Environmental Modelling and Software 126 (2020) 104622

Contents lists available at ScienceDirect

Environmental Modelling and Software

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

An open-data open-model framework for hydrological models' integration, evaluation and application

Daniel Salas [a], Xu Liang [a,*], Miguel Navarro [b], Yao Liang [b,**], Daniel Luna [a]

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

ARTICLE INFO

Keywords:
Open architecture
Scientific workflow
Data-source integration
Model coupling
Machine-to-machine automation
Reproducible process

ABSTRACT

To tackle fundamental scientific questions regarding health, resilience and sustainability of water resources which encompass multiple disciplines, researchers need to be able to easily access diverse data sources and to also effectively incorporate these data into heterogeneous models. To address these cyberinfrastructure challenges, a new sustainable and easy-to-use Open Data and Open Modeling framework called Meta-Scientific-Modeling (MSM) is developed. MSM addresses the challenges of accessing heterogeneous data sources via the Open Data architecture which facilitates integration of various external data sources. Data Agents are used to handle remote data access protocols, metadata standards, and source-specific implementations. The Open Modeling architecture allows different models to be easily integrated into MSM via Model Agents, enabling direct heterogeneous model coupling. MSM adopts a graphical scientific workflow system (VisTrails) and does not require re-compiling or adding interface codes for any diverse model integration. A study case is presented to illustrate the merit of MSM.

Software/data availability

Name	Author	Year first available	Size	Data format	Access
VisTrails	New York University	2007	300 Mb	Python language	https://www.vistrails.org/index.php/Main_Page
GES DISC through OPeNDAP	NASA	2011		OPeNDAP	https://developer.earthdata.nasa.gov/opendap/quickstart
USGS Water Service	USGS			Xml webservices	http://waterservices.usgs.gov/
NOAA Radar Precipitation	NOAA	2004		Shapefile	https://water.weather.gov/precip/archive/ https://water.weather.gov/precip/downloads/

1. Introduction

1.1. Context and motivation

Models are developed for various applications, such as predictions of weather, floods, landslides, erosions, droughts, water quality, air quality, climate variability, evolution of landscape, and pollutant transport within soil column and rivers. To improve our understanding of the complex behaviors of the various processes (e.g., physical, hydrological, atmospheric, biological, chemical) and the interactions among them, as well as to improve the accuracy and reliability of the models' predictions and simulations, researchers need to make an effective use of various

Abbreviations: MSM, Meta Scientific Modeling; MKS, Multiscale Kalman Smoother; NOAA, National Oceanic and Atmospheric Administration; OPeNDAP, Open-source Project for a Network Data Access Protocol; VIC, Variable Infiltration Capacity; XDAC, Data-Agent-Component Service that offers the services to access external data making use of Data Agents. It returns DataSets as its results.

* Corresponding author.
** Corresponding author.
E-mail addresses: dsalas0@gmail.com (D. Salas), xuliang@pitt.edu (X. Liang), mignavar@iupui.edu (M. Navarro), yaoliang@iupui.edu (Y. Liang), del47@pitt.edu (D. Luna).

https://doi.org/10.1016/j.envsoft.2020.104622
Received 17 September 2019; Received in revised form 5 January 2020; Accepted 6 January 2020
Available online 12 January 2020
1364-8152/

available data across disciplines to improve theories, algorithms, models, and validations. However, a large amount of such valuable data often goes unused, due to the significant overhead of time and effort needed to discover, access, understand, and process these heterogeneous data (Kannan et al., 2000; McPhillips et al., 2009; Parada and Liang, 2004; Ravindran et al., 2010; Salas and Liang, 2013). While a number of modeling systems exist for scientific communities (Bhat and Jadhav, 2010; Bryant, 2007; DeLuca et al., 2008; Peckham et al., 2013; Rajib et al., 2016; Salas et al., 2012), most of them do not support directly accessing to heterogeneous data sources over the Internet. Also, the complexity of the existing models makes them difficulty to be shared and used by others. Consequently, the strengths and limitations of these various developed models are not well evaluated, understood, and widely used.

1.2. Strategy for solution

To address the aforementioned challenges, we develop an Open-data and Open-model framework. Our approach is based on several fundamental concepts and strategies. First, we design an open system architecture which includes two types of open system interfaces, by which individual models and heterogeneous data sources are connected to the framework through model agents and data agents respectively. Thus, this open system architecture makes it easy for the individual models to be integrated into the framework by their model agents to access the different data sources by making use of the data agents. Our framework provides data agents for some popular sources, like NASA OPeNDAP and radar data from NOAA. A data owner (or user) can also write his/her own data agent if he/she wants his/her data to be widely disseminated to the community through this framework. Second, we adopt a non-proprietary workflow system that organizes the scientific tasks in a reproducible manner. These tasks can be of different types including data-retrieval, data pre-processing, model execution, model coupling, and data visualization. Third, we develop a system-wide data fusion scheme enabling the user to fuse data from several data sources and/or models in his/her modeling exploration.

Our proposed Open-data Open-model framework MSM (Meta-Scientific-Modeling), whose implementation is called *msm*, has effectively addressed the drawbacks of the existing modeling systems, and thus has the following unique features:

(1) Other systems use proprietary workflows with preliminary functionalities (Dou et al., 2008). In contrast, the *msm* incorporates with an independent open source and graphical based Workflow engine, which is benefited by the future improvements of the generic open source workflow engine.
(2) Other systems suffer the lack of access support to external data sources. On the contrary, the *msm* facilitates users to obtain data directly from popular sources such as NASA OPeNDAP and USGS web-services by just dragging a component inside a working board area.
(3) The *msm* does not require a central administration, and each modeler has the responsibility of his/her own model and its corresponding model agent. This makes *msm* de-centralized and more sustainable than the approaches that require adaption and recompilation to be made by a central administration team.
(4) Unlike other modeling systems, the *msm* enables models to be freely connected with each other in the *msm* framework. That is, once the user has written a single Agent for his/her specific model to connect into the *msm*, he/she is able not only to access all the data sources but also to integrate it with other models already connected to the *msm* framework without any additional effort.

The remainder of this paper is organized as follows. Section 2 briefly overviews the relevant existing work and background. Section 3 presents the design of our framework and its implementation of the *msm*. Section 4 illustrates our framework's functionality and usability via examples. Section 5 presents an end-to-end modeling process to demonstrate the current version of *msm* framework. Finally, Section 6 gives our conclusions and planned future work.

2. Related works and background

2.1. Existing systems

There have been several efforts in developing different frameworks to integrate models to meet different needs in the broad Earth science and environmental domains, including Open Modeling Interface (OpenMI[1]) (Gregersen et al., 2007; Knapen et al., 2011; Moore and Tindall, 2005), Community Surface Dynamics Modeling System (CSDMS[2])(Peckham et al., 2013), Object Modeling System (OMS[3]) (Ahuja et al., 2005), Interactive Component Modeling System (ICMS[4]) (Vertessy et al., 2002), Earth System Modeling Framework (ESMF[5]) (Hill et al., 2004), and Community Hydrological Prediction System (CHPS[6]). CHPS is a main hydrological modeling framework used by the NOAA's River Forecast Centers (RFCs). Its main hydrologic modeling software is FEWS (Flood Early Warning System) (Krzhizhanovskaya et al., 2011; Werner et al., 2013). All of these existing model integration systems have their strengths and shed lights on this important research area. Nevertheless, these existing frameworks have several weaknesses which are briefly summarized below.

Lack of Data Access: As mainly focusing on modeling environments, open data access is not considered in the existing modeling frameworks by design, although a few of them (e.g., FEWS) support some data sources. In general, the access to data sources in those modeling frameworks relies on user's own application implementation, for example, as illustrated in CSDMS (University of Colorado Boulder, 2012).

No Data Fusion: Data fusion for multiple datasets and/or model results is not provided by the existing modeling frameworks.

Preliminary Workflow: While many of these frameworks have some basic ad-hoc workflow-like schemes to link models, a standard and sophisticated scientific graphical workflow engine is missing. As a result, workflow-provenance management is not available (Deelman and Chervenak, 2008).

Lack of Model Source-Code Transparency: While the requirements on models to be linked to a framework vary from one framework to another, most frameworks require the source code of the models. For example, CSDMS requires (1) to implement a Basic Modeling Interface (BMI) for a model to become CSDMS compliant model, which requires some refactoring of the model source code (Hutton et al., 2014); and (2) to implement an additional Component Model Interface (CMI) by CSDMS Integration Facility staff that wraps the CSDMS-compliant model to be used in the CSDMS framework. Some frameworks are not centrally administrated such as OpenMI, OMS, and ESFM, while others need central administration for framework development and use, such as CSDMS and FEWS.

2.2. Desirable features

Table 1 shows a list of functionalities that are important for hydrological modeling systems. It also compares the functionalities of the existing systems: OpenMI (Gregersen et al., 2007; Knapen et al., 2011), CSDMS (Peckham et al., 2013), FEWS (Gijsbers, 2010; Werner et al.,

[1] https://www.openmi.org/.
[2] https://csdms.colorado.edu/wiki/Main_Page.
[3] https://alm.engr.colostate.edu/cb/wiki/16961.
[4] http://www.clw.csiro.au/products/icms/index.html.
[5] https://www.earthsystemcog.org/projects/esmf/.
[6] http://www.nws.noaa.gov/ohd/hrl/chps/.

Table 1
Comparison of available features among model/integration systems. An "X" in the table represents a feature offered by the corresponding tool. Colors of green, orange, and white represent, respectively, features that are desirable and offered by the corresponding tools, desirable but not offered by the tools, and not required and not offered either by the tools.

FUNCTIONALITIES			Desirable	OPEN-MI	CSDMS	FEWS	OMS	ICMS	ESMF	msm
MODELS	Execution location	Central Server			X	X				
		Individual	X	X		X	X	X	X	X
	Adding existing models	Central process required			X	X				
		End user can add models alone	X	X			X	X	X	X
	Models' languages supported	Any (or many) languages	X	X	X	X	X			X
		Specific language						X	X	
	Models' source openness	Open source	X	X	X	X	X		X	X
		Commercial	X			X				X
	Code refactor/recompile not required for integration		X			X				X
	Offer public repository of code of contributed models				X					
DATA SOURCES	Location for data sources	Third party - remote data sources	X							X
		Central Server			X	X				
		Local	X			X	X			X
	Adding new data sources	User can add sources	X			X				X
FITTING I/O	Framework provides space re-scaling	Controlled by end user through workflow	X							X
		Controlled by code			X	X			X	
	Framework provides time re-scaling	Controlled by end user through workflow	X							X
		Controlled by code			X	X				
	Data fusion		X							X
	Re-gridding		X		X	X			X	X
DATA ANALYSIS	Framework provides data analysis				X	X		X	X	X
	Data visualization		X	X	X	X		X		X
DATA MANAG.	Framework system's data storage		X			X				X
	Model-to-model data transferring strategy	Easy to integrate	X					X		X
		Execution Performance		X	X	X	X		X	
WORKFLOW TOOLS	Workflow engine		X	X	X	X		X		X
	Graphical workflow engine		X	X	X			X		X
	Automatic process provenance		X			X				X
	Non-proprietary open workflow		X		X					X
	Tasks I/O ports	Explicit connection variable to variable	X	X						X
		Graphical connection variable to variable	X							X
		Variables grouped in several configurations			X			X		
	Workflow checks pre-required tasks			X						X

2013), OMS (Ahuja et al., 2005), ICMS (Rizzoli et al. (1998), ESMF (Hill et al., 2004) with those from our *msm* framework. The ones marked with an "*X*" in the column "*Desirable*" are critical in terms of reducing the time spent by researchers in tedious activities such as codifying, retrieving, and administering data, and/or improving the modeling systems' sustainability. In the following, the features of each category shown in Table 1 are discussed.

Models: The first feature listed under this category in Table 1 is the Execution Location, which has been chosen to be individual/local for our *msm* to save the time required to upload and manage remote

information. The choice of "individual" makes it easier to run models that are installed individually and locally but are not required at a centrally administrated system. It also makes it easier for the user to add his/her models to the framework without the need of a central team. In other words, the user does not need to wait for a centralized administration to have his/her model integrated into the framework but can do it by himself/herself.

Regarding the programming language, some existing systems (e.g., ICMS and ESMF) require specific languages. However, the language in which a model is written should not be a restriction. A desirable framework should be able to support models written in any languages by supporting the models in their compiled forms or at least to support a number of languages.

The framework should also be able to support the integration of not only the Open Source models but also the Commercial models. This feature is important since the commercial models usually do not provide source codes, but they are often useful in research activities. Besides, even though the re-compilation is possible with the Open Source models, it may not be always honored successfully. This is because the difficulties of finding and/or rebuilding dependencies (as third-party libraries of particular versions) are enormous in many cases. Besides, the re-compiling and releasing processes would invalidate all the testing and verifications that a model has already achieved. On the other hand, when the added code is minimal, it is not efficient to have the entire Open Source model be recompiled.

Data sources: With respect to the category of data sources, it is desirable that the framework can retrieve the information directly from the external sources and make it usable to models. Learning, retrieving, downloading and administering these data usually take a considerable portion of the researchers' work-time. Thus, a desirable framework should be able to help retrieve the data via the framework system itself and also allow the user to easily connect new data sources into the system.

Fitting I/O: Each model or data source has its own set of specific requirements of input and output formats. In order to integrate models and data sources together, the desirable framework should provide necessary conversions between their inputs and outputs to enable them fitting each other. Re-scaling data in time and space provides such a "glue" facility to connect models with data sources and to link models with each other. Doing this manually would take a considerable portion of researchers' time (Kouzes et al., 2009). For a desirable framework, this feature should be not only provided but also represented in a way that the user can easily use it, for example, through a graphical workflow instead of coding.

