International Environmental Modelling and Software Society (iEMSs)
7th International Congress on Environmental Modelling and Software
San Diego, California, USA, D.P. Ames, N. Quinn (Eds.)
http://www.iemss.org/society/index.php/iemss-2014-proceedings

Plug and Play Component Modeling — The CSDMS2.0 Approach

James P Syvitski, Eric Hutton, Mark Piper, Irina Overeem, Albert Kettner, and Scott Peckham
CSDMS Integration Facility, INSTAAR, University of Colorado, Boulder CO, 80309
(James.syvitski@colorado.edu)

Abstract: The CSDMS2.0 focus is on developing a software modelling environment that offers the earth and ocean communities products to enable easier penetration into the world of high performance computing, plug-and-play component modelling, and access to vetted open source surface-dynamics models. Protocols and standards define modelling interfaces, standard names, service components, and DOIs labelling.

***Keywords**:* Semantic mediation; model framework; model coupling.

1 INTRODUCTION

The Community Surface Dynamics Modeling System, or CSDMS, develops, integrates, archives and disseminates software to define the earth's surface dynamics. CSDMS coordinates a large (67 country) international community in building a toolbox of surface dynamics component models. The challenge encapsulates the variety of users, the volunteer effort, and the hundreds of very different models. The CSDMS Integration Facility develops the cyber-architecture and framework, to populate a plug-and-play component-modeling environment, able to operate within a cloud-sourced High Performance Computing environment.

2 WMT: THE CSDMS WEB MODELING TOOL

The CSDMS Web Modeling Tool (WMT) is the web-based successor to the desktop Component Modeling Tool (Peckham et al., 2013). WMT provides a client-side drag-and-drop graphical interface and a server-side database and application programming interface (API) that allows users to build and run coupled surface dynamics models on a high-performance computing cluster (HPCC) from a web browser on a desktop, laptop or tablet computer. With WMT, a user can:

- Select a component model from a list to run in standalone mode,
- Build a coupled model from multiple components organized as nodes of a tree structure,
- View and edit the parameters for these model components,
- Upload custom input files to the server,
- Save models to a server, where they can be accessed on any Internet-accessible computer,
- Share saved models with others in the community, and
- Run a model by connecting to a remote HPCC where the components are installed.

Although WMT is web-based, the building and configuration of a model can be done offline. Reconnection is necessary only when saving a model and submitting it for a run.

2.1 Client Overview

WMT presents a streamlined graphical interface, consisting of three scrollable panels, or views, and one menu (Fig. 1).

- *Components view* — a list of Common Component Architecture (CCA) components (Armstrong et al., 1999) that are available on the HPCC.
- *Model view* — a component can be dragged from the Components view into the tree structure of the Model view. Once in the tree, the component displays its CCA uses ports as leaves on the tree. By adding other components that provide ports for these open leaves, a coupled model can be created. A component instance that provides feedback to the coupled model is displayed as a link (e.g. CEM in Fig. 1).
- *Parameters view* — displays model parameters in the Model view for viewing and editing. Type and range checks are performed immediately on any parameter that is modified.
- *Model menu* — provides selections for opening, closing, saving, deleting and running models. Models developed with WMT are currently saved to a server at CSDMS. When a model run is initiated, the user is provided with a list of available HPCC nodes on which it can be run, and prompted to provide login credentials for the selected HPCC.

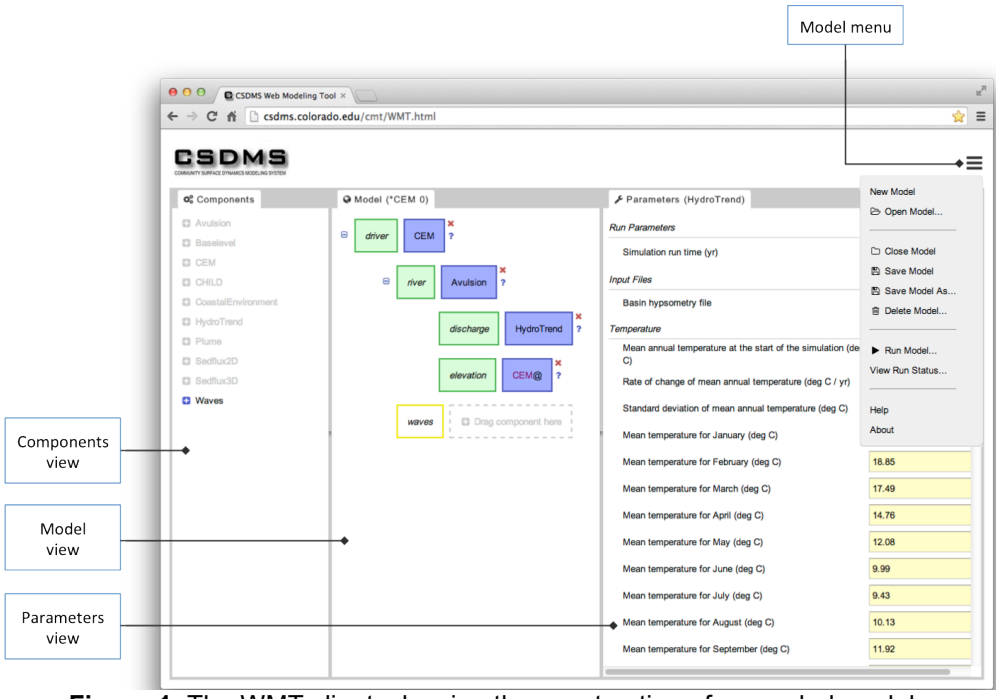

Figure 1. The WMT client, showing the construction of a coupled model.

A model run can be initiated and its status (uploaded, staged, launched, complete) viewed; on completion, the model output can be retrieved by FTP.

2.2 Client Architecture

The WMT client is written with GWT (GWT Project, 2013), a toolkit for building browser-based applications. Using GWT over native JavaScript offers the advantage that the client code is written in Java, which allows the developer to employ object-oriented design principles and mature Java development tools such as Eclipse. GWT provides a development mode for rapid prototyping and debugging, and a production mode, where the Java source is compiled to JavaScript for deployment on the web. GWT is used in several Google projects, and boasts a large user community. GWT is supported on all modern browsers,

including Firefox, Internet Explorer (6+), Safari (5+), Chromium/Chrome and Opera. The WMT client uses the model-view-presenter (MVP) pattern (Fowler, 2006):

- *Model*: The layer providing data for the application.
- *View*: The user interface for viewing and modifying the application data (Fig. 2).
- *Presenter*: The mediator between the Model and the View. Messages are passed between View and Presenter, and between Model and Presenter, but the View and Model are designed to have no knowledge of the other.

MVP architecture separates the domain logic of an application, where rules are set for how data are stored and modified, from the client interface, where the user can interact with the data. This separation of responsibilities makes it easier to test, modify and maintain an application. MVP is particularly useful in applications that have a graphical user interface, since the testing of the interface often must be done manually (Wellman, 2008). The GWT Project recommends MVP for GWT applications (GWT Project, 2010).

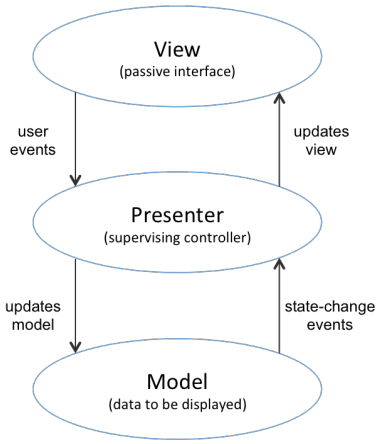

Figure 2. The Model-View-Presenter (MVP) architecture pattern is adapted from Wellman (2008).

2.3 Server Overview

The WMT server is a RESTful (Fielding, 2000) web application that provides a uniform interface through which client applications interact with the CSDMS model-coupling framework. Although opaque to a client, behind the WMT server is a layered system that consists of the following resources:

- A database server that contains component, model, and simulation metadata
- One or more execution servers on which simulations are launched
- A data server from which simulation output is stored and can be downloaded.

The database server provides, as JSON encoded messages, the component metadata necessary for an end-user to couple components, and set input parameters. The metadata includes descriptions of component exchange items, uses and provides ports, as well as user-modifiable input parameters. It is held on a server separate from the execution server so that it is easily and quickly accessed without need to connect to a firewalled or inaccessible execution server. Execution servers are computational resources that contain the software stack needed to run a coupled or uncoupled model simulation. These servers can range from large high performance computing clusters, to smaller web servers, or even to an end-user's personal computer. The requirements are only that the WMT server has network access to the execution server and that the CSDMS software stack is installed on the server. This includes the CCA-toolchain, the CSDMS framework tools, and compiled shared libraries for each of the component models. Once a

simulation completes, its output is packaged and uploaded to a data server where it is stored and from which the end-user is able to download it as a single compressed archive file.

2.4 Incorporating BMI Models into the CSDMS Modeling Framework

The CSDMS Basic Modeling Interface (BMI) specification (Peckham et al., 2013) describes an application-programming interface (API) for scientific numerical models. The interface identifies entry points into software components to provide a calling application with the necessary level of control over the components that is necessary for two-way model coupling. CSDMS as well as other modeling frameworks, such as ESMF (Hill et al., 2004), OpenMI (Gregerson), and OMS (David et al., 2002), have identified the minimum granularity of control to be an interface that provides functionality to initialize, update, and finalize a component model. BMI establishes precise names, calling signature and return types for each of these functions in a language agnostic manner and also provides bindings for each of the CSDMS supported languages (Python, C, C++, Fortran, Java). Because modeling-coupling frameworks share this common requirement, any model that exposes a BMI can be incorporated into any number of frameworks, not just the CSDMS model-coupling framework.

A component model that strictly follows the BMI specification allows for a streamlined workflow that enables it to function inside the CSDMS model-coupling framework. Templates exist for each supported language, and consists of boilerplate code that makes functions calls to CSDMS and CCA services but will only access the underlying component model through BMI function calls. Wrapper templates will never make reference to component-specific functions or data. Rather, component control (initialize, update, finalize) and data access (getters, setters) is always through BMI functions. Then, at run-time, these function references are linked dynamically to the shared library that contains the compiled BMI implementation for the appropriate component. BMI functions provide most component metadata (names of input / output exchange items). Additional metadata the CSDMS framework needs to incorporate a new component include:

- Source code: author(s), license, version, link to source code, etc.
- Input files: File templates that contain placeholders for adjustable parameters
- Input parameters: description of user-adjustable input parameters

These additional source code metadata provide end users with standardized model information. If a component requires input files to operate, the component contributor must provide template versions of these files. Additionally, if the contributor would like some of these parameters to be editable by an end user, the template files should include placeholders for the adjustable parameters. A placeholder is simply a key name, which refers to the parameter, enclosed in curly-braces. Each input parameters must be described (float, int, string, etc.), along with suggested ranges, and a short description of the parameter. This additional metadata is used by various CSDMS tools to enhance the end-user experience and help overcome the “black-box syndrome” that results from users running models without being aware of the model’s inner workings.

3 CSDMS Standard Names and Model Metadata

In order to develop a modeling framework that would allow automated coupling of models and data sets from different contributors, semantic mediation or matching is required. Each model and data set uses its own terms or labels for input and output variable names, often domain-specific or abbreviated. To ensure that one model’s output variable is appropriate for use as another model’s input, a precise

description of the variable, its units and certain other attributes are required. To address this need, a semantic matching called the CSDMS Standard Names was developed. These standardized names avoid domain-specific terms and abbreviations, are based on a set of rules or conventions and are designed to eliminate ambiguity. Contributors of models or data sets are asked to map each of their own terms to the appropriate "long name" in the CSDMS Standard Names. For models or data sets, this can be done by implementing a CSDMS Basic Model Interface (BMI) that provides standardized self-description as well as model control functions (i.e. initialize, update, finalize). The model control functions provide the modeling framework with fine-grained control of the model and allow heterogeneous models to be coupled within the CSDMS framework. Contributors create the mapping (e.g. Python dictionary) from their model's internal variable names to CSDMS standard names, and supply information about the spatial grid, time-stepping scheme, and assumptions.

The CSDMS Standard Names provide a comprehensive set of naming rules and patterns for creating unique labels for model variables that are not specific to any particular modeling domain. These naming conventions consist of an extensive set of patterns that cover a wide variety of cases gleaned from models in the CSDMS repository as well as from the CF Standard Names. They are designed to have features such as parsability and natural alphabetical grouping. CSDMS Standard Names for variables always consist of an object part and a quantity/attribute part and the quantity part may have an operation prefix that can consist of multiple operations. Unlike the CF Standard Names, assumptions and explanations are not included in the name itself; they are instead selected from a standardized list and specified with <assume> tags in a Model Metadata File (XML) that clarifies how a given model uses the name. The additional metadata in this file supports the names by including assumptions, units, equations used, boundary conditions, object name source, geo-referencing information (e.g. standard ellipsoid, datum and projection names), and so on, thereby fully describing the model and its associated input and output variables.

At the highest level, CSDMS Standard Names (v. 0.7.1) consist of Model Variable Names and Model Metadata Names, and consist of numerous supporting parts. Model Variable names are constructed from valid Object Names, Operation Names and Quantity Names, and the Quantity Names often include a Process Name. Model Metadata Names attempt to provide complete metadata for describing key attributes of a model other than the input and output variable names and are stored in Model Metadata Files. The Model Metadata Names include additional metadata to support the variable names, such as units, object name source and geo-referencing data (e.g. standard ellipsoid, datum and projection names) and different types of Assumption Names. For further detail, readers are referred to *http://csdms.colorado.edu/wiki/CSDMS_Standard_Names*. Developers can continue to use whatever variable names they want to in their model code or data set, but must then "map" each of their internal variable names to the appropriate CSDMS standard name in their BMI implementation.

3 CSDMS SERVICE COMPONENTS

CSDMS employs two versions of ESMF regridding tools, in combination with CSDMS **regridding tools**. The serial version is used on single-processor platforms; Message Passing Interface (MPI) is employed for use with multiple processors. The parallel version of the mapper scales nearly linearly up to several dozen processors. These mappers map elements from one unstructured grid to another. While grid elements are typically either three or four sided, ESMF offers a more general tool that supports polygonal cells with an arbitrary number of sides. This makes it possible for a model that uses watershed polygons as its

"computational cells" to obtain spatially interpolated rainfall data from a data source that uses rectangular cells.

Earth surface process models may use fixed or adaptive time-stepping schemes, and coupled models may use time-steps that are significantly different in size. A snowmelt model may employ hourly time-steps and be coupled to a channelized flow model that uses time-steps of several seconds. "Temporal misalignment" may have unintended consequences. Application of a smooth interpolation function to each of the state variables in the model with the larger time-step allows the smaller time-step model to retrieve and use interpolated values that vary more smoothly and which can be updated (with every time-step) with very low computational cost. A new time **interpolation service component** is made available too components run through the CSDMS WMT framework.

CSDMS has created **file-writing tools** for use within the CSDMS framework. The new writer class receives data from a component model and outputs the data to either a VTK file or a NetCDF file. VTK files are written in binary using the “new-style” XML format for VTKs. For structured grids, NetCDF files follow the CF conventions. Since there are currently no CF standards for storing unstructured meshes in NetCDF format, we provide for an additional format: (1) Values of the x- and y-coordinate for each node; (2) Array of integers as indices into data arrays for each element of the mesh; and (3) Array of integers that indicate the shape of each element (triangle, polygon, cube, etc.). Element types are defined in the same way as the VTK standard. Variable values (at either nodes or elements) are then listed with the same ordering as the x and y, or connectivity arrays.

4 BEYOND THE BLACK BOX MODEL

A “black box” model can be manipulated in terms of its input and then generates output for a user without having knowledge of its internal workings or without being able to get insight in the model engine, or its process routines. The model algorithms and their implementation are then "opaque" or “black”. CSDMS strives to take models and components beyond black box state. Science practice in principle condemns a “black box”; it is of crucial importance to know the level of process simplification within a model engine and the implementation into equations and a numerical scheme. Without such transparency the analysis of model output is of much less value.

CSDMS also offers web-based metadata on each model, submitted by the original developers, and maintained as a wiki database and thus updatable by users themselves. CSDMS maintains an online model repository where the original code can be downloaded, viewed, compiled and run. The model engines are thus available to any user. WMT components are documented in more detail on the CSDMS wiki (Figs. 3 and 4). With WMT, a user can access: 1) more extensive model description, 2) notes on input parameters, 3) key model equations, 4) notes on coupling ports, and 5) essential references provided by the original developer.

Pedagogical research shows the importance of hands-on activities in learning (Campbell et al., 2013). Students show significant learning gains when they work with inquiry-based modules and receive instantaneous feedback (Fogleman et al., 2011). The CSDMS Educational Working Group noted that hands-on modeling labs are more valuable if they are combined with mathematical and physics problems based on the careful analysis of the underlying model engine (Schwarz et al., 2009). CSDMS offers an educational repository with modeling labs for graduate and advanced undergraduate students. These labs support students to run models, analyze output and highlight some critical aspect of the modeled processes and model engine, the selection of which depend on the learning objective and lesson plan.

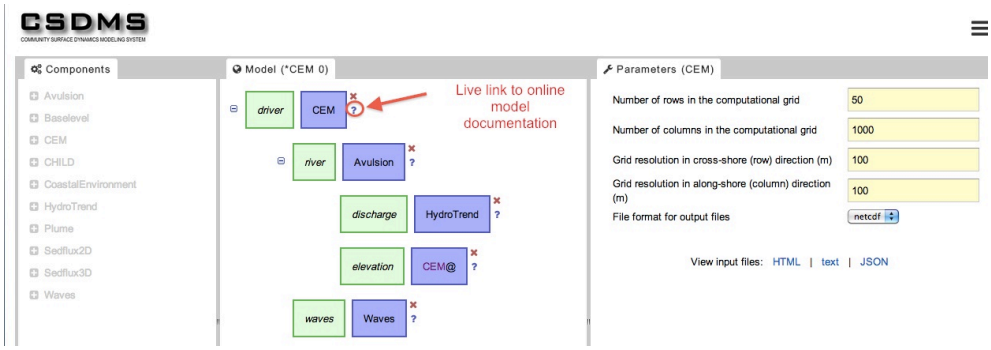

Figure 3. All components in the WMT have live links to online detailed documentation maintained on the CSDMS wiki.

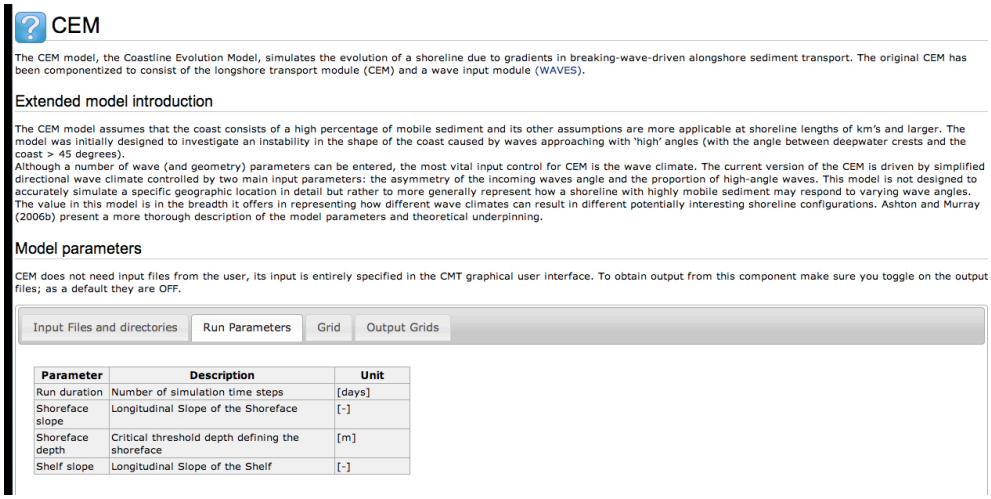

Figure 4. Detailed model description of the CEM-Coastline Evolution Model as displayed within WMT.

5 DIGITAL OBJECT IDENTIFIERS FOR NUMERICAL MODELS

All code in CSDMS is open source (see Ince et al., 2012). Source code exposes the scientific hypotheses embodied in a numerical model, and the solution to the set of equations. Code transparency allows for full peer review and replication of results — the foundation of modern science. Code transparency allows for reuse in new and clever ways, and reduces redundancy. CSDMS ensures that model developers receive recognition for their work, even when code is submitted and not yet described in a scientific journal by adopting the Digital Object Identifier (DOI). The DOI system provides a unique identification to content that is available on digital networks. Since 2005 DOIs were made available for research data (Paskin, 2005). CSDMS is the first to assign Digital Object Identifiers (DOIs) to numerical source code. The advantages of adopting a DOI system for models include:

- Guarantee credit to a model developer.
- Reuse and replication of research with direct access to a referenced code.
- Higher visibility — content with a DOI is 5 times more likely to deliver active links.
- The opportunity for funding agencies to track usage, so to measure impact.

CSDMS collaborates with Integrated Earth Data Applications (IEDA), a formal Publication Agent of the DOI system through the German National Library of Science and Technology, to assign unique identifiers for those models that contain metadata and are physically part of the CSDMS repository. An archive of all numerical models of the CSDMS model repository that have a DOI, together with

limited metadata and source code is provided to IEDA to guaranty access beyond the CSDMS program; a DOI for an object is permanent, whereas its location and other metadata may change in future. A new DOI is provided for each new version of a model (i.e. major upgrade/version of the source code). CSDMS uses Apache Subversion, better known as SVN, for tracking source code versioning and revision control so that current and past releases and changes can be accessed through the web. As of March 2014, 109 models within the CSDMS model repository have a DOI. Model source code can be viewed as 'data' and therefore CSDMS endorses citations defined by DataCite guidelines (Brase, 2010). Following these guidelines, CSDMS strongly recommends the following structure for citing a model: *ModelDeveloper (PublicationYear). ModelName, ModelVersion. Identifier.*

ACKNOWLEDGMENTS

The CSDMS Integration Facility operates under a continuing grant 0621695 from the U.S. National Science Foundation.

REFERENCES

Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., McInnes, L., Parker, S., Smolinski, B., 1999, Toward a common component architecture for high-performance scientific computing. In: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing.

Brase, J., 2010, Datacite: A global registration agency for research data, *Working Paper Series des Rates für Sozial- und Wirtschaftsdaten*, No. 149.

Campbell, K., Overeem, I., Berlin, M., 2013, Taking it to the streets: the case for modeling in the geosciences undergraduate curriculum. Computers & Geosciences 53, 123-128.

David, O., Markstrom, S.L., Rojas, K. W., Ahuja, L.R., Schneider, I. W., 2002, The object modeling system. In: Ahuja, L.R., Ma, L., Howell, T. (Eds.), Agricultural System Models in Field Research and Technology Transfer. Lewis Publ. CRC Press, 317-331.

Fielding, R. T., 2000, Architectural styles and the design of network-based software architectures. Doctoral dissertation, University of California.

Fogleman, J., McNeill, K., Krajcik, J., 2011, Examining the effect of teachers adaptations of a Middle School Science Inquiry-Oriented Curriