Environmental Modelling & Software 92 (2017) 107–118

Contents lists available at ScienceDirect

Environmental Modelling & Software

journal homepage: www.elsevier.com/locate/envsoft

A service-oriented architecture for coupling web service models using the Basic Model Interface (BMI)

Peishi Jiang [a], Mostafa Elag [a], Praveen Kumar [a, *], Scott Dale Peckham [c], Luigi Marini [b], Liu Rui [b]

[a] Ven Te Chow Hydrosystem Laboratory, Civil and Environmental Engineering, University of Illinois, Urbana, IL 61801, USA
[b] National Center for Supercomputing Applications, University of Illinois, Urbana, IL 61801, USA
[c] CSDMS, University of Colorado, Boulder, CO 80303, USA

ARTICLE INFO

Article history:
Received 31 December 2016
Accepted 27 January 2017
Available online 28 February 2017

Keywords:
Integrated modeling
Service-oriented architecture
The Basic Model Interface
EMELI-Web

ABSTRACT

Service-oriented approach for model coupling is gradually gaining momentum. By leveraging the World Wide Web, the service-oriented approach lowers the interoperability barrier of integrating models in terms of programming languague and operating system. While such paradigm has been applied to integrate models wrapped with some standard interfaces, this paper considers the Basic Model Interface (BMI) as model interface. The advantages of BMI are that it (1) enrich the semantics of variable names, and (2) is framework-agnostic. We exposed the BMI-enabled models through web services. Then, a smart modeling framework, the Experimental Modeling Environment for Linking and Interoperability (EMELI), was enhanced into a web application (i.e., EMELI-Web) to integrate the BMI-enabled web service models. By implementing the whole orchestration in coupling TopoFlow components, we demonstrate that BMI helps connect web service models by reducing the heterogeneity of variable names, and EMELI-Web makes it convenient to couple BMI-enabled web service models.

Software availability

- The code for the BMI-enabled web service TopoFlow components is available at: https://opensource.ncsa.illinois.edu/bitbucket/projects/ECGS/repos/bmi-flask/browse
- The code for EMELI-Web is available at: https://opensource.ncsa.illinois.edu/bitbucket/projects/ECGS/repos/emeli-web-application/browse

1. Introduction

There is an increasing need of integration and re-use of models from different disciplines in the geoscience community to simulate and model complex environmental systems. Due to different scientific conventions and vocabulary usage in different disciplines, the associated numerical and physical models usually differ in programming languages, variable names, variable units, and spatial and temporal grids for solution, causing the difficulties of model integration(Argent, 2004). To address the issues of coupling multidisciplinary heterogeneous models, a lot of solutions have been put forward during the past decade (Sui and Maggio, 1999; Hill et al., 2004; Syvitski et al., 2004; Maxwell and Miller, 2005; Moore and Tindall, 2005; David et al., 2013). Among these solutions, a loosely-coupled, service-oriented approach is gaining momentum recently due to its ability of leveraging the World Wide Web for integrated modeling (Geller and Melton, 2008; Goodall et al., 2011; Laniak et al., 2013; Nativi et al., 2013). However, most applications of such service-oriented architecture (SOA) are limited to models wrapped with the model interfaces which (1) does not reduce the heterogeneity of variable names, and (2) is used in a specific model integration framework thus causing the difficulties of being applied in other frameworks (Goodall et al., 2011, 2013; Castronova et al., 2013). Therefore, the goal of this study is to develop a service-oriented modeling framework for coupling models by adopting the Basic Model Interface (BMI, Peckham et al. (2013)) which not only enriches the semantic information of variable names but also is framework-independent.

Prior to the emergence of the loosely-coupled, service-oriented approach, researchers have proposed multiple solutions to

* Corresponding author.
E-mail address: kumar1@illinois.edu (P. Kumar).

http://dx.doi.org/10.1016/j.envsoft.2017.01.021
1364-8152/

integrate heterogeneous models. Generally, there are two types of methods: tightly-coupled and loosely-coupled, integration approaches. The tight coupling approach has been adopted by a number of researchers by porting codes from different models into a single modeling application (Sui and Maggio, 1999; Facchi et al., 2004; Maxwell and Miller, 2005; Yu et al., 2006). Despite its capability in fully controlling the modeling process, the tightly-coupled approach requires the consistent internal conventions within the models (e.g., data structures). In contrast, following a loosely-coupled approach, researchers only need to standardize the model interface and integrate models within a specific modeling framework, thus allowing the internal structure of the model to be unchanged. Examples of such modeling frameworks include the Earth System Modleing Framework (ESMF, Hill et al. (2004)), the Open Modeling Interface (OpenMI, Moore and Tindall (2005)), the Object Modeling System (OMS, David et al. (2013)), and the Community Surface Dynamics Modeling System (CSDMS, Peckham et al. (2013)).

Compared with the traditional loose coupling approach (for example, the component-based modeling approach where the models are integrated in one computing platform in a plug-and-play manner (Van Ittersum et al., 2008; Elag et al., 2011; Peckham et al., 2013; Theurich et al., 2015)), the application of loosely-coupled service-oriented method in model integration has the following advantages (Goodall et al., 2011). First, it allows the independence of operating system and programming languages for models, lowering the interoperability barrier of model integration. Second, by exposing models through web service, one can utilize the functionality of the model without installing it in his or her own computing system. Third, the web service feature allows the maintenance or update of a model while still providing the original functionality to the clients over the web. In geoscience domain, the idea of adopting SOA in modeling is also termed as “Model Web” (Geller and Melton, 2008) and has been applied in modeling water resource systems (Goodall et al., 2011; Castronova et al., 2013).

Despite the existing efforts in applying SOA in model integration, the model interfaces used do not provide sufficient semantic information of variable names and are heavily dependent on model integration framework and model interface standard. For instance, hydrologic models encapsulated with OpenMI are transformed into web services and loosely coupled in a service-oriented architecture (Goodall et al., 2011; Castronova et al., 2013). However, OpenMI standard fails to address the heterogeneity issue of variable names. Furthermore, it is usually difficult to integrate models standardized in different modeling frameworks (e.g., ESMF and OMS), therefore limiting a wider application of these modeling frameworks and standards.

Hence, in this study, we adopt the Basic Model Interface (BMI) as the model interface, which is originally developed in the Community Surface Dynamics Modeling System (CSDMS) (Peckham et al., 2013). There are two unique features of BMI. First, BMI is able to map a model's internal variable names to CSDMS standard names, which is a set of “cross-domain naming conventions for describing process models, data sets, and their associated variables” (Peckham, 2014a), so that models adopting different variable naming convention can still be properly connected before being further coupled. The second feature of BMI is that it is framework-agnostic. Namely, there is no need for researchers to adjust their models to a specific modeling framework (Peckham, 2014b). It also suggests that a BMI-enabled model can be used in any other framework once a corresponding adapter is developed. This is reflected in an ongoing EarthCube project, Earth System Bridge whose goal is to allow different modeling frameworks to be “interoperate” by using BMI as the “bridge” connecting different frameworks (Peckham et al., 2014). Another example of utilizing BMI's framework-agnostic property is the development of Experimental Modeling Environment for Linking and Interoperability (EMELI), a smart modeling framework for integrating BMI-enabled models (Peckham, 2014b).

In this research, we develop a service-oriented modeling framework by enhancing EMELI to integrate web service models using BMI as depicted in Fig. 1. To achieve the BMI-based service-oriented framework, the following technical challenges are addressed: (1) how to convert the BMI-enabled models into web service models (i.e., constructing the web service exposing the functionality of BMI); (2) how to advance the EMELI framework to EMELI-Web which enables the “integration” of the BMI-enabled web serviced models. Also, a web interface is established for EMELI-Web, which allows the convenient usage of EMELI-Web through the browser. The entire architecture is then tested by (1) transforming components of a set of spatially-distributed hydrologic model TopoFlow (Peckham, 2009) into BMI-enabled web service models and (2) executing the models in Owl watershed of the Upper Sangamon River Basin in Illinois through EMELI-Web.