Data fusion allows the user to fuse data from several data sources and/or models that represent the same variable at the same time and for the same spatial area. By having a data fusion feature available, the researcher can analyze additional hypothesis. Finally, the re-gridding is also a desirable binding feature used to process the data from different sources that use disparate coordinate systems or resolutions.

Data analysis: Even though the user may need to create complex views of the data using their preferred and/or familiar tools, a desirable framework should be able to provide the user a minimum number of functionalities to display spatial information for any given time step. In that way, the researcher can save time by having a quick graphical view of the results from his/her work such as hypothesis testing, data analyses, model simulations, etc., and only needs to spend additional time to create graphic views using external tools for specifically chosen results.

Data management: Creating and administering data folders, detecting errors injected by accidentally deleted or moved files, dealing with machine-oriented data file names in thousands of files, etc. are not only a considerable workload for the researcher, but also error prone due to its tediousness. Since this is a repetitive machine-doable process, manual execution, as it is typically done at present, not only distracts the researchers from doing their real research but also easily injects errors (Ma et al., 2010; Sonntag et al., 2011). Because of this, a desirable framework should provide data management functionalities, particularly from the very beginning when the data are retrieved from the data sources, through the workflow and coupling process until the results are obtained and displayed (or exported).

Several techniques can be used to transfer data from one model to another. Some techniques are easier for the user to implement and modify the model integration with a reasonable model execution speed for research (i.e., at a small scale), but may not be sufficient for execution performance for a production system (i.e., at a large scale like the daily operation at the National Weather Forecast Offices). Other techniques may provide better execution performance, but are more difficult to make changes or to have additional models integrated. A desirable research framework should focus more on easy integration and user-friendly experience because the goal is to help researchers to analyze various hypotheses and to conduct real research. Once a new hypothesis is discovered, for example, a second tool (or the same tool with a different configuration) can be used to thoroughly test the hypothesis in depth where the system's performance becomes more critical.

Workflow tools: To improve the understanding and explanation of the modeling results and to ensure a reproducible process of researcher's modeling work, it is desirable to have the execution of his/her modeling process controlled by a graphical workflow (Callaghan et al., 2010; Deelman et al., 2009).

A desirable workflow engine should provide the provenance service automatically to track the changes of the workflow itself (Barkstrom, 2010). In this way the user can investigate the trace of data not only in the current workflow version, but also in previous workflow versions, and determine how the changes in the workflow affected the results of his/her hypotheses (Ludäscher et al., 2006). Since storing such information is a repetitive task, performing it manually would add a heavy workload to the researcher and be also error prone. A desirable workflow engine should also be an open source and an independent tool, since its maintenance and evolution would benefit the framework and the end user.

In addition, the inputs and outputs of each component involved in a desirable workflow need to be graphically explicit to ensure that different modeling simulations correspond to different inputs and outputs and that they are clear and visible for the researcher.

It can be seen from Table 1 that each of the existing hydrological modeling tools/systems offers only some of the desirable features, but not all of them. Also, some of the existing systems in Table 1 have their features only focused on certain specific categories, e.g., OpenMI has more features in the categories of the "models" and "workflow tools", while FEWS has more features in the "model" category.

In general, while most of the existing systems only offer the basic functionality in providing connectivity among models, they do not support the other important functionalities including remote data retrieval, transformations, and traceability required for actual workflow execution. Without those important functionalities supported from the framework system, the models cannot be easily connected to each other in a workflow with compatible time scale, space scale, units and/or gridding geometry, which has to be taken care of by users themselves.

As most of these systems do not provide support to access external data-sources, consequently, the researcher has to manually download, administer, transform, and feed the data into models, adding more workload to the researcher and increasing the risks of injecting errors. And none of the existing systems have a strategy to add any data-source oriented plug-ins.

In addition, the existing systems do not provide any functionality to fuse datasets, yet such capability of easily fusing different datasets is fundamentally important in this data rich era. This is because, oftentimes, there are different techniques to measure or simulate the same data variables, where each of them has its own strengths and weaknesses. The researcher then has to either develop his/her own code to fuse information from different data sources or use just one data source at a time.

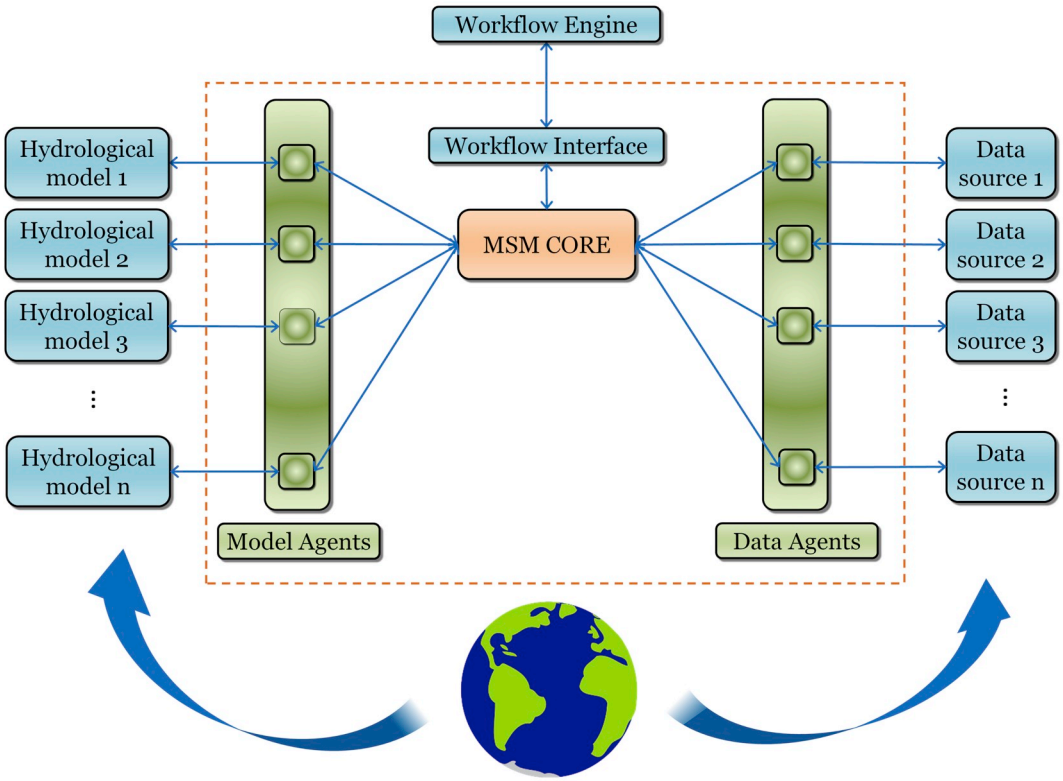

Fig. 1. The top-level design and overall architecture of *msm* system.

While most existing systems provide visualization tools for viewing the results and conducting data analysis, they have no graphical tools for controlling the workflow sequences. This means that the user needs to deal with text or XML-based workflow, which is not only tedious but also error-prone. Thus, it is desirable to have user oriented graphical inputs and outputs, allowing the user to graphically design the workflow (Davidson and Freire, 2008; Wang et al., 2009).

From Table 1, it can also be seen that most of the existing systems require recompiling the sources when new models are added. This reduces the flexibility of the system because the researcher may become dependent on the central administration organization to have his/her model connected. In some cases, the user is attempting to connect a model developed by a third party. By requiring code changes and recompiling, the user needs to contact the model authors and have them changing the code, compiling, testing and releasing another version before the model can get integrated into the system. In the case of CSDMS, for example, the framework team keeps the central control of the source code of all the models connected. This may make it easier for a researcher who just needs to use the models which are already connected to the system. However, if the researcher wants to connect his/her own models and run simulations with his/her own data, the process becomes complex.

3. The msm design

3.1. Overall architecture

We present our design and development of a novel Open-data Open-model framework as an integrated solution to provide researchers and practitioners with a sophisticated workflow-controlled modeling environment that ensures traceability and reproducibility for hypothesis tests in hydrological studies. From a top-level design point of view, our framework system *msm* is composed by a core (referred to as *msm* core), an interface with Workflow engine, Data Agents, and Model Agents (Fig. 1). Basically, *msm* interacts with the selected Workflow engine through the workflow interface, interacts with data sources through Data Agents, and interacts with models through Model Agents.

In the overall architecture of our framework in Fig. 1, the *msm* core controls all the other components and reaches external plugged-in models and connected remote data sources by the corresponding model and data agents, respectively. The *msm* core is connected to the workflow engine to provide end-users with all workflow control functionality. The *msm* core includes a data persistency service that can be used by all the *msm* components. In particular, an instance of the framework system is administered by an individual researcher who configures and runs it, not by any centralized entity. Thus, the *msm* framework allows the user/researcher not only to easily access external data from diverse sources but also to easily and efficiently execute, couple, and evaluate/intercompare various and complex models. The former is achieved via the Open Data architecture, while the latter is achieved via the Open Modeling architecture. The Open Data architecture adopts a common internal data model and representation to facilitate the integration of various external data sources into the *msm* framework using Data Agents. These Data Agents hide the heterogeneity of the external data sources and provide a common interface to the *msm* core. The Open Modeling architecture allows different models or modules to be easily integrated into the *msm* framework via Model Agents. The *msm* architectural design offers a general many-to-many connectivity between all individual models and external data sources, instead of specific one-to-one pair-wise connectivity. In other words, assume there are M heterogeneous data sources and N diverse models that need to be fully coupled and integrated in a modeling system. To accomplish this task, existing model frameworks would typically require $MN + N(N-1)$ pair-wise agents, but the *msm* framework only requires $(M + N)$ agents due to its architecture design. The number of $(M + N)$ agents represents the lowest linear complexity for such a model integration task.

In our *msm* framework the modeling processes are dynamically

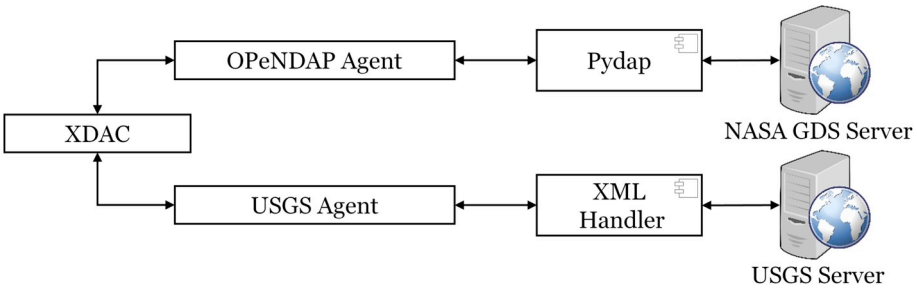

Fig. 2. Data agents developed in our *msm*.

defined by the user through a workflow engine. Workflow provides the user with a capacity of defining the sequence of activities to be performed. Activities are user-defined standalone tasks by which a user can easily build up any workflow sequence for his/her modeling process. Each activity has to be pluggable with others, and should not impose any implicit restrictions to the workflow sequence. To this end, activities have been defined with the following properties:

(1) Each workflow activity has a well-defined responsibility. Each activity is independent.
(2) No any hidden activity exists in any workflow. All workflow activities must be explicitly specified and controlled by the user.
(3) There is no any additional communication channel for an activity other than its inputs and outputs (loose coupling) to be connected with other activities, which are controlled only by the workflow engine.
(4) The activities are self-sufficient. As soon as the inputs are ready, the activities are ready to be executed. That is, no other tasks need to be performed prior to or at the same time of their execution.
(5) The activities are stateless. The memory is a responsibility of the persistency data management components with which the activity can use. Therefore, once any activity finishes its execution and returns the outputs, it must release any acquired resource or data that could be used by a subsequent execution.

To satisfy the above properties in our *msm* framework, we introduce a general data abstract called *msmDataSet* to ensure the type compatibility of any workflow activity input/output connections. This *msmDataSet* is a data representation composed by a list of time-steps, and a 2-dimensional image per time-step; it is the only data abstract type (apart from basic int, boolean and string types) that an activity can accept for its inputs and generate for its outputs. Our framework currently supports four types of activities in a workflow:

(1) Data retrieval: The framework automatically generates a graphical activity (a box in the workflow) for the corresponding data agent (provided and pre-configured by *msm* or added by the user).
(2) Data transformation: These are activities provided by the framework directly, and performed by the core by executing an internal algorithm (Data fusion, time/space re-scale, change of units, etc.).
(3) Model execution: The framework automatically generates a graphical activity (a box in the workflow) for the corresponding model agent (provided by the user or pre-configured).
(4) Data visualization: Even though the workflow engine can provide many of the visualization tools, it requires the core to prepare the data to be reported.

The responsibility of the workflow control is to determine the sequence of activities, and the inputs/outputs for each activity. In all cases, the workflow will request the *msm* core to perform the activity for the given inputs and will send back the corresponding outputs. Once the activity is done, the workflow can use its outputs as inputs for the subsequent activity. Most of the time, for example, the first activity defined in the workflow will be a retrieval of data from a data source. The workflow control requests the core to execute such a task. The core will call the data Agent to execute the task. The data Agent will retrieve the data and store them using services offered by the core.

One of our design criteria is to have a general and flexible open system architecture so that our framework can be easily integrated with different open source workflow engines. This way, our *msm* framework can take the advantage of existing (and future) workflow engines that the scientific communities use or prefer, and at the same time, can also avoid any reinventing of the wheels. In our current implementation of *msm*, we select *VisTrails* (Bavoil et al., 2005) as the workflow engine. This is because VisTrails provides a nice and convenient graphical workflow interface, in addition to its unique capabilities, such as, provenance and data visualization, which serve well to the goals of our *msm* framework. The other components in the *msm* framework shown in Fig. 1 are described below.

