This is an Accepted Manuscript of an article published in Environmental Modelling & Software in 2011, available online: doi:10.1016/j.envsoft.2010.11.013

Modeling Water Resource Systems using a Service-Oriented Computing Paradigm

Jonathan L. Goodall[a,c], Bella F. Robinson[b], Anthony M. Castronova[a]

[a] *Department of Civil and Environmental Engineering*
University of South Carolina
300 Main Street, Columbia, South Carolina 29208 USA
[b] *ICT Centre*
Commonwealth Scientific and Industrial Research Organisation
Building 108, North Road, Acton ACT 2601, Australia
[c] *Corresponding Author: goodall@cec.sc.edu, v: 803.777-8184, f:803.777.0670*

Abstract

Service-oriented computing is a software engineering paradigm that views complex software systems as an interconnected collection of distributed computational components. Each component has a defined web service interface that allows it to be loosely coupled with client applications. The service-oriented paradigm presents an attractive way of modeling multidisciplinary water resource systems because it allows a diverse community of scientists and engineers to work independently on components of a larger modeling system. While a service-oriented paradigm has been successfully applied for integrating water resource data, this paper considers service-oriented computing as an approach for integrating water resource models. We present an interface design for exposing water resource models as web services and demonstrate how it can be used to simulate a rainfall/runoff event within a watershed system. We discuss the advantages and disadvantages of using service-oriented computing for modeling water resource systems, and conclude with future work needed to advance the application of service-oriented computing for modeling water resource systems.

Keywords: Integrated Modeling, Systems Analysis, Web Services, Water Management

1. Introduction

Modeling regional-scale water resource systems requires new modeling paradigms for capturing the complex biogeochemical and socio-economic interactions operating within natu-

ral systems (Argent, 2004; del Barrio et al., 2006; Clark and Gelfand, 2006; Engelen et al., 1995; Jakeman and Letcher, 2003). Many of the questions commonly asked by scientists and policy-makers require an integration of models across disciplinary boundaries, for example hydrology, ecology, agriculture, and socio-economic disciplines, and therefore a challenge facing the modeling community is how to most efficiently make existing disciplinary models interoperable. In addition to making existing models interoperable, there should also be the goal of creating a new generation of water resource system models where interoperability across disciplinary boundaries is considered a priority at the outset. The objective of this paper is to explore the applicability of a service-oriented computing paradigm as the thought model for building integrated water resource modeling systems that allow existing and new models to interoperate.

Despite the fact that commonly used water resource models have had decades of development effort, most effort has either been directed toward improving numerical algorithms, data preparation tools, or user interface capabilities. The core software architectural principles used within water resource models have largely remained unchanged. This is beginning to change through efforts to modernize water resource model architectures (Hill et al., 2004; Leavesley et al., 1996; Moore and Tindall, 2005; Syvitski et al., 2004). These efforts emphasize similar concepts of modularity and componentization as a key principle to achieve more open, flexible, and extensible modeling systems. Concurrent to model framework development efforts within the water domain, new software engineering approaches are being put forth by computer scientists (Allan et al., 2006; Foster, 2005; Jennings, 2001), and so there is a perpetual need for reevaluation and consideration of new software architecture paradigms to better understand their appropriateness and utility for modeling water resource systems.

Recent attention has focused on service-oriented architectures as a means for building environmental decision support systems (Mineter et al., 2003; Granell et al., 2010; Goodall et al., 2008; Horsburgh et al., 2009). In service-oriented computing, a software system is viewed as independent components or services that are loosely coupled and able to exchange data with one another over a computer network (Curbera et al., 2002; Huhns and Singh, 2005). Service-orientation is a core concept behind distributed computing where the Internet

is used not only for delivering information from machines to humans, but also between machines themselves (Huhns and Singh, 2005; Foster et al., 2001). Service-oriented science is a term coined by Foster (2005) that refers to scientific research enabled by distributed networks of interoperating services. The concept of service-orientation shares much in common with component-based modeling as described by Argent (2004). Each service acts as a component that is an autonomous entity capable of responding to requests instantiated by outside entities. The primary distinction between service-oriented computing and component-based modeling is that service-orientation implies distributed components interoperating over a network (Huhns and Singh, 2005). This distinction caries with it important design and use case implications that are discussed in this paper.

Service-oriented modeling implies a loose-coupling approach for integrating models. This differs from many of the past approaches for integrating water resource simulation models that have followed a tight-coupling integration approach (Figure 1). In a tight-coupling approach, as demonstrated in such models as Band et al. (1993), Facchi et al. (2004), Maxwell and Miller (2005), and Qu and Duffy (2007), the originally independent models are integrated by porting code into a single modeling application. The advantage of a tight-coupling approach is that it provides complete control over the process representations and data structures within all parts of the model. The model can therefore make use of the most efficient algorithms to solve complicated numerical problems, for example fully-coupled systems of differential equations representing flow between groundwater and surface water systems (Qu and Duffy, 2007). The disadvantage of tight coupling is that, because internal conventions such as data structures and semantics within a model are fixed, it becomes difficult to integrate models that do not conform to a existing conventions. For example, a tight integration approach may not be the ideal means for integrating physical models and models that are not easily described by physically-based governing equations to create a holistic water resources modeling system. In contrast, a loose-coupled architecture requires only the standardization of interfaces and data exchanges, giving each model component developer more control over how to structure the internal algorithms required to represent a particular system.

The advantages of adopting a loose-coupling, service-oriented paradigm for modeling extend beyond allowing for the integration of multidisciplinary water resource models. By using web services, modeling system components that require complicated software configurations, large data demands, or significant computational resources, for example, might be more easily integrated into a decision support system by keeping each model in its own hardware environment, but then exposing functionality through web service interfaces. In other cases, it might be practical or necessary to keep models under the control of certain groups. By exposing such models as web services, groups will be able to maintain control and more easily update their models, while still having their model provide a service within a larger modeling system. In addition to these advantages of using a loose-coupling, service-oriented paradigm for modeling, there are also potential disadvantages to the approach. While the approach has proven successful for data integration, model integration has a different set of challenges including the need for models to be calibrated and model users to be clear on what the model was calibrated to optimize. There are also technical challenges such as allowing remote users to create sessions through web services in order to have interactive control over how the model steps through time, set boundary conditions or manipulate state variables as the model advances through time.

Past work in the application of web services within the water resources community has focused primarily on exposing historical databases (Goodall et al., 2008), integrating water data across heterogeneous data providers (Horsburgh et al., 2009), or data processing workflows using web services (Granell et al., 2010). The Consortium of Universities for the Advancement of Hydrologic Science, Inc. (CUAHSI) Hydrologic Information System (HIS) provides an example of how web services are being used in the United States to address data integration challenges in the water community. The CUAHSI HIS integrates hydrologic observational data from heterogeneous federal, state, and local data providers into a "virtual database" using a service-oriented architecture (Maidment, 2008). Although data is physically stored with distributed data providers, a standard means for automatic retrieval of data and metadata from providers using web services gives end users the view that HIS is a central database of information (Horsburgh et al., 2009). Granell et al. (2010) demonstrate

how the role of services can be expanded to not only handle data delivery, but also data processing and visualization. The authors use hydrologic simulation models as part of their service-oriented architecture, however as web applications ingesting data from web services and not as web services themselves.

