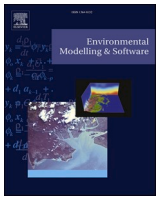
Environmental Modelling and Software 152 (2022) 105384

Contents lists available at ScienceDirect

Environmental Modelling and Software

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



Open data and model integration through generic model agent toolkit in CyberWater framework

Ranran Chen [a], Daniel Luna [b], Yuan Cao [a], Yao Liang [a,**], Xu Liang [b,*]

[a] Department of Computer and Information Science, Indiana University-Purdue University Indianapolis, 723 W. Michigan Street, SL 280, Indianapolis, IN, 46202, USA
[b] Department of Civil and Environmental Engineering, University of Pittsburgh, 728 Benedum Hall, 3700 O'Hara Street, Pittsburgh, PA, 15261, USA

ARTICLE INFO

Keywords:
Open data and model
Model integration
Generic model agent
Scientific workflow
Reproducible process

ABSTRACT

The CyberWater project is created to develop an open data and open model integration framework for studying complex environmental and water problems, where diverse online data sources can be directly accessed by diverse models without any need of users' extra effort on the tedious tasks of data preparation for their models. We present our design and development of a novel generic model agent toolkit in the context of CyberWater, which enables users to integrate their models into the CyberWater system without writing any new code, significantly simplifying the data and model integration task. CyberWater adopts a visual scientific workflow system, VisTrails, which also supports provenance and reproducible computing. Our approach and the developed generic model agent toolkit are demonstrated, via CyberWater framework, with automated and flexible workflows through integrating data and models using real-world use cases. Two popular hydrological models, VIC and DHSVM, are used for illustrations.

Software/data availability

Name	Author/Contact	Year first available	Format/Language	Cost	Size	Availability
VisTrails	New York University (https://www.vistrails.org/index.php/People)	2007	Python	Free	300 MB	https://www.vistrails.org/index.php/Main_Page
VIC	University of Washington (http://uw-hydro.github.io/team/)	1994	C	Free	2 MB	https://vic.readthedocs.io/en/master/
DHSVM	Pacific Northwest National Laboratory (ning.sun@pnnl.gov; mark.wigmosta@pnnl.gov)	1994	C	Free	60 MB	https://www.pnnl.gov/source-code
USGS Water Services	USGS (https://water.usgs.gov/contact/gsanswers)	N/A	XML webservices	Free	N/A	https://waterdata.usgs.gov/nwis/dv
NCA-LDAS	NASA (gsfc-dl-help-disc@mail.nasa.gov)	2018	Tiff, GRIB or NetCDF Webservices	Free	N/A	https://doi.org/10.5067/7V3N5DO04MAS
NLDAS	NASA (gsfc-dl-help-disc@mail.nasa.gov)	2009	Tiff or NetCDF Webservices	Free	N/A	https://doi.org/10.5067/6J5LHHOHZHN4

* Corresponding author.
** Corresponding author.
E-mail addresses: ranrchen@iu.edu (R. Chen), del47@pitt.edu (D. Luna), caoyua@imail.iu.edu (Y. Cao), yaoliang@iupui.edu (Y. Liang), xuliang@pitt.edu (X. Liang).

https://doi.org/10.1016/j.envsoft.2022.105384
Received 1 August 2021; Received in revised form 2 March 2022; Accepted 26 March 2022
Available online 29 March 2022

1. Introduction

With diverse data across disciplines being increasingly available on today's internet, it has become more urgent and important to have a framework facilitating data and computational model integration for studying problems involved in many complex processes (e.g., physical, hydrological, biological, and atmospheric) such as predictions of floods, droughts, water quality, and air quality. While there are many online data sources available on the internet, the key challenge is how to make these data sources directly accessible to various models without users' additional effort on data preparation or data preprocessing for their models. Such model input data preparation task is usually very time-consuming, tedious, and error-prone due to the fact that individual data sources offer their different access protocols and have their different data organizations and formats. Previous research focused more on model integration (Argent, 2004; Belete et al., 2017) and the papers therein), with little consideration or treatment on the general problem of integrating data and models together. For example, existing modeling systems for different needs in environmental and water fields include Open Modeling Interface (OpenMI)[1] (Moore and Tindall, 2005; Gregersen et al., 2007; Knapen et al., 2013; Harpham et al., 2019), Community Surface Dynamics Modeling System (CSDMS)[2] (Peckham et al., 2013; University of Colorado Boulder, 2012), Object Modeling System (OMS)[3] (Ahuja et al., 2005; David et al., 2013), Earth System Modeling Framework (ESMF)[4] (Collins et al., 2005; DeLuca et al., 2012; Hill et al., 2004), The Invisible Modeling Environment (TIME) (Rahman et al., 2003; Stenson et al., 2011), and Geographic Modeling and Simulation Systems (OpenGMS) (Chen et al., 2019; Zhang et al., 2019; Wang et al., 2020). These modeling systems do not directly address the challenge of enabling users' models to access the heterogeneous data sources automatically and seamlessly over the internet to carry out their modeling studies. While some of these systems support public sharing of data, modeling, and/or simulation resources, they are generally limited to some specific models and data rather than providing a general framework to integrate diverse data and models like CyberWater. Although OpenMI2.0 and OpenGMS, for example, increase the flexibility by turning numerical models into linkable components to receive exchange data from each other, there is remaining work for these systems to wrap up the existing models to be feasible components running in these frameworks. Salas et al. (2020) provided comprehensive reviews of these systems and compared them with the earlier version of the CyberWater framework – a recent effort aiming at developing a systemic solution to this challenge. The key idea and feature of the CyberWater system is its focus on building an open architecture framework to facilitate realizations of open science (Salas et al., 2020) which facilitates resource sharing (e.g., data and models), reproducibility of the work, and community participation. To this end, an open data and open modeling framework (Salas et al., 2020), referred to as MSM (Meta Scientific Modeling) that provides the foundation for the CyberWater framework, is developed. MSM is designed to offer scientific researchers and practitioners a sophisticated open modeling environment, where data agents are developed to access heterogeneous online data products provided by different data sources so that the data can directly flow seamlessly from the data sources to the users' models without any need for the users to preprocess the input data for their models. The model input data preparation has been performed by each individual data agent developed in MSM for its corresponding data source, such as the USGS and NASA data products. The implementation of MSM is called *msm*, which incorporates the workflow system of VisTrails[5] to provide not only a graphical workflow mechanism for achieving the modeling task but also data provenance to ensure traceability and reproducibility (Salas et al., 2020). To make use of the facilities of data agents in *msm*, however, one needs to integrate his/her model into the *msm* system by means of writing a model agent based on the interface provided by *msm*. Once a user's model is integrated into *msm* via its agent, the user's model can use all the data agents available in the *msm* system for direct data and model integration and can also couple his/her model with another model already integrated into *msm* for model-to-model coupling/integration without coding. Another recent work on the model input data preparation is HydroDS web services (Gichamo et al., 2020), which is applicable to two specific models, the Utah Energy Balance (UEB) snowmelt model (Tarboton et al., 2014) and the TOPNET hydrologic model (Bandaragoda et al., 2004). The use of HydroDS, an integration of model UEB with data, is demonstrated through the development of a web application (Gan et al., 2020). The strength of HydroDS, a webservice based data and model integration approach, is that there is no need for local software installation. Its limitations include not supporting other models beyond UEB and TOPNET, nor providing users with any mechanism to integrate new models into HydroDS. CyberWater is a standalone system at present, which avoids the potential bottleneck of the centralized server, facilitating scalability and sustainability. CyberWater will be extended to include web services as well in the future, to further complement its standalone system. For the webservice-based systems, Chen et al. (2020) provided a comprehensive review.

While the *msm* framework has its unique merit for open data and open model integration, it nevertheless still requires programming a model agent per new model integration. To write a model agent means that the user needs to do programming/coding. To overcome such a limitation, this work extends the previous *msm* framework. Here, we present a general approach to construct a generic model agent template using workflow for handling models' input interfaces for the environmental and hydrological models. We then design and develop a set of tools called generic model agent toolkit for our approach, with which a user can construct his/her model agent by simply doing parameter-based configuration steps without coding for most of the environmental and hydrological models. This effort, together with other extensions to and improvements on the *msm* system, including accessing high performance computing on demand (Li et al., 2021), constitutes the latest development of our CyberWater framework software system.

In this paper, we present the design and development of the generic model agent toolkit in CyberWater. We then demonstrate how this set of tools can significantly eliminate the user's task of writing a model agent in order to integrate a new model into the system, and thus further enhance the usability and effectiveness of the CyberWater framework for open data and model integration and greatly improve the capability of the community to share diverse models for model validation, evaluation, scientific explorations, etc. Different versions of the Variable Infiltration Capacity (VIC) model (Hamman et al., 2018; Liang et al., 1994, 1996a, 1996b; Liang and Xie, 2003)and the Distributed Hydrology Soil Vegetation Model (DHSVM) (Wigmosta et al., 1994) are used in this study for illustration via use cases over several watersheds in Pennsylvania, USA.

The remainder of this paper is organized as follows. Section 2 briefly overviews the background to set up the context. Section 3 presents our approach and design, while Section 4 describes the implementation of the generic model agent toolkit. Section 5 provides two use cases of reproducible end-to-end model simulations to demonstrate the use of the developed generic model agent toolkit for open data and model integration in CyberWater and offers our insights through discussions. Finally, Section 6 concludes the presented work and provides planned future work.

[1] https://www.openmi.org/.
[2] https://csdms.colorado.edu/wiki/Main_Page.
[3] https://alm.engr.clolstate.edu/cb/wiki/16961.
[4] https://earthsystemmodeling.org/.
[5] https://www.vistrails.org/index.php/Main_Page.

2. Background

2.1. MSM architecture

MSM (Meta-Scientific-Modeling) is an Open-data Open-model framework that provides a sophisticated workflow-controlled modeling environment for heterogeneous data and model integration without a need for a central administration (Salas et al., 2020). The implementation of the MSM framework, having VisTrails (Bavoil et al., 2005) incorporated in it, is called *msm* system which, running on the user's local desktop or laptop, constitutes several components, including the core (i.e., *msm* core), Data Agents, and Model Agents. Basically, the *msm* system interacts with the workflow engine of VisTrails, accesses remote heterogeneous data sources through Data Agents, and invokes users' computational models for simulations through Model Agents.

The *msm* core is the center of all the components in the Open-data Open-model framework. Through its connection with the workflow engine of VisTrails, the *msm* system invokes external models and couples various data sources with respective model agents and data agents via a friendly and feasible workflow controlled working environment. Therefore, the MSM enables access to external data of diverse sources conveniently and effectively, and execution of sophisticated numerical models efficiently. The open-model feature allows various models or modules to be smoothly integrated into the MSM system via Model Agents. The architecture of MSM offers a novel "information bus" connection pattern of linear complexity (i.e., the complexity of the development of data and model agents is $O(m + n)$ for integration of m data sources and n computational models), the lowest possible complexity, among all independent models and external heterogeneous data sources, as opposed to quadratic complexity involved in a pair-wise based one-to-one integration architecture (i.e., the complexity of $O(mn + n^2)$ for integration of m data sources and n computational models) (Salas et al., 2020).

In the MSM framework, the modeling processes are dynamically decided by the end-users through a workflow engine. Users can customize the sequence of activities and construct any workflow series for their modeling process based on the building blocks or modules provided by the *msm* system. Each workflow activity conducts interactively with the other workflow activities in the same workflow.

MSM's design criteria enable a creation of a general and flexible system so that the MSM framework can be easily plugged into various open-source workflow engines. The MSM takes advantage of the existing workflow engines and avoids reinventing the wheel. The overall architecture of the MSM system is shown in Fig. 1.