(1) Data Agents: They are dynamically loaded components that are connected to external data sources through the Internet, and store retrieved information in *msm* using services provided by the core. They are loaded at run time, and *msm* comes with developed data agents for some popular data sources. The framework automatically generates a Workflow Activity template per each Data Agent.
(2) Model Agents: They are dynamically loaded components that are able to run external models. The framework automatically generates a Workflow Activity template per each Model Agent. A Model Agent performs the following three main steps:
    - Preparing inputs: The Agent receives the inputs from the workflow (i.e., *msmDataSets*) and transforms them into the input files with the format needed by the model.
    - Executing model: The Agent will call the model executable file.
    - Storing outputs: The Agent reads the output files generated by the model and transforms them into *msmDataSets*. Then, the Agent uses the framework to store them.
(3) Core data management: This is a layer inside the *msm* core that provides persistency for the *msmDatasets*, allowing other components to write datasets and read datasets created during previous workflow activities. This component has a cache to improve its performance.

3.2. Open data

Our framework addresses the challenges of accessing multiple and heterogeneous external data sources via its Open Data architecture, which adopts a common internal data model and representation. In this architecture, data sources are integrated into the system using Data Agents that handle all source-specific implementations and remote data

```
from msm_core.msm..... import GenericDataAgent
class MyAgent (GenericDataAgent):
  def obtain_data (self, inputs, parameters):
```

```
def create_full_dataset (self, values, dataset_name, units, left, right,
top, bottom, time_ini, time_end, timestep):
```

access protocols (e.g., OPeNDAP, Web services), metadata standards, and source-specific implementations, thus abstracting the data sources' heterogeneity and providing a common interface to the *msm* core system.

The Open Data architecture not only allows integrating external data sources in an abstract and extensible manner, but also allows end users to load data agents dynamically at runtime, without having to stop current operations or recompiling the entire system. In this way, end users can develop their custom data agents for immediate use by the workflow engine.

The framework's internal data model integrates a relational representation of the system's metadata and a non-relational representation to store the actual data received from external sources and/or the results produced from model computation. The data model allows persisting data sets associated with gridded data as well as time series data (all currently represented by the *msmDataSet* structure).

We have developed and integrated two data agents into the framework (see Fig. 2), which already covers a broad range of data sources and data access protocols: NASA (OPeNDAP-GDS) and USGS (REST Web services) for bringing in data from both gridded (netCDF) and point time series (WaterML) services using the major cataloging services (e.g., NASA GES DISC).

3.2.1. Develop an agent for a data source

To enable users to easily develop new data agents so as to add new data sources into the *msm* framework for their scientific explorations, we have designed a Generic Data Agent class. It offers the generic *obtain_data* method. Individual data agents inherit from the Generic Data Agent and override the generic *obtain_data* method, which enables the use of a specific data retrieval protocol from the workflow. The data agent also uses services from the Generic Data Agent for the creation and edition of the output datasets, simplifying the development of new Data Agents.

All Data Agents share the following basic responsibilities: (1) Securely contact the data-offering server; (2) Download the data files to the local machine if they do not already exist; and (3) Read the data from the downloaded files based on their format(s) and create a new Dataset. If, at some point during the downloading process the connection is interrupted, the simulation workflow will stop. In such a case, the user can restart the workflow, and *msm* will continue to download the data from where it last stopped from the data source since there will be a local copy of the data downloaded so far. The user is free to delete these locally downloaded data or maps at any time.

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

After that, the user can add, in the *obtain_data* function, all the necessary codes to download the data and create the dataset object. When the information is extracted and is ready to be stored in a Dataset object, the user can call, for example, the function *create_full_dataset* of the Generic Data Agent, which will ensure the proper building of the Dataset object, and the persistence of it.

3.2.2. Register data agents into msm framework

The data agents are added to the system by copying the folder with the Agent's code into the *upload* folder of the framework. The folder needs to be named exactly as the class representing the Agent. Also, the file containing the class needs to be named exactly as the class. The folder may contain more folders and more Python code files.

The system needs to be refreshed to generate the graphical representation of the Agent in the Open Data list. After that, the Agent will be visible in the graphical interface on the panel, and can be considered successfully registered.

3.3. Open model

To take the full advantages of the framework, such as the *M2M*[7] automated data retrieval from popular data sources (e.g., NASA data centers and USGS), graphical workflow, process provenance, data transformations and rescaling, data fusion, and data visualizations, researchers/users need to connect their model agents into the *msm* framework. There are two steps required to accomplish this: (1) Develop agents for their models and (2) Register the agents into the framework.

3.3.1. Develop an agent for a model

To enable users to easily develop their model agents so as to add their scientific models into the *msm* framework for their scientific explorations, we have designed a Generic Agent class. The Generic Agent declares the generic *run_model* method. Individual model agents inherit from the Generic Agent and override the generic *run_model* method, which enables the invocation of a specific user model 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 database, which simplifies the integration of a user's model into the framework. This model agent is independent of the language used in writing the model. That is, the language used by the model can be in C, C++, Fortran, Python, Java, etc.

To facilitate the model agent development by individual users, we have also developed an Agent Development Kit[8] that contains only the basic API necessary to write and test an Agent, without the entire *msm* core and the workflow. This makes it easier to set up the development environment to start writing and testing a Model Agent.

The responsibility of the model agent is to provide the steps to: (1) Read the inputs from the *msm* core and save the data into files from which the model can read; (2) Run the model; and (3) Read the model's output files and save the information into the *msm* core. This is implemented by overriding the *run_model* function.

To write the model agent, the user has to create a text file including a

[7] *M2M*: Machine-to-Machine.

[8] The ADK (Agent Development Kit) is a set of libraries that contains the interfaces and utilities required to write, compile and test an Agent.

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

```
def get_time_series (self, dataset_id, time_ini, time_end, row, col):
```

```
dataset_id = self.create_dataset (variable_name, left, right,
top, bottom, side, base)
def save_timestep (dataset_id, timestamp, matrix)
```

class in Python as outlined below:

After that, the user can add, in the *run_model* function, all the necessary codes to prepare the inputs, execute the model and store the outputs. To read variables from the database, the user can call, for example, the function *get_time_series* of the GenericAgent, which will return an array with the values of the time series for the dataset requested.

To write variables (i.e., model outputs) into the database, the user can call the following provided functions:

The *create_dataset* function serves to initialize a *msmDataSet* and provides the dataset id. The *save_timestep* function saves a timestep inside the dataset. Both are provided by the GenericAgent.

The user also needs to implement the function *getInputs*. The *msm* system will use such functions to create appropriate ports in the workflow activities.

3.3.2. Register agents into msm framework

The model agents are added to the system by copying the folder with the Agent's code to the *upload* folder of the framework. The folder needs to be named exactly as the class representing the Agent. Also, the file containing the class needs to be named exactly as the class. The folder may contain more folders and more Python code files. Also, the folder must contain the executable files required to run the model.

The workflow needs to be refreshed to generate the graphical representation of the Agent. After that, the Agent will be visible in the graphical interface, and can be considered successfully registered.

3.3.3. An example of agent

To illustrate, we describe the model agent written for the *VIC* model (Cherkauer and Lettenmaier, 1999; Liang et al., 1996a, 1996b; 1994; Liang and Xie, 2003, 2001; Parada and Liang, 2004), a land surface model that is widely used in the hydrology and water communities (Maurer et al., 2002; Nijssen et al., 2001). The *run_model* function for the VIC model agent (*VicAgent*), which performs all the responsibilities described above, include:

(1) Initialization of environment;
(2) Preparation of input files: In this part, the VIC model agent (VicAgent) uses several functions to build the configuration and input files required by the VIC model. The VicAgent prepares one input forcing file for each modeling cell of the study area over which the VIC model is to be run. This forcing file includes all the forcing variables needed to run VIC. Since VIC can be run at different levels of complexities which require different forcing data, the users need to be careful with the different requirements on the forcing files for each case. It is important to mention that when obtaining the information (e.g., forcing data) from the database, the agent does not directly query the database. Instead, the model agent only uses the services offered by the Generic Agent from the *msm* core. Such a design in the framework's architecture ensures a minimum complexity for the user when he/she writes the model agent. The VicAgent will automatically create the parameter files (e.g., soil, vegetation, and snow) required for the model to run in case they do not exist yet in the working directory. Otherwise, the model will use the existing parameter files. This allows the user to perform model calibration manually through changing the values in the parameter files.
(3) Execution: In this step, the VIC's model agent builds the command that calls the VIC executable with its given global parameter file (for instance: prompt > vic –config_file myconfig.txt –input_dir inputs/files). The command is sent to the operating system for its execution and the agent waits until the VIC model run is completed at which time the output and error streams, if any, of the VIC model are directly sent back to the console of *msm* (which is visualized in the console of Vistrails).
(4) Processing of outputs: Once the command is executed, the agent processes the output files for each of the VIC model's output variables. VIC produces results in time series if it is run with a choice of time first then space. Because of that, the agent needs to open all of the output files (one per cell) and reads all of them for each time-step in order to generate area-images and store them for each time step in the output *dataset* in *msm*. To save this in the database, the agent again uses the services offered by the Generic Agent from the *msm* core.

3.4. Data management layer

The persistence of information in *msm* is implemented as an in-memory and local data caching, featuring a non-relational database scheme that supports high throughput. Our data management facilitates caching the retrieved data, storing previous results, and making the task of coupling different models more efficient where only a dataset's identifier is passed among coupled models.

Our framework persists metadata information, *msmDataSets* and

Table 2
A list of components/functions supported by the *msm* framework.

Group	Component	Description
Configuration	msmListVariables	Show the list of all variables configured in *msm*. Allow the user to add or remove a variable by name.
	msmVariable	Show the information of a variable. Allow the user to add or remove sources by name.
	msmSourceVariable	Show the information of a source for a given variable. Allow the user to add or remove properties of the source.
Control	*Iterate*	Build control loops (both *for* style and *while* style) in workflow.
Data Preprocess	msmCreateDataset	Create datasets. The spatial and temporal dimensions can be initialized and the values of the grids are filled with a default value.
	msmExpandDict	Extract a max of 10 values from a dictionary given the keys where the dictionary is a map of variable names and their values.
	msmFill	Fill the empty values of a dataset with a constant value or by inverse squares interpolation.
	msmGridding	Change the gridding of a study area by interpolation.
	msmMakeList	Generate a python list using all the objects connected to the port.
	msmPrepareDict	Create a python dictionary from a max of 10 static keys and values.
	msmSetNones	Set empty values for the same cells that have empty values in a reference DataSet. This is done for each timestep.
	msmWhiteNoise	Add white noise to a DataSet.
I/O Files	FileToDataset	Create a one-timestep DataSet by reading a text file that stores the data as an image: North in top, South in bottom, West in left.
	TimeSeriesFileToDataSet	Create a dataset by reading a file with a timeseries.
	DirToDataSet	Create one dataset from each file in the directory. The resulting datasets have one timestep each.
	TimeSeriesFileToDataSet	Create only one Dataset. Each file in the directory is used to create one timestep for the dataset.
	msmOutFile	Export a timestep to a text file.
	DatasetToTimeSeriesFile	Export all timesteps of a chosen cell in a dataset to a timeseries file.
Open Data	RetrieveDataSetAgent	Retrieve data given the variable name and optionally the source. If no source is given, it will pick the default source and its Data Agent.
Open Model	RoutingAgent	Run the routing model.
	VicAgent	Run the VIC model.
Statistics	msmComputeCorrelation	Given two datasets with the same time period and the timestep and the same spatial dimensions (i.e., the size and the spatial resolution of the study area), compute the correlation coefficient for the two time series for each corresponding cell. The result is returned in a dataset with the correlation coefficient values in the same spatial dimensions as those in the inputs.
Time-Space	msmAddTime	Receive a starting timestamp, the duration of each timestep and a number of timesteps. It returns a final timestamp, adding the elapsed time-steps to the starting time.
	msmMKS	Use the MKS framework to fuse two or many datasets. It can return the fused dataset for one given scale or many scales.
	msmTimeRescale	Rescale cell by cell, for a target timestep (i.e., a specified time duration of one time step) using one of four operations: Aggregation, Average, Maximum and Minimum.
Visualization	msmShowDataSet	Print all the timesteps of a DataSet as images.
	msmShowPlot	Plot the time series represented by each of the cells of the DataSet.
	msmShowTimeStep	Print a single timestep as an image.

remote cached data. The metadata contains configuration information, physical variables used by the Workflow Activities and the inventory (by id) of *msmDataSets*. The *msmDataSets* storage in our framework contains the time-steps, which can be the big data volume. Each time-step contains a reference to the *msmDataSet* it belongs to, and a Document (the data oriented structure) with a 2D image. The remote-cached data are useful in storing raw data received from external data sources. In this way, the data agents do not have to download the same data every time the execution repeats; they can reuse the data even when the researcher tests a slightly different hypothesis, allowing the system to work offline.

4. Implementation

The current version of our open data and open model *msm* prototype is implemented in Python and integrated with *VisTrails*. At present, the *msm* prototype supports various components summarized in Table 2.

The functions of the *msm* framework described in Table 2 are presented below.

4.1. Graphical workflow

The msm runs inside *Vistrails*, an open source of graphical data-workflow that deals with control of the sequence of activities, the versions, provenance and visualization tools (see Fig. 3).

During execution, *VisTrails* will start to analyze the activities of the workflow. For each one, *VisTrails* will track back the origin of the required inputs. Such tracking is repeated until tasks without inputs are found. With this analysis, *VisTrails* marks and detects all the required tasks for execution. Unmarked tasks are not executed.