In the remainder of this paper, the concept of SOA and its application is reviewed in Section 2. Section 3 details the design of the BMI-based service-oriented modeling paradigm, including the conversion of BMI-enabled models into web services and the construction of EMELI-Web. An implementation is then performed by converting TopoFlow components into the BMI-enabled web service models in Section 4. A short discussion of the whole architecture is provided in Section 5, and the paper is briefly concluded in Section 6.

2. Background

Service-Oriented Architecture (SOA) is a way of using web service to model a large software system, where sub-softwares or computing components are distributed on different remote servers that provide services for other clients (Erl, 2004; Huhns and Singh, 2005). A typical communication between a service and a client (which can be either a human being or another service) is as follows. Following a specific communication protocol, a client sends a request to a web service operated on another server via the internet. After receiving the request, the service reads the incoming information, carries out a certain processing, and sends a response back to the client. Such communication in SOA implies a loosely-coupled architecture where the software or computing component behind the web service can be conserved in any hardware environment and run in any programming language.

Until recently, SOA has been adopted as an alternative in model integration for component-based modeling approach, which has been widely applied in coupling heterogeneous models in a ‘plug-and-play’ manner (Geller and Melton, 2008). Compared with the component-based approach, which usually has to reply on a specific modeling framework in a single computing resource, the service-oriented modeling paradigm allows the independence of both operating system and programming language as well as the avoidance of code duplication (Nativi et al., 2013). By exposing a model as web services, users can utilize the functionality of the models directly by calling the service, without considering the dependent platform and programming language. Also, because of the ability to run a model via the Internet, there is no need to duplicate the code in the users' own computer. There have been some efforts to apply SOA for coupling models in geoscience. For example, Goodall et al. (2011) transformed water resource models into web services by using the Open Geospatial Consortium (OGC) Web Processing Service (WPS) and demonstrated how it can be encapsulated as OpenMI-compliant models. Later, Castronova et al.

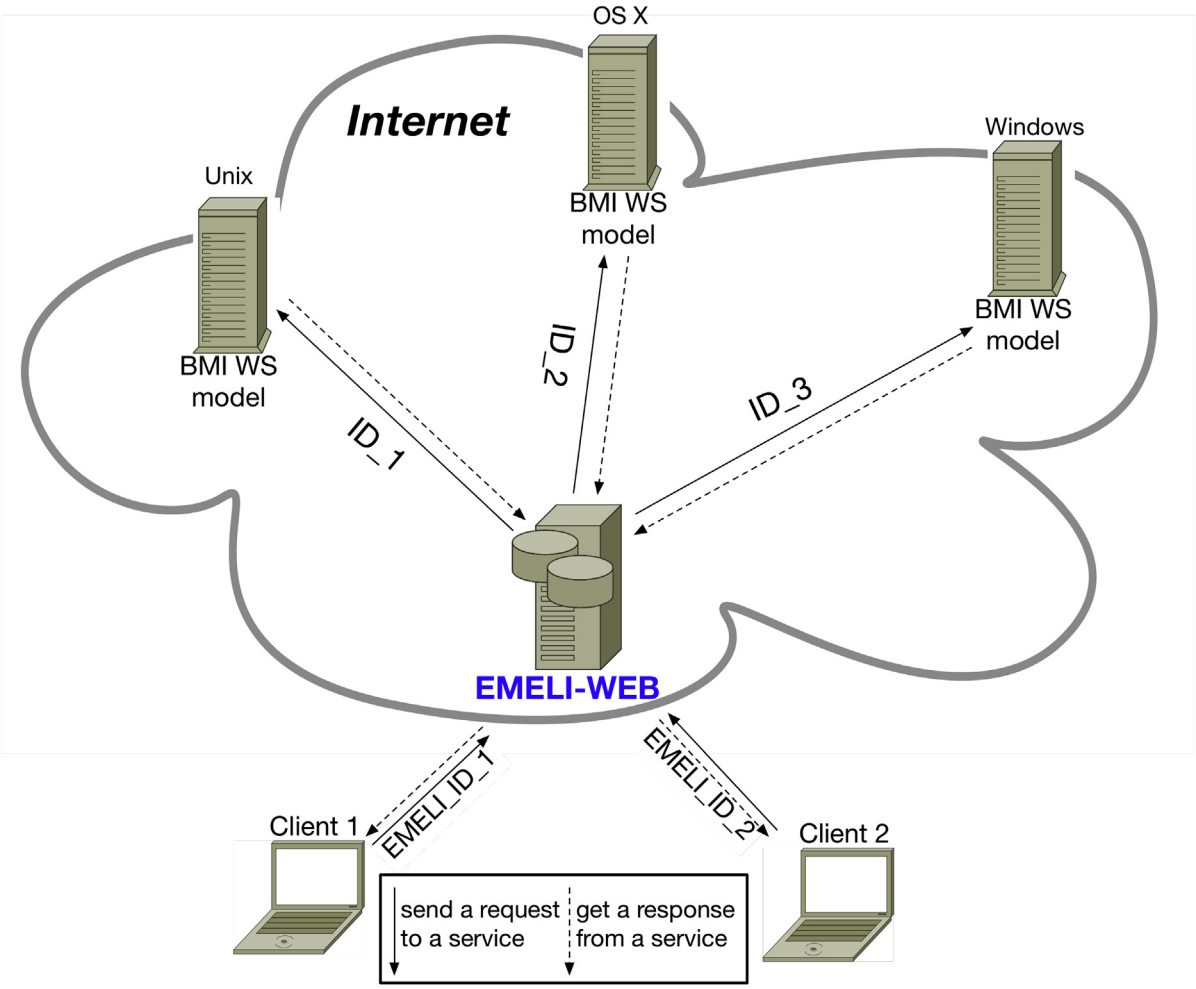

Fig. 1. The architecture of integrating BMI-enabled web service models through EMELI-Web.

(2013) furthered the idea of servicing an OpenMI-compliant model by considering the case of time-dependent models. In addition, an OpenMI-ESMF web service wrapper was developed to couple a climate model implemented via ESMF web service with an OpenMI-compliant hydrologic model running on a personal computer (Goodall et al., 2013). Exposing models through web services would significantly lower the interoperability barrier of the communication between models, and a client only needs to configure the communication protocol used by the modeling service.

Two common web service communication methods are: Simple Object Access Protocol (SOAP, SOAP (2004)) and REpresentational State Transfer (REST, Fielding (2000)) specifications. SOAP has been widely applied for setting up web services, usually combined with the Web Service Description Language (WSDL). However, a web service adopting SOAP/WSDL protocol is complex in that SOAP is designed for structured information and thus requires the incoming data to follow a sophisticated prototype (Mulligan and Gračanin, 2009). Meanwhile, constructing a RESTful application is relatively simple because it only relies on a stateless communication protocol, most commonly Hypertext Transfer Protocol (HTTP). By using a set of HTTP methods (GET, POST, PUT, DELETE, HEAD (Fielding et al., 1999)), a RESTful application is able to interact with the resources exposed by a web service.

In this study, we adopt the RESTful approach due to its simplicity and easy of use. Also, in terms of the specification encoding messages transferred between the RESTful web services, JavaScript Object Notation (JSON) is utilized due to its capability of transmitting messages in a lightweight data-interchange format.

3. Design

The design of our SOA includes (1) a modeling service for exposing BMI-enabled models as web service models, and (2) the EMELI-Web framework for coupling web service models, which is an enhanced version of EMELI to handle BMI-enabled web service model. As discussed in Section 1, BMI is chosen as the model interface because of its framework-agnostic property and capability of enriching the semantic information of variable names. Therefore, despite the adoption of BMI as the model interface, models encapsulated with other standards or in other frameworks can also be integrated in this SOA once a wrapper like the one being developed in Earth System Bridge is ready.

By exposing BMI-enabled models as web services, clients can get the information of a model and execute it by sending requests to the services. To couple the BMI-enabled web service models, EMELI-Web is equipped with a user interface and introduces a port to receive and ingest the response from the modeling service.

3.1. Exposing BMI-enabled models through web service

The interface of a web service model developed in this work is

based on a combination of the Basic Model Interface (BMI) and a set of JSON-based RESTful web service APIs (application programming interfaces) exposing information provided by BMI. As a model interface, BMI allows a model to self-describe in that the basic features of a model (e.g., input and output names) are able to be retrieved through a set of BMI functions (Peckham et al., 2013). In terms of web service interface, despite the wide applications of different web service specifications developed in the Open Geospatial Consortium (OGC), there is no specification for geoscience modeling. Even though a combination of several existing standards (e.g., the Web Processing Service, the Geography Markup Language and Water Markup Language) might provide a solution to address this issue, as suggested by Castronova et al. (2013), these XML-based standards would make the conveyed information heavily encapsulated, slowing down the entire service-based communication. Hence, in this study, we develop a set of JSON-based web service APIs for conveying information provided by BMI.

To convert a BMI-enabled model into a web service, we construct a wrapper by exposing most BMI functions through web service APIs. In addition, when an instance of a model resource is initialized in the server, its own ID and file system are generated as well. In order to reduce the web latency of service communication, variable values of a model are saved in a binary-formate netCDF (i.e., network Common Data Form) file and transferred over the web.

3.1.1. Web service design for BMI-enabled models

The design of BMI-enabled web service models is to allow the integration of the web service models under a EMELI-based web service framework in our study. To this end, the following efforts are made: (1) exposing almost all BMI functions into RESTful APIs; (2) designing a RESTful API by combining several BMI functions to reduce web latency; and (3) introducing utility RESTful APIs to smooth the SOA-based coupling in EMELI-Web. All the APIs designed for a BMI-enabled web service model, including BMI-based APIs and utility APIs, are listed in Table 1 and Table 2, respectively.

A BMI-enabled model not only is self-describing but also facilitates clients to have a full control of the model. The BMI functions are categorized into five groups: model control functions, model information functions, variable *getter* and *setter* functions, variable information functions, and grid information functions. The basic information of a model is available through performing model information functions, variable information functions and grid information functions. For example, by using model information functions, the input(s) and output(s) of a model can be retrieved in CSDMS standard names, which would help check the connections between models in a model coupling workflow. Also, the variables' values can be obtained and reset through variable *getter* and *setter* functions. Furthermore, model control functions enable clients to initialize the model based on a configuration file, update the model at each time step and finalize the model by releasing all computational resources. More detailed functionality of BMI are provided in Peckham et al. (2013).

For the conversion of a BMI-enabled model into web services, most BMI-based APIs are directly constructed upon the original BMI functions as shown in Table 1. This design allows a convenient revision of EMELI in that most BMI functions utilized by EMELI can then be easily revised to consume the responses from calling the corresponding BMI-based APIs. For example, in model control functions, the BMI capability of *initializing*, *updating* and *finalizing* a model is converted into three corresponding HTTP PUT methods. Similarly, in model information functions, three HTTP GET methods

Table 1
The BMI-based RESTful APIs of a BMI-enabled web service model where ⟨ model⟩ and ⟨ id⟩ represent the model name and the ID of the model instance, respectively.

Model Control Functions		
/instantiate	POST	Instantiates the model and return back an ID
/⟨ id⟩/initialize	PUT	Initializes the model after it is instantiated by sending the configuration file
/⟨ id⟩/update	PUT	Updates the model after it is initialized
/⟨ id⟩/finalize	PUT	Finalizes the model after the simulation is done
Model Information Functions		
/⟨ id⟩/get_input_var_names	GET	Returns the inputs of the model
/⟨ id⟩/get_output_var_names	GET	Returns the outputs of the model
/⟨ id⟩/get_attribute	GET	Returns the attributes of the model
Variable Information Functions		
/⟨ id⟩/get_time_step	GET	Returns the time step of the model after model initialization
/⟨ id⟩/get_time_units	GET	Returns the time units of the model after model initialization
/⟨ id⟩/get_time/⟨ when⟩	GET	Returns the time of the model after model initialization
Variable *Getter* and *Setter* Functions		
/⟨ id⟩/set_values_for_vars	PUT	Resets the values of some variables in the model by sending a netCDF file including the variable values after model initialization.
/⟨ id⟩/set_vars_provided_list	PUT	Informs the model of the variables which are used by clients after model initialization
Grid Information Functions		
/⟨ id⟩/get_grid_properties	GET	Returns the grid properties of the variable with name var after model initialization

Table 2
The utility RESTful APIs of a BMI-enabled web service model where ⟨ id⟩ represents the ID of the model instance.

/⟨ id⟩/send_cfg_sup_files	GET	Sends a configuration file along with any input files to the model after model instantiation
/⟨ id⟩/mode/update/⟨ status⟩	PUT	Updates the mode status of the model after model instantiation
/⟨ id⟩/output	GET	Lists the model's output files after model instantiation
/⟨ id⟩/download_output/⟨ file⟩	GET	Downloads the model's output files after model completion
/⟨ id⟩/download_temp_nc/⟨ file⟩	GET	Downloads the temporary netCDF files of the model when the model is initialized updated or finalized
/⟨ id⟩/remove	DELETE	Removes the model instance after the model is instantiated

are built up to retrieve the input and output names of the model as well as the model's attributes.

Moreover, to reduce web latency, some BMI-based APIs are established by including several BMI functions. For instance, for grid information function, one single API is provided to obtain variable's spatial information which is originally available through several BMI functions. In this way, the number of modeling services called decreases so that the web latency is diminished. Also, for BMI's variable *getter* and *setter* functions, instead of getting and setting values of one variable at each time, the web service design allows retrieving and changing values of multiple variables by saving the variables' values in a netCDF file, thereby reducing web service calling times. The details of using netCDF file to store variable values are further explained in Section 3.1.3.

Some utility RESTful APIs of a BMI-enabled web service model are designed for performing specific tasks under a service-oriented environment (as shown in Table 2). For example, once the model instance is finalized, the client can obtain the list of outputs through calling the API/⟨ id⟩/output, and download them by sending a request to the API/⟨ id⟩/download_output/⟨ file⟩ to download the output.

3.1.2. Identification of a web service model instance

To avoid the conflicting executions of a modeling resource by different clients, a unique ID is assigned to the model instance when the model is instantiated by calling the model instantiation API (i.e.,/⟨ id⟩/instantiate). Universally unique identifier (UUID), a widely adopted identifier standard in software construction, is employed as the mechanism for generating the ID (Leach et al., 2005). Once a model instance is created in the server, the assigned ID is required for further interactions with the specific instance of the modeling resource through the APIs shown in Tables 1 and 2.

Furthermore, a file system is needed for each model instance to store input and output files. During the model instantiation, three file folders (i.e., input folder, output folder and temp folder) are created for the specific model instance to store the input files, the output files and the files containing temporary variable values during the model simulation, respectively. Specifically, the temp folder contains files either created when the model is executed and stores the variable values, or uploaded from a client for resetting the variable values, in netCDF format. The file system belonging to a specific model instance is deleted once the model instance is removed by using the API/⟨ id⟩/remove.

3.1.3. Variable value transfer via the internet through netCDF files

To enable unambiguous, structured and efficient client-service communication, the variable values of a model are saved in a netCDF file at each time step. NetCDF is a "self-describing, machine-independent" and binary data format to store and access the array-oriented data, developed in Unidata program by the University Corporation for Atmospheric Research (UCAR) (Rew and Davis, 1990). It is also a standard for defining data in terms of variable name, variable unit and spatio-temporal property. As a consequence, adopting the binary-based netCDF files to store variable values is extremely helpful in both structuring data values and reducing information storage compared with the traditional XML-based data transfer format.

We use netCDF files to store variable data in the following scenarios. First, netCDF files are utilized to contain several variable values and returned back to the client when a model is either initialized, updated or finalized. The variables of a model required by the clients should be informed when the modeling service is initialized by sending a list of the required variables to the API/⟨ id⟩/set_ vars_provided_list. In a workflow, the required variables are usually parts of the model's BMI outputs which are utilized as the inputs of other models. Once the model instance is informed of what variables should be provided for the client, a netCDF file storing the values of these required variables is sent back to the client during each model execution (i.e., model initialization, update and finalization). The second case of using netCDF files is for resetting values of multiple variables in a model by posting a netCDF file to the API/⟨ id⟩/set_values_for_vars. This is useful in a scenario when the inputs of the model need to be updated based on the outputs of other models before the model is further executed.