The goal of this research is to add to the previous work on the application of web services for data gathering, processing, and visualization to consider how web services can be used for exposing hydrologic simulation models. In this paper we demonstrate how a water resource model can be exposed using a service-oriented approach, focusing specifically on the role of existing standards and their applicability to service-oriented hydrologic modeling. Following this introduction is a background section that provides further context for our work. We then present a general methodology for exposing a hydrologic process model as web services. We describe a simple case study demonstration where we implement the Muskingum Routing method as a web service, wrap that web service as an OpenMI-compliant component (Gregersen et al., 2007), and then include the component within a rainfall/runoff model configuration. We provide a discussion of the benefits and limitations of our proposed approach for modeling water resource systems using web services, and conclude with future work needed to move the presented case study to a more robust and complete implementation.

2. Research Context

A water resource decision support system using a service-oriented architecture would consist of databases, analysis routines, and models distributed across the Web and available for use in workflow orchestrations (Figure 2). Each database, processing routine, and model would be available to client applications through a web service interface, and scientists would use software to create workflow orchestrations that define the data flow between services. All services within the system would be registered in a central repository with metadata describing its capabilities, inputs, and outputs. This would allow scientists to discover new modeling services for use in their modeling applications. A key attribute of web services is that all communication between service providers and service consumers follows predefined

data exchange standards. Communication standardization allows services to be loosely coupled, i.e. plug-and-play, and reusable within multiple software systems.

Web service standards begin with lower-level specifications that target a broad user group. At this level, a service interface is defined using the Web Service Description Language (WSDL) (Curbera et al., 2002). Each service has an associated WSDL document that acts as a contract between the service provider and service consumer, specifying service operations, input data, and output data. Data transfers are often encoded using the Simple Object Access Protocol (SOAP), which allow for the exchange of both simple data types (e.g. string, integer, double array, etc.) or complex data types (e.g. classes). These core web service standards provide a level of interoperability between machines, but they are generic protocols that must often be supplemented with domain-specific standards (Foster, 2005).

Domain specific web service standards applicable to the water resource community are primarily aimed at exposing data and analysis routines as web services. Examples include the CUAHSI HIS and its standards for service interface design (WaterOneFlow) and data transmission (Water Markup Language), the Open Geospatial Consortium (OGC) and its standards for transmitting geospatial data such as Web Feature Service (WFS) and Web Coverage Service (WCS), and the OPeN Data Access Protocol (OPeNDAP), widely used within the scientific community for accessing gridded model output data. While standardization across these services is still ongoing, each service defines a protocol for requesting and receiving data from remote data providers. In addition to standards for serving data, the OGC has also established a standard for data processing called the Web Processing Service (WPS). Because WPS is particular relevant to our work, we will provide a more detailed discussion of WPS in the following section.

Although web service standards for modeling do not exist in the water resources-related communities, there has been considerable related work on establishing standards for component-based modeling of environmental systems (Argent, 2004). Examples include the Earth System Modeling Framework (ESMF), the Community Surface Dynamics Modeling System (CSDMS), and the Open Modeling Interface (OpenMI). Modeling frameworks often target different disciplines and therefore have design goals targeted to best address each

community's needs. For example, both ESMF and CSDMS emphasize High Performance Computing (HPC) for simulating complex climate and surface dynamic systems (Hill et al., 2004; Peckham, 2008). OpenMI emphasizes interoperability between otherwise independent models, in part because it was designed to address integrated modeling challenges that necessitate the coupling of proprietary simulation models (Moore and Tindall, 2005). Few if any modeling frameworks have been designed using a service-oriented architecture, although there has been some work to explore extending existing frameworks to accommodate web services (Čurn, 2007; Turuncoglu et al., 2009).

In addition to specifying standards for exposing models as services and for exchanging data between modeling services, another critical part of a service-oriented modeling system envisioned in Figure 2 is software that enables web services integration. This process, called service orchestration (Peltz, 2003), allows users to define workflows where the output from one service becomes the input to a second service. Workflows can vary greatly in their level of sophistication, from simple serial execution of a digraph of linked services, to parallel, asynchronous execution of a graph with loops, branching, and peer-to-peer data transfers (Yu and Buyya, 2005). Two example workflow systems are the Kepler scientific workflow application (Ludscher et al., 2006) and the Open Modeling Interface (OpenMI) Configuration Editor (Gregersen et al., 2007).

Kepler is built on a lower-level dataflow-oriented system called Ptolemy II that includes many of the workflow elements defined in the Yu and Buyya (2005) taxonomy. In Kepler, workflow components are called actors, and each actor has a set of ports that define input and output data associated with that actor. Scientists using a Graphical User Interface (GUI) define workflow orchestrations by linking the output from one actor to serve as the input to a second actor. Kepler allows a web service to be leveraged within Kepler workflows by the user inputting the web service's WSDL document to embed that service as a Kepler actor (Ludscher et al., 2006). Kepler also includes actors for executing and monitoring Grid computing tasks (GlobusJob, GridFTP, etc.), along with local data processing and visualization using R, Python, and a suite of prebuilt actor components.

The Open Modeling Interface (OpenMI) Configuration Editor is an application for de-

signing and running linked models that adopt the OpenMI protocol for model integration (Gregersen et al., 2007; Moore and Tindall, 2005). The OpenMI was designed specifically for integrating water resource models and, for this reason, it includes domain-specific specifications for digitally describing water resource data passed between coupled models. The OpenMI Software Development Kit (SDK) provides an implementation of the OpenMI standard along with supporting tools for creating components and the OpenMI Configuration Editor is a GUI software tool for creating and running component workflows. OpenMI is a less general tool than Kepler, in that it is specific for water resource modeling, and therefore it does not natively include the capability of ingesting web services as components within a workflow.

We elected to use the OpenMI Configuration Editor as our workflow environment in our case study demonstration section because it includes conventions specific for water resource modeling. This required the creation of a web service client component that allows one to ingest a web service model within a OpenMI workflow. An important point to stress, however, is that a model web service could be incorporated within multiple workflow environments in addition to other analysis and visualization software applications. Therefore, while our case study demonstration makes use of the OpenMI configuration editor, potential applications are not limited to this specific framework. Kepler could likewise be used to perform web service orchestration of water resource models. This level of interoperability is one of the key features of web services and service-oriented architectures. It is enabled by standardizing service interfaces and data exchanges, which in turn simplifies the work required to leverage the service as a resource within a particular application environment.