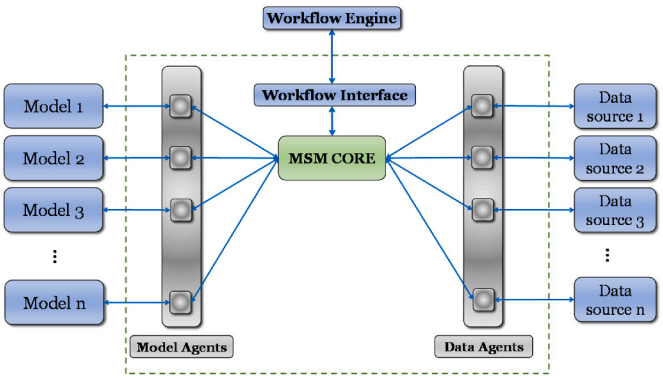


Fig. 1. An overall architecture of the MSM framework system, in which each model (i.e., Model 1, Model 2, …, and Model n) represents an individual computational model integrated to the MSM framework to perform its model simulation instance.

2.2. Model agent

In *msm*, various computational models are added into the MSM modeling framework by means of their corresponding model agents. Any model added into MSM can then be freely integrated with various data agents as well as other models in a workflow constructed by the user, where model agents represent and execute the corresponding external models in the workflow (Salas et al., 2020). Thus, the effort to add a model into the MSM modeling framework is to develop the model's agent. Since no modification of the original model code is required in the *msm* system, a model agent needs to perform three main tasks as follows:

- Preparing model's inputs: The agent collects the model's required input data from the workflow and transforms them into the model's input files, which are needed by the model's execution.
- Executing model: The agent invokes the model executable file, according to the workflow control, and runs the model.
- Retrieving model's outputs: After finishing the model's execution, the agent needs to retrieve the model's output files and transform the output data into the *msm* system in the format of *msmDataSet* before the subsequent workflow item gets to be executed.

The Generic Agent is a class developed in the *msm* system to assist users in their development process of model agents. In the Generic Agent base class, a generic *run_model* method is declared. Each model's agent is inherited from the Generic Agent class and overrides the generic *run_model* method, enabling the customized model's invocation from the workflow. The model agent uses services offered by the Generic Agent class to read the inputs from and write the outputs into the *msm* core datastore, which simplifies the agent's coding for integrating an external model into the *msm* system.

The model agent's primary responsibilities are to be implemented by overriding the *run_model* function. That is, first, read the model inputs from the *msm* core and save the data into the model's input files; then, run the model executable file. Finally, read the model's output files after the model's execution, transform them into the *msm* dataset format, and save the model's outputs into the *msm* core.

To write a model agent, the user has to create a text file including a class in Python as outlined here:

```
from msm_core.msm…… import GenericAgent
class MyAgent (GenericAgent):
    def run_model (self, inputs, parameters):
```

In the *run_model* function, the user writes all the necessary codes to prepare the inputs, execute the model and store the outputs. These are the three main tasks that the model agent is supposed to perform to integrate the user's model into the *msm* system. For more details, readers are referred to Salas et al. (2020).

3. Approach

3.1. The idea

While the Generic Agent class in the *msm* system reduces users' burden to develop their model agents to some extent, it would be desirable if users' model agents can be constructed without any coding. Since, in general, any model agent will conduct the three tasks described in Section 2.2, our idea is to construct a generic model agent *template* for most environmental and hydrological models, which predefines/outlines tasks of preparing model inputs, executing model, and storing outputs in a workflow environment. In the *msm* system, all workflow modules can be dynamically defined and configured. Thus, using the *divide and conquer* technique, a generic model agent can be achieved by first composing a generic workflow segment (template) of predefined generic modules as part of the overall workflow in the *msm* system. Such a workflow segment (template) will then be *configured* by individual users for their specific models, where each model's variability is accommodated via the configuration of the generic workflow segment (template). The predefined generic modules represent the abstraction of each aspect of the generic model agent. This way, the variability of an individual model is treated and represented through parameter-based configuration options of the workflow segment where specific information concerning an individual model, such as the model's parameter values, is configured via a graphic user interface of the input panel associated with the configuration option. Therefore, a user no longer needs to write any specific model agent code to integrate an external model into the *msm* system, which greatly enhances the usability of the MSM modeling framework for achieving the data and model integration.

3.2. The design

To fulfill our idea, we attempt to systematically predefine and design a set of generic workflow modules (i.e., components) to be used for constructing a model agent for an individual model in the form of a generic workflow segment (i.e., template), where each component represents and fulfills an individual task that a user's specific model agent would accomplish. To this end, our design strategy is as follows.

a) We developed a set of basic template-based workflow components based on an analysis of the representative model interface structures and organizations of most environmental and hydrological models. Those workflow components are used as building blocks for constructing a generic model agent, where each component (i.e., building block) is called a generic model agent tool.
b) A specific model agent for a given scientific model is constructed by composing the predeveloped component templates into a workflow segment (template) which is then incorporated into the user's overall modeling workflow sequence.
c) The workflow segment (template) is configured by the user for the model simulation needs to serve as the model agent. This way, the model agent's construction and its corresponding configurations are realized within the *msm* system via an interactive graphical workflow interface without any coding.

Our design includes the following six major generic model agent tools (i.e., components). Users are only expected to be knowledgeable about their own models' input/output details when they use these generic model agent tools for integrating their models into the CyberWater system. A model's inputs are typically classified into three categories: static parameters, forcing data, and the initial states of the model.

1 ***MainGenerator***: This is the first component (i.e., the root component) to build up a model agent in the form of a workflow segment (template) for a user's specific model. This component is responsible for setting up a working directory for performing the user's model simulation. Inside this working directory folder, all the input information needed for executing the user's model and the output information after the model's execution will be placed. The path of the working directory is specified by the user. This component is also responsible for receiving the forcing information required for the model's execution. The order in which these forcing data are collected to the *MainGenerator* as datasets through its *module information panel* matters since the final forcing data files created and placed in the working directory folder for the model execution will have these forcing data organized in columns following the same order. In addition, this *MainGenerator* reads in other important information needed for setting up the required execution condition/environment for the user's specific model, for example, information specifying certain options such as the number of soil layers to be used by the model, the water or energy balance mode to be executed, etc. Such information is typically provided through a model's global file or model's configuration file whose path is provided to this component, and also through the use of its *module information panel*. This module outputs two pieces of information through its two output ports. One is called WD_Path, which provides the working directory's path information where all the model's needed input information and the model's output information will reside. The other is called DataSet_Class, which is a list of all the datasets comprising the model's forcings.
2 ***AreaWiseParamGenerator***: This component organizes parameter files (e.g., vegetation and soil related parameter files), and places the imported parameter files into the parameter folder created by it. This parameter folder will be automatically placed under the working folder provided by the *MainGenerator* module. Information on all the paths of the parameter files needed for executing the user's model is collected in this module by specifying them through the *module's information panel* from the user interface. This module outputs a "ready" signal which will be discussed later.
3 ***ForcingDataFileGenerator***: This component is responsible for creating the forcing data for the user's model. Specifically, this component organizes the forcing data brought in by the *MainGenerator*. The user should be aware of the expected format of the model's forcing data. By default, this module creates the forcing data to be used by the user's model in a folder named "Forcing" inside the working directory. Such forcing data are always created as a time-series, where different columns hold different variables according to the order specified in the *MainGenerator*. The data are divided into separate files, where each one represents a single modeling cell/unit of the forcing inputs. This module has two input ports, one is the working directory, and the other is the list of forcing datasets, both of which are provided by the *MainGenerator* module. Like the *AreaWiseParamGenerator*, this module also outputs a "ready" signal.
4 ***InitialStateFileGenerator***: This is an optional component which organizes the data of initial states for the user's model, as the initial state files are not always required for the execution of a given model. However, if a user would like to conduct data assimilation to improve the user's model forecast reliability, the user may need to use this module. In any event, if required, this module receives the information of the working directory from the *MainGenerator* module. Also, the initial states' file paths will be specified through this *module's information panel* and be placed either inside the working directory or a new directory created by the *InitialStateFileGenerator*, with a name given as an input from the user. The output of this module is again a "ready" signal.
5 ***RunModuleAgent***: This component is responsible for invoking the user's model (e.g., executable) on the local machine, and for retrieving the model's outputs. It executes the model in the same way as the model runs alone without the *msm* system environment, where the user must manually prepare all the input files (e.g., forcing data, parameter files, initial state files) of the model. This module has three pieces of input information. First, it connects to the output of the

AreaWiseParamGenerator, *ForcingDataFileGenerator*, and *InitialStateFileGenerator* (if required), separately, to receive their "ready" signals. Second, it receives the information of the working directory from the *MainGenerator*. Third, it receives the dataset information outputted from the *MainGenerator*. This module also requires the path where the model's executable file is located. By setting up all the aforementioned information and receiving all the required information passed onto it from the other modules/components, this *RunModuleAgent* component is now ready to execute the user's model. The model's simulated results will be written back to the CyberWater system for the workflow to continue.

6 ***HPC***: This component is the module responsible for accessing remote High-Performance Computing (HPC) facility on-demand. This module aims to provide high-performance computing capacity for executing the user's model by seamlessly connecting it to either academic supercomputers or commercial cloud platforms (Li et al., 2021). It retrieves the results back to the workflow when the execution of the user's model on the remote HPC platform is completed.

In a nutshell, regarding the three tasks that a model agent needs to perform (Sec. 2.2), *preparing model's input* is fulfilled by a combination of three corresponding generic components (i.e., components 2, 3, and 4 described above), while the tasks of both *executing model* and *retrieving model's outputs* are combined and fulfilled by the generic component *RunModuleAgent* (component 5 above) or HPC (component 6 above), depending on whether the model is executed on a local machine or a remote HPC platform. Based on our approach, users construct their model agents, using the generic model agent toolkit (i.e., the six modules or building components) described above, at the same time as they employ the *msm* system to build their overall modeling workflows. Fig. 2 illustrates a typical workflow construction for hydrological/environmental modeling in the *msm* system with a graphical workflow interface. The dotted block indicates how the generic workflow components presented above are used to construct and configure a user's model agent in the workflow. After the completion of the user's construction and configuration of the workflow, the workflow is then executed in the *msm* system. A sequence diagram of a typical modeling workflow with the constructed model agent using the generic model agent toolkit is illustrated in Fig. 3.

In summary, the generic model agent toolkit is designed to help the user construct a model agent to integrate the user's model into CyberWater without writing any codes. Such a goal is achieved through some combinations of the six modules in the toolkit together to perform the following main functionalities:

- Set up the user's model working directory.
- Write data sets retrieved from data agent(s) or local files into model input files. That is, prepare input data files for the user's model.
- Invoke the user's model codes to run the user's model.
- Write the user's model output results back to the CyberWater system.

4. Implementation

The generic model agent toolkit, called *GenericModelAgentTools*, is developed and implemented in Python outside of the *msm* system as an extensive VisTrails package, which is a significant extension to the original *msm* system in the overall CyberWater system. However, the *GenericModelAgentTools* need the support of the corresponding DAO and Cache modules of the *msm* system to access data sets flowed into the workflow modules. The hierarchical architecture of the CyberWater system is shown in Fig. 4.

Each generic model agent tool in the toolkit is designed and implemented as a template-based workflow module. To illustrate, we present, in the following, the implementation of the template-based generic model agent tool *MainGenerator* in Sections 4.1, and its configuration in Section 4.2, respectively.