3.2. Service orchestration based on EMELI-Web

As illustrated in Fig. 1, EMELI-Web, a web-based implementation of EMELI, is responsible for creating the workflow of multiple web service models, orchestrating the execution of web service models running at different time steps, and coordinating the information (e.g., netCDF data files) flowing between EMELI-Web and modeling services. To enable EMELI to integrate BMI-enabled web service models, EMELI is first enhanced by introducing a port to call the modeling service and receive its response. Then, a web interface is constructed for users to conveniently utilize EMELI-Web through web browsers.

3.2.1. EMELI-web design for integrating BMI-enabled web service models

We choose EMELI as the basis for integrating BMI-enabled web service models. EMELI is a Python class whose goal is to conveniently integrate component models wrapped with BMI into a composite model (Peckham, 2014b). EMELI creates a runtime environment for the BMI-enabled models, and it couples the models by sequentially going through model instantiation, initialization, update and finalization stages. For details of how EMELI works, please see the Peckham's description of EMELI (Peckham, 2014b).

The port, a Python class, is developed for converting the JSON-based response from the modeling service into the data structure that EMELI can consume. Based on the combination of EMELI and the port for calling modeling service, EMELI-Web is set up to interact with a BMI-enabled web service model, as illustrated in Fig. 2.

- **Model preparation**: After BMI-enabled models are exposed through web service, their URL namespaces are registered in an XML file (i.e., component_repository.xml). The registration file is used later for calling the web service model in EMELI-Web.
- **Model instantiation**: A new instance of EMELI-Web is created. The instance first obtains the available web service models by reading the model registration file (i.e., component_repository.xml). Then, the EMELI-Web instance reads a text file (named provider_file in Fig. 2), which lists the models to be coupled in the execution order, and instantiates the selected web service models.
- **Model initialization**: In this stage, the EMELI-Web instance checks the connections between the web service models. Basically, the connection check ensures whether the inputs of a models can be provided by the outputs of other models. This connection process is improved by reading standard-based variable names between models through the BMI functions. Once all the chosen models are *connected*, by using the model initialization API, the EMELI-Web instance then initializes the models through sending the input files and obtains the IDs of the modeling web service instances for further interaction usage.

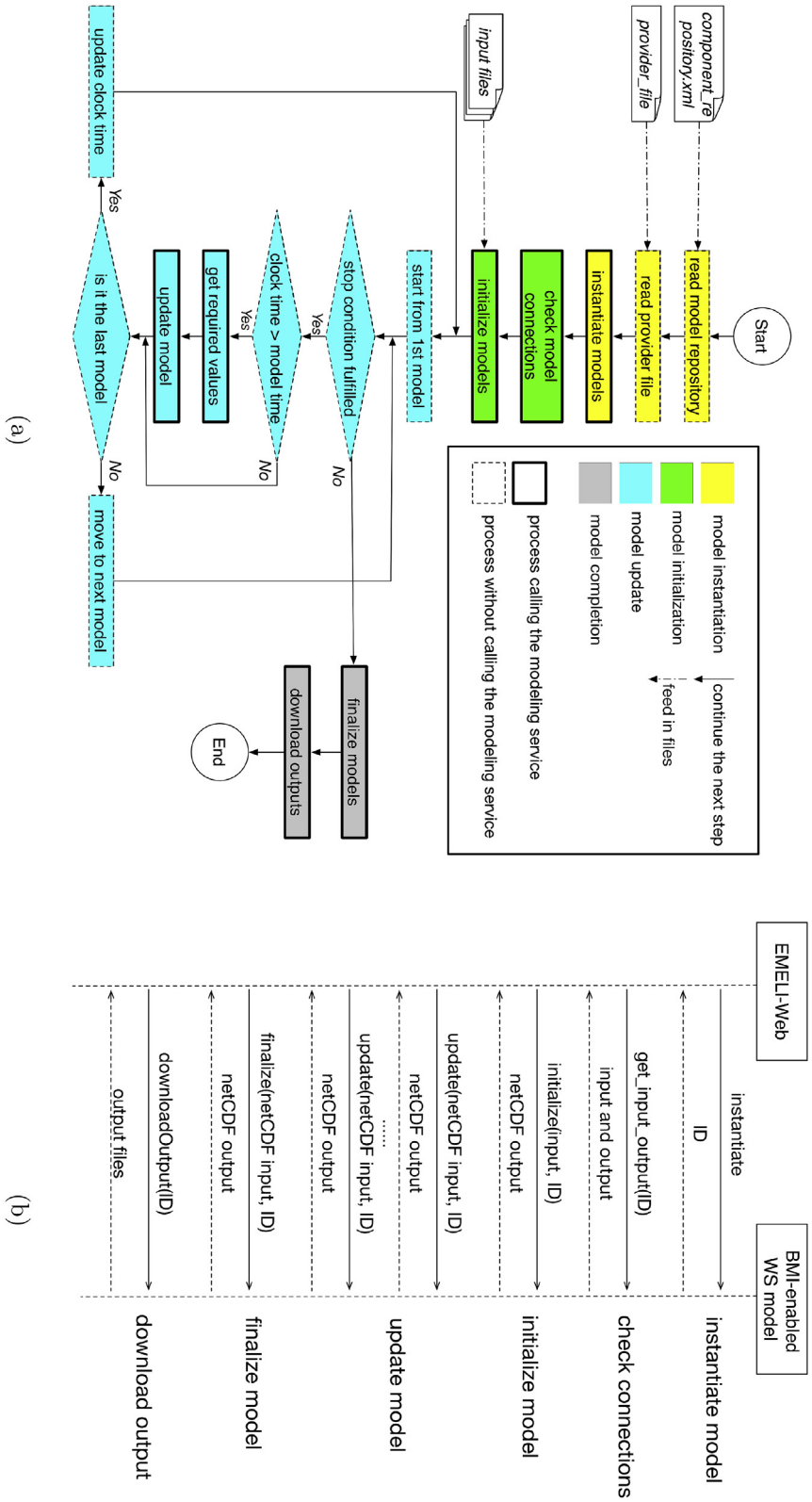

Fig. 2. The internal mechanism of EMELI-Web. (a) The workflow of running EMELI-Web with model instantiation (yellow), model initialization (green), model update (cyan), and model finalization (gray) stages. (b) Communication between EMELI-Web and a BMI-enabled web service model.

- **Model update**: After the web service models are initialized, the update API is used to update each model according to its own time step. In each time step, to judge whether a model needs to be updated, a framework environment *clock time* is compared with the internal time of a model, which is available by calling the API for BMI variable information functions. The model is

updated if *clock time* is larger than model's internal time. Once the internal times of all the models are larger than *clock time*, *clock time* is updated and compared again with models' current time for the update of the model. Furthermore, before a model is updated, EMELI-Web resets the input values of the model by (1) extracting other models' output values (by calling the APIs for BMI variable *getter* and *setter* functions), and (2) conducting numerical alignments (e.g., unit conversion and temporal alignment) based on the variable features (e.g., variable unit and its spatio-temporal property) of each model.

- **Model completion**: When a certain stopping condition (e.g., the total runtime or the excedance of a specific variable value threshold) is fulfilled, EMELI-Web sends a request to the finalization API of each modeling service, and downloads any existing output files.

3.2.2. Creating a web interface for EMELI-Web

To make it easy for users to utilize web service model coupling capability, EMELI-Web provides a web interface, which is constructed by using Flask — a Python web development package (Flask, 2016). This web interface is similar to CSDMS Web Modeling Tool (WMT), enabling the model selection, model configuration, model running and output download based on a web interface (CSDMS, 2016). However, it is noted that, different from CSDMS WMT, which interacts with BMI-enabled models existing in CSDMS High Performance Computing Cluster, the web application supported by EMELI-Web aims to couple models under a service-oriented architecture.

As a web application, EMELI-Web employs a simple relational database by using SQL (Structured Query Language) to enable the authorization of using EMELI-Web and also record the coupling activities of each user, which is illustrated in Fig. 3. The *User* database model is utilized to record a user's basic information (e.g., id, email, user_name, password, etc.) and authorize the usage of EMELI-Web. The other database model is *EMELI_Instance*, which is used for recording coupling activities of each user. The affiliation of a *EMELI_Instance* to a *User* is referred by the connection between *user_id* in *EMELI_Instance* and *id* in *User*. In addition, it is noted that the *id* field in *EMELI_Instance* serves a similar purpose as the ID in identifying the instance of a modeling service (see Section 3.1.2) — to avoid the conflicting uses of the same resource (EMELI in this case) from different users (see Fig. 1).

After registration, a user can start to use EMELI-Web to couple BMI-enabled web service models following the procedures depicted in Fig. 4 (i.e., selecting, configuring, and coupling, and getting results). In the **Select model** page, a new *EMELI_Instance* is generated to record this coupling activity based on the fields of *EMELI_Instance* (see Fig. 3). The user is required to: (1) select the models from the available BMI-enabled web service models in the execution order, (2) check the connections between the selected models by clicking **check model connections**, and (3) click **submit** to go to the **Configure model** page. In the **Configure model** page, the user needs to provide the values or files of the configuration variables. Once the configuration variables are ready, the user can click the **submit** button to trigger both the generation of the configuration file for each model and the creation of a runtime environment by EMELI-Web. After the coupling completes, the **Results & Outputs** page is shown, listing all the outputs.

4. Implementation

In this section we present an implementation of the BMI-based service-oriented modeling paradigm in loosely coupling a set of hydrological models. TopoFlow, a family of spatially distributed hydrological process components, is deployed as BMI-enabled web service models. EMELI-Web is then used to create a runtime environment for coupling these web service models.

4.1. Model description

TopoFlow is a spatially-distributed, D8-based hydrologic model (Peckham, 2009) that simulates several hydrologic processes, including meteorology, channel/overland flow, snow, evaporation, infiltration and subsurface flow. Its capability of reproducing different processes in hydrological cycle is well tested in modeling the Imnavait Creek watershed, Alaska (Bolton, 2006; Schramm et al., 2007). Furthermore, each process in TopoFlow can be simulated by using at least one method. For example, the infiltration process includes three methods: Green-Ampt method, Smith-Parlange method, and 1D Richards' equation with 3 layers. In addition, each process runs at a different time step (see Fig. 5) and is modular such that each process can be simulated as an independent model.

The modular feature of TopoFlow components allows the models being wrapped with BMI interfaces individually and work as plug-and-play components in the CSDMS architecture. The latest version of TopoFlow is written in Python and wrapped with BMI

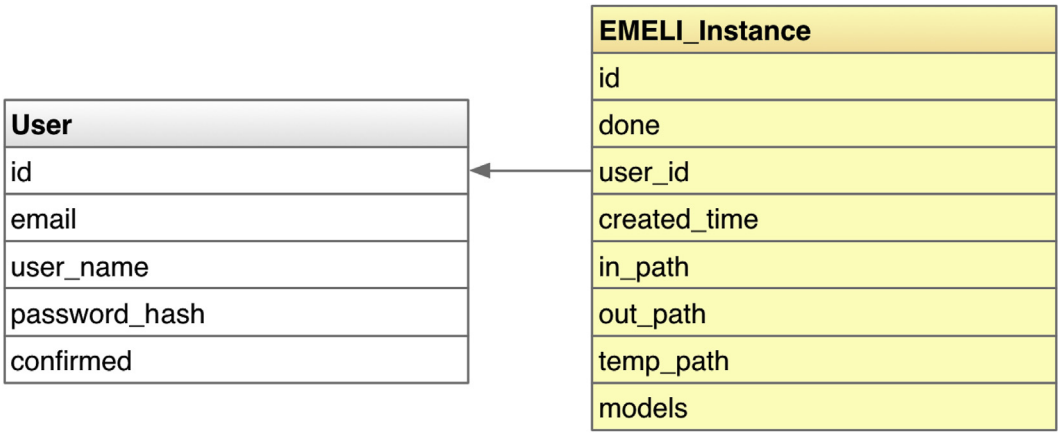

Fig. 3. The SQL relational database used for EMELI-Web includes two database model: User and EMELI_instance. The user_id in EMELI_instance is used as the foreign key pointing to the primary key id in User so that a user may have multiple EMELI-Web instances.

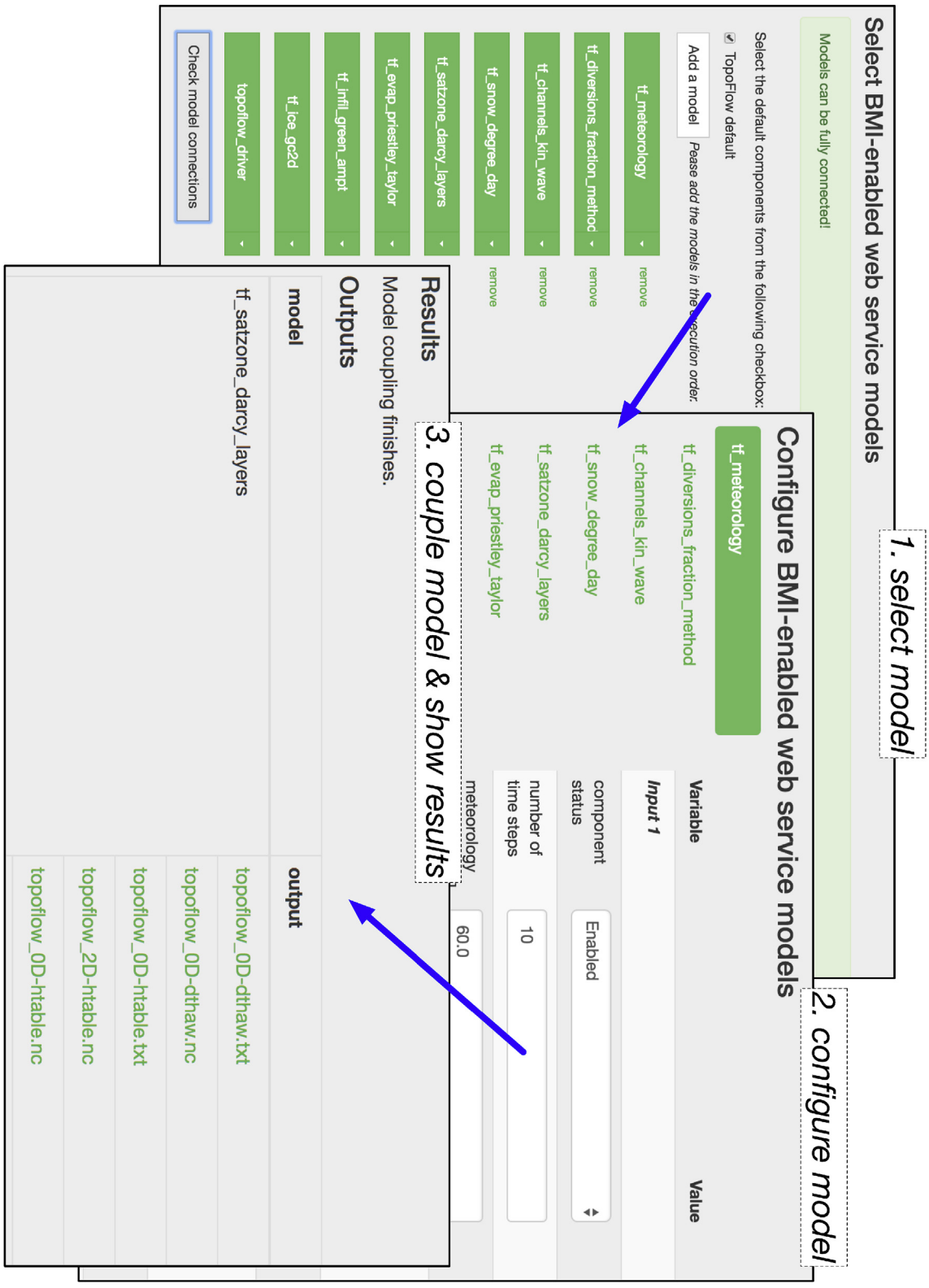

Fig. 4. Screenshots of the three procedures of using EMELI-Web: (1) selecting web service models to be coupled, (2) configuring the selected models, and (3) coupling models and showing the results.

interfaces (Peckham, 2013). The BMI-enabled feature of TopoFlow components facilitates the mapping between the CSDMS standard names and the models' internal variable names in BMI's model information functions for retrieving the BMI's inputs and outputs of a model in standard names. The semantic mapping between variable names helps connect models. For instance, one of the BMI output of the meteorology component is rainfall volume flux with the internal symbol *P_rain*, while the BMI input of the channel flow

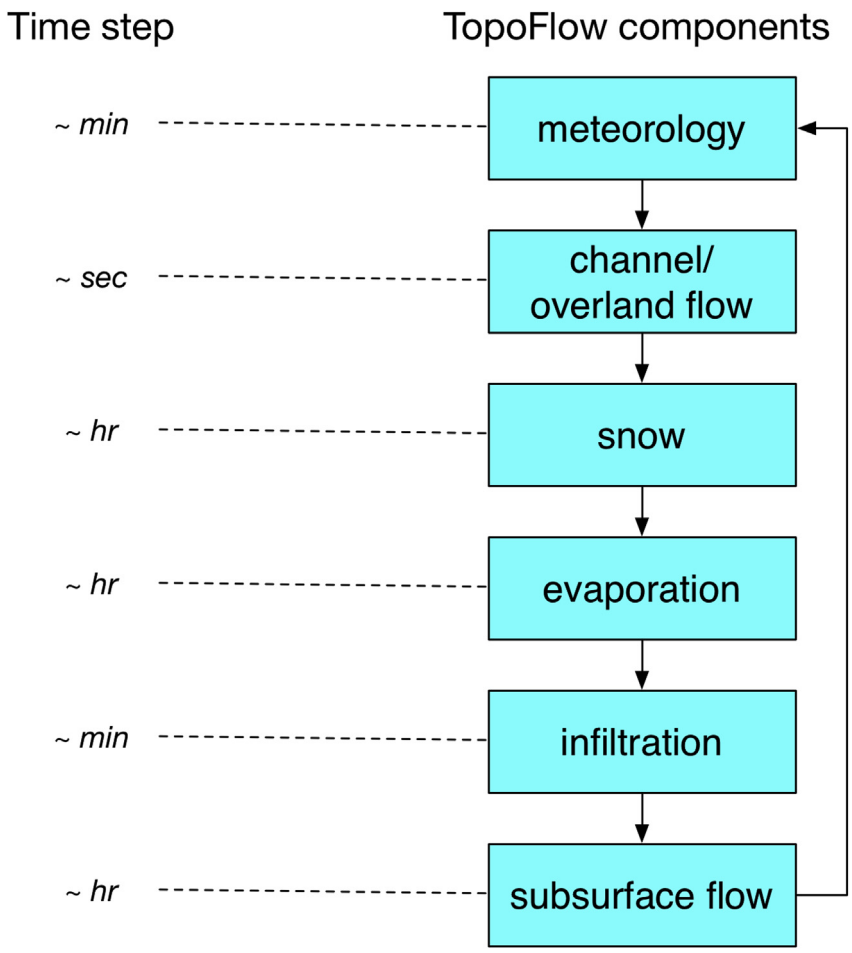

Fig. 5. An example of running TopoFlow components at different time steps.

component uses a different internal symbol *P* for rainfall. By mapping both *P_rain* and *P* to the CSDMS standard name *atmosphere_water__rainfall_volume_flux* in the meteorology component and the channel flow component, respectively, the two models can be connected through the rainfall variable even though their internal representations are different. The integration of the BMI-enabled TopoFlow components has been successfully implemented not only in the CSDMS framework but also in EMELI (Peckham, 2014b).

4.2. Integrating TopoFlow components as BMI-enabled web service models in EMELI-WEB

We expose the BMI-enabled TopoFlow components through web services, based on the endpoints listed in Tables 1 and 2 The Python web development package, Flask, is used for creating the web services.

Once the web service version of BMI-enabled TopoFlow components is ready, the basic information of the components (e.g., model name, author, URL, etc.) are registered in the component_repository.xml file. The XML file is later read by EMELI-Web in model instantiation stage as described in Section 3.2.1. Also, a web interface allowing users to provide information of configuration variables for each model is constructed in EMELI-Web, of which an example is illustrated in the configure model page screenshot in Fig. 4.

A simulation of TopoFlow is conducted on the Owl watershed, located in the Upper Sangamon River Basin in Illinois (see Fig. 6). The integrated TopoFlow components listed in the execution order are the meteorology component, the channel flow component of the kinematic wave method, the snow component of the degree day method, the evaporation component of Priestley-Taylor method, the infiltration component of Green-Ampt method and the saturated zone process component. To drive the above TopoFlow components, the following data files are used: (1) the digital elevation model of the Owl watershed and (2) the time series data collected from the sites of Intensively Managed Landscapes of Critical Zone Observatory (IML-CZO, 2014), including air temperature, precipitation and relative humidity. The remaining configuration variables of the models are assumed to be constant in that the main purpose of the simulation is to study the feasibility of coupling BMI-enabled web service models rather than seeking simulation accuracy. The procedures of using EMELI-Web to run BMI-enabled TopoFlow components have been detailed in Section 3.2.2. First, select the TopoFlow components in the **Select model** page, and check the model connections. Then, go to the **Configure model** page, and provide the configuration variables' information (i.e., upload data files from IML-CZO data repository site or enter default values). Next, when the configuration information is provided, click the **submit** button and EMELI-Web would start to couple the web service models. Once the model simulation finishes, the output files are available for downloading in the **Results & Outputs** page, as shown in Fig. 4.

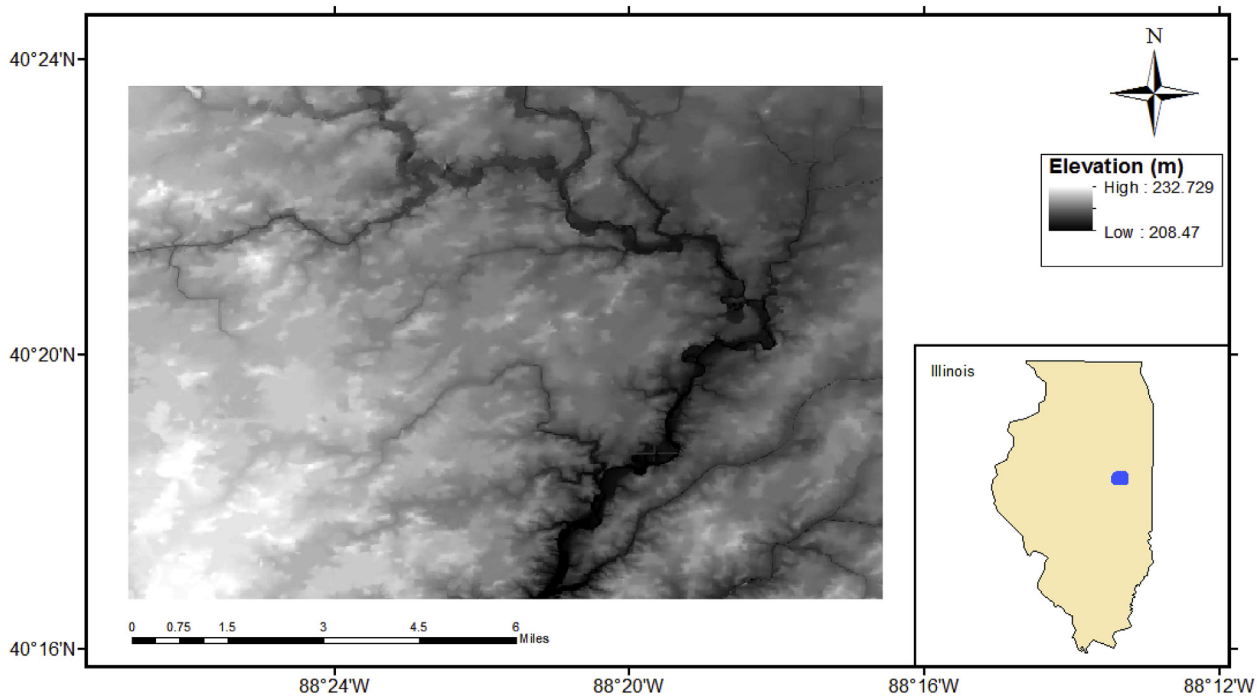

Fig. 6. The digital elevation model of Owl watershed and its location in Illinois (shown as the blue area of the figure in the lower right corner). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

5. Discussion

The motivation of this study is to create a service-oriented architecture for coupling models using Basic Model Interface (BMI). BMI is chosen as the standard model interface because of its two unique advantages in terms of model integration. First, the framework-agnostic property of BMI allows a BMI-enabled model to be easily set up in any other modeling integration frameworks. This is proved by the successfully developed EMELI, which is the basis of EMELI-Web. Also, BMI's capability of mapping between models' internal variable names and CSDMS standard names helps semantic mapping during the connection of models. We illustrated how TopoFlow components are connected to create a composite model in EMELI-Web through the BMI's input and output functions (Section 4.2).

A set of web service APIs are developed for converting a BMI-enabled model into web services, with JSON as the schema for encoding the information to and from the endpoints. Contrary to XML, which has been widely used as the basic data transfer schema in many standards (e.g., different conventions developed by OGC), JSON is adopted as the data-interchange format because of its light-weight format. Furthermore, we didn't adopt any existing web service standard due to the lack of a domain-specific web service standard for geoscience modeling, despite the existence of web service standard for different subdomains (e.g., OGC WPS standard for processing general data, the Water Markup Language (WaterML) for describing time series data (Taylor, 2012) and the Geography Markup Language (GML) for describing spatial information (Portele, 2013)). Besides a call for a web service standard of geoscience modeling, the combined employment of OGC WPS, WaterML and GML is also suggested as an alternative (Castronova et al., 2013). This effort requires a separate publication which demands not only how the different web service standards are combined but also whether the combination is suitable for a new standard. Hence, we suggest not only the development of a standard for geoscience modeling web service but also more JSON-based web service standards (some of which are being developed in the community (Cox and Taylor, 2015)).

One important consideration in our service-oriented modeling is reducing network latency caused by numerous information flowing between services. Specifically, netCDF files are utilized to store the numerical values of variables for updating variables between models through EMELI-Web. Compared with storing numerical values in XML format, less data storage in a binary-based netCDF file can be of benefit and leads to a faster information communication over the web.

Further, to identify different usages of the same resource (which can be either a BMI-enabled web service model or a model integration environment based on EMELI-Web in this study), a unique ID is assigned to a resource instance when the instance is generated by a specific client during the stage of model instantiation. However, the design of conserving different instances in both the memory and the file system raises the issue of system reliability and service performance, which is also a key aspect in service-oriented architecture. For example, how to optimize the execution procedures at the server side when multiple requests are sent to the server? To answer this question, a thoughtful parallel programming design considering the estimated execution time spent on each request would be helpful to achieve the minimum running time in the server, and requires further investigations.

Moreover, such service-oriented architecture can be not only applied in model coupling, but used for integrating a model with either an online data repository or a model assessment tool as well. For instance, a modeling service can be coupled with a model analysis toolkit such as DAKOTA, which has already been integrated into CSDMS modeling framework (Peckham et al., 2016). Also, a service-oriented wrapper can be set up between a modeling service and an online data repository (e.g. CUAHSI-HIS (the Consortium of Universities for the Advancement of Hydrologic Science, Inc - Hydrologic Information System)). Peckham and Goodall (2013) demonstrated the interoperability between CUAHSI-HIS and CSDMS by developing a prototype CSDMS component. Such

component can be revised to allow the interoperability between a EMELI-Web coupling activity and CUAHSI-HIS.

6. Summary and conclusions

We have deployed a release of a service-oriented modeling paradigm by adopting the Basic Model Interface (BMI) in this paper. We developed a set of web service APIs to expose the functionality of a BMI-enabled model via the internet. We enhanced EMELI to EMELI-Web enabling the integration of BMI-enabled web service models by using their web service APIs. Finally, we developed a user-friendly web interface for the convenient use of EMELI-Web. The whole orchestration was then implemented in coupling TopoFlow components, a set of spatially distributed hydrologic models.

The key contributions of this study in integrating web service models are as follows. First, employing the BMI as the standard model interface in modeling service not only enriches the semantic information of variables names which facilitates checking connections between web service models, but also allows the utilization of EMELI as a base to construct EMELI-Web due to its framework-independent property. In addition, to reduce network latency of executing web-serviced models, saving input/output values in a binary-format file transferred over the internet is more efficient than storing the values in a standard data-interchange protocol (e.g., JSON and XML). In this study, netCDF file is utilized to convey the variable values between EMELI-Web and a BMI-enabled web service model. Finally, the proposed way of identification of the model instance is effective in the avoidance of conflicting executions towards the web service model.

As stated in the introduction, employing a service-oriented architecture in model coupling is able to (1) lower the interoperability barrier of model integration by enabling the independence of programming language and operating system of models, (2) facilitate a model to be accessed via the internet without being downloaded thus popularizing the model, and (3) allow the easier maintenance of the model with it original functionality exposed through a web service. Despite these benefits of SOA, issues such as the network latency and the reliability of the architecture should also be taken into consideration for modelers and decision makers when applying a service-oriented modeling paradigm in simulating a specific phenomenon. Finally, even though further investigations need to be emphasized on the reduction of the web latency and the improvement of the execution performance, we are confident that service-oriented modeling paradigm implying a loosely coupling style is more suitable and convenient for integrating models where models are set up at different platform.

Acknowledgements

The authors wish to acknowledge the support of the study by the National Science Foundation (ICER-1440315 & EAR-1331906). Gratitude is also expressed to Yan Zhao from National Center for Supercomputing Applications for her tremendous help and advices in constructing the web interface of EMELI-Web.

References

Argent, R.M., 2004. An overview of model integration for environmental applications components, frameworks and semantics. Environ. Model. Softw. 19, 219–234. http://dx.doi.org/10.1016/S1364-8152(03)00150-6.

Bolton, W.R., 2006. Dynamic Modeling of the Hydrologic Processes in Areas of Discontinuous Permafrost. Ph.D. thesis. University of Alaska Fairbanks.

Castronova, A.M., Goodall, J.L., Elag, M.M., 2013. Models as web services using the open geospatial consortium (OGC) web processing service (WPS) standard. Environ. Model. Softw. 41, 72–83. http://dx.doi.org/10.1016/j.envsoft.2012.11.010.

Cox, S.J.D., Taylor, P., 2015. OGC Observations and Measurements JSON Implementation. Technical Report Open Geospatial Consortium. URL. https://portal.opengeospatial.org/files/?artifact_id=64910.

CSDMS, 2016. The CSDMS web modeling tool. URL. https://csdms.colorado.edu/wmt/. accessed: 2016-12-01.

David, O., Ascough, J., Lloyd, W., Green, T., Rojas, K., Leavesley, G., Ahuja, L., 2013. A software engineering perspective on environmental modeling framework design: the object modeling system. Environ. Model. Softw. 39, 201–213. http://dx.doi.org/10.1016/j.envsoft.2012.03.006.

Elag, M.M., Goodall, J.L., Castronova, A.M., 2011. Feedback loops and temporal misalignment in component-based hydrologic modeling. Water Resour. Res. 47 http://dx.doi.org/10.1029/2011WR010792.

Erl, T., 2004. Service-oriented Architecture: a Field Guide to Integrating XML and Web Services. Prentice Hall PTR.

Facchi, A., Ortuani, B., Maggi, D., Gandolfi, C., 2004. Coupled SVAT–groundwater model for water resources simulation in irrigated alluvial plains. Environ. Model. Softw. 19, 1053–1063. http://dx.doi.org/10.1016/j.envsoft.2003.11.008.

Fielding, R.T., 2000. Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis. University of California, Irvine.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., 1999. Hypertext transfer protocol–http/1.1. URL. https://www.w3.org/Protocols/HTTP/1.1/draft-ietf-http-v11-spec-01. accessed: 2016-12-01.

Flask, 2016. Flask web development, one drop at a time. URL. http://flask.pocoo.org/. accessed: 2016-07-04.

Geller, G.N., Melton, F., 2008. Looking forward: applying an ecological model web to assess impacts of climate change. Biodiversity 9, 79–83. http://dx.doi.org/10.1080/14888386.2008.9712910.

Goodall, J.L., Robinson, B.F., Castronova, A.M., 2011. Modeling water resource systems using a service-oriented computing paradigm. Environ. Model. Softw. 26, 573–582. http://dx.doi.org/10.1016/j.envsoft.2010.11.013.

Goodall, J.L., Saint, K.D., Ercan, M.B., Briley, L.J., Murphy, S., You, H., DeLuca, C., Rood, R.B., 2013. Coupling climate and hydrological models: interoperability through web services. Environ. Model. Softw. 46, 250–259. http://dx.doi.org/10.1016/j.envsoft.2013.03.019.

Hill, C., DeLuca, C., Suarez, M., Da Silva, A., et al., 2004. The architecture of the earth system modeling framework. Comput. Sci. Eng. 6, 18–28. http://dx.doi.org/10.1109/MCISE.2004.1255817.

Huhns, M.N., Singh, M.P., 2005. Service-oriented computing: key concepts and principles. Internet Comput. IEEE 9, 75–81. http://dx.doi.org/10.1109/MIC.2005.21.

IML-CZO, 2014. Intensively managed landscapes - critical zone observatory. URL. http://imlczo.ncsa.illinois.edu/ (Accessed: 04.07.16).

Van Ittersum, M.K., Ewert, F., Heckelei, T., Wery, J., Olsson, J.A., Andersen, E., Bezlepkina, I., Brouwer, F., Donatelli, M., Flichman, G., et al., 2008. Integrated assessment of agricultural systems–a component-based framework for the European Union (SEAMLESS). Agric. Syst. 96, 150–165. http://dx.doi.org/10.1016/j.agsy.2007.07.009.

Laniak, G.F., Olchin, G., Goodall, J., Voinov, A., Hill, M., Glynn, P., Whelan, G., Geller, G., Quinn, N., Blind, M., et al., 2013. Integrated environmental modeling: a vision and roadmap for the future. Environ. Model. Softw. 39, 3–23. http://dx.doi.org/10.1016/j.envsoft.2012.09.006.

Leach, P.J., Mealling, M., Salz, R., 2005. A universally unique identifier (UUID urn) namespace. URL. https://tools.ietf.org/html/rfc4122.

Maxwell, R.M., Miller, N.L., 2005. Development of a coupled land surface and groundwater model. J. Hydrometeorol. 6, 233–247. http://dx.doi.org/10.1175/JHM422.1.

Moore, R.V., Tindall, C.I., 2005. An overview of the open modelling interface and environment (the OpenMI). Environ. Sci. Policy 8, 279–286. http://dx.doi.org/10.1016/j.envsci.2005.03.009.

Mulligan, G., Gračanin, D., 2009. A comparison of SOAP and REST implementations of a service based interaction independence middleware framework. In: Simulation Conference (WSC), Proceedings of the 2009 Winter. IEEE, pp. 1423–1432. http://dx.doi.org/10.1109/WSC.2009.5429290.

Nativi, S., Mazzetti, P., Geller, G.N., 2013. Environmental model access and interoperability: the GEO model web initiative. Environ. Model. Softw. 39, 214–228. http://dx.doi.org/10.1016/j.envsoft.2012.03.007.

Peckham, S., 2009. Geomorphometry and spatial hydrologic modelling. Dev. Soil Sci. 33, 579–602. http://dx.doi.org/10.1016/S0166-2481(08)00025-1.

Peckham, S.D., 2013. TopoFlow. URL. https://csdms.colorado.edu/wiki/Model:TopoFlow (Accessed: 04.07.16).

Peckham, S., 2014a. The CSDMS Standard Names: Cross-domain Naming Conventions for Describing Process Models, Data Sets and Their Associated Variables. International Environmental Modelling and Software Society (iEMSs), San Diego, California, USA.

Peckham, S.D., 2014b. EMELI 1.0: An Experimental Smart Modeling Framework for Automatic Coupling of Self-Describing 483 Models. CUNY Academic Works. http://academicworks.cuny.edu/cc_conf_hic/464.

Peckham, S.D., Goodall, J.L., 2013. Driving plug-and-play models with data from web services: a demonstration of interoperability between CSDMS and CUAHSI-HIS. Comput. Geosci. 53, 154–161. http://dx.doi.org/10.1016/j.cageo.2012.04.019.

Peckham, S.D., Hutton, E.W., Norris, B., 2013. A component-based approach to integrated modeling in the geosciences: the design of CSDMS. Comput. Geosci. 53, 3–12. http://dx.doi.org/10.1016/j.cageo.2012.04.002.

Peckham, S., DeLuca, C., Gochis, D., Arrigo, J., Kelbert, A., Choi, E., & Dunlap, R. (2014). EarthCube-earth system bridge: spanning scientific communities with

interoperable modeling frameworks. In AGU Fall Meeting Abstracts (p. 3754). volume 1.
Peckham, S.D., Kelbert, A., Hill, M.C., Hutton, E.W., 2016. Towards uncertainty quantification and parameter estimation for earth system models in a component-based modeling framework. Comput. Geosci. 90, 152–161. http://dx.doi.org/10.1016/j.cageo.2016.03.005.
Portele, C., 2013. OpenGIS Geography Markup Language (GML) Encoding Standard. Technical Report Open Geospatial Consortium. URL. http://www.opengeospatial.org/standards/gml (Accessed: 01.12.16).
Rew, R., Davis, G., 1990. NetCDF: an interface for scientific data access. IEEE Comput. Graph. Appl. 10, 76–82. http://dx.doi.org/10.1109/38.56302.
Schramm, I., Boike, J., Bolton, W.R., Hinzman, L.D., 2007. Application of TopoFlow, a spatially distributed hydrological model, to the Imnavait Creek watershed, Alaska. J. Geophys. Res. Biogeosci. 112 http://dx.doi.org/10.1029/2006JG000326.
SOAP, 2004. Simple object access protocol. URL. https://www.w3.org/TR/soap/ (Accessed: 04.07.16).
Sui, D., Maggio, R., 1999. Integrating GIS with hydrological modeling: practices, problems, and prospects. Comput. Environ. urban Syst. 23, 33–51. http://dx.doi.org/10.1016/S0198-9715(98)00052-0.
Syvitski, J., Paola, C., Slingerland, R., Furbish, D., Wiberg, P., Tucker, G., 2004. Building a Community Surface Dynamics Modeling System: Rationale and Strategy. A Report to the National Science Foundation. Penn State University, State College, p. 41.
Taylor, P., 2012. OGCWaterML 2.0: Part 1 Timeseries. Technical Report Open Geospatial Consortium. URL. https://portal.opengeospatial.org/files/?artifact_id=57222.
Theurich, G., DeLuca, C., Campbell, T., Liu, F., Saint, K., Vertenstein, M., Chen, J., Oehmke, R., Doyle, J., Whitcomb, T., et al., 2015. The earth system prediction suite: toward a coordinated US modeling capability. Bull. Am. Meteorol. Soc. http://dx.doi.org/10.1175/BAMS-D-14-00164.1.
Yu, Z., Pollard, D., Cheng, L., 2006. On continental-scale hydrologic simulations with a coupled hydrologic model. J. Hydrol. 331, 110–124. http://dx.doi.org/10.1016/j.jhydrol.2006.05.021.