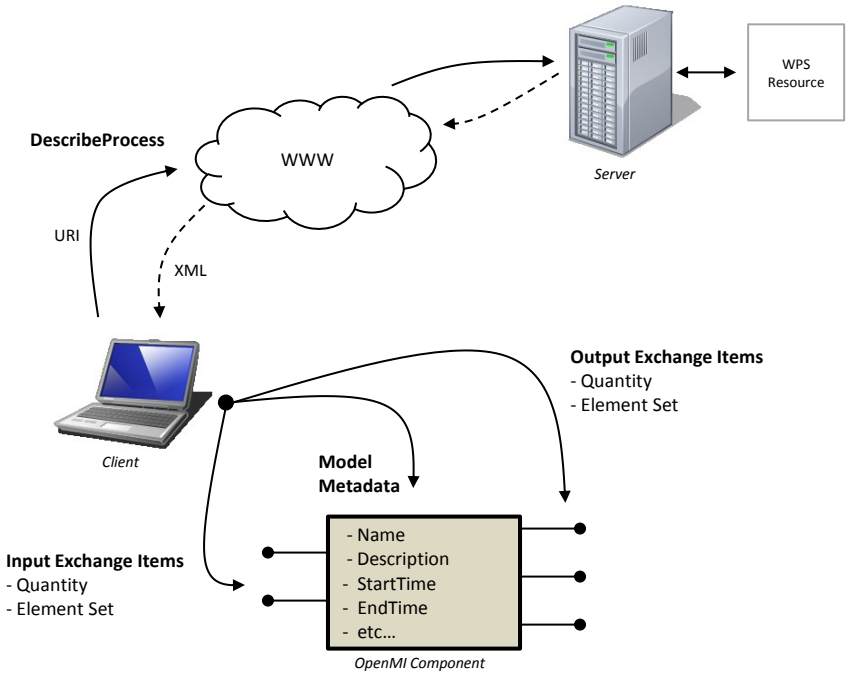

Figure 3: During the initialize phase of client-side simulation, the Wps-Wrapper queries the web resource using the WPS Describe Process operation. The XML response is used to create OpenMI exchange items and to populate component parameters.

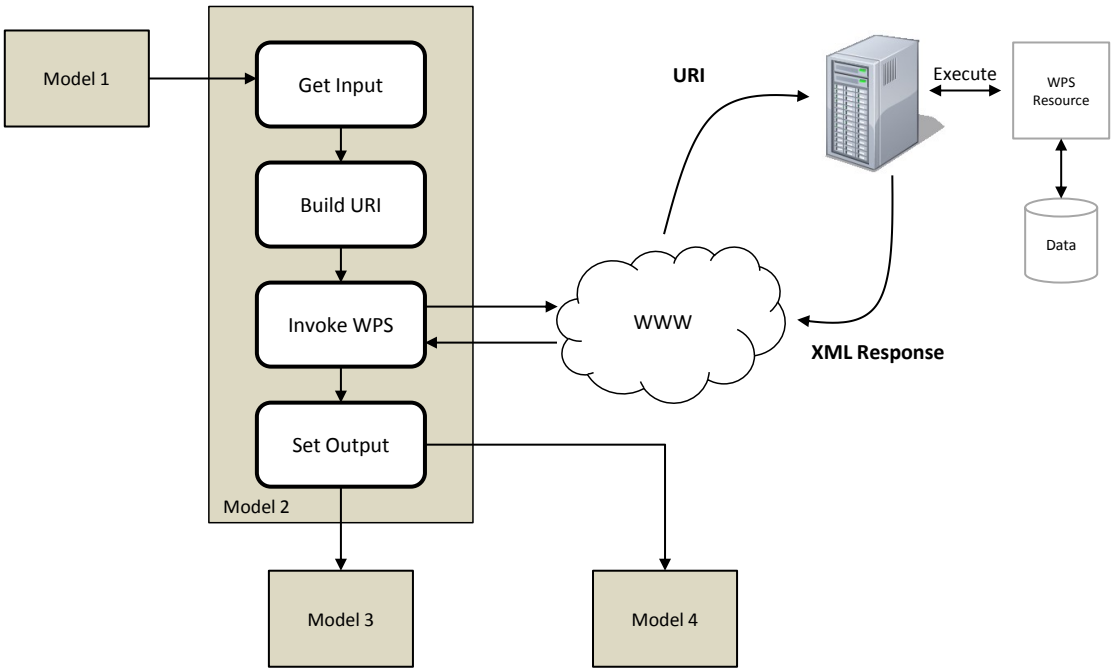

Figure 4: During the run phase of the simulation the client retrieves time-dependent input data from other the components, then builds a URI using this input data and invokes the WPS resource. The resource returns its output (encoded in the WPS XML specification) to the the client Wps-Wrapper which exposes these values to other components within a model configuration.