4.1. Generic model agent tool as a template

The *MainGenerator* creates a working directory folder as a workspace for the user's model, where it hosts all the input information required for executing the user's model and the output results of the model after its execution, copies the model's global parameter file or configuration file into this created working directory for carrying out the model's execution task, and converts the datasets into DATASET_CLASS elements in the form of Python dictionaries. This component also gathers all the forcing datasets required by running the user's model. The maximum

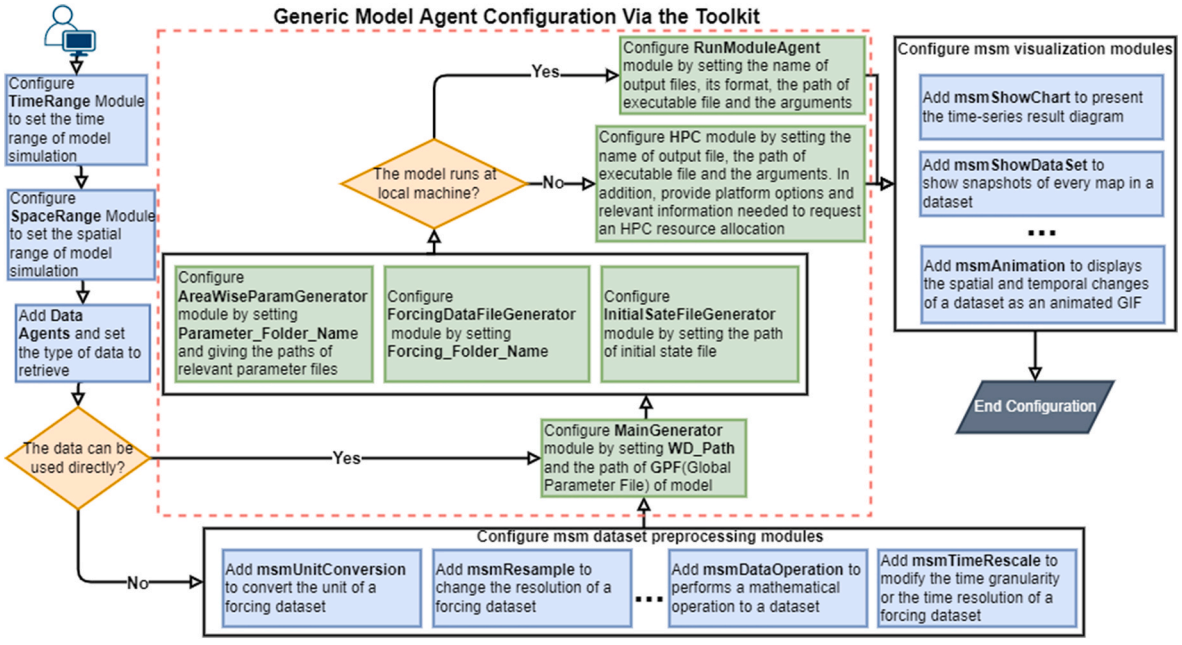


Fig. 2. A typical construction process of hydrological/environmental modeling workflow in the msm system, where the dotted block indicates how the generic workflow components are used to construct and configure a user's model agent in the workflow.

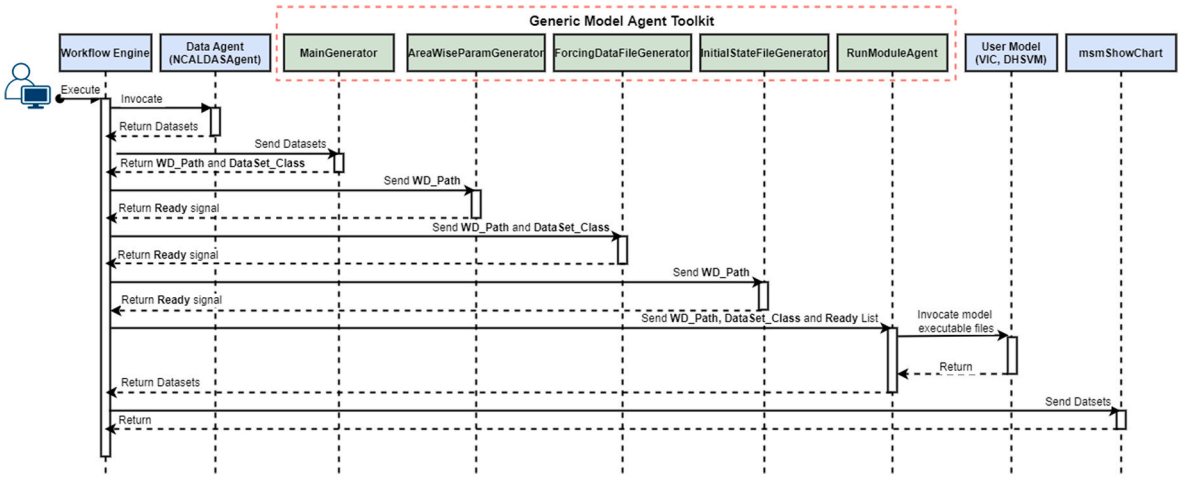


Fig. 3. An illustration of a sequential execution diagram of a typical modeling workflow with the constructed model agent by the agent toolkit in the msm system.

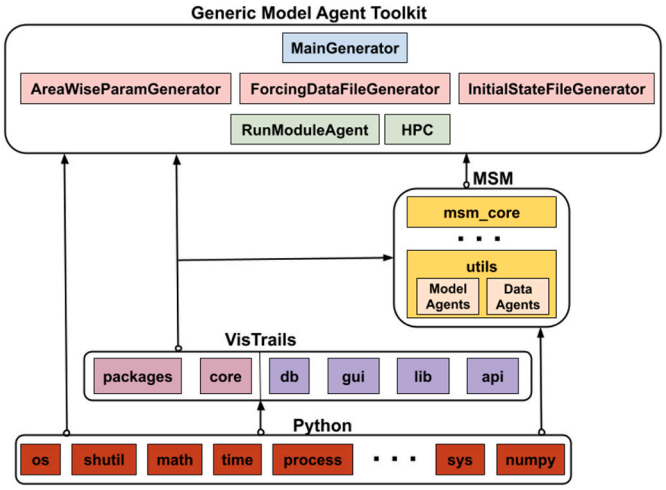


Fig. 4. Diagram of the hierarchical architecture of the CyberWater system.

number of the *MainGenerator's* input ports for datasets is set to be 15 as a default for convenience. However, the input ports of the *MainGenerator* can be expanded if needed. Hence, in the template-based component *MainGenerator*, the path of the working directory, the path of the model's global parameter file, and the datasets of the forcing to be used are all *configurable* by the user. For illustration, the pseudocode of the *MainGenerator* is given in Fig. 5.

In general, there are three types of input files needed for executing a numerical model, namely parameter files, forcing data files, and initial state files. The *AreaWiseParamGenerator* organizes the parameter files for performing the numerical model; the *ForcingDataFileGenerator* creates the model's forcing data files; and the *InitialStateFileGenerator* is engaged in preparing the initial state files, if needed, for performing the numerical model simulation. The *RunModuleAgent* module is for setting up the path where the model's executable file is located. It also obtains the information of the arguments for executing the model and the configuration information for the model's output variables. During the workflow execution, the *AreaWiseParamGenerator*, *ForcingDataFileGenerator*, and *InitialStateFileGenerator* modules process the datasets to generate the needed input files for the user's model. The module of the *RunModuleAgent* invokes the user's model execution based on the datasets passed. The model's simulated results will be written back to the CyberWater system. Similar to the *MainGenerator*, each of these template-based components also has its places configurable for the user to construct part of the user's model agent.

4.2. Configuration of generic model agent modules

A configuration example with the *MainGenerator* (Fig. 6) is used here to illustrate how the user can configure a generic model agent module.

a. Input Port Specification

01_Path: The path of the working directory folder where files for executing the user's model will reside.

02_GPF: The path of the global or configuration file if such a file is required by the user's model. This file is commonly used for setting up the variables for the model's execution.

Dataset_x: Collecting the forcing ***Dataset_x*** (x = 01, 02, 03, …, 15) from the *msm* system, in which *x* indicates the index of the dataset, and passing the forcing datasets to the *ForcingDataFileGenerator* to generate the required forcing files for the user's model to run.

b. Output Port Specification

WD_Path: An output port that passes the working directory path information to the next module.

DataSet_Class: Another output port that passes the information of a dataset cluster, i.e., the information of the forcing datasets, to the next module.

5. Use cases

There are two approaches to integrate a new model into CyberWater. One is to program a model agent by the user, while the other is to use the generic model agent toolkit provided in CyberWater to build the model agent without coding. This section shows how to achieve the integration of a user's model into the CyberWater system using our developed generic model agent toolkit without writing a single line of code. The user's model integration will be illustrated using our generic model agent toolkit versus adopting the user's manually programmed model agent. Two use cases with the VIC model and DHSVM model integrations are provided in Sections 5.1 and 5.2, respectively.

5.1. VIC model simulation

CyberWater provides various modules that help users construct their model simulation workflow to directly access online data from diverse data sources (e.g., forcing data from NASA and streamflow data from USGS), execute models, and produce model simulation results. In this use case, the Variable Infiltration Capacity (VIC) model (Liang et al., 1994, 1996a, 1996b; Liang and Xie, 2003) is employed for illustration. First, we show the integration of the VIC model version 4.0 (VIC4) into the CyberWater system via the user's manually coded model agent called VIC-Agent. The integrated VIC4 runs a water balance simulation of the French Creek basin in CyberWater. This is a small watershed with

```
1.  PROGRAM MainGenerator
2.
3.  START
4.
5.  //Get Inputs
6.  GET the WD_PATH, which points at an empty folder on user's machine and where the relative simulation will occur as
    a working directory.
7.  GET the GPF, which points at the global parameter configuration file.
8.  GET the datasets by connecting the previous modules that brings needed dataset.
9.
10. FOR every dataset connected to the input ports
11.       GET the Dataset_x. //x indicates the index of the dataset.
12. ENDFOR
13.
14. //Start computing
15. INIT the variable DATASET_CLASS as an empty dictionary
16.
17. //Fill the DATASET_CLASS with the datasets obtained from previous modules.
18. FOR every dataset connected to the input ports
19.       SET the key as a string variable with the index of the dataset
20.       SET the value to be the initial address of dataset
21.       INSERT the pair into the DATASET_CLASS dictionary
22. ENDFOR
23.
24. //Create working directory where simulator occurs according to the WD_PATH
25. IF the folder which variable WD_PATH indicates does not exist
26.       CREATE the responding folder
27. ENDIF
28.
29. //Copy the global parameters configuration file to the working directory, which is created by last procedure.
30. BEGIN
31.       COPY GPF file into the folder created
32. EXCEPTION
33.       WHEN fails to copy files
34.             PRINT "Could not copy GPF file to folder"
35.  END
36.
37. //Pass the value of WD_PATH and pointer of dictionary for DATASET_CLASS to next modules.
38. OUTPUT WD_PATH and DATASET_CLASS
39.
40. END
```

Fig. 5. Pseudocode of *MainGenerator,* where the variables in bold and italic are to be configured by the user.