3. Design of a Modeling Service

Because standards are critical to service-oriented modeling, the first objective of our work was to design a standard for exposing models as web services. We intentionally took inspiration from existing standards for specifying service interfaces and data exchanges to limit the proliferation of repetitive standards. Following this section, we present an implementation of the web service design for a specific hydrologic model.

3.1. Modeling Service Interface Design

The interface for a service defines how service consumers request information from that service. We considered two existing specifications for providing a water resource modeling web service interface. The first is the Open Geospatial Consortium (OGC) Web Processing Service (WPS) and the second is the Open Modeling Interface (OpenMI). We considered the WPS because it offers a standards-based approach for processing data, and we considered the OpenMI because it offers a interface specification for water resource simulation models. Neither standard is ideal for service oriented modeling, for the reasons described in the following paragraphs. However, we have concluded that by combining ideas from these two standards, it is possible to design a web service interface appropriate for exposing water resource models. In this section we will first describe both WPS and OpenMI and then describe how we used the two standards to design a web service standard for water resource modeling.

The Web Processing Service (WPS) standard has three methods: GetCapabilities, DescribeProcess, and Execute (Open Geospatial Consortium, 2007; Michaelis and Ames, 2008). The GetCapabilities method, included in many OGC web services, allows client applications to explore and discover service capabilities as a list of processes. The DescribeProcess method provides metadata specific to an individual process on the service. For example, a service might have a process named "buffer" and a DescribeProcess response would inform the client that the buffer process requires an input feature class and a buffering distance and will result in a buffered feature class. The Execute method is called to perform a specific process. The response from an Execute method summarizes the inputs and outputs for the process and information on whether the process ran successfully. The WPS is designed for data processing where one or more inputs are used to generate one or more outputs. The WPS is a general service that would need to be extended to handle water-specific data structures and semantics. Furthermore, maintaining session state, meaning the server stores information associated with a specific user between service calls, this not often used with a WPS. It is, however, needed for providing clients with run-time control over a model's execution.

The OpenMI standard is designed for component-based modeling of water resource systems. The standard includes an interface specification called ILinkableComponent for defining models as system components. The OpenMI Software Development Kit (SDK) defines a second interface called IEngine that inherits from ILinkableComponent and provides a higher-level means for wrapping numerical simulation models as components. The IEngine interface provides a syntactically and semantically rich interface for describing water resource models. For example, the IEngine interface and the OpenMI standard in general include class specifications for ElementSets to *where* model elements exist, Quantities to describe *what* is being passed into or out of the model, and TimeHorizon to describe *when* the model is able to produce outputs. That said, OpenMI was designed for integrating models that are implemented as local resources. This design feature is evident by the fact that IEngine interface includes over twenty methods and attributes for encapsulating a model as a component. In a service-oriented architecture, it is likely that models may be distributed across the Internet and so there is a need to minimize network call and data transfers between the model components and the modeling framework.

Because the WPS lacks domain specificity and the OpenMI assumes that all models are local resources, neither approach is ideal for service-oriented water resource modeling. We propose that the two protocols, when combined, would leverage the strength of each to create an appropriate protocol for service-oriented water resource modeling. We designed a web service interface that begins with the OpenMI and modified the interface in order to achieve a level of complexity closer to the WPS. The OpenMI methods can be broadly grouped into two categories. The first category are methods meant for getting and setting properties of the model that will remain static during the time which the model is initiated until it is deleted from the server's memory. The second category includes methods which get or set values that will be dynamic during the model's lifetime. Examples of static methods that fit into the first category include the OpenMI IEngine GetModelDescription and GetTimeHorizon methods. These methods will likely not be updated and will return the same information each time they are called by a client. The prime example of a dynamic method that fits into the second category is GetValues. As the model progresses through

time, each GetValues call will likely return a unique data object.

Our web service specification is summarized in Figure 3a. The model metadata, which is retrieved through the IEngine interface by using method calls for each property, would instead be retrieved from the service using a DescribeModel method call that lumps metadata into an XML string. This change would bring the IEngine interface closer to the Web Processing Service interface that uses a DescribeProcess method to inform the client application about the service capabilities, inputs, and outputs. The other service methods should include an Initialize method for preparing the model for use, a PerformTimeStep method for advancing the model in time, and a Finalize method for removing the model from memory. Finally, it is important to state that this design would not strictly follow the OpenMI standard, but it is intentionally designed to provide sufficient metadata and capabilities in order for it to be wrapped as an OpenMI-compliant component. The service design would, therefore, be a lower level interface that allows but does not require OpenMI-compliance.

3.2. Data Transfer Schema

In addition to specifying a standard interface for services, it is also necessary to define the data exchanges between services. Precise definition of these data exchanges, both syntactically and semantically, is required to allow client applications to properly understand and operate on data exchange objects. This is perhaps the largest challenge in achieving interoperability within a service-oriented architecture for water resource modeling. Because covering both the syntactic and semantic interoperability challenges is beyond the scope of a single paper, we focus our discussion primarily on the syntactic interoperability challenges. The organization of water data into a core set of data exchange objects must include sufficient detail for proper interpretation of information, but must not be verbose to avoid large data transfer sizes. One of the core design decisions is how to group space, time, and variables into data exchange objects. Two potential organizations are time series (one variable, one location, and multiple times) used by the CUAHSI HIS in its Water Markup Language (WaterML) schema or time slices (one variable, one time, and multiple locations) used by the OpenMI in its ExchangeItem class. While it would be possible to support both

time series and time slice data structures, keeping the variety of data exchange objects to a minimum reduces the complexity of the server and client code required for interpreting the data exchange objects.

Our design begins from the OpenMI data transfer objects for data exchange between models within a service-oriented architecture because the data exchanges are designed to be the minimal information required for properly interpreting values passed between models during simulation run-time. The organization of OpenMI data transfer objects is presented in Figure 3b. The basic object passed between client and server is an ExchangeItem object (Gregersen et al., 2007). The ExchangeItem object includes two child objects as properties: an ElementSet and a Quantity. Geographic features are encapsulated within an ElementSet object that include one or more point, line, polygon, or volume elements. Variables are defined by a Quantity object that includes a variable name, measurement units, measurement unit dimensions, and conversion factors to standard units. An exchange item is therefore a time slice organization of information meaning it is valid for one moment in time, for one variable type, but for many spatial elements.

While the OpenMI ExchangeItem object is an appropriate starting point for designing data exchange standards in a water resource modeling web service, there are still some limitations of this exchange standard. OpenMI does not restrict the vocabularies of variable names, unit names, or geographic referencing systems (Gregersen et al., 2007). These attributes of an exchange item are simply string data types in which the user input is unrestricted. The assumption is that a human will properly interpret variable and unit names when matching output and inputs between components. Unit dimensions, however, are limited to a defined list of terms, and thus are checked for consistency when interlinking two components during a configuration stage (Gregersen et al., 2007). To realize the full potential of service-oriented modeling, additional work is needed to extend the OpenMI ExchangeItem objects using semantics and controlled vocabularies, such as what is available in the NetCDF Climate and Forecast (CF) Metadata Conventions (Eaton et al., 2009) or in the CUAHSI HIS ontology used to support search over disparate databases (Beran and Piasecki, 2009).