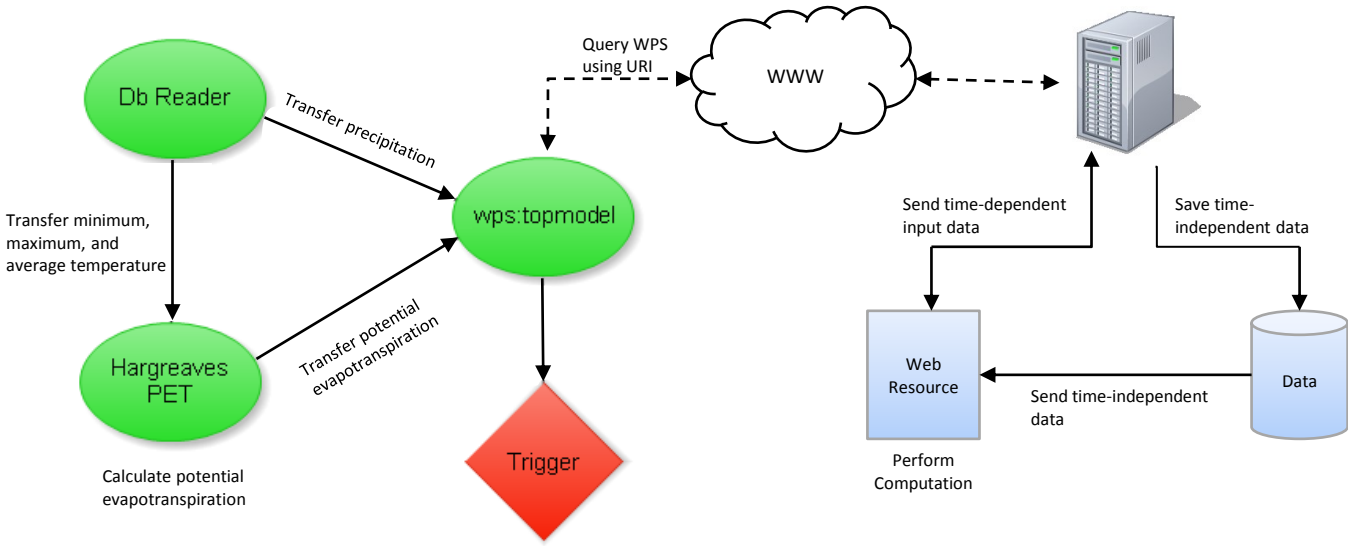

Figure 5: An OpenMI model configuration consisting of the DbReader, Hargreaves PET, and WPS TOPMODEL components. The links established between components define the flow of data during a single time step of the simulation.

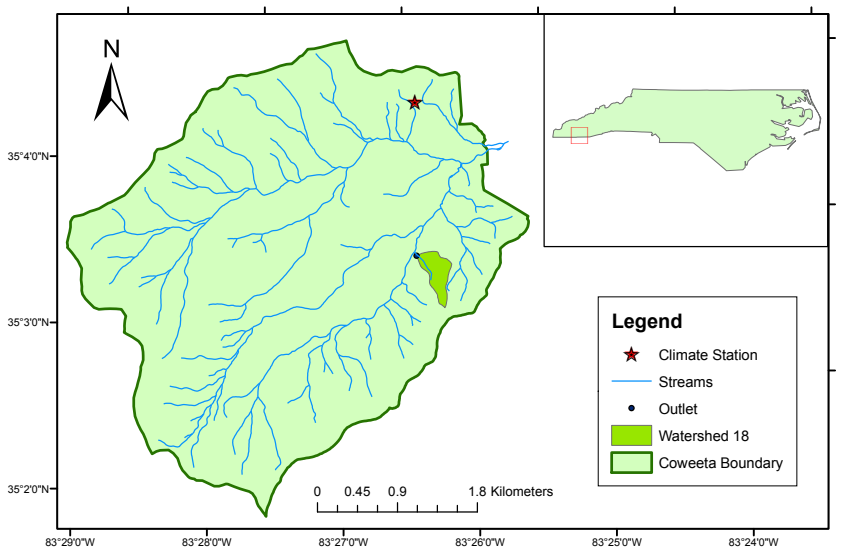

Figure 6: The Coweeta watershed, located in western North Carolina, is part of the Long Term Ecological Research (LTER) project maintained by the U.S. Department of Agriculture (USDA) and the U.S. Forest Service. It includes an experimental catchment that is used in this study named Watershed #18.

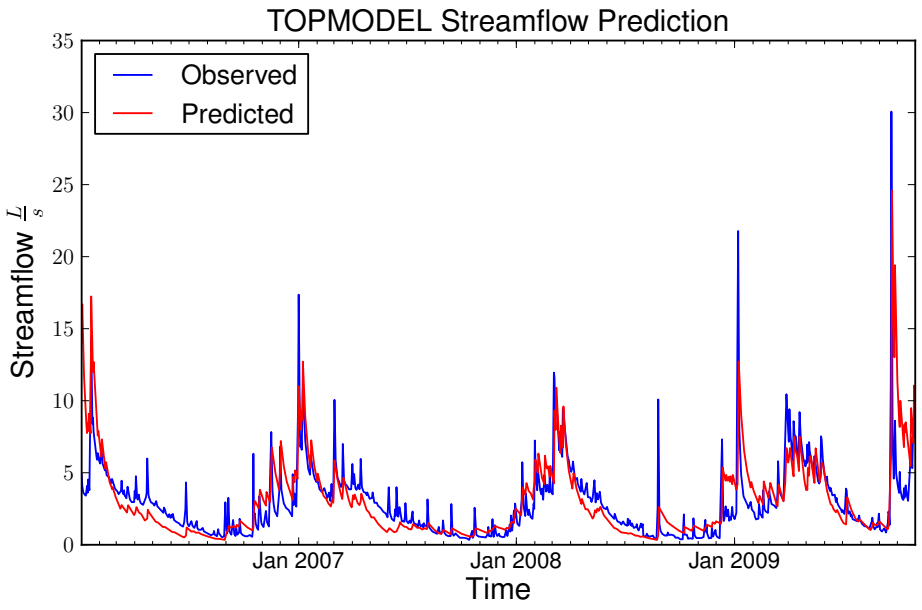

Figure 7: The predicted streamflow computed by the TOPMODEL web service utilized within a client-side OpenMI component that serves as part of a HydroModeler workflow configuration.