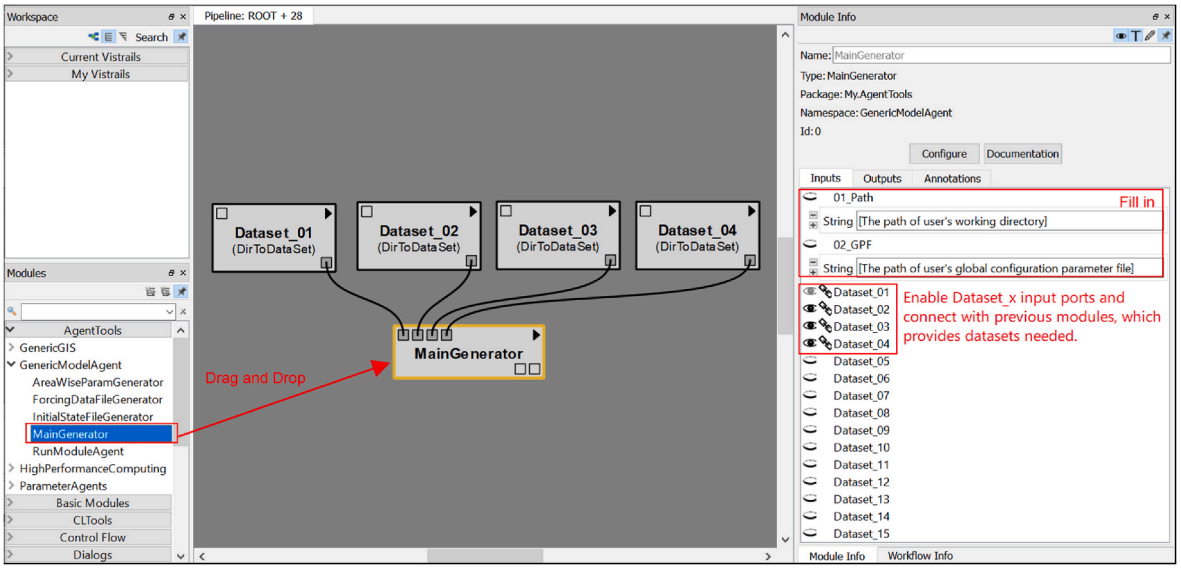


Fig. 6. User Interface of the *MainGenerator* module in Generic Model Agent toolkit with ***01_Path*** and ***02_GPF*** filled, and four datasets input ports enabled and connected with the data agent modules retrieving the four datasets.

a drainage area of 153 km^2 located in the southeast of Pennsylvania, United States. Note that all the model parameters in this simulation are default values that the user can further modify through a parameter calibration process.

5.1.1. *Workflow with manually developed VIC-model-agent*

We first show how to build a complete example of a model simulation workflow using the VIC model via a user's manually developed VIC-Agent. In general, creating a model simulation workflow starts with model input data preparation based on various data sources for a given study area over a given time period. Thus, the workflow starts first with a *TimeRange* module which is used to specify the simulation time extend. For this example, the simulation will go from 2010/01/01 00:00:00 (time start) to 2011/01/01 00:00:00 (time end); then a *SpaceRange* module is added to specify the study area with four bounds (i.e., east, west, north, and south) as: −75.58, −75.86, 40.24, 40.10 (x_max, x_min, y_max, y_min, respectively).

To access the forcing information required for this simulation, the *NCALDAS* Data Agent will be used. This agent accesses NASA NCALDAS[6] Earth Data database to automatically bring NASA's spatially organized daily hydrometric information to the local machine. Four NCALDAS-Agent boxes are added into the workflow area. Each one of these four new boxes will be used to retrieve a different variable of the forcing data. The four forcing variables required to execute VIC4 include: Wind Speed [m/s], Total Precipitation Rate [mm/s], Temperature Max [K], and Temperature Min [K].

Now, bring the VIC4 model into the workflow. The user's model is added into the workflow through its model agent. So, the *VICAgent* module developed by us (Salas et al., 2020) is added to the workflow area. Basically, this *VICAgent* enables the usage of the VIC model in the CyberWater workflow. It collects VIC forcing information either extracted via CyberWater's Data-Agents (e.g., Wind Speed and Precipitation) online or reads in the forcing data from files locally provided by the user. It also reads in all the parameter files provided by the user to run the VIC model. If these parameter files are not provided, the *VIC-Agent* will generate all the parameter files required for the simulation using default values automatically. Values of the parameters in these files can later be modified or calibrated by the user for re-running the VIC model. The *VICAgent* also provides users the flexibility of selecting their desired output variables from the *VICAgent* interface for visualizing them through the workflow execution. The workflow chart shown in Fig. 7 represents the main functionalities accomplished by the *VICAgent* code.

Note that VIC, like any other model, requires inputs to be included in specific units, depending on the individual variables. Details of such input information are included in the *VICAgent* documentation available via the Documentation button in CyberWater. For example, the temperature must be in Celsius, and for this case study, the precipitation will need to be in mm per day, since the expected results will be daily. Thus, the user only needs to check if the units of the retrieved data from data providers match the units required by the user's model. If not, unit transformations are needed. The *msmUnitConversion* module can be used to perform such transformations. For instance, to convert the temperature from the *NCALDAS* dataset in Kelvin into Celsius, the user needs to set the input "*operation*" to be "*x-273.15*". This "*operation*" allows the user to execute simple mathematical operations using "x" as the variable representing the dataset given as an input. To transform the input precipitation data from mm per second to mm per day, for example, the user just needs to set the "*operation*" to "*x*86400*" by creating another *msmUnitConversion* box. This module also allows the user to indicate the new resulting units of the transformed dataset. The last step for creating this workflow is to set up visualization tools, which will allow the user to see the model simulated results, for example, the baseflow and surface runoff. This visualization can be achieved by adding two "*msmShowChart*" boxes into the workflow and naming them "*Baseflow*" and "*Surface Runoff*", respectively. The completely created workflow for this VIC simulation case is shown in Fig. 8.

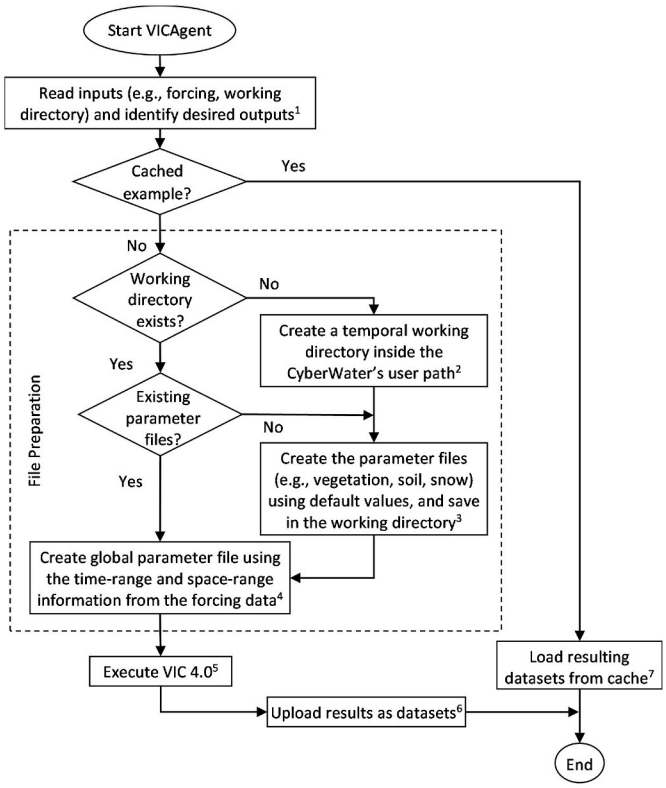


Fig. 7. Flowchart of the *VICAgent* that is manually programmed. The number of lines of code for each respective element in the flowchart is: (1) 78, (2) 9, (3) 269, (4) 184, (5) 12, (6) 115, (7) 56. Total number of lines of code is 723.

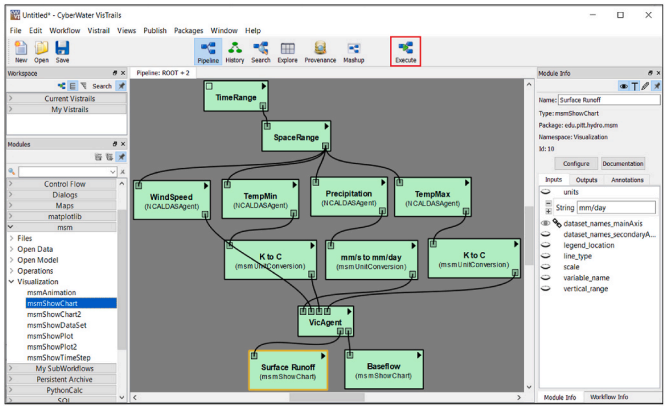


Fig. 8. A created workflow for a VIC simulation with its results plotted as time series using CyberWater. Execution of the workflow starts when the user clicks the "Execute" button. The green color displayed in each module of the workflow indicates that the workflow is successfully executed.

After executing this workflow, two charts, shown in Fig. 9(a) and Fig. 9(b), will be prompted. This example produces outputs of these two variables offered by default in the VIC Agent. However, if the user goes to the Outputs tab in the Module Info panel, more model output results available to be displayed are listed.

5.1.2. *Constructing VIC4 model agent using generic model agent toolkit*

This section shows how to use the generic model agent toolkit

[6] https://ldas.gsfc.nasa.gov/nca-ldas/forcing.

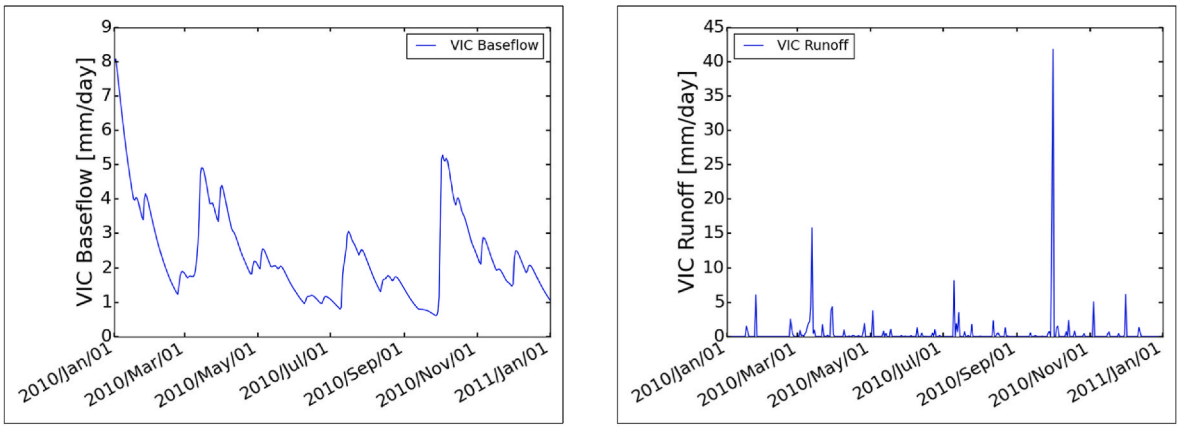


Fig. 9. VIC4 model simulated results of the French Creek watershed obtained using the *VicAgent*: **(a)** baseflow time series, (**b**) surface runoff time series.

presented in Sections 3 and 4 to construct/configure the VIC4 model agent for the model integration instead of manually writing the VIC4 model agent code. The important benefit here is that a user can integrate the user's own model into the CyberWater environment using this generic toolkit with little or no programming effort. To use this toolkit, users are only expected to be knowledgeable about their own models' inputs and outputs organizations. Our following example of the VIC4 model illustrates a simulation for the same period of time over the same study area described in Section 5.1.1. Thus, the time range, space range, and forcing data construction part of the workflow is the same as those described in Section 5.1.1. Here, we focus on steps to construct the VIC4 model agent using the toolkit.

5.1.2.1. MainGenerator. To start constructing the VIC model agent, first, bring a *MainGenerator* box, the main control component of the generic model agent toolkit, into the workflow working area. This component is responsible for setting up the working directory for the model simulation where it receives all the forcing datasets as inputs. The users can adjust the paths accordingly for their specific folder structure for the working directory. For this example, this working directory folder will be saved on "C:\temp\CYBERWATER\GT\VIC4\French_Creek"; thus, the inputs of *Path* and *GPF* can be set up, respectively, as: C:\temp\CYBERWATER\GT\VIC4\French_Creek\MainAgent and C:\temp\CYBERWATER\GT\VIC4\French_Creek\Source\vic_global_file_val. In addition, all forcing data are collected by the *MainGenerator* by connecting the forcing data modules to this *MainGenerator*. The working directory path information will be passed (via its WD_Path output port) to all the other components in the toolkit to be used for constructing the user's model agent.

5.1.2.2. AreaWiseParamGenerator. Add an *AreaWiseParamGenerator*, which is in charge of setting up the parameter files (e.g., soil parameter files and vegetation parameter files) of the model. These files need to be previously created by the user. The user needs to enter the information of the file's path for each of the parameter files required as the model's inputs in the *AreaWiseParamGenerator* module. For this example, the files used are stored in the folder: <C:\temp\CYBERWATER\GT\VIC4\French_Creek\Source >.