4.2. Retrieving data

The Open Data Agents (see Fig. 4) can be used to retrieve data and create *msmDataSets*. Each Data Agent can define its own inputs and they can produce one or more datasets. Using the already developed data agents, *msm* can retrieve OPeNDAP information from NASA and access data from USGS using web-services.

In case the data retrieval is interrupted, *msm* is capable of restarting downloading the data from where the last time step downloaded was. This is done not only to reduce the time it takes to retrieve the information into the local machine, but to resolve the internet interruption problems. The user is free to delete these locally downloaded maps or datasets from his or her machine.

4.3. Provenance

The provenance is a *VisTrails*' functionality from which *msm* benefits. It keeps track of all of the executions from all versions of the workflow. It helps users find and review previous executions, the graphical configuration and the potential errors. Each execution is identified by the name of the version (which is labeled in the history tab) and the start- and end-time. Fig. 5 shows a failed and a successful execution (the color code shown in the upper right side of the screen shot), each using a different version of the workflow. The row that shows the execution can be expanded to show the list of components used with the execution time, completion condition and whether it was cached or not.

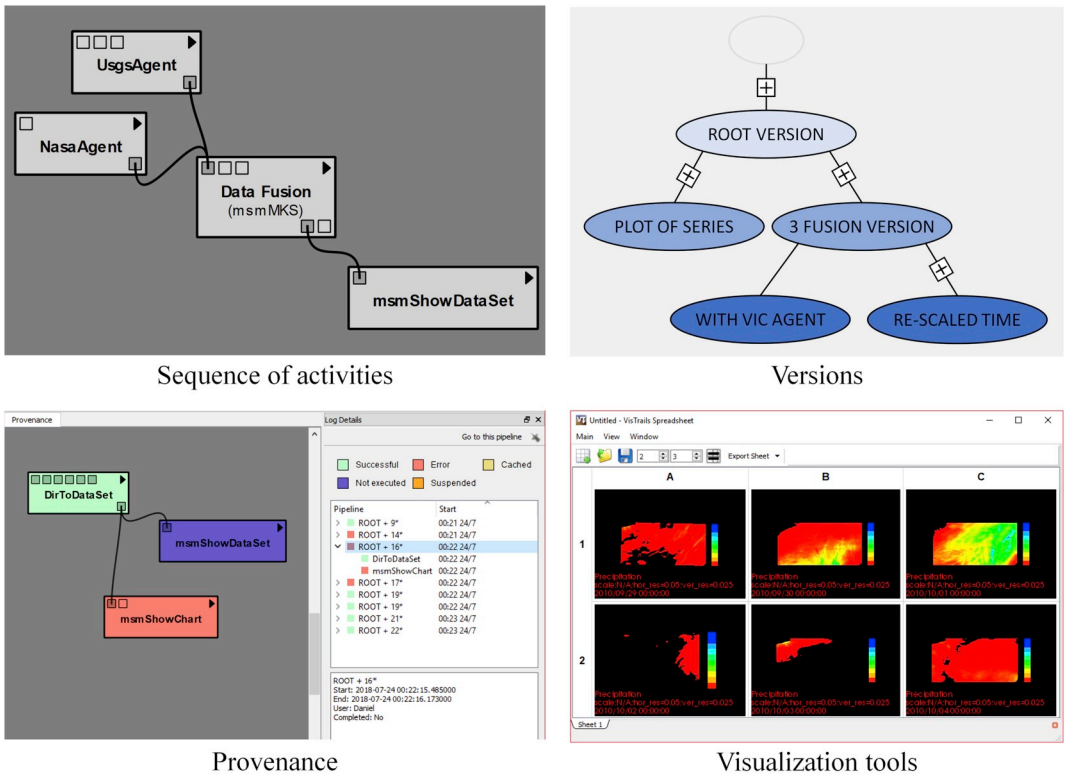

Fig. 3. An illustration of main capabilities of the Graphical Workflow (VisTrails).

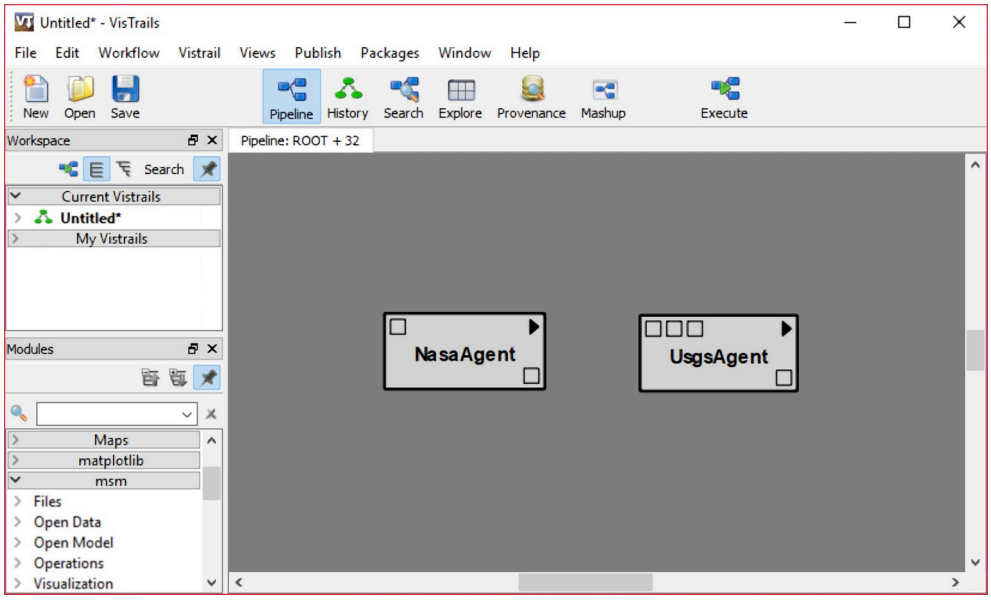

Fig. 4. An illustration of RetrieveDataSetAgent component via Data Agents.

4.4. Re-scale space and re-gridding

The re-gridding component (see Fig. 6) transforms a dataset from its original spatial configuration to a desired target configuration. The spatial configuration is defined by:

(1) Shape of the cells: Although the shapes of the data cells are in squares most of the times, they can also be in rectangular or rhomboidal shapes. In the case of the radar data, each cell is a different tetragon.
(2) Resolution: Size of the cells.
(3) Borders starting point: Even when two grids have the same shape and resolution, if their borders are not the same, their spatial configurations will not be the same.

The re-gridding module first identifies the portions of the original data cells that are located within the study area (i.e., represented by

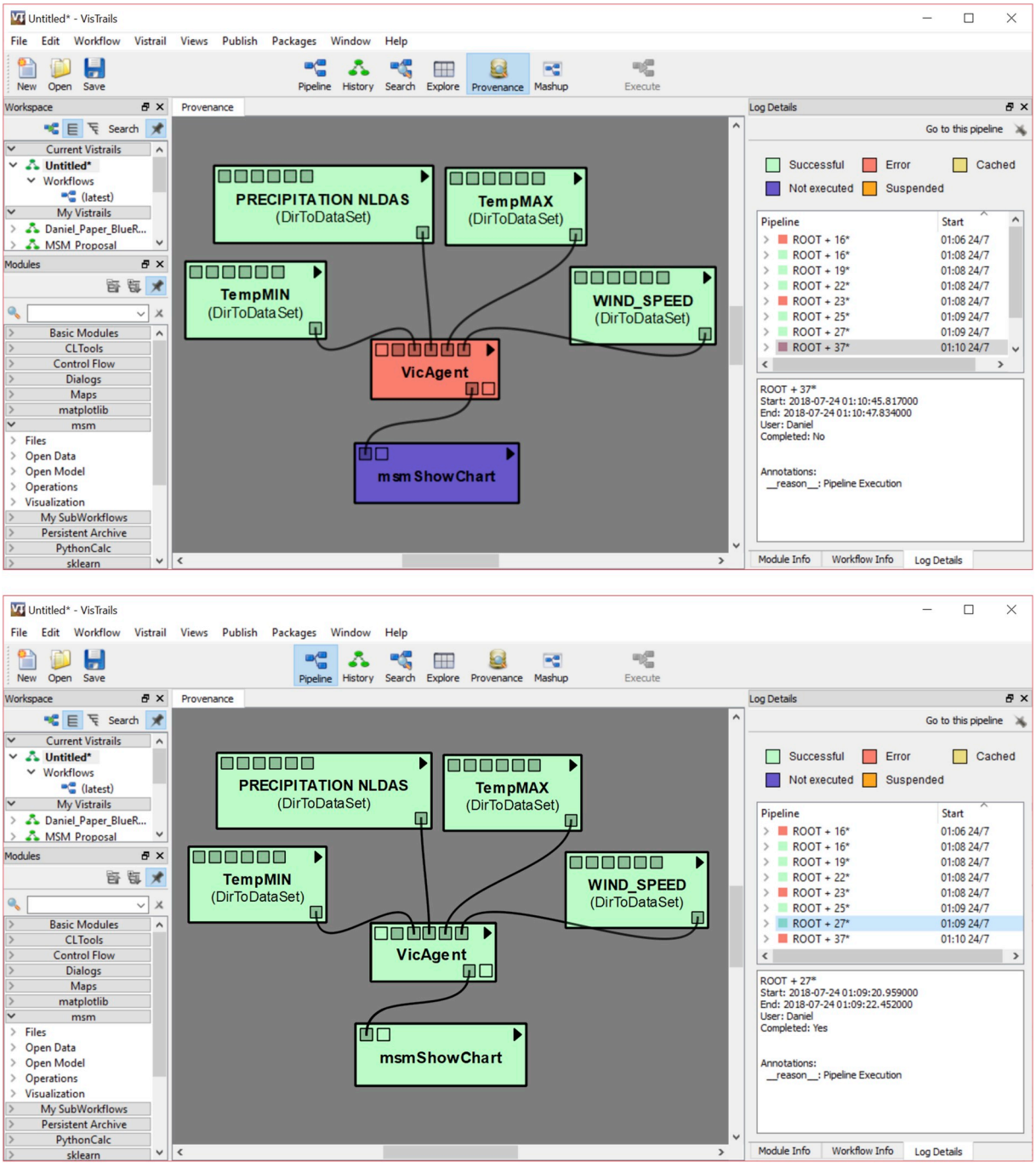

Fig. 5. An illustration of provenance example (top: failed execution; bottom: successful execution) with VisTrails. In the right pane of the above VisTrials screen shot, there are five different small colored squares which are status indicators stored by the provenance feature of VisTrials after each execution of the workflows. The red/blue boxes mean that there was an error at execution; the orange boxes mean that the user suspended the execution, and only yellow/green boxes mean that the workflow was successfully executed. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

destination cells). Then, it computes the data values for these destination cells using an areal interpolation method whenever the data cells and the destination cells do not match with each other in terms of spatial resolution, shapes, or borders.

4.5. Re-scale time

Using the *msmTimeRescale* component (see Fig. 7), the *msmDataSets* can be re-scaled to the time resolution required by a model or another input. This component can use one of the following aggregation functions: average, maximum, and minimum. It also interpolates when data is unavailable in a required timestep.

4.6. Fusion of DataSets

Using the MKS (Parada and Liang, 2004) framework, the *msmDataSets* can be fused when more than one source is included in an analysis for a given physical variable (see Fig. 8). For example, if precipitation data are available from NASA OPeNDAP and from radar, they can be fused to form a single precipitation input to be used by a model.

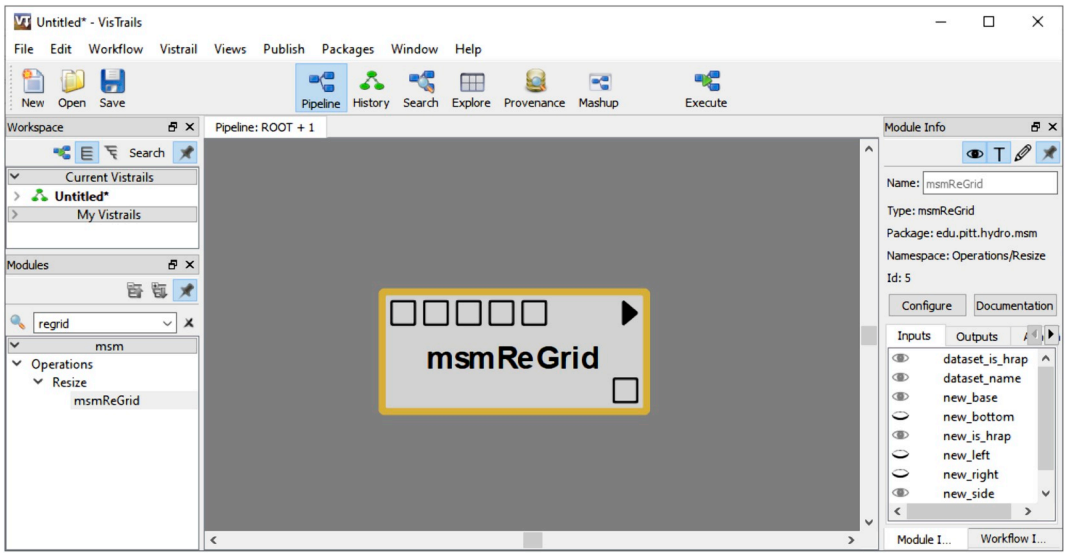

Fig. 6. An illustration of Re-gridding in *msm*.

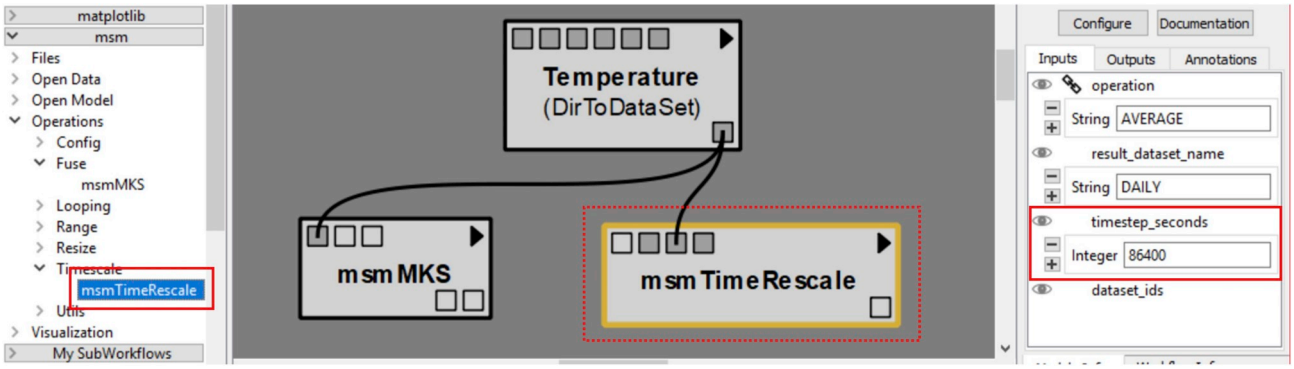

Fig. 7. An illustration of Time re-scale in *msm* via VisTrails.

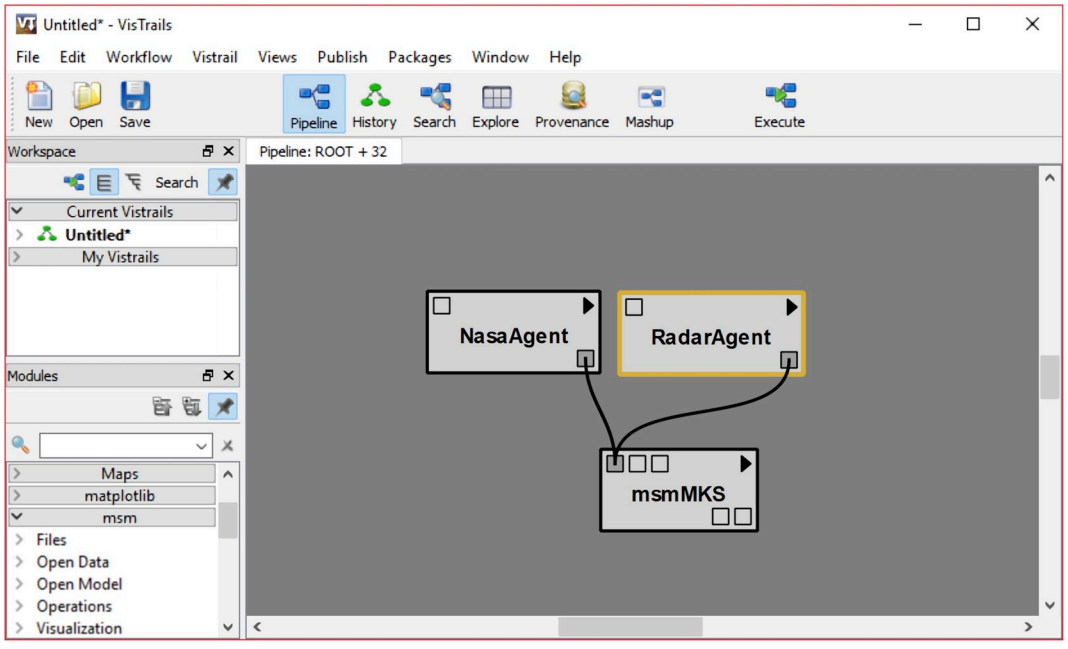

Fig. 8. An illustration of Fusion of *msmDataSets in msm*.

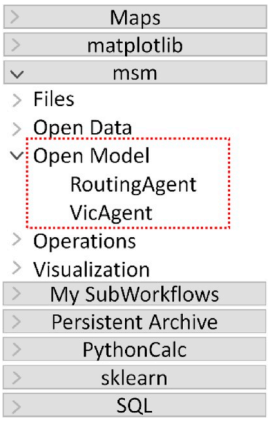

Fig. 9. Graphical components created automatically based on registered Model Agents.

4.7. Running models

The workflow creates a graphical component for each model agent registered in the *msm* framework. Fig. 9 shows that two models are registered in the *msm* framework: one Model Agent for the VIC model and the other one for the routing model (see highlighted square). The *msm* framework creates graphical inputs and outputs as defined in the Model Agents for VIC model (see Fig. 10).

4.8. Coupling models

The user can couple the components using any of the available ports in the way he/she desires because all components in *msm* accept and return *msmDataSets* as shown in Fig. 11, where the VIC model is coupled to a routing model through their corresponding model agents, VicAgent and RoutingAgent.

In this example the VicAgent outputs its baseflow and surface runoff to be used as inputs for the Routing Model Agent. This is the way model agents interact with each other in *msm* from the workflow perspective. New Model Agents implemented by the users need to clearly specify the datasets and units required on each port.

Since not all models provide outputs for every single time step, some preprocessing may be necessary for models' appropriate interactions with each other. One solution offered by *msm* is to use the *msmTimeRescale* component to modify the temporal granularity of the datasets flowed between models. Another possibility involves the usage of the *Iterate* component (see Fig. 21) to perform a full spatial simulation for a single time step. This way, the user can create a time series with the snapshots of each simulation. A user can also modify his/her particular model in order to accommodate a time-step wise integration with other models.

4.9. Adding new data agents

The *msm* framework can be expanded to access data from other data sources by adding new corresponding Data Agents created by users. After writing a data agent, the user just needs to copy the code into a folder provided by the framework, as described in Sections 3.2.1 and 3.2.2.

4.10. Visualization of results

The *msm* makes use of the *VisTrails* visualization tools to show 2D images in time, point time-series and snapshots of variables for a given time. Fig. 12 illustrates the time series plots generated using the msmTimeRescale component shown in Fig. 7.

4.11. Loading local information

Fig. 13 shows that data stored locally in files can be loaded into the *msm* framework using the functions built in *msm*. After the local data are loaded into the *msm* framework, they can be accessed just by using the id of the Dataset.

5. A modeling example

In this section, we present an end-to-end hydrological modeling example to demonstrate how the *msm* framework can facilitate modeling with complex data retrieval and processing it in an automated manner. It starts with retrieving precipitation, wind and temperature from the NASA OPeNDAP data sources, which follows by running the VIC model

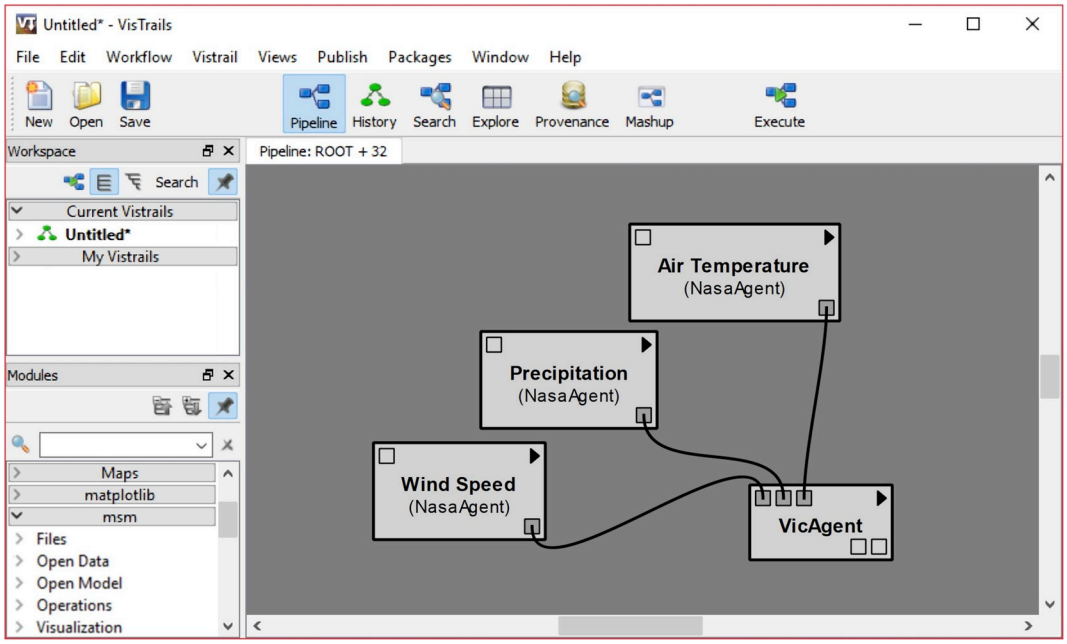

Fig. 10. An illustration of graphically explicit inputs/outputs in *msm* via VisTrials.

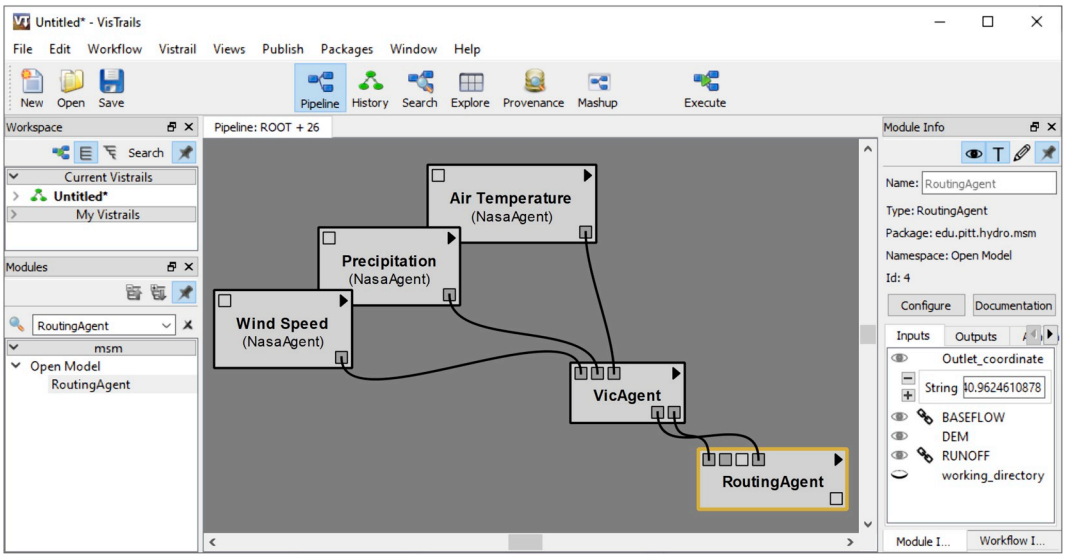

Fig. 11. An illustration of coupling of the VIC and Routing models in *msm* via VisTrials.

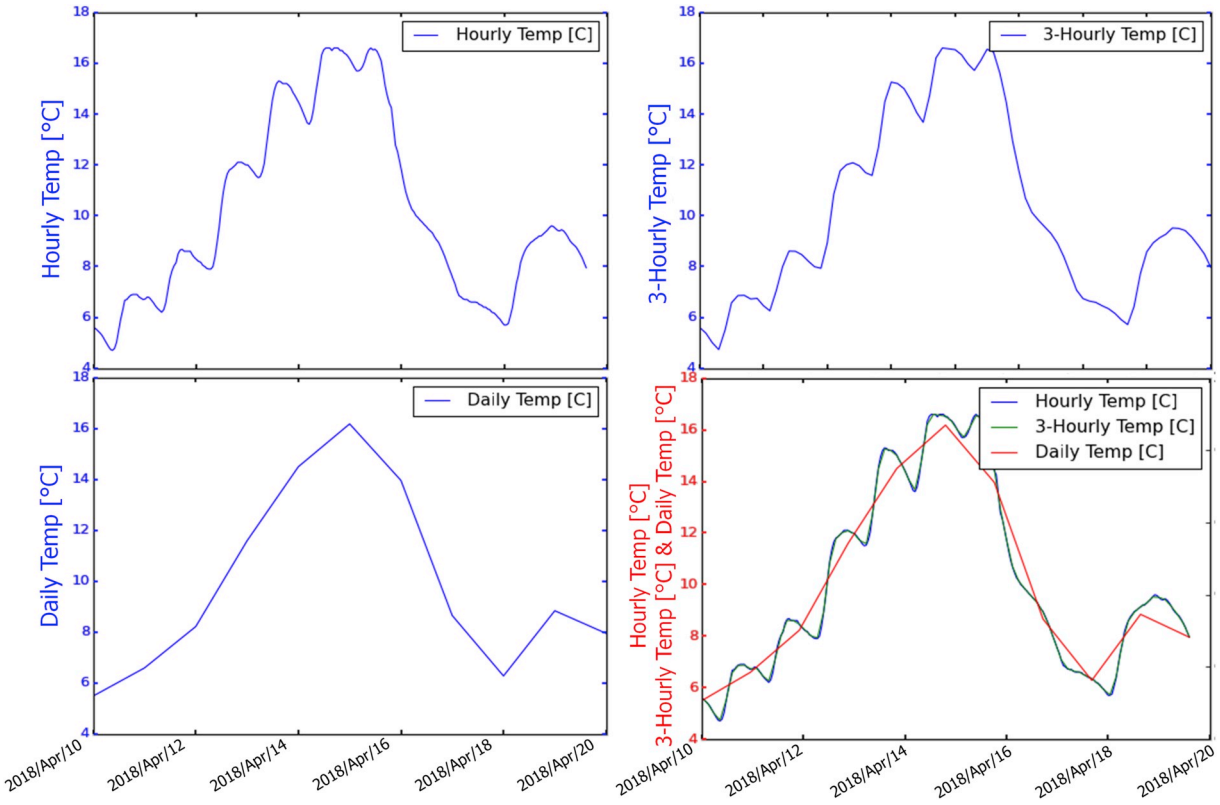

Fig. 12. An illustration of time series of temperature plots in *msm* via VisTrails: (1) at original hourly resolution, (2) with temporal resolution rescaled to 3 h, (3) with temporal resolution rescaled to daily resolution, and (4) a plot with all three different temporal resolutions.