3.3. Service Orchestration

The final step in service-oriented modeling is defining workflow orchestrations that interlink services to perform analysis or modeling tasks. In this subsection we discuss three types of workflows that are important for modeling and analysis of water resource systems: (1) time independent, (2) component-level time looping, and (3) workflow-level time looping (Figure 4). This section is intended to provide a high level discussion of how each type of workflow would be implemented within a service-oriented computing paradigm, and the next section provides an example implementation. The three approaches are listed in order of increasing difficulty in terms of the sophistication of the software required to address each approach.

In the first case the workflow might perform data processing or analysis tasks with no need to track model run time as part of the workflow execution (Figure 4a). Such a workflow may be used to execute a series of data processing tasks needed to transform model input or output data. An example of a time-independent workflow is a series of linked geoprocessing operations required to create a watershed model input file (Figure 4a). Time-independent workflows use a directed graph workflow structure where each processing step is performed sequentially (Yu and Buyya, 2005). It is possible to implement such workflows as multi-threaded tasks where independent branches of the graph are computed in parallel. The data movement between components within a time-independent workflow is mediated by a central job scheduler and intermediate files are often written to standard file formats (Yu and Buyya, 2005). The ArcGIS Model Builder environment is an example of a workflow environment that processes what we refer to as time-independent workflows.

The second case of workflows takes the form of running multiple models in series or, where possible, in parallel with model time being handled at the individual component level (Figure 4b). Having time handled at the component level means that each model component begins at time zero and runs through all time-steps before communicating data to downstream components. Ideally components within such a workflow run on the same time horizon and time step, although there is the possibility of including translation code between component interfaces. Because each component runs without run-time input from other components,

it is not possible to allow feedbacks between model components. An example of a water resources model that would be implemented using this approach is a lumped rainfall/runoff model where hydrographs are generated for subwatersheds and then routed downstream through the river network (Figure 4b). Like the time independent workflow, it is possible to achieve performance gains by executing components on independent branches of the workflow in parallel. Our case study provides an example of implementing a component-level time looping workflow where one of the workflow components is implemented as a web service.

The third case of workflows is a dynamic model where model components communicate during the workflow execution and time looping is controlled at the workflow rather than the component level (Figure 4c). Because time-looping occurs at the workflow level, components will communicate data as the simulation advances in time. An example of a workflow that could be implemented using this approach would be a simulation of a hydrologic system that includes predictions of river stage, infiltration rate, groundwater level, and precipitations rates by a set of coupled models (Figure 4c). Data movement in such a workflow would ideally follow a peer-to-peer pattern to minimize bottlenecks associated with centralized and mediated data movements (Yu and Buyya, 2005). There is a great deal of flexibility in such a workflow structure compared to the others because model time is handled externally from individual services so that services communicate during a simulation. This allows for a more dynamic interaction between services including feedback loops and component-to-component interactions necessary for modeling dynamic system interactions. The OpenMI Configuration Editor accomplishes such workflows with its peer-to-peer data flow implementation where components request information from linked components as the workflow progresses in simulation time (Gregersen et al., 2007).

4. Prototype Implementation

In this section we present a prototype implementation of the web service for the case of workflow-level time looping service interactions. In related work we have implemented a rainfall/runoff model using a typical hydrologic engineering approach for a small watershed in Columbia, SC USA as OpenMI components (Castronova and Goodall, 2008).

Excess rainfall was computed by a component that implements the Curve Number method, incremental runoff by a component that implements the Unit Hydrograph method, and streamflow throughout the river network by a component that implements the Muskingum Routing method (Chow et al., 1988) (Figure 5). The components were implemented as OpenMI components using the Simple Model Wrapper (SMW) approach designed to ease the process of creating new, process-level OpenMI components (Castronova and Goodall, 2010). The calculations were verified by comparison with an industry-standard hydrologic engineering model, the Hydrologic Engineering Center (HEC) Hydrologic Modeling System (HMS).

In the related work, each of the three components were all implemented as local resources and not as web services. To test the concepts presented in this paper, we re-implemented the Muskingum component as a web service using the proposed interface standard presented in Section 3.1. We then wrote an OpenMI component to act as a client for the web service and handle data transfers to and from the Muskingum web service. The end result of this re-implemented model was that it performed the same simulation as in our previous model where all components were local resources, despite the fact that one component of the overall model, the Muskingum Routing method, was performed by a web service. The difference is largely hidden from the end user when viewed through the OpenMI Configuration Editor GUI (Figure 5), which was done by design to ease the process of switching model components between local and remote services depending on the needs of the modeler.

The interaction between the OpenMI client component and web service is described in Figure 6. The Initialization method reads in a data structure representing the river network passed from the client OpenMI component to the web service and encoded as an XML string. The PerformTimeStep method takes as input the runoff values for the river network nodes and performs the routing calculation on those runoff values to estimate streamflow along all edges in the river network. The method then returns these streamflow values to the client. The web service maintains state (meaning the attributes of the river network) until the user makes a Finalize call on the web service. The finalize method clears the river network object from the server's memory and frees the service for use by another client application.

The Muskingum service was implemented using the Python programming language to demonstrate the potential of web services for providing programming language and cross-platform interoperability. It leveraged open source libraries for numerical (Oliphant, 2006) and graph algorithms (Hagberg et al., 2008). The prototype used a low-level web service communication protocol, XML-RPC, although SOAP/WSDL and REST standards would also be possible for a production implementation. We choose to use this lower level protocol for ease of implementation in the prototype and for generality because there are competing web service protocols one may wish to implement.

Although this model could follow what we earlier described as a component-level time looping structure, the web service was actually programmed to allow for the more complicated class of workflows where time looping is handled at the workflow level. This was done because the OpenMI 1.4 standard requires time looping to be handled external to each model component. Model components are written to respond to GetValues request where it is most often the case that a GetValues request is for a specific instant in time. Thus, the PerformTimeStep method of the Muskingum web service is called for each time step in the model and not just once at the beginning of the model execution, therefore making the example more similar to the workflow-level time looping structure discussed in the previous section.

Lastly it should be noted that for this prototype implementation, the service was implemented as a single threaded application, meaning that it can serve only one client application at a time. Thus an Initialize call locks the service for use by other client applications until a Finalize method is called. In a production environment, a multithreaded service could be implemented using tokens or some other means for uniquely identifying client applications and creating session states to support multiple simultaneous users.

5. Discussion

The motivation for this work was to design a web service standard appropriate for water resource modeling. Such a standard does not currently exist, yet is needed to achieve an end-to-end cyberinfrastructure for modeling water resource systems. We explored existing

interface and data exchange standards and leveraged these existing standards in our proposed web service interface design. While a complete solution to the challenges of water resource modeling within a service-oriented architecture is beyond the scope of a single paper, our intention is to begin a dialog on service-oriented water resource modeling in order to address key research challenges for applying the technology to the water resources domain. A large part of this effort is identifying both benefits and challenges of service-orientation. We have alluded to such benefits and challenges throughout this paper, but present a more complete discussion in this section.

A key advantage of service-oriented computing is avoiding code duplication (Papazoglou and Georgakopoulos, 2003). This is accomplished by creating targeted services with specific objectives and boundaries that are reusable across applications. Because applications often rely on the same set of underlying services, and because services are loosely-coupled with applications in a service-oriented architecture, it is possible for application developers to reuse the same underlying services within multiple applications (Huhns and Singh, 2005). Another important benefit of service-oriented computing is that the technology is built to provide an open architecture where new services can be easily created and then consumed within client applications. Services created by one user group become available for use by any other group connected through the Internet, although it is of course possible to restrict use to only authorized clients. A related benefit is that, because all communication between services and clients is through a defined interface, there is platform independence between clients and servers. This means a service could be implemented using Fortran on a Linux operating system, yet that service could still be consumed by a Windows client programmed using Visual Basic .Net.

If services adopt industry-standard specifications like WSDL for defining a service interface, then it becomes possible to use software tools to automatically generate an Application Programming Interface (API) for communicating with the service. Many software languages, including mathematical and statistical languages, include tools for automatically generating an API from a WSDL document. Because of these tools, the model developer using the web service API is hidden from details of the messages passed between the modeling

system and the service. The service execution remains external to the modeling system, therefore making for a loosely-coupled integration of the modeling system and service. This means that instead of system components acting as standalone applications communicating with one another through standardized file formats, as is the typical approach for loosely integrating water resource models, services instead become dependent libraries embedded within a modeling system application.

Service-oriented modeling allows for a hierarchical representation of complex water resource systems. A service can be a conglomeration of other services, allowing for different levels of system abstraction. In some cases modeling water resource systems requires a coarse view of the natural world (e.g. decomposing a system into atmosphere, land surface, subsurface, and other model services), while other cases may require a more detailed view into individual process-level components (e.g. infiltration, evaporation, overland flow, and other process-level services). Coarse decompositions minimize the complexity in setting up a system representation as a workflow by specifying how data is exchanged between system components. Because tight-coupling approaches are generally more computationally efficient than loose-coupling approaches, coarse decompositions also improve computational performance (Pingali and Stodghill, 2006). Refined decompositions allow for increased flexibility because smaller units within the modeling system can be more easily added or removed. Ideally systems would be represented with a hierarchical structure where coarse services are themselves workflows consisting of more refined services. This would allow the modeler to select the most appropriate level of system abstraction for a particular application.

Service-oriented computing also enables the use of high performance computing (HPC) through computational architectures like the Grid (Foster et al., 2001). The Grid provides important extensions for service-oriented modeling including state and fault handling, authentication, and resource management. Even without Grid computing, services can aid in improving model performance and efficiency. For example, web services could be used to expose a model with large data input requirements but relatively small output datasets. A spatially distributed watershed model, for example, might be exposed as a web service keeping the terrain, soil, and other parameterizations of the model on the server side and

only transmitting soil moisture outputs to client applications. In such a case, the model would be stored geographically near the large input datasets, saving the end user that is only interested in soil moisture conditions from having to download and process these input data required for running the model locally.

The primary disadvantages of a web services approach for environmental modeling are related to the loosely-coupled, web-based architecture of services that can result in performance, reliability, and security issues. Performance challenges in using web services for water resource modeling primarily relate to tightly-coupled process interactions that may require large data transfers, particularly when initializing and parameterizing the modeling domain, or from computation tasks with long compute times. In addition to performance issues, reliability of services can also be a disadvantage of service-oriented modeling approaches. There is the possibility of remote servers becoming temporarily unavailable, thus breaking all client applications dependent on that service. Finally security must be considered to prohibit unauthorized users and track overuse and abuse of services. Many of these issues are addressed by existing technologies such as Grid infrastructures (Foster et al., 2001) that can be used to enhance the applicability of service-oriented architectures for modeling.

In many but certainly not all modeling scenarios, these disadvantages of service-oriented approaches can be minimized through thoughtful system design (Pingali and Stodghill, 2006). When designing services, it is important to consider the response time and size of messages passed over the Internet. Model processes that require numerous communications with data transfers during runtime should be tightly-coupled within a single service to avoid network latency. Intelligent caching of data can also be used to minimize data transfers of repetitive information. For example, if a large dataset is required to initialize a modeling domain within a service, the data can be maintained within the service's state so that it does not need to be passed to the service with each service method call. Services can be implemented with complex back-end hardware architectures ranging from single machine servers to clusters. Thus the compute time and uptime reliability for services can be addressed with the proper infrastructure investment. While this may not be possible for prototyping environments, it can and should be implemented within production environments.

Perhaps the most important and challenging design decision for building a service-oriented water resource modeling system is deciding how to decompose a system into a set of representative services. This is the "granularity problem" associated with deciding the scope for each service within workflow configurations. As services become smaller in scope, for example modeling an individual process such as infiltration or evapotranspiration, the flexibility with which one can reuse and reconfigure services within workflow configurations increases. However, fine granularity also increases the frequency of communication required between services and clients. These communications could become a bottleneck within the application and make its use for real-time operations infeasible. However, as services increase in scope, the scientist or engineer using the services within workflow configurations will have less control over the individual processes used within the workflow. Additional research is needed to quantify performance costs associated with modeling fully-coupled processes with large data transfers within a loosely coupled service-oriented architecture. If the cost is prohibitive, then ideally services would be created across granular sizes (from individual process to logical groupings of processes) and the modeler would be able to decide the exact service needed for a specific application.

6. Summary

The basic design for a modeling service presented in this paper highlights the existing and new standards needed for implementing the service. The model exchange language and web service interface definition presented here combine elements from existing standards into a new protocol appropriate for exposing water resource models as services. No current modeling web service standard exists, and this work is meant to facilitate a discussion of the important aspects for such a web service modeling standard. As stated before, modeling of complex water resource systems depends on integration of multidisciplinary models, and integration depends on standards. Our hope is that the work presented here informs standards organizations like the Open Geospatial Consortium (OGC) and the Open Modeling Interface (OpenMI) Association on a potential design of a web service standard for water resource system modeling.

The simple implementation of the service standard for a process-level hydrologic model demonstrates the advantages of the approach for building water resource modeling systems. Services provide an open platform for exposing models because they allow models to be authored in most modern programming languages and need not be used by the same programming language and operating system in which the service was authored. They are flexible because they are loosely-coupled, allowing modelers to design workflow orchestrations that reconfigure services to achieve different end goals. Finally services are also extensible because anyone can author new services, host them on their own server environments, and allow anyone they wish to use the service for their own purposes. To fully capitalize on these core properties of service-orientation, however, basic standards must be agreed upon for exposing models as web services (Foster, 2005).

Thoughtful design is required to minimize potential performance, reliability, and security issues associated with service-oriented computing (Pingali and Stodghill, 2006). Because services follow a loosely-coupled integration approach, it is unlikely that a service-oriented paradigm will deliver maximum performance in terms of computational time for complex models. However, well thought out model and service designs could lessen these disadvantages and make service-oriented computing attractive for a wide variety of applications. An obvious application is for models that require a large amount of data for parameterization but produce a relatively small amount of data as output. By co-locating such a model along with its input database and then exposing the model as a web service, multiple end users can leverage the same model within workflow orchestrations without having to each parameterize and manage the model input data. They would be able to set certain boundary conditions for the model, run the model, and receive model outputs by using a web service.

Finally, we have provided here only a simple implementation of this approach in our prototype section to serve as a proof of concept for service-oriented hydrologic modeling. More work is needed to fully test the impact of model performance as the data transfers between services increase in both size and frequency. Furthermore, we have only briefly described the semantic issues associated with service-oriented modeling in this paper. To reach the true potential of service-oriented computing, services are required across disciplinary boundaries,

and so ontologies are needed to map between domain specific semantics. This work is being addressed for data integration efforts in the earth sciences, and such efforts will largely be applicable to model integration as well.

Acknowledgments

This work was funded in part by the National Science Foundation under project number EAR-0622374 and by a CSIRO Chief Executive's Study Award.

References

Allan, B. A., Armstrong, R., Bernholdt, D. E., Bertrand, F., Chiu, K., Dahlgren, T. L., Damevski, K., Elwasif, W. R., Epperly, T. G. W., Govindaraju, M., Katz, D. S., Kohl, J. A., Krishnan, M., Kumfert, G., Larson, J. W., Lefantzi, S., Lewis, M. J., Malony, A. D., McInnes, L. C., Nieplocha, J., Norris, B., Parker, S. G., Ray, J., Shende, S., Windus, T. L., Zhou, S. J., 2006. A component architecture for high-performance scientific computing. International Journal of High Performance Computing Applications 20 (2), 163–202.

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

Band, L., Patterson, P., Nemani, R., Running, S., 1993. Forest ecosystem processes at the watershed scale: incorporating hillslope hydrology. Agricultural and Forest Meteorology 63 (1-2), 93–126.

Beran, B., Piasecki, M., 2009. Engineering new paths to water data. Computers & Geosciences 35 (4, Sp. Iss. SI), 753–760.

Castronova, A. M., Goodall, J. L., 2008. Design and implementation of a simple model interface for component-based modeling. In: Eos Trans. AGU. Vol. Abstract H21G-0961.

Castronova, A. M., Goodall, J. L., 2010. A generic approach for developing process-level hydrologic modeling components. Environ. Model. Softw. 25 (7), 819–825.

Chow, V. T., Maidment, D. R., Mays, L. W., 1988. Applied Hydrology. McGraw-Hill, New York.

Clark, J., Gelfand, A., 2006. A future for models and data in environmental science. Trends in Ecology & Evolution 21 (7), 375–380.

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., Weerawarana, S., Apr. 2002. Unraveling the Web services web: An introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing 6 (2), 86–93.

del Barrio, G., Harrison, P., Berry, P., Butt, N., Sanjuan, M., Pearson, R., Dawson, T., 2006. Integrating multiple modelling approaches to predict the potential impacts of climate change on species distributions

in contrasting regions: comparison and implications for policy. Environmental Science & Policy 9 (2), 129–147.

Eaton, B., Gregory, J., Drach, B., Taylor, K., Hankin, S., Caron, J., Signell, R., Bentley, P., Rappa, G., 2009. NetCDF Climate and Forecast (CF) Metadata Conventions. Version 1.4, 27 February, 2009. URL `http://cf-pcmdi.llnl.gov/documents/cf-conventions/1.4`

Engelen, G., White, R., Uljee, I., Drazan, P., 1995. Using cellular automata for integrated modelling of socio-environmental systems. Environmental Monitoring and Assessment 34 (2), 203–214.

Facchi, A., Ortuani, B., Maggi, D., Gandolfi, C., 2004. Coupled SVATgroundwater model for water resources simulation in irrigated alluvial plains. Environmental Modelling & Software 19 (11), 1053–1063.

Foster, I., 2005. Service-oriented science. Science 308 (5723), 814–817.

Foster, I., Kesselman, C., Tuecke, S., 2001. The anatomy of the grid: Enabling scalable virtual organizations. International Journal of High Performance Computing Applications 15 (3), 200–222.

Goodall, J., Horsburgh, J., Whiteaker, T., Maidment, D., Zaslavsky, I., 2008. A first approach to web services for the National Water Information System. Environmental Modelling & Software 23 (4), 404–411.

Granell, C., Daz, L., Gould, M., 2010. Service-oriented applications for environmental models: Reusable geospatial services. Environmental Modelling & Software 25 (2), 182–198.

Gregersen, J. B., Gijsbers, P. J. A., Westen, S. J. P., 2007. OpenMI: Open Modelling Interface. Journal of Hydroinformatics 9 (3), 175.

Hagberg, A. A., Schult, D. A., Swart, P. J., Aug. 2008. Exploring network structure, dynamics, and function using NetworkX. In: Proceedings of the 7th Python in Science Conference (SciPy2008). Gel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), Pasadena, CA USA, pp. 11–15.

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

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

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

Jakeman, A., Letcher, R., 2003. Integrated assessment and modelling: features, principles and examples for catchment management. Environmental Modelling & Software 18 (6), 491–501.

Jennings, N. R., 2001. An agent-based approach for building complex software systems. Communications of the ACM 44 (4), 35–41.

Leavesley, G. H., Markstrom, S. L., Brewer, M. S., Viger, R. J., 1996. The Modular Modeling System (MMS): The physical process modeling component of a database-centered decision support system for

water and power management. Water, Air, & Soil Pollution 90 (1-2), 303–311.

Ludscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E. A., Tao, J., Zhao, Y., 2006. Scientific workflow management and the kepler system. Concurrency and Computation: Practice and Experience 18 (10), 1039–1065.

Maidment, D. R., 2008. Bringing water data together. Journal of Water Resources Planning and Management 134 (2), 95.

Maxwell, R. M., Miller, N. L., 2005. Development of a coupled land surface and groundwater model. Journal of Hydrometeorology 6 (3), 233.

Michaelis, C. D., Ames, D. P., 2008. Evaluation and implementation of the OGC Web Processing Service for use in Client-Side GIS. GeoInformatica 13 (1), 109–120.

Mineter, M., Jarvis, C., Dowers, S., 2003. From stand-alone programs towards grid-aware services and components: A case study in agricultural modelling with interpolated climate data. Environmental Modelling & Software 18 (4), 379–391.

Moore, R. V., Tindall, C., 2005. An overview of the open modelling interface and environment (the OpenMI). Environmental Science & Policy 8 (3), 279–286.

Oliphant, T. E., 2006. Guide to NumPy. Trelgol Publishing, USA.
URL `http://numpy.scipy.org`

Open Geospatial Consortium, 2007. OpenGIS®Web Processing Service. OGC 05-007r7, Version 1.0.0.

Papazoglou, M. P., Georgakopoulos, D., 2003. Service-oriented computing: Introduction. Communications of the ACM 46 (10), 24.

Peckham, S., 2008. CSDMS handbook of concepts and protocols: A guide for code contributors. `http://csdms.colorado.edu/wiki/Tools/_CSDMS/_Handbook`.

Peltz, C., 2003. Web services orchestration and choreography. Computer 36 (10), 46–52.

Pingali, K., Stodghill, P., 2006. A distributed system based on web services for computational science simulations. In: Proceedings of the 20th Annual International Conference on Supercomputing - ICS '06. Cairns, Queensland, Australia, p. 297.

Qu, Y., Duffy, C. J., 2007. A semidiscrete finite volume formulation for multiprocess watershed simulation. Water Resources Research 43 (8).

Syvitski, J., Paola, R., Slingerland, R., Furbish, D., Wiberg, P., Tucker, G., Apr. 2004. Building a community surface dynamics modeling system: Rational and strategy. A Report to the National Science Foundation, INSTAAR, University Colorado, Boulder.

Turuncoglu, U., Murphy, S., DeLuca, C., 2009. Towards CCSM Self-Describing workflows. poster presented at the 2009 CCSM Workshop.

Čurn, J., 2007. Distribution for Open Modelling Interface and Environment. Master's thesis, Charles Uni-

versity, Prague.

Yu, J., Buyya, R., 2005. A taxonomy of scientific workflow systems for grid computing. ACM SIGMOD Record 34 (3), 44.

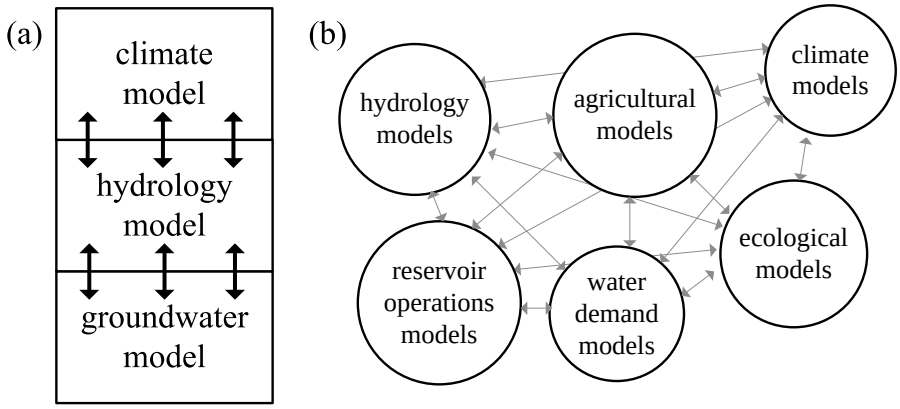

Figure 1: Contrasting a (a) tight-coupling and (b) loose-coupling approach for model integration. In a tight coupling approach, models must adopt common internal data structures. In a loose coupling approach, model interfaces are standardized but internal implementation is not.

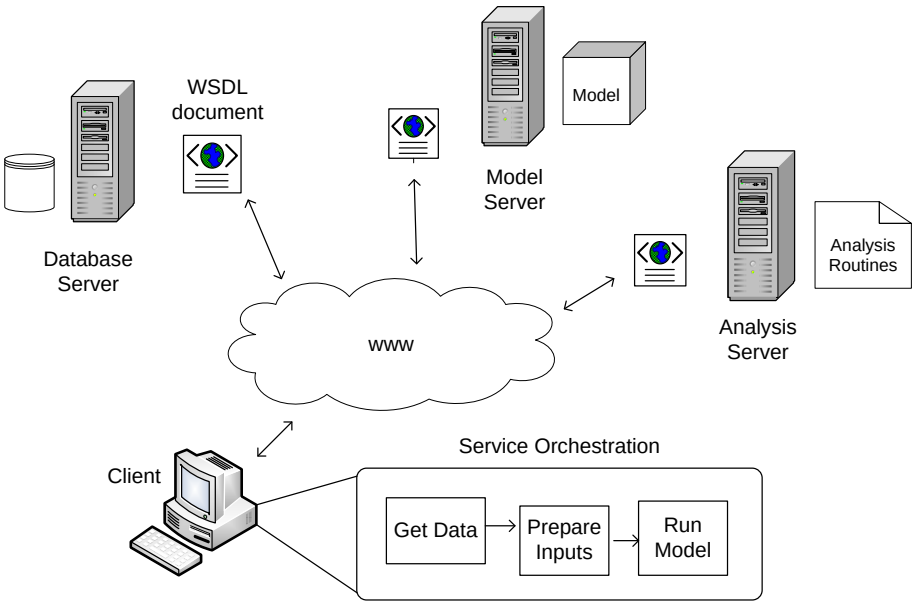

Figure 2: A vision for service-oriented modeling where data, models, and analysis routines are exposed as web services and integrated into workflow orchestrations designed to address specific scientific or management questions

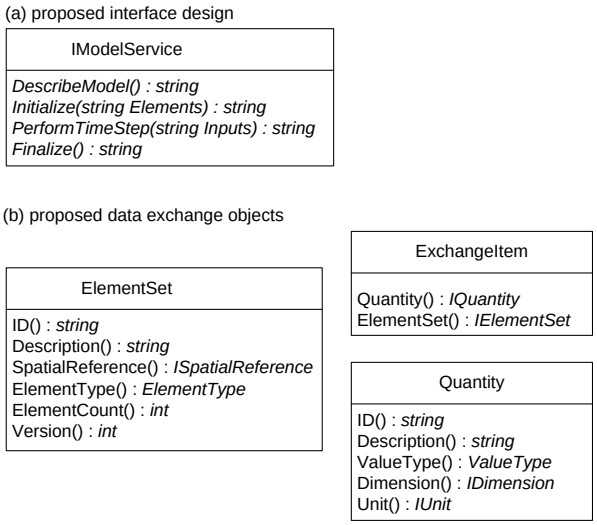

Figure 3: Proposed modeling service design including (a) interface and (b) data exchange specifications

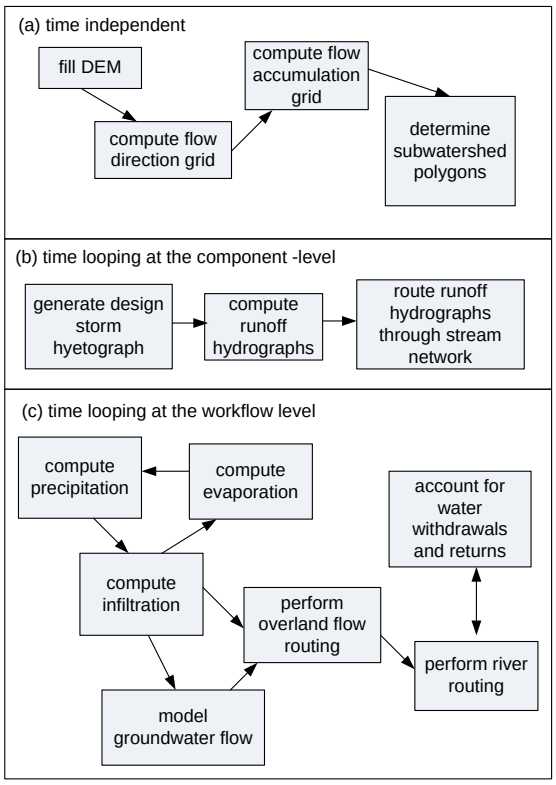

Figure 4: Three classes of workflow orchestrations relevant for modeling water resource systems

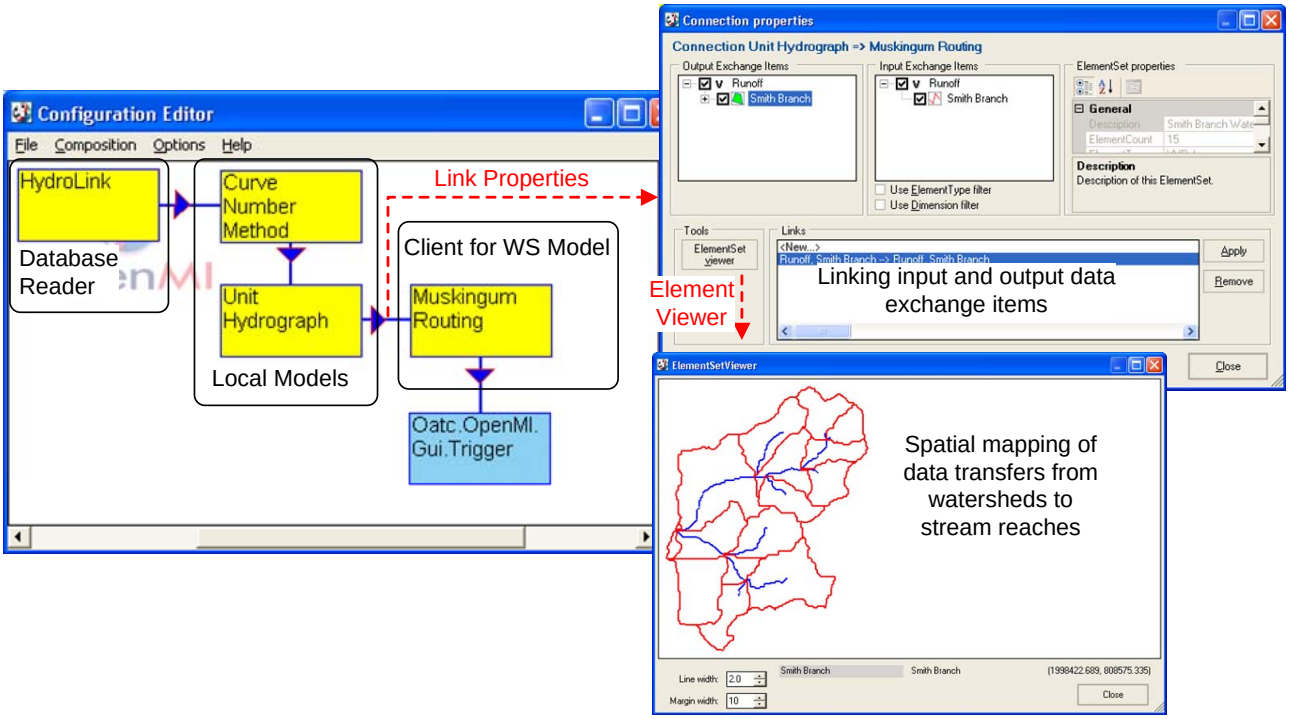

Figure 5: Prototype Implementation of a service workflow for modeling a rainfall/runoff event using the Open Modeling Interface (OpenMI) Configuration Editor

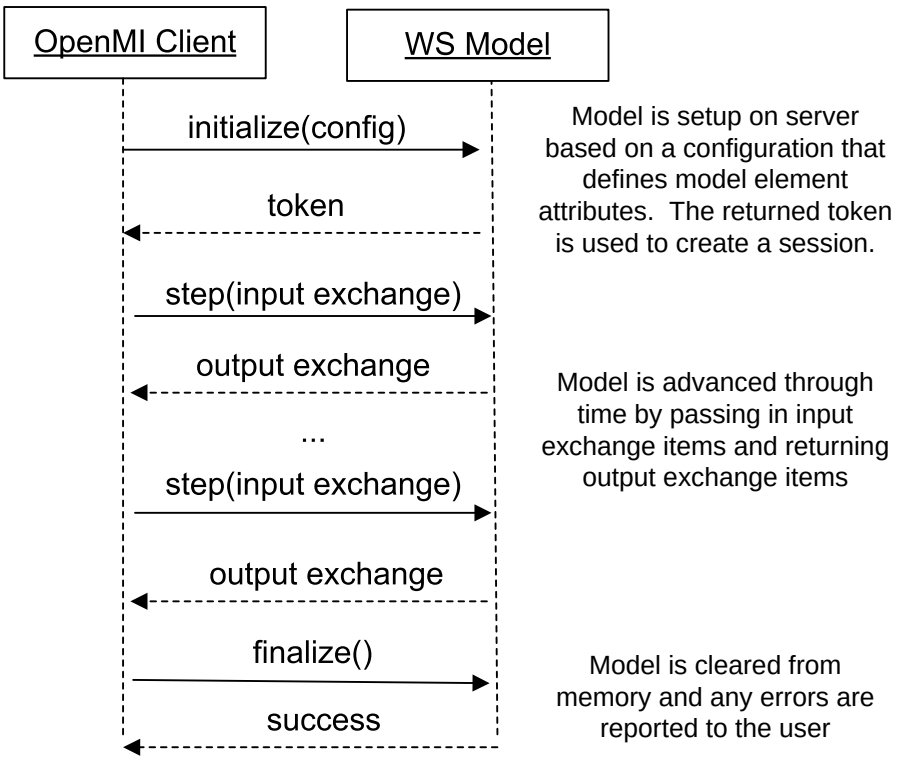

Figure 6: Communication between OpenMI client component and Muskingum web service