5.1.2.3. ForcingDataFileGenerator. Add a *ForcingDataFileGenerator*, which is responsible for the creation of the forcing data files of the model. This component takes the forcing information gathered from the *MainGenerator* and saves it into the user's specified folder.

5.1.2.4. InitialStateFileGenerator. Add an *InitialStateFileGenerator*. This module is responsible for placing the initial state files in the right folder in the working directory.

5.1.2.5. RunModuleAgent. Add the *RunModuleAgent*. It is responsible for setting up the path where CyberWater can locate the model's executable file (e.g., C, Fortran, Python, Java, and MATLAB which have been tested so far). Also, the arguments of the execution, as well as the model output variables selected, and their formats are configured here. The user's model can only be executed when all of its inputs are ready, which is guaranteed by connecting the output port of each of the three previous components (i.e., *AreaWiseParamGenerator*, *ForcingDataFileGenerator*, and *InitialStateFileGenerator*) to the *Ready_List* input port of the *RunModuleAgent*.

As in Section 5.1.1, the user can now add the *msmShowChart* modules to plot the surface runoff and the baseflow for viewing. After adding two *msmShowChart* modules to plot the Output01 (Surface Runoff) and Output02 (Baseflow) ports of the *RunModuleAgent*, the user will be able to execute the created workflow shown in Fig. 10. The VIC4 model simulated baseflow and surface runoff results obtained using the generic model agent toolkit are exactly the same as shown in Fig. 9.

5.1.3. Coupling the routing agent with the VIC4 model simulation using generic model agent toolkit

The generic model agent toolkit can also be used to couple different models in the workflow. To illustrate, we use the outputs of the VIC model as the inputs to the routing model. This routing model takes datasets of the surface and subsurface runoff (i.e., baseflow) as inputs (through the *MainGenerator* and the *ForcingDataFileGenerator*) and computes the resulting streamflow of a given watershed following the Muskingum method. Other required input parameter files are provided to the routing model through the *AreaWiseParameterGenerator* module, which are prepared offline, including those related to geographic information. The routing model's core functionalities are compiled in a *.jar file, which requires the presence of Java in the local machine to be able to work. The *RunModuleAgent* component takes this *.jar file through the "**exe**" input as "java -jar 'jar_compiled_model_file.jar'". The workflow of coupling these two models using the generic model agent toolkit is given in Fig. 11. The simulated streamflow results from the coupled models are depicted in Fig. 12.

5.1.4. Constructing model agent for VIC5

The more recent available version of the VIC model is VIC5 (VIC version5.0) which has some significant differences in the model's I/O from those in the VIC4 version. Consequently, the model agent previously developed or coded for VIC4 no longer works for VIC5 (v5.0). However, with the generic model agent toolkit, a user can easily reconstruct a new model agent for VIC5. The following example

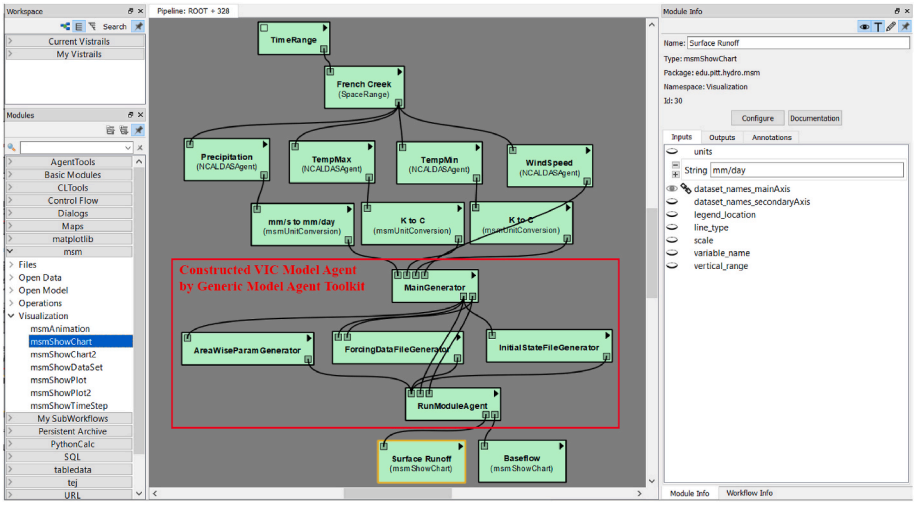


Fig. 10. Workflow where the French Creek watershed's fluxes are simulated using the generic model agent toolkit to execute the VIC4 model.

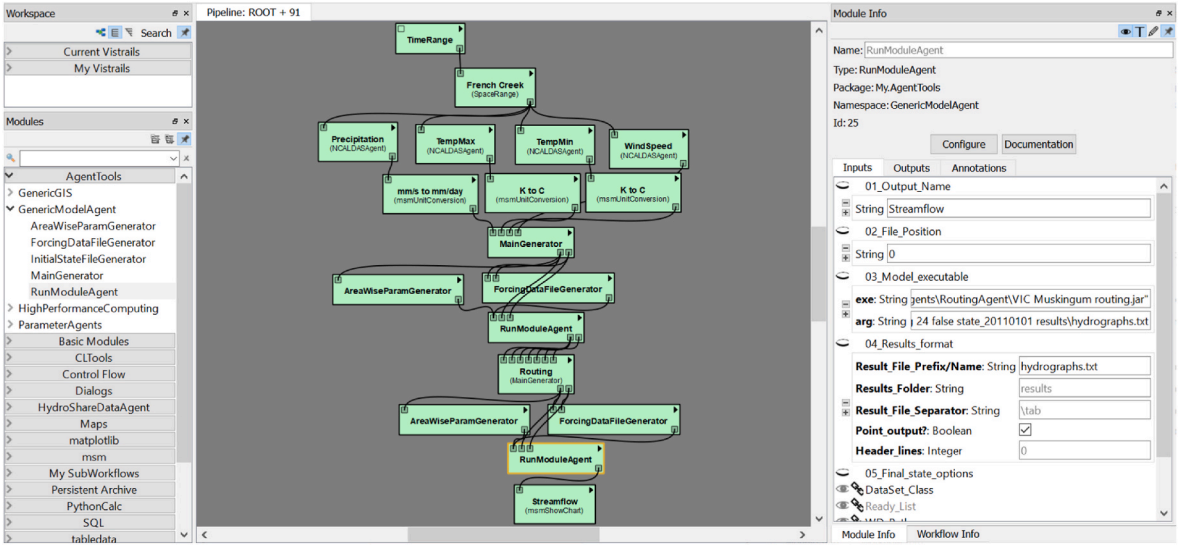


Fig. 11. Workflow with the complete setup of using the generic model agent toolkit to perform the VIC model and routing model coupling execution.

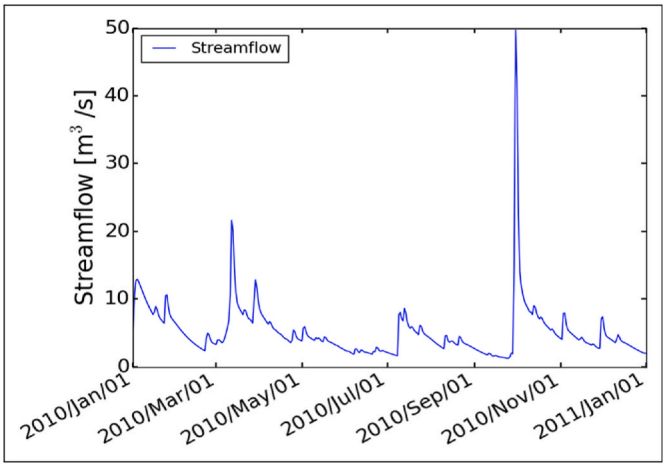


Fig. 12. Simulated streamflow time series of the French Creek watershed in PA from the coupled VIC and routing models using the generic model agent toolkit.

illustrates the construction of the VIC5 model agent and its associated workflow to simulate the water and energy fluxes for a different and large watershed using the CyberWater system.

In this example, an hourly model simulation of a large watershed inside the state of Pennsylvania is conducted. The watershed is the West-Branch Susquehanna (WBS) river basin covering more than 17,700 square kilometers. This river basin is selected to illustrate that Cyber-Water can scale up to handle large watersheds. In this case, a new VIC Agent is constructed using the generic model agent toolkit and is executed in an energy-balance mode using VIC5. This watershed includes 299 modeling cells at 1/8th-degree resolution. Like the French Creek watershed example, all the parameter values used in this simulation are default values that can be modified later by the user to perform parameter calibrations.

The simulation is set up for a period of time between 2010/01/01 00:00:00 and 2010/03/01 00:00:00. Since this simulation is hourly, it requires more than 1,400 data files per forcing variable as each data file represents 1 h (i.e., one time step) covering the entire watershed with 299 modeling cells at a spatial resolution of 1/8° per cell. In other words, each forcing data file is a map with 299 cells representing 1 h time step to be automatically retrieved from NASA. The space range of the WBS

watershed is given by the limits: −76.213, −78.9155, 41.933, and 40.454 (i.e., x_max, x_min, y_max, y_min, respectively). Eight *NLDAS-Agent* boxes are used to bring hourly forcing data from the NASA data source into CyberWater for the following eight variables: Temperature [K], Pressure [Pa], Radiation Flux Long Wave [W/m^2], Radiation Flux Short Wave [W/m^2], Total Precipitation [mm/h], Specific Humidity [kg/kg], U-Wind [m/s], and V-Wind [m/s].

CyberWater offers the *msmDatasetOperation* module, facilitating performing operations between two datasets. Thus, it can be used to compute the magnitude of the wind speed based on information of the two components associated with two directions of the wind speed, i.e., U-Wind and V-Wind, given by the NASA data source. For this example, add the *msmDatasetOperation* module to compute the magnitude of the wind speed $(x^2 + y^2)^{1/2}$ required as a forcing variable by VIC5, where x and y represent obtained U-Wind and V-Wind from the NASA data source respectively. Then, use the *msmUnitConversion* to transform the units of the Temperature and Pressure datasets. In addition, the energy-balance mode of VIC requires the user to provide Water Vapor Pressure as a forcing variable. To do so, the user needs to read in the specific humidity information provided by NLDAS data product using the *NLDASAgent*, then convert it to vapor pressure using the equation below as an approximation,

$$VP = 1.61 \cdot q_h \cdot P \tag{1}$$

where VP is the water vapor pressure [kPa], q_h is the specific humidity [Kg/Kg], and P is the pressure [kPa]. This operation can be performed using again the *msmDatasetOperation* module.

5.1.4.1. MainGenerator. After the data preparation, one is ready to construct the VIC5 model agent using the generic model agent toolkit. First, bring a *MainGenerator* box, the main control component of the generic model agent toolkit, into the working area. Then, have each required input information provided in a similar way as it is illustrated in Section 5.1.2. Note that the order of the input Datasets is determined by the order given in the VIC model's global file, "vic_global_file_val".

CyberWater-VisTrails offers a nice feature that allows the user to group multiple modules to facilitate displaying a larger workflow. Select all the modules between *SpaceRange* and *MainGenerator* and go to the Workflow −> Group menu on the top toolbar as shown in Fig. 13 to group these modules.

5.1.4.2. AreaWiseParamGenerator. Add an *AreaWiseParamGenerator* to set up the parameter files (e.g., soil parameter files and vegetation parameter files) for the model. These files need to be previously created by the user before using the *AreaWiseParamGenerator* module. Again, the user needs to provide the required information in a similar way as it is illustrated in Section 5.1.2 for the *AreaWiseParamGenerator* module.

5.1.4.3. ForcingDataFileGenerator. Add a *ForcingDataFileGenerator* for taking care of the forcing data file preparation for the model. This component takes the forcing information collected from the *MainGenerator* and saves it into the user specified folder.

5.1.4.4. InitialStateFileGenerator. Add an *InitialStateFileGenerator* to place the initial state files into the right folder in the working directory.

5.1.4.5. RunModuleAgent. Add the final component, the *RunModuleAgent*, to complete the construction of the VIC5 model agent. It is responsible for setting up the path where *the executable file of the model is located*. Also, the arguments of the execution, as well as the selected outputs, should be configured here.

Now, the user can add the *msmShowChart* module to plot the surface runoff and baseflow for visualization. After adding the two *msmShowChart* modules to plot the Output01 (Surface Runoff) and Output02 (Baseflow) ports of the *RunModuleAgent*, the user will be able to execute the workflow shown in Fig. 14. The two time-series obtained are depicted in Fig. 15(a) and Fig. 15(b).

5.2. DHSVM model simulation

The Distributed Hydrology Soil Vegetation Model (DHSVM) is a high-resolution hydrological model that simulates water and energy fluxes by considering the effects of terrain features (Wigmosta et al., 1994). One of the main differences in the modeling structures between VIC and DHSVM is that in the latter, terrain or topography information is needed as routing process is part of DHSVM, i.e., the routing process is internally coupled to other processes inside DHSVM. In this use case, we illustrate simulations of the DHSVM model, through both its manually developed *DHSVMAgent,* and the agent constructed using the generic model agent toolkit, separately. The Indiantown Run basin, a small watershed in Pennsylvania, is used to illustrate these features. Seven types of forcing data are required for DHSVM: pressure, precipitation, relative humidity, temperature, wind speed, longwave radiation, and shortwave radiation. Since the simulation will be performed at an hourly time step, the NLDAS data agent will be used again. However, this data source does not provide relative humidity but the specific humidity. Thus, one needs to calculate the relative humidity based on the other given forcing data, employing an approach similar to that used in the VIC5 example with the *msmDatasetOperation* module.

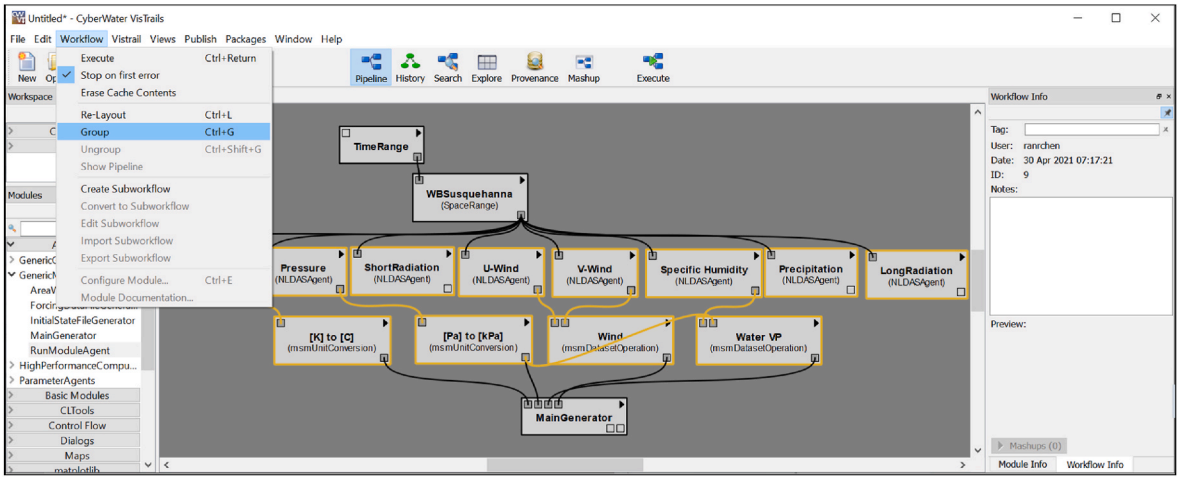


Fig. 13. Illustration of the grouping feature used to group modules related to the forcing data for the VIC5 model simulation.

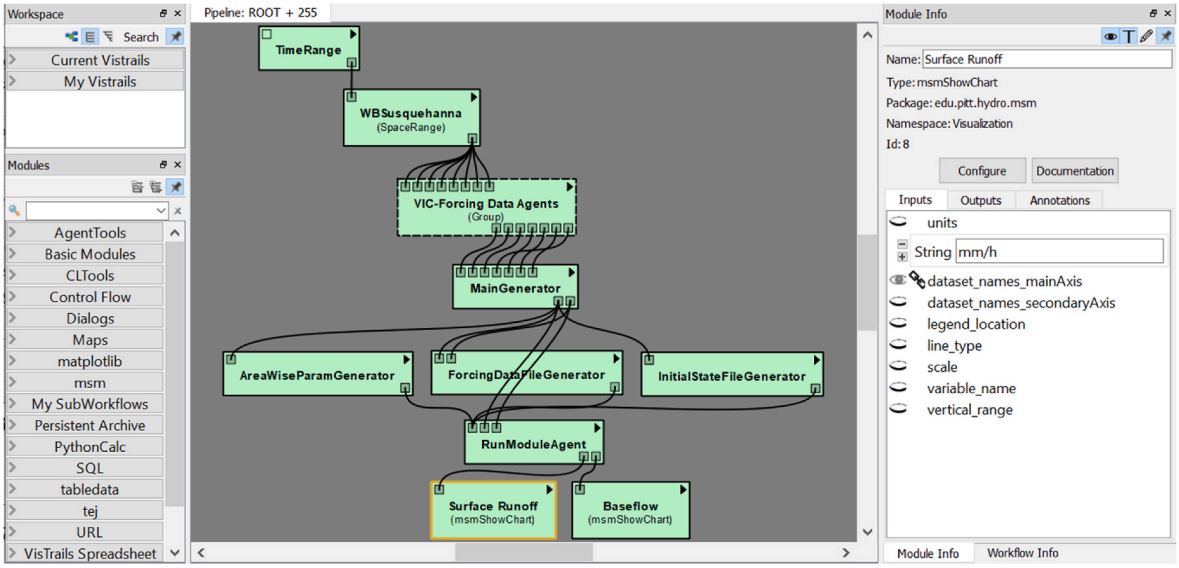
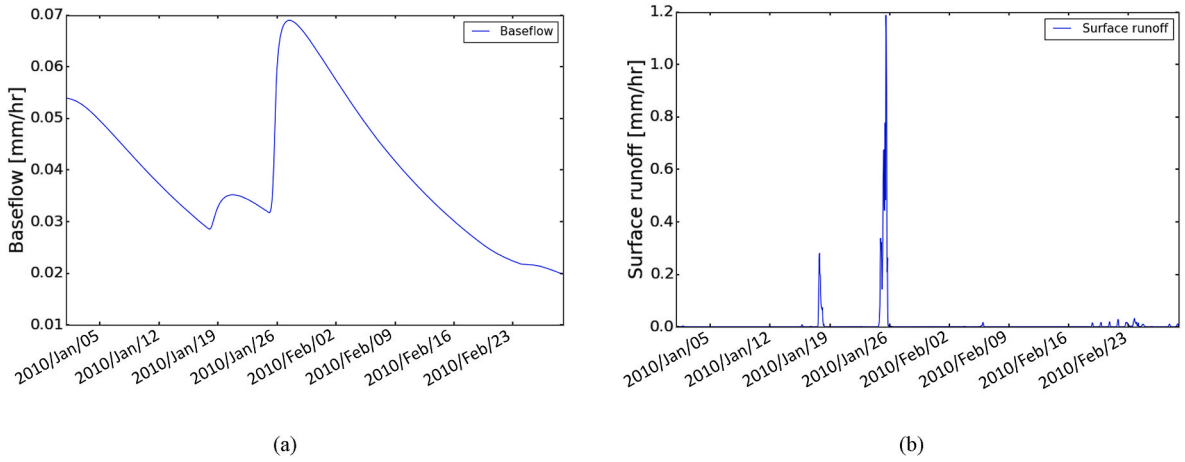


Fig. 14. A successfully executed workflow for VIC5 where the WBS river basin's fluxes are simulated in CyberWater using the generic model agent toolkit to build the model agent without coding. The dashed line box of VIC-Forcing Data Agents in the workflow is the grouped portion of workflow including multiple data agents and their related unit conversions and dataset operations as illustrated in Fig. 13.



(a) (b)

Fig. 15. VIC5 model simulated results of the WBS river basin using the generic model agent toolkit: **(a)** baseflow, and **(b)** surface runoff.

5.2.1. Constructing DHSVM model agent using generic model agent toolkit

In this section, we show how to integrate the DHSVM model by using the generic model agent toolkit and escape steps that are similar to those described before.

To construct a workflow running the DHSVM model, the user needs to add *TimeRange*, *SpaceRange*, *NLDASAgent* modules like before, and use *msmDatasetOperation* and *msmUnitConversion* modules to calculate certain forcing input variables based on available input data and to convert units. For this example, the user needs to use eight *NLDASAgent* boxes to retrieve eight different hourly forcing variables which are Temperature [K], Pressure [Pa], Radiation Flux Long Wave [W/m^2], Radiation Flux Short Wave [W/m^2], Specific Humidity [kg/kg], Total Precipitation [mm/h], U-Wind [m/s] and V-Wind [m/s], and then to compute the relative humidity and magnitude of wind required by DHSVM.

To run the DHSVM model one needs to use GIS[7] (Geographic Information System) tools to prepare data for the routing related processes, like those involved in the routing model described in subsection 5.1.3, as these routing related processes are part of the DHSVM model. One can either prepare these GIS related files offline and have them inputted through the *AreaWiseParameterGenerator* as illustrated in subsection 5.1.3 or prepare them as part of the overall workflow as in this example. To facilitate the in-workflow preparation of these GIS related files required for the integration of models, such as the DHSVM model, CyberWater has incorporated GRASS GIS,[8] an open-source GIS software, in its *msm* system. A module called *GISEngine* is developed to interface between the *msm* system and the GRASS GIS system. Through this *GISEngine* module and other GIS related modules developed in CyberWater, one can automatically and seamlessly access the GRASS GIS system, and easily and effectively make use of the various functionalities offered by GRASS GIS to accomplish the user's various tasks, such as identifying the flow directions, flow paths, and the stream network within the studied area, given its DEM[9] (Digital Elevation Model) map file. The resulting files from GIS are then automatically passed back to the *msm* system. Thus, all the input files required to be produced by GIS

[7] https://en.wikipedia.org/wiki/Geographic_information_system.

[8] https://grass.osgeo.org/.

[9] https://en.wikipedia.org/wiki/USGS_DEM.

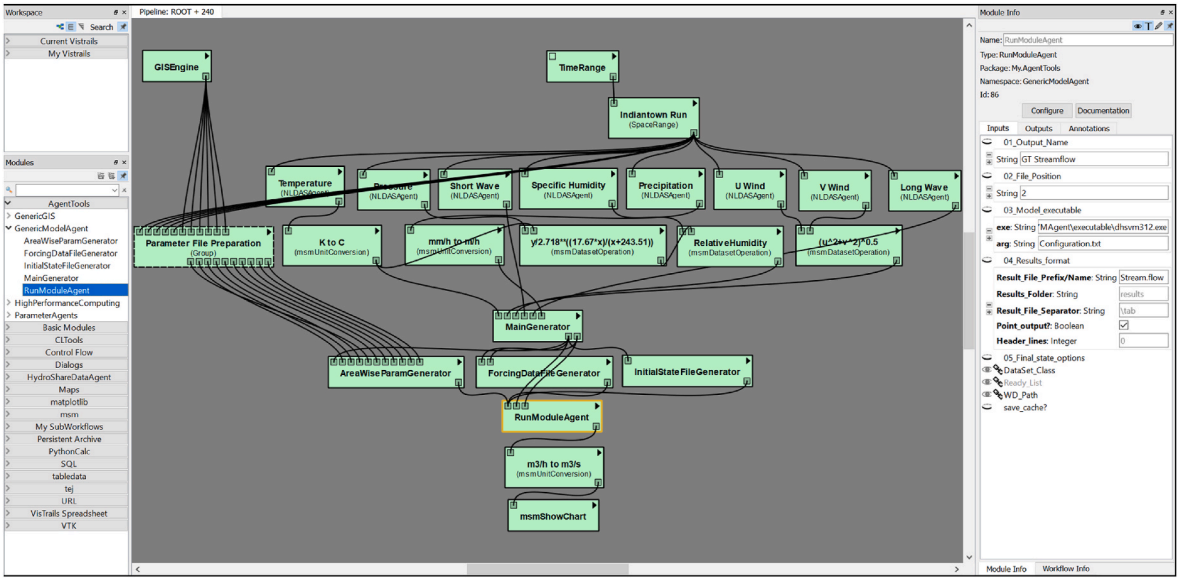


Fig. 16. Full workflow of DHSVM model simulation via the constructed model agent using the generic model agent toolkit.

can be obtained using our developed GIS related modules (Luna et al., 2020, 2021a). Details on the integration of GRASS GIS and its relevant modules developed in the CyberWater system are presented in Luna et al. (2021b) and will be discussed in detail in a separate paper (Luna et al., 2022). In this example, most of the GIS-related input files are automatically prepared and are passed to the *AreaWiseParameterGenerator* as indicated by the dashed line box "Parameter File Preparation" in Fig. 16 which is achieved by using our developed *GISEngine* and the static parameter agent modules (Luna et al., 2021b, 2022) which are grouped together in the "Parameter File Preparation" box and described below.

5.2.1.1. Parameter files preparation. This preparation mainly includes the following major steps:

1) Use four *StaticBinaryMapAgent* modules which are configured with the soil depth map, soil map, vegetation map, and DEM map paths, respectively;
2) Use a *GisDefineWatershed* module, configured with the path of the DEM map;
3) Use a *StreamMapAndRoutingFiles_DHSVM* module, configured with the north coordinator, south coordinator, west coordinator, east coordinator as mask limits;
4) Group all the modules above in a module box for a neat and concise workflow outlook, and name these grouped components as "Parameter File Preparation", as shown in Fig. 16.

5.2.1.2. DHSVM model agent construction with the toolkit.

1) Add a *MainGenerator* module and create a new working directory folder on the user's computer, where the DHSVM simulation will occur. For this example, this folder will be saved on "C:\temp \CYBERWATER". The inputs are set up in a similar way as described in subsection 5.1.4 for VIC5, but the variables corresponding to each dataset are different since they are determined by the model's global file or configuration file.

2) Add an *AreaWiseParamGenerator* module, and the inputs are set up in a similar way as described in subsection 5.1.4 for VIC5.

3) Add *ForcingDataFileGenerator* to take care of the forcing data file preparation for the DHSVM model. This component takes the forcing information collected from the *MainGenerator* and saves it into the user specified folder.

4) Add an *InitialStateFileGenerator* module, and let the path be the folder path of "C:\temp\CYBERWATER\GT\DHSVM\IRun10years \Source".

5) Add a *RunModuleAgent* module, and its inputs are set up in a similar way as described in subsection 5.1.4 for VIC5.

The newly created workflow after execution is shown in Fig. 16. The resulting plot after the execution of this workflow is depicted in Fig. 17. In contrast, Fig. 18 shows the complete workflow if the user employs the manually written DHSVM model agent for integrating the DHSVM model.

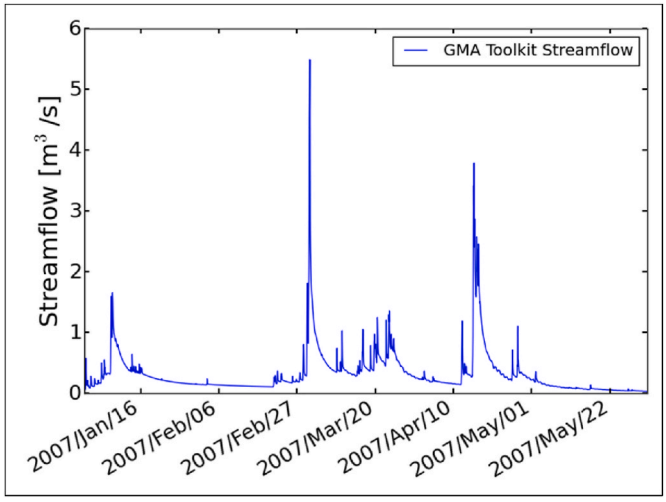


Fig. 17. The DHSVM model simulated streamflow time series of the Indiantown Run watershed obtained using the generic model agent toolkit.

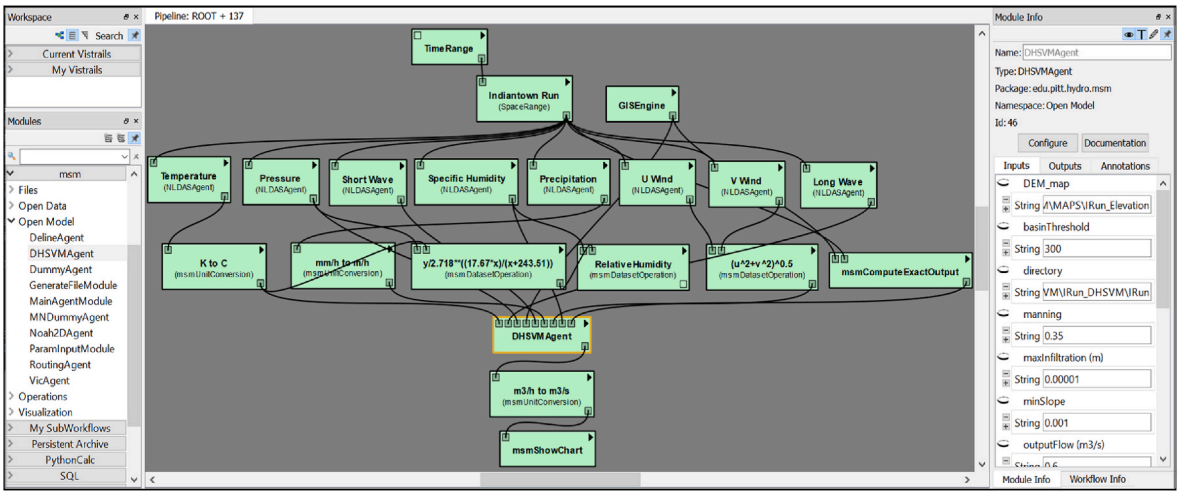


Fig. 18. A complete workflow to execute DHSVM model simulation in CyberWater via user's programed *DHSVMAgent*.

5.3. Discussion

Based on our investigation of the popular hydrological and environmental models, it shows that the complexity of these models' interface structures often lies in the organizations of models' input files with many different options/types (e.g., soil and vegetation related libraries and parameter files, snow related information, and GIS related information), rather than the organizations and structures of the models' output files. Thus, we designed three components to handle a given model's input interface organizations, i.e., *AreaWiseParamGenerator*, *ForcingDataFileGenerator*, and *InitialStateFileGenerator*, so that each component would not be too complicated for users to configure. These three components correspond to the three major types that a typical model would have. On the other hand, as the model's output interface organizations and structures are usually straightforward, e.g., values of computed state variables, fluxes, and input information, organized either in time series or spatial maps, we thus combined the receiving of the model's outputs into the component handling the model's execution (e.g., *RunModuleAgent*), rather than creating a dedicated component for that. Furthermore, the organizations and structures of the computational models' interfaces can be briefly classified into the following two categories: (1) The model itself does not include the use of GIS related information and/or functionalities, such as VIC; (2) The model itself utilizes GIS related information and/or functionalities as well, such as the routing model, and DHSVM where the routing processes are included. The former does not directly involve GIS to handle its area-wise parameter files, while the latter usually directly involves the use of GIS for handling its area-wise parameter files. To facilitate the latter case, the GRASS GIS is integrated into the CyberWater system via our developed *GISEngine* and its related modules; in addition, the static parameter agent modules are developed to automatically prepare GIS-related input files for such models, which will be discussed in a separate paper (Luna et al., 2022). Once the GIS-related input files are created, they are passed to the *AreaWiseParameterGenerator* as indicated in the use case of subsection 5.2.1.

We note that among the three types of the input data files, the structure and format interfaces of the forcing data files are less heterogeneous and complex compared to those of the parameter files and initial state files among the different hydrological and environmental models. For the forcing data files, the majority of the models (e.g., VIC, DHSVM, Routing, and CASA-CNP) have their formats fall into two types, namely, single-point time series or spatio-temporal maps. These two types are both taken care of inside CyberWater's *ForcingDataFileGenerator* module which also allows for customizable time-stamps for the data. In addition, the *ForcingDataFileGenerator* module allows the users to modify the order of the columns inside the forcing data files to represent different forcing variables read in so that the order of the variables involved in the forcing data files can be adjusted to match the structure requirements of different models' forcing data files. For the input parameter files and initial state files, however, interfaces of the structures and formats of the individual files are then taken care of by the static parameter agent modules developed in CyberWater (Luna et al., 2021b, 2022).

We also note that the generic model agent toolkit not only makes the integration of one's model an easier task, but also significantly saves user's time and efforts and reduces the error-prone process of writing the model agent codes, since one does not need to write any code to integrate a model into the CyberWater system. By manually writing a model agent to integrate a model, one needs to not only write codes initially, but also to re-write the codes whenever the model's I/O interface changes as shown in the example of using VIC4 (version 4) versus VIC5 (version 5). On the other hand, due to the existing large number of diverse types of models, it is possible that some unusual model structures and features might not be covered by the generic model agent toolkit we have developed. In such a case, the option of manually writing a model agent in CyberWater provides more flexibility. Consequently, CyberWater provides users with the flexibility to either use the generic model agent toolkit for most models or to manually write their own model agents when needed for any model with unusual I/O structures.

From the illustrations of the use cases in Sections 5.1 and 5.2, we can see the flexibility and straightforwardness of using the generic model agent toolkit in integrating various computational models with significantly different model input structures and formats without the need of writing any code. The strengths of using the visual drag and drop approach of the generic model agent toolkit to construct a model agent for integrating a model into the CyberWater system as opposed to the traditional approach of writing programming codes, e.g., the VICAgent and DHSVMAgent, include:

1) Usability (Ease of use): One can easily construct a model agent with the toolkit to integrate a model into CyberWater without writing computer codes, in which the functionalities and responsibilities of the individual modules are clear. The relationships/connections between different modules in the toolkit provide a clear overview and big picture of a model's data file structure/organization, logic, and its workflow processes, making model integration an easier task, reducing the error-prone code writing process, and saving time.
2) Clarity: the organizations and structures of a model's input/output files can be clearly seen from the five/six modules included in the generic model agent toolkit. For example, from the configuration

panel associated with each module, users can easily see where the model's input files are located and what they are. Such a neat visual presentation helps users identify missing information and debugging.

3) Generality: Our toolkit is applicable to a wide range of computational models since most of the models have similar input file organizations, e.g., global file/configuration file, parameter file, forcing data file, and initial state file. In fact, we have been successfully applied the toolkit to integrate other hydrological and environmental models with ease (Luna et al., 2020), such as the biogeochemical model – CASA-CNP (The Carnegie-Ames Stanford Approach model with Carbon, Nitrogen, and Phosphorous cycles) which computes the net primary productivity of terrestrial ecosystems, together with detailed distributions of the amount of Carbon, Nitrogen, and Phosphorous present in the biosphere and its surroundings (Wang et al., 2010), and the USGS environmental model – PHREEQC model which is used to calculate a variety of aqueous geochemical reactions and processes in natural waters or laboratory experiments (Parkhurst and Appelo, 1999).
4) Agility: Any changes in a model's interface can be easily accommodated with the toolkit. For example, when a model's inputs' structure is changed as shown in Section 5.1, VIC4 versus VIC5, one can easily reconstruct the model agent for VIC5 using the toolkit instead of rewriting hundreds of new codes to have a new VICAgent.