(in its water balance mode) to compute the water budget related variables, such as evapotranspiration, soil moisture and runoff. Then, the VIC simulated runoff time series is used as an input to a routing model. The example finishes by comparing the routed discharge based on the VIC simulation with the USGS measured streamflow data which is automatically read into the *msm* framework from the USGS website. This is a typical hydrological modeling use case scenario which involves retrieving the weather (forcing) data needed for running a sophisticated model, executing it, and comparing the model simulation results with observations. Compared to the typical modeling practice in the hydrology community, the main difference shown in this example is that all of those tasks are done automatically, under the *msm* framework, with machine-to-machine communication for data retrieval from external data sources. This example also demonstrates several important features of the *msm*, including using the modules of Open Data, Open Model, *MKS-based* data fusion and visualization. Fig. 14 shows the study area of the example. Fig. 15 shows the selected watershed and its river network.

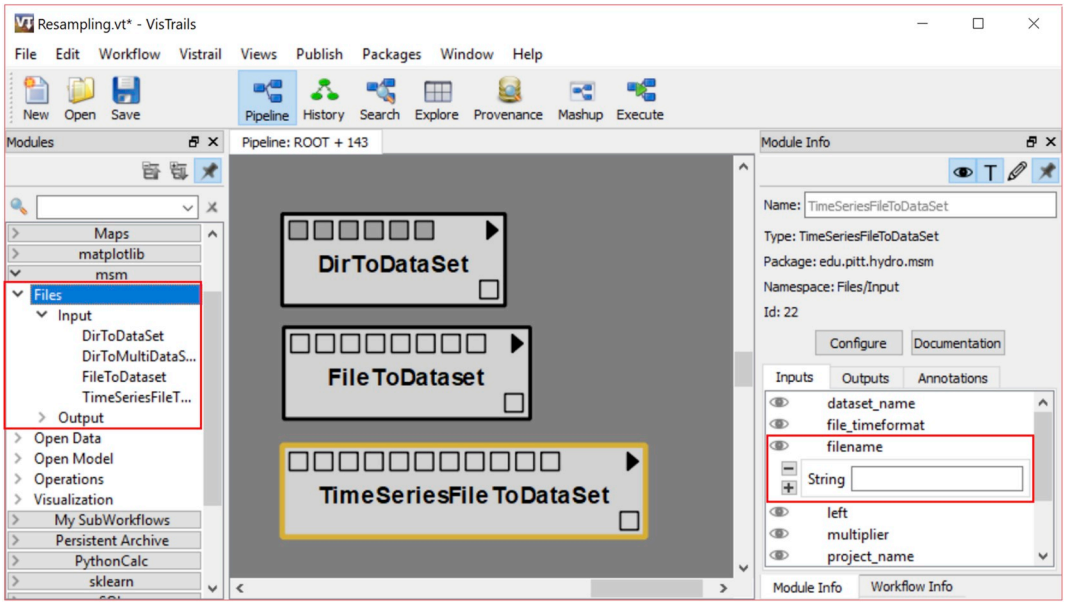

Fig. 13. An illustration of access to local data.

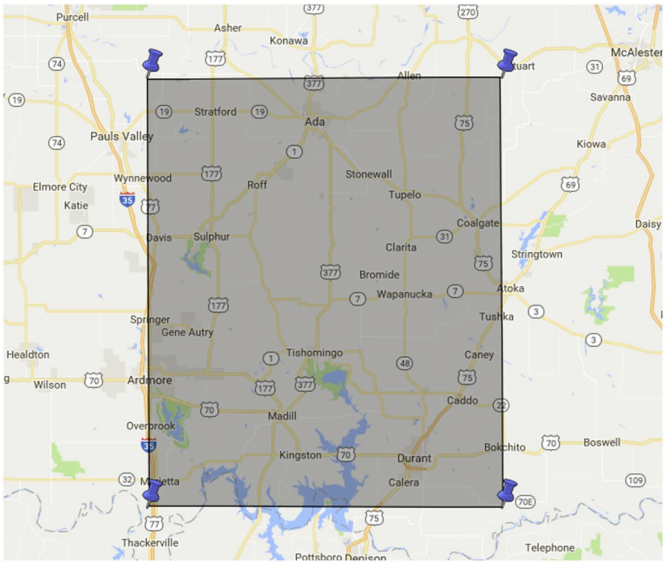

Fig. 14. Study area and the outlet of the watershed (Blue River). Map is generated using Google maps engine https://www.google.com/maps/d/viewer?mid=15BL122LhCkzMGiUg4hAStCvXt7vsmXkt&ll=34.37762642783292%2C-96.63872839999999&z=9. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

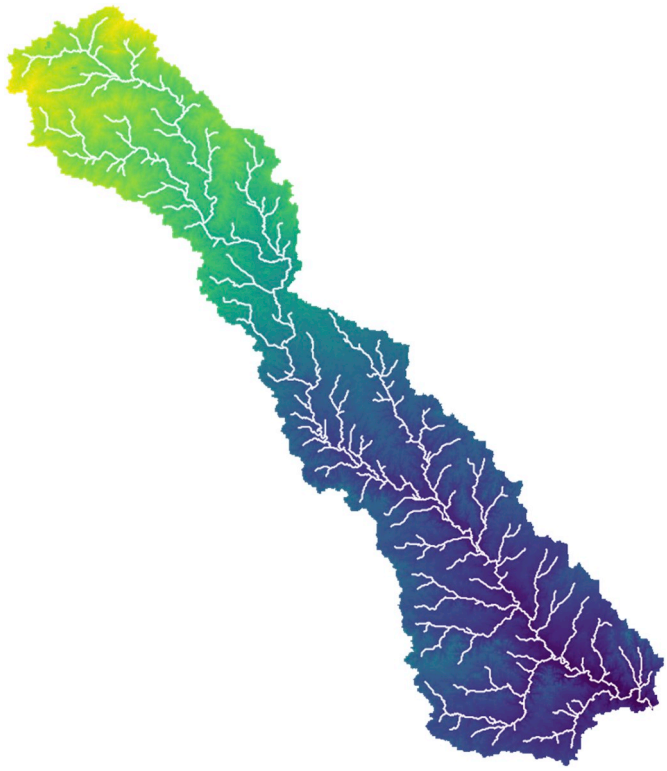

Fig. 15. Watershed (Blue River) and its stream flow network. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

5.1. Configuration of the workflow

The first step is to set up the components to retrieve various types of information from the corresponding data sources: precipitation, temperature and wind from NLDAS (NASA), for example, when VIC is run in its water balance mode. Fig. 16 shows the workflow designed for conducting this modeling task, in which each component for retrieving the forcing data is created by dragging the corresponding data agent component from the Open Data list on the panel shown in Fig. 17 into the working area. In addition, the *VicAgent* component is dragged from the Open Model list into the working area as well as shown in Fig. 16.

For each of the three *RetrieveDataSetAgent* components, the configuration inputs showing in the right column of Fig. 16 next to the work area are filled with values shown in Table 3 below:

Note that some of the fields listed in Table 3 cannot be seen Fig. 16. To see all of them, the user just needs to scroll down using the arrow in the right-bottom corner of the right panel shown in Fig. 16.

Next, the VIC model is set up by connecting the *VicAgent* component to the appropriate input data sources as well as the output displays shown in Fig. 16. For the model input connection, the *dataset_id* output ports from the *RetrieveDataSetAgent* components are connected to the

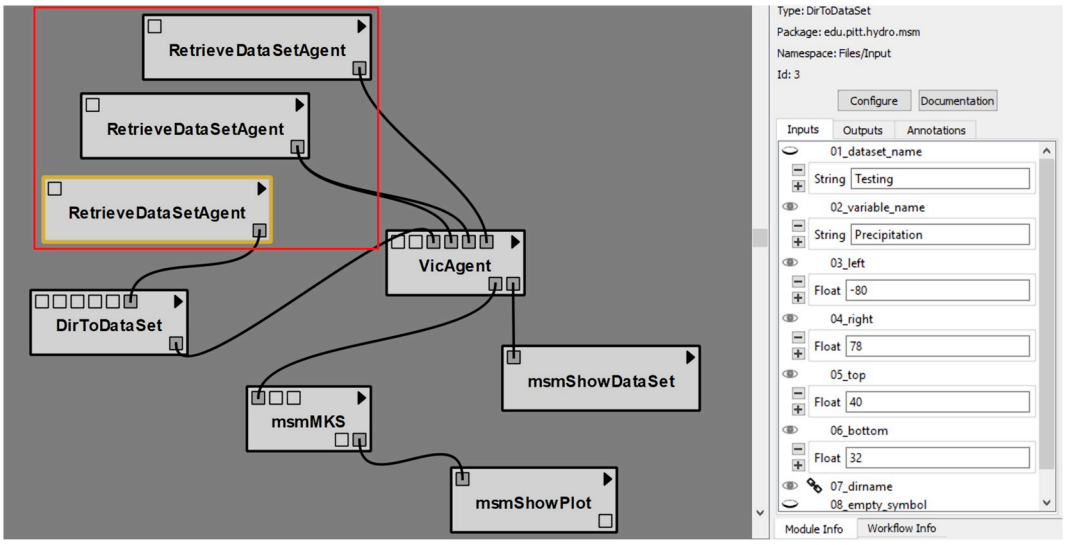

Fig. 16. Workflow designed for running the VIC model.

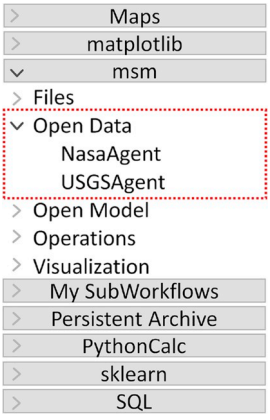

Fig. 17. An illustration of components within Open Data.

VicAgent input ports. We note that *VIC* requires the min and max temperature as inputs, but the same variable *temperature* is used in this case for simplicity.

The *DirToDataSet* component is used here to load the static soil parameters used by VIC. If it is not provided, VIC will use the default values. This component reads all the files from a folder and then creates a map of the specified study area for each file read in. Each such file contains a 2D matrix corresponding to the modeling grids over the study area for a time step. Since the soil parameters are static and thus they are stored as a one time-step image. The component also has other input ports that can be used to provide information such as the coordinates and the desired scale. But in this case, the information of coordinates and desired scale is received from the *RetrieveDataSet* component.

For the model output connections, two different functions, *msmShowDataSet* and *msmShowPlot* are used to illustrate two types of visualization for the model simulation results. The *msmShowDataSet* is connected to the soil moisture output of the *VicAgent* to view VIC generated soil moisture results as an image, while the *msmShowPlot* is connected to the evapotranspiration output to view it as a time series. More specifically, once the computation of the VIC model is finished, the soil moisture output will be shown as an image for each time-step whereas the evapotranspiration output will be shown as a time-series plot.

The evapotranspiration output is being re-scaled to scale zero to show the averaged evapotranspiration of the study area as a time-series where the *msmMKS* component is used to achieve the areal average through the rescaling operation based on the MKS algorithm.

5.2. Execution

The execution of the workflow starts when the user clicks the *Execute* button shown in the 2nd top row of the VisTrails window (see Fig. 13). The *RetrieveDataSet* components will attempt to automatically download the information from the specified data sources. If the data to be downloaded are too large, the *msm* component will retrieve it in chunks of one time-step each. If the communication is lost at any time, the user can re-run the workflow and the components will continue downloading the data from the last time-step it was saved. Before the VIC Model component starts, all the *RetrieveDataSet* components, and the *DirToDataSet* must have been finished. They will provide the *dataset_ids* for

Table 3
Information for each field shown in the right column of Fig. 16.

Field	First	Second	Third
Username (Manually hidden in Fig. 16)	Admin	Admin	Admin
Project name	TESTING	TESTING	TESTING
Dataset name: Any name the user wants	PREC_NLDAS	TEMP_NLDAS	W_NLDAS
Left coordinate (i.e., the left coordinate of the study area)	−80	−80	−80
Top coordinate (i.e., the top coordinate of the study area)	40	40	40
Width (in degrees for the study area)	2	2	2
Time initial (i.e., the start time of the data)	2001/01/01 00:00:00	2001/01/01 00:00:00	2001/01/01 00:00:00
Time end (i.e., the end time of the data)	2001/01/10 00:00:00	2001/01/10 00:00:00	2001/01/10 00:00:00
Variable name	PRECIPITATION	TEMPERATURE	WIND
Source-variable name (i.e., the name of the data source)	Empty (Default is NLDAS)	Empty (Default is NLDAS)	Empty (Default is NLDAS)

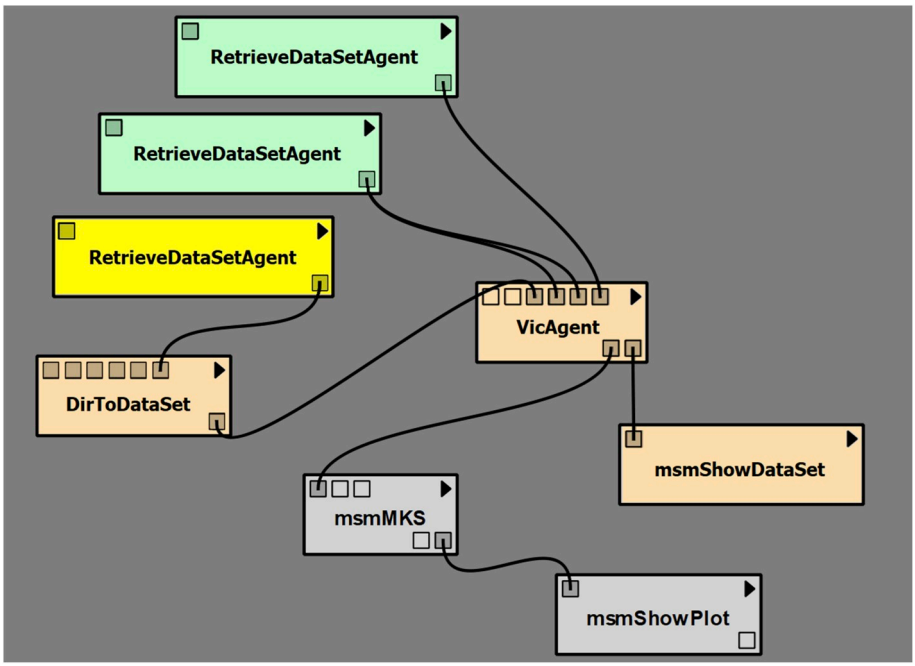

Fig. 18. An illustration of different colors of a workflow during its execution. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

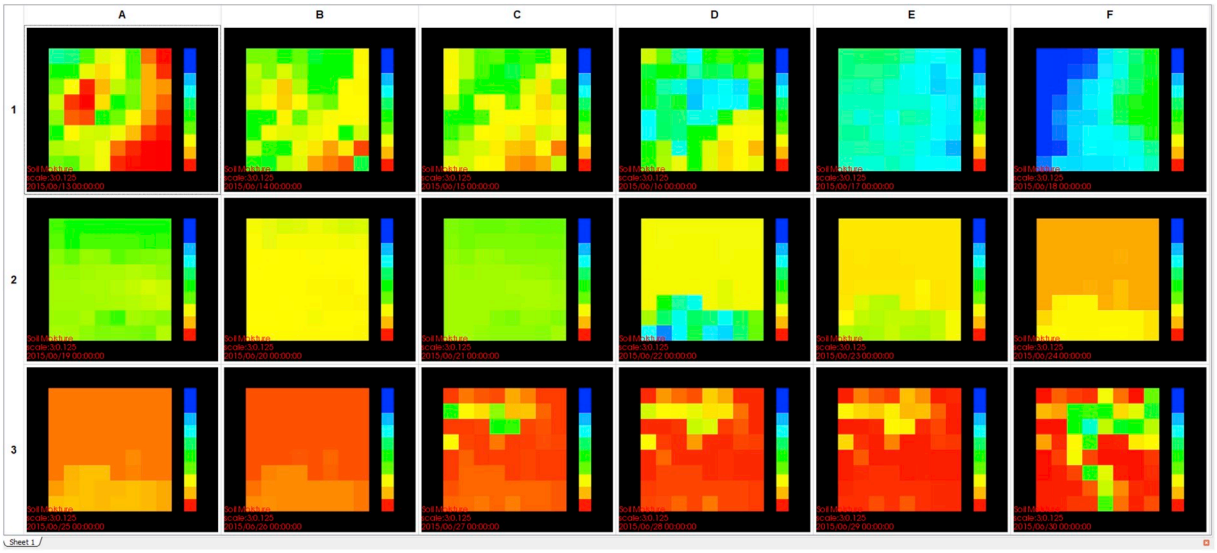

Fig. 19. Spatial distribution of the soil moisture from the VIC model simulation for eighteen different time steps.

all the forcing inputs and the model parameters. The *VicAgent* will then use such *dataset ids* to create the inputs, run VIC and wait for it to finish. Then, the *VicAgent* will read the VIC outputs and save them as datasets. *VicAgent* will send the *dataset_ids* of the outputs to the visualization components to generate plots.

The execution of the RetrieveDataSetAgent components is completed when all the information specified is downloaded. Only at that point, the execution of the subsequent component *VicAgent* is invoked. VisTrails has a color code for the workflow boxes shown in the working area (e.g., see the snapshot shown in Fig. 18) during an execution phase. For example, VisTrails will show green for the boxes already computed, yellow for the boxes under execution, orange for the boxes waiting for the preceded boxes to complete, gray for the boxes that have not yet been started or even considered for execution, red for the boxes that failed, and blue for the boxes that cannot be executed because of failures in the inputs. Fig. 18 shows a few different statuses of the boxes during the execution phase of one workflow case. It is worth mentioning that these color indicators are different from the colors used for the workflows saved in the provenance as shown in Fig. 5.

5.3. Results generated by the VIC model

The VIC model with a general *VicAgent* generates more than 160 output variables. For this specific illustration, the *VicAgent* is configured to generate only 2 outputs: soil moisture and evapotranspiration. Fig. 19 shows the VIC model simulated *soil moisture* plotted as an image per time step for the study area and for eighteen consecutive time steps.

The VIC simulated *evapotranspiration* is averaged over the study area for each time step and is shown in Fig. 20 as a time series. In order to generate such time series, the VIC results for the study area were re-

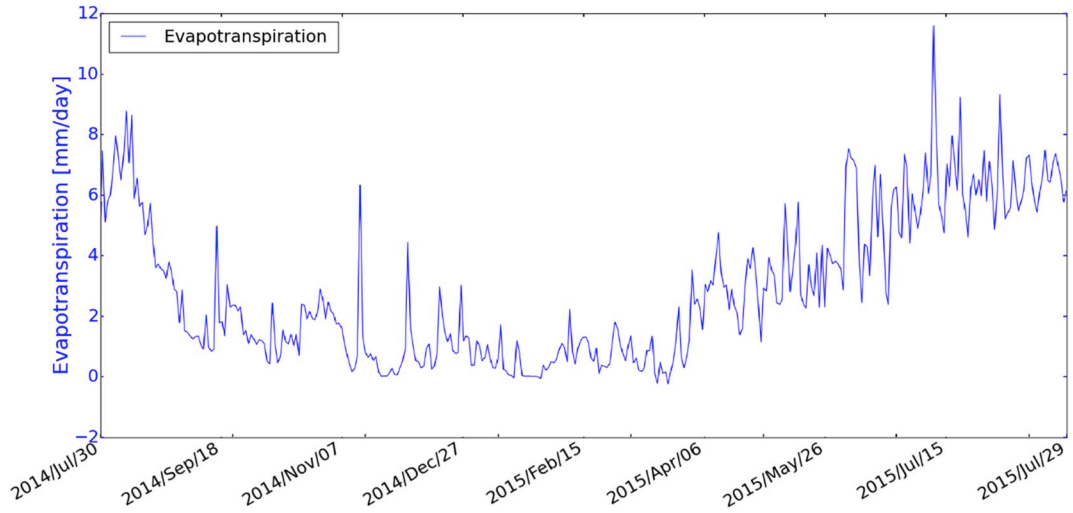

Fig. 20. The VIC generated time series of mean evapotranspiration over the Blue River basin. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

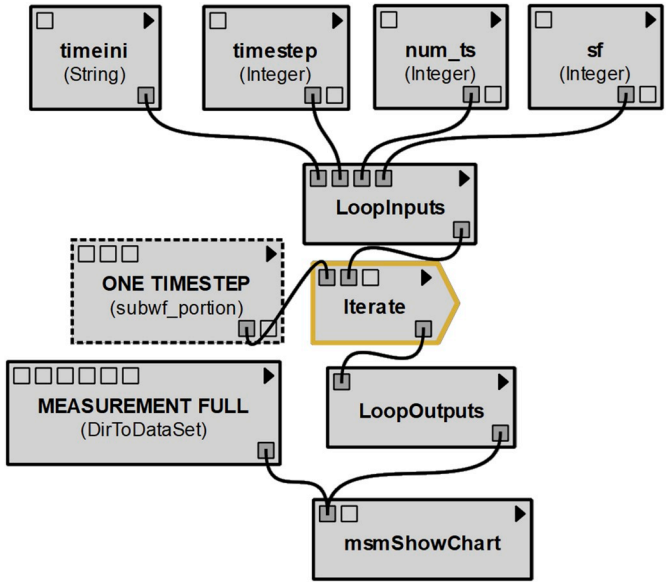

Fig. 21. Re-organized/re-designed workflow for input/output comparison with an iterative execution.

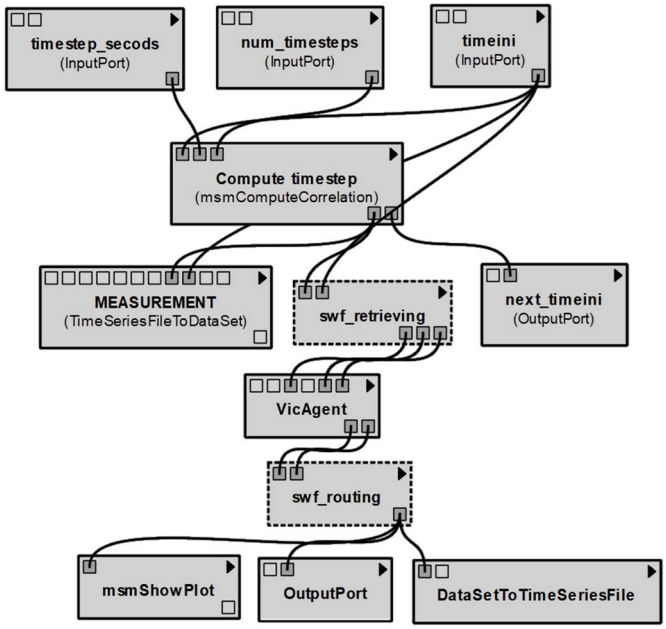

Fig. 22. Sub-workflow of the ONE_TIMESTEP for the coupling of VIC model and routing model.

scaled to scale zero using the MKS component to obtain the average.

In this example, we further illustrate how to change the configuration during the task execution in a timestep-by-timestep fashion. For instance, the execution described above has an overall retrieval phase in which all the inputs are downloaded for the entire simulation period, before the *VicAgent* is executed. This approach is widely used in model applications when using historical data, and is called time first then space. However, in the forecast applications with real-time data, a model needs to be run one step at a time for the entire study area and then move on to the next time step in an iterative fashion. In other words, one needs to download the model input data from each data source for only one or a few timesteps each time, then, run the model for that (or those) time step(s) for the entire study area, then, download the input data again for another time step or few time steps, and then, run the model for the new time step or new time steps again, etc.

To achieve this, the user can use the iterative functionality developed in *msm*. That is, by using the configuration with an iterative approach, it is possible to control the tasks executed at each time step or even repeat a set of activities until the convergence is reached before moving forward the next time step.

The readability of such workflow can be improved by using sub-workflows. This is a VisTrails feature that allows the user to group a set of modules to make them usable as an entire new aggregated module. This new component can be used as a single box in other workflows. In that way, workflows with many modules become simpler and organized in a hierarchical way for easier understanding.

Figs. 21 and 22 illustrate the use of the iterative function we developed in conjunction with the sub-workflow feature of VisTrails, running the VIC model coupled to a routing scheme. Specifically, Fig. 21 shows the main workflow after grouping some of the activities into the "ONE TIMESTEP" box and using the looping component (i.e., *Iterate*). The measured streamflow is also included in this workflow through the *TimeSeriesFileToDataSet* component. In this way, the VIC model simulated runoff after routing from the *msm* framework can be compared with the USGS measured streamflows. The iterative nature of this

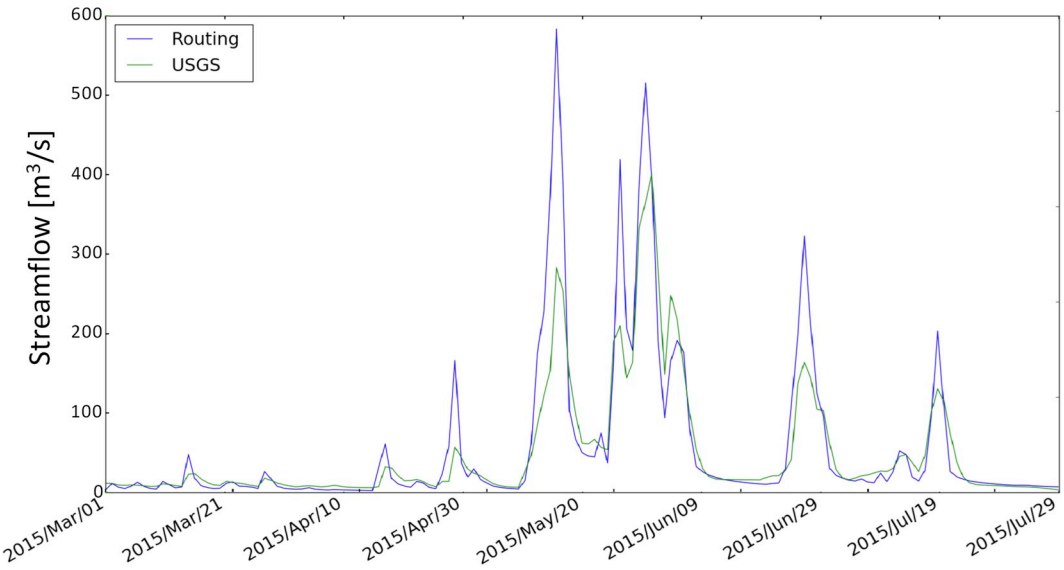

Fig. 23. Comparison of streamflows between measured and model computed for the study area.

workflow is accomplished through the *Iterate* component which makes repetitive use of the task named "ONE_TIMESTEP".

The "ONE TIMESTEP" box in Fig. 21 represents a subworkflow which is shown in Fig. 22. This component contains all the tasks needed to be performed at each time step. It starts by setting the looping parameters (time per step, number of steps and initial time), then retrieves all the inputs (subworkflow *swf_retrieving*), then runs the VIC model (VicAgent), and then sends the surface *runoff* and *baseflow* outputs to the routing model (subworkflow *swf_routing*). At the end of the execution of this subworkflow, one can compare the model simulated streamflows (i.e., after the VIC and routing models) with the USGS measured ones. In addition, the model results can also be saved as text files via the component *DatasetToTimeSeriesFile* shown in Fig. 22. Such saved model simulated results can be used for further analysis or other purposes by the user. For instance, the VIC model results obtained here are based on the default VIC model parameters. No calibration for the model parameters is conducted. If one wants to do model calibration and compare the results, one can save the results for later comparison studies. The saved results can also be plotted with different software tools at the user's preference. The VIC + routing model simulated results shown in Fig. 23 are obtained by running the VIC model with default parameters plus the routing model without conducting any parameter calibration. Such a plot is generated by the msmShowChart component in *msm*.

6. Conclusions

The fields of geosciences are entering a new era of big data, in which physically-based computational models, with easy access to and use of increasingly available and diverse data, are fundamentally important for scientific explorations. Our presented open data open modeling framework innovatively combines open data access and open modeling framework together and hence *simultaneously* addresses several critical issues faced by the broad community: (1) direct open machine-to-machine (M2M) access of data to models from external data sources; (2) graphical scientific workflow environment for scientists; (3) easy model sharing and heterogeneous/diverse model couplings without the need to share or change model source codes; (4) no required central administration of shared models and the modeling framework system; (5) data fusion of datasets from different sources and/or computational model outputs; (6) provenance management for reproducible computing; and (7) data exploration and visualization. These seven challenges have been repeatedly identified and highlighted by the geosciences communities. The unique features of the presented open data open modeling framework make it very suitable as an integrated solution for general geoscientists to overcome the above challenges in their explorations.

The prototype system of our framework, *msm*, has successfully demonstrated its usefulness and effectiveness in various aspects of scientific modeling activities. The openness of our framework by design indicates that when a user integrates his/her model(s) into *msm*, the impact of the integration process on user's model is none since there is no inter-dependency between the *msm* framework and the model codes. Moreover, the approach of the integration of an open scientific workflow engine (i.e., VisTrails) into our framework provides a unique provenance management among other desirable graphical workflow features, which directly facilitates reproducibility study in model couplings and thus is critical for users' research and its reliability.

Our plan of future work includes continuing to add more external data sources into the *msm*, such as hydro-meteorological datasets offered by NASA, NLDAS (North American Land Data Assimilation System) and NCALDAS which are composites of measurements from land stations and model simulations, as well as the satellite datasets like GPM for precipitation and SMAP for soil moisture. In addition, in the near future, we will add more NOAA datasets, such as NAM (North American Mesoscale Forecast System) and GFS (Global Forecast System), which are datasets from the model forecasts. Also, there are plans to add more computing facility choices (e.g., high performance computing, or cloud computing) into the *msm* framework for users to select based on the demand of each workflow activity. At present, we are developing new model agents to integrate models like the Distributed Hydrology Soil and Vegetation Model (DHSVM) and the PH-Redox Equilibrium model in C (PHREEQC) into the *msm* system. Moreover, a newer version of VIC (VIC5.0) is being integrated to *msm*. We are also working on developing generic model agent tools to help the user to develop his/her own model agent(s) with little to no coding to integrate the model(s) of the user's choices into *msm.* Our framework software, msm, is intended to be made available to the geosciences community as an open source at its due course. Part of this work involves making it available on the Ubuntu Operative System, since the *msm* has only been fully tested on Windows 10 OS so far.

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 [EAR-1245067] and [OAC-1835785] to the University of Pittsburgh and under [EAR-1245171] and [OAC-1835817] to Indiana University-Purdue University Indianapolis (IUPUI), respectively. We would like to thank the support from Thomas Adams, Jerad Bales, Richard Hooper, Steve Kempler, Pedro Restrepo, and William Teng. The second author also acknowledges the support from the William Kepler Whiteford Professorship from the University of Pittsburgh.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.envsoft.2020.104622.

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.

Barkstrom, B.R., 2010. A mathematical framework for earth science data provenance tracing. Earth Sci. Inf. 3, 167–196. https://doi.org/10.1007/s12145-010-0057-0.

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.

Bhat, U., Jadhav, S., 2010. Moving towards non-relational databases. Int. J. Comput. Appl. 1, 975–8887. https://doi.org/10.5120/284-446, 2010.

Bryant, R.E., 2007. Data-intensive supercomputing: the case for DISC. Sc, 1–22.

Callaghan, S., Deelman, E., Gunter, D., Juve, G., Maechling, P., Brooks, C., Vahi, K., Milner, K., Graves, R., Field, E., Okaya, D., Jordan, T., 2010. Scaling up workflow-based applications. J. Comput. Syst. Sci. 76, 428–446. https://doi.org/10.1016/j.jcss.2009.11.005.

Cherkauer, K.A., Lettenmaier, D.P., 1999. Hydrologic effects of frozen soils in the upper Mississippi River basin. J. Geophys. Res. 104, 19599–19610. https://doi.org/10.1029/1999jd900337.

Davidson, S.B., Freire, J., 2008. Provenance and scientific workflows. Proc. 2008 ACM SIGMOD Int. Conf. Manag. data - SIGMOD '08 1345. https://doi.org/10.1145/1376616.1376772.

Deelman, E., Chervenak, A., 2008. Data management challenges of data-intensive scientific workflows. 2008 Eighth IEEE Int. Symp. Clust. Comput. https://doi.org/10.1109/CCGRID.2008.24. Grid 687–692.

Deelman, E., Gannon, D., Shields, M., Taylor, I., 2009. Workflows and e-Science: an overview of workflow system features and capabilities. Future Gener. Comput. Syst. 25, 528–540. https://doi.org/10.1016/j.future.2008.06.012.

DeLuca, C., Oehmke, R., Neckels, D., Theurich, G., O'Kuinghttons, R., de Fainchtein, R., Murphy, S., Dunlap, R., 2008. Enhancements for Hydrological Modeling in ESMF. AGU Fall Meeting Abstracts, pp. 3–5.

Dou, W., Chen, J., Liu, J., Cheung, S.C., Chen, G., Fan, S., 2008. A workflow engine-driven SOA-based cooperative computing paradigm in grid environments. Int. J. High Perform. Comput. Appl. 22, 284–300. https://doi.org/10.1177/1094342007086227.

Gijsbers, P., 2010. Opportunities and limitations of DelftFEWS as a scientific workflow tool for environmental modelling. In: International Congress on Environmental Modelling and Software. Deltares, USA.

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

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.

Hutton, E.W.H., Piper, M.D., Peckham, S.D., Overeem, I., Kettner, A.J., Syvitski, J.P.M., 2014. Building Sustainable Software - the CSDMS Approach 1–6.

Kannan, A., Ostendorf, M., Karl, W.C., Castanon, D.A., Fish, R.K., 2000. ML parameter estimation of a multiscale stochastic process using the EM algorithm. IEEE Trans. Signal Process. 48, 1836–1840. https://doi.org/10.1109/78.845950.

Knapen, M.J.R., Janssen, S.J.C., Roosenschoon, O.R., Verweij, P.J.F.M., 2011. Evaluating and improving OpenMI as a model integration platform across disciplines. MODSIM 2011 - 19th Int. Congr. Model. Simul. - Sustain. Our Futur. Underst. Living with Uncertain. 1223–1229.

Kouzes, R., Anderson, G., Elbert, S., 2009. The changing paradigm of data-intensive computing. IEEE Comput. Soc. 42, 26–34. https://doi.org/10.1109/MC.2009.26.

Krzhizhanovskaya, V.V., Shirshov, G.S., Melnikova, N.B., Belleman, R.G., Rusadi, F.I., Broekhuijsen, B.J., Gouldby, B.P., Lhomme, J., Balis, B., Bubak, M., Pyayt, A.L., Mokhov, I.I., Ozhigin, A.V., Lang, B., Meijer, R.J., 2011. Flood early warning system: design, implementation and computational modules. Procedia Comput. Sci. 4, 106–115. https://doi.org/10.1016/j.procs.2011.04.012.

Liang, X., Xie, Z., 2001. A new surface runoff parameterization with subgrid-scale soil heterogeneity for land surface models. Nonlinear Propag. Multi-scale Dyn. Through Hydrol. Subsystems 24, 1173–1193. https://doi.org/10.1016/S0309-1708(01)00032-X.

Liang, X., Xie, Z., 2003. Important factors in land-atmosphere interactions: surface runoff generations and interactions between surface and groundwater. Glob. Planet. Chang. 38, 101–114. https://doi.org/10.1016/S0921-8181(03)00012-2.

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, 14415–14428. https://doi.org/10.1029/94JD00483.

Liang, X., Lettenmaier, P., Wood, E.F., 1996a. One-dimensional statistical dynamic representation of subgrid spatial variability of precipitation in the two-layer variable infiltration capacity model. J. Geophys. Res. Atmos. 101 https://doi.org/10.1029/96JD01448.

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

Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E.A., Tao, J., Zhao, Y., 2006. Scientific workflow management and the Kepler system. Concurrency Comput. Pract. Ex. 18, 1039–1065. https://doi.org/10.1002/cpe.994.

Ma, J., Cao, J., Zhang, Y., 2010. Efficiently supporting secure and reliable collaboration in scientific workflows. J. Comput. Syst. Sci. 76, 475–489. https://doi.org/10.1016/j.jcss.2009.11.008.

Maurer, E.P., Wood, A.W., Adam, J.C., Lettenmaier, D.P., 2002. A long-term hydrologically based dataset of land surface fluxes and states for the conterminous United States: update and extensions. J. Clim. 26, 9384–9392. https://doi.org/10.1175/JCLI-D-12-00508.1.

McPhillips, T., Bowers, S., Zinn, D., Ludäscher, B., 2009. Scientific workflow design for mere mortals. Future Gener. Comput. Syst. 25, 541–551. https://doi.org/10.1016/j.future.2008.06.013.

Moore, R., Tindall, I., 2005. An overview of the open modelling interface and environment (the OpenMI). Environ. Sci. Pol. 8 (3 SPEC. ISS.), 279–286. https://doi.org/10.1016/j.envsci.2005.03.009. In this issue.

Nijssen, B., O'donnell, G.M., Hamlet, A.F., Lettenmaier, D.P., 2001. Hydrologic sensitivity of global rivers to climate change. Clim. Change 50, 143–175. https://doi.org/10.1023/A:1010616428763.

Parada, L.M., Liang, X., 2004. Optimal multiscale Kalman filter for assimilation of near-surface soil moisture into land surface models. J. Geophys. Res. D Atmos. 109, 1–21. https://doi.org/10.1029/2004JD004745.

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.

Rajib, M., Merwade, V., I Luk, K., Lan, Z., Carol, S., Shandian, Z., 2016. SWATShare - A web platform for collaborative research and education through online sharing, simulation and visualization of SWAT models. Environ. Model. Software 75, 498–512. https://doi.org/10.1016/j.envsoft.2015.10.032.

Ravindran, N., Liang, Y., Liang, X., 2010. A labeled-tree approach to semantic and structural data interoperability applied in hydrology domain. Inf. Sci. 180, 5008–5028. https://doi.org/10.1016/j.ins.2010.06.015.

Rizzoli, A.E., Davis, J.R., Abel, D.J., 1998. Model and data integration and re-use in environmental decison support systems. Decis. Support Syst. 24, 127.

Salas, D.E., Liang, X., 2013. An introduction to multi-scale Kalman smoother-based framework and its application to data assimilation. L. Surf. Obs. Model. Data Assim. 275–334. https://doi.org/10.1142/9789814472616_0010.

Salas, D., Liang, X., Liang, Y., 2012. A systematic approach for hydrological model couplings *. Int. J. Commun. Netw. Syst. Sci. 5, 343–352. https://doi.org/10.4236/ijcns.2012.56045.

Sonntag, M., Görlach, K., Karastoyanova, D., Leymann, F., Malets, P., Schumm, D., 2011. Views on scientific workflows. Lect. Notes Bus. Inf. Process. 90 LNBIP 321–335. https://doi.org/10.1007/978-3-642-24511-4_25.

University of Colorado Boulder, 2012. CSDMS Final Report.

Vertessy, R.A., Rahman, J.M., Watson, F.G.R., Argent, R.M., Cuddy, S.M., Seaton, S.P., 2002. A regional water quality model designed for a range of users and for retrofit and re-use. In: International Environmental Modelling and Software Society (IEMSs) 2012 International Congress on Environmental Modelling and Software. Managing Resources of a Limited Planet: Pathways and Visions under Uncertainty. Sixth Biennial Meeting. Lugano, Switzerland, pp. 312–317.

Wang, L., Lu, S., Fei, X., Chebotko, A., Victoria Bryant, H., Ram, J.L., 2009. Atomicity and provenance support for pipelined scientific workflows. Future Gener. Comput. Syst. 25, 568–576. https://doi.org/10.1016/j.future.2008.06.007.

Werner, M., Schellekens, J., Gijsbers, P., van Dijk, M., van den Akker, O., Heynert, K., 2013. The Delft-FEWS flow forecasting system. Environ. Model. Softw 40, 65–77. https://doi.org/10.1016/j.envsoft.2012.07.010.