6. Conclusion

In the fields of sciences and engineering, the need for open data and open model integration is urgent. While our previous work on the open data and open modeling framework of MSM lays a solid foundation for effectively and efficiently addressing the challenge faced by the broad community, this new work on the development of the generic model agent toolkit enables modelers to construct their own model agents without coding, therefore further simplifying the data and model integration task by eliminating users' effort of writing model agent code. This is achieved by the innovative design and development of a suite of feature-oriented code templates that extract each individual model agent's characteristics, and therefore reduce the model agent programming work to the template-based components' configuration task. This approach of the generic model agent toolkit in CyberWater is capable of applying to diverse models developed in the broad community without modifying any code of the original models or writing any model agents' code. With CyberWater, users can individually and easily make their models pluggable into comprehensive end-to-end workflows with automated and seamless access to various online data sources for their scientific study without any need for a central control administration or service. Therefore, our presented approach not only provides a general and elegant solution for open data and open model integration for complex modeling systems, but also offers a truly sustainable and scalable solution for broad applications. The current limitation of the generic model agent toolkit is that it only works for grid-based models. As we know, there are hydrological models developed based on hillslope units or sub-watershed units, all of which have irregular modeling grids. Our future work is to extend the generic model agent toolkit to handle irregular grid-based models. We also plan to extend CyberWater to include web services to further complement its current standalone system, which can better support education in colleges and universities as most universities have very strict limitations for software installations on their laboratory computers.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the U.S. National Science Foundation under [OAC-1835817] to Indiana University-Purdue University Indianapolis (IUPUI), and [OAC-1835785] to the University of Pittsburgh, respectively.

References

Ahuja, L.R., Ascough, J.C., David, O., 2005. Developing natural resource models using the object modeling system: feasibility and challenges. Adv. Geosci. 4, 29–36. https://doi.org/10.5194/adgeo-4-29-2005.

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

Bandaragoda, C., Tarboton, D.G., Woods, R.A., 2004. Application of TOPNET in the distributed model intercomparison project. J. Hydrol. 298, 178–201. https://doi.org/10.1016/j.jhydrol.2004.03.038.

Bavoil, L., Callahan, S.P., Crossno, P.J., Freire, J., Scheidegger, C.E., Silva, C.T., Vo, H.T., 2005. VisTrails: enabling interactive multiple-view visualizations. Proc. IEEE Vis. Conf. 18 https://doi.org/10.1109/VISUAL.2005.1532788.

Belete, G.F., Voinov, A., Laniak, G.F., 2017. An overview of the model integration process: from pre-integration assessment to testing. Environ. Model. Software 87, 49–63. https://doi.org/10.1016/j.envsoft.2016.10.013.

Chen, M., Voinov, A., Ames, D.P., Kettner, A.J., Goodall, J.L., Jakeman, A.J., Barton, M. C., Harpham, Q., Cuddy, S.M., DeLuca, C., Yue, S., Wang, J., Zhang, F., Wen, Y., Lü, G., 2020. Position paper: open web-distributed integrated geographic modelling and simulation to enable broader participation and applications. Earth Sci. Rev. 207, 103223 https://doi.org/10.1016/j.earscirev.2020.103223.

Chen, M., Yue, S.S., Lu, G.N., Lin, H., Yang, C.W., Wen, Y.N., Hou, T., Xiao, D.W., Jiang, H., 2019. Teamwork-oriented integrated modelling method for geo-problem solving. Environ. Model. Software 119, 111–123. https://doi.org/10.1016/j.envsoft.2019.05.015.

Collins, N., Theurich, G., DeLuca, C., Suárez, M.J., Trayanov, A., Balaji, V., Li, P., Yang, W., Hill, C., Silva, A.M., 2005. Design and implementation of components in the Earth System Modeling Framework. Int. J. High Perform. Comput. Appl. 19, 341–350. https://doi.org/10.1177/1094342005056120.

David, O., Ascough II, J.C., Lloyd, W., Green, T.R., Rojas, K.W., Leavesley, G.H., Ahuja, L. R., 2013. A software engineering perspective on environmental modelling framework design: the Object modelling system. Environ. Model. Software 39, 201–213.

DeLuca, C., Theurich, G., Balaji, V., Valcke, S., Redler, R., Budich, R., 2012. The Earth System Modeling Framework. Earth System Modelling - Volume 3: Coupling Software and Strategies, pp. 43–54. https://doi.org/10.1007/978-3-642-23360-9_6, 978-3-642-23360-9.

Gan, T., Tarboton, D.G., Dash, P., Gichamo, T.Z., Horsburgh, J.S., 2020. Integrating hydrologic modeling web services with online data sharing to prepare, store, and execute hydrologic models. Environ. Model. Software 130, 104731. https://doi.org/10.1016/j.envsoft.2020.104731.

Gichamo, T.Z., Sazib, N.S., Tarboton, D.G., Dash, P., 2020. HydroDS: Data services in support of physically based, distributed hydrological models. Environ. Model. Software 125, 104625. https://doi.org/10.1016/j.envsoft.2020.104623.

Gregersen, J.B., Gijsbers, P., Westen, S., 2007. OpenMI: open modelling interface. J. Hydroinf. 9, 25847377 https://doi.org/10.2166/hydro.2007.023.

Hamman, J.J, Nijssen, B., Gergel, D.R., Mao, Y., 2018. The Variable Infiltration Capacity model version 5 (VIC-5): infrastructure improvements for new applications and reproducibility. Geosci. Model Dev. 11, 3481–3496. https://doi.org/10.5194/gmd-11-3481-2018.

Harpham, Q.K., Hughes, A., Moore, R.V., 2019. Introductory overview: the OpenMI 2.0 standard for integrating numerical models. Environ. Model. Software 122, 104549. https://doi.org/10.1016/j.envsoft.2019.104549.

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

Knapen, R., Janssen, S., Roosenschoon, O., Verweij, P., de Winter, W., Uiterwijk, M., Wien, J.-E., 2013. Evaluating OpenMI as a Model Integration Platform across Disciplines, vol. 39. Environmental Modelling & Software, pp. 274–282.

Li, F., Chen, R., Fu, Y., Song, F., Liang, Y., Ranawaka, I., Pamidighantam, S., Luna, D., Liang, X., 2021. Accelerating complex modeling workflows in CyberWater using on-demand HPC/Cloud resources. In: 2021 IEEE 17th International Conference on eScience, pp. 196–205. https://doi.org/10.1109/eScience51609.2021.00030, 978-1-6654-0361-0/21.

Liang, X., Lettenmaier, D.P., Wood, E.F., Burges, S.J., 1994. A simple hydrologically based model of land surface water and energy fluxes for general circulation models. J. Geophys. Res. Atmos. 99 (D7), 14415–14428. https://doi.org/10.1029/96JD01448.

Liang, X., Wood, E.F., Lettenmaier, D.P., 1996a. Surface soil moisture parameterization of the VIC-2L model: evaluation and modification. Global Planet. Change 13 (1–4), 195–206. https://doi.org/10.1016/0921-8181(95)00046-1.

Liang, X., Lettenmaier, D.P., Wood, E.F., 1996b. A one-dimensional statistical dynamic representation of subgrid spatial variability of precipitation in the two-layer variable infiltration capacity model. J. Geophys. Res. 101 (D16), 21,403-421,422.

Liang, X., Xie, Z., 2003. Important factors in land-atmosphere interactions: surface runoff generations and interactions between surface and groundwater. Global Planet. Change 38 (1–2), 101–114.
Luna, D., Chen, R., Li, F., Ranawaka, I.J., Pamidighantam, S., Song, F., Liang, Y., Liang, X., 2020. Advances in CyberWater Development—An Open Source Framework for Data and Model Integration. American Geophysical Union Fall Meeting, AGU Fall Meeting (virtual). Dec. 1-17, 2020.
Luna, D., Chen, R., Young, R., Liang, Y., Liang, X., 2021a. CyberWater Manual (V6), p. 87. August 2021.
Luna, D., Chen, R., Young, R., Liang, Y., Liang, X., 2021b. Static Parameter Agents of CyberWater Framework for Geoscience Data and Model Integration. American Geophysical Union Fall Meeting (hybrid), New Orleans, LA, USA. Dec. 13-17, 2021.
Luna, D., Chen, R., Young, R., Liang, Y., Liang, X., 2022. Generic Parameter Agent Toolkit of CyberWater Framework for Geoscience Data and Model Integration, p. 2022 in preparation.
Moore, R., Tindall, I., 2005. An overview of the open modelling interface and environment (the OpenMI). Environ. Sci. Pol. 8 (3), 279–286. https://doi.org/10.1016/j.envsci.2005.03.009.
Parkhurst, D.L., Appelo, C.A.J., 1999. User's guide to PHREEQC (version 2)–A computer program for speciation, batch-reaction, one-dimensional transport, and inverse geochemical calculations. U.S. Geological Survey. Water-Resources Investigations Report 99–4259, 4312.
Peckham, S.D., Hutton, E.W.H., Norris, B., 2013. A component-based approach to integrated modeling in the geosciences: the design of CSDMS. Comput. Geosci. 53, 3–12. https://doi.org/10.1016/j.cageo.2012.04.002.
Rahman, J.M., Seaton, S.P., Perraud, J.M., Hotham, H., Verrelli, D.I., Coleman, J.R., 2003. It's TIME for a new environmental modeling framework. In: Post, D.A. (Ed.), MODSIM 2003 International Congress on Modeling and Simulation. Modeling and Simulation Society of Australia and New Zealand Inc, Townsville, pp. 1727–1732.
Salas, D., Liang, X., Navarro, M., Liang, Y., Luna, D., 2020. An open-data open-model framework for hydrological models' integration, evaluation and application. Environ. Model. Software 126, 104622. https://doi.org/10.1016/j.envsoft.2020.104622.
Stenson, M.P., Littleboy, M., Gilfedder, M., 2011. Estimation of water and salt generation from unregulated upland catchments. Environ. Model. Software 26 (11), 1268–1278.
Tarboton, D.G., Idaszak, R., Horsburgh, J.S., Heard, J., Ames, D., Goodall, J.L., Band, L., Merwade, V., Couch, A., Arrigo, J., Hooper, R., 2014. HydroShare: advancing collaboration through hydrologic data and model sharing. Int. Congr. Environ. Model. Softw., no. JUNE 2014.
University of Colorado Boulder, 2012. CSDMS final report. https://csdms.colorado.edu/mediawiki/images/2012_CSDMS_semiannual_report.pdf.
Wang, J., Chen, M., Lü, G.N., Yue, S.S., Wen, Y.N., Lan, Z.X., Zhang, S., 2020. A data sharing method in the open web environment: data sharing in hydrology. J. Hydrol. 587 https://doi.org/10.1016/j.jhydrol.2020.124973.
Wang, Y.P., Law, R.M., Pak, B., 2010. A global model of carbon, nitrogen, and phosphorus cycles for the terrestrialbiosphere. Biogeosciences 7, 2261–2282. https://doi.org/10.5194/bg-7-2261-2010.
Wigmosta, M.S., Vail, L.W., Lettenmaier, D.P., 1994. A distributed hydrology-vegetation model for complex terrain. Water Resour. Res. 30 (6), 1665–1679.
Zhang, F.Y., Chen, M., Ames, D.P., Shen, C.R., Yue, S.S., Wen, Y.N., Lu, G.N., 2019. Design and development of a service-oriented wrapper system for sharing and reusing distributed geoanalysis models on the web. Environ. Model. Software 111, 498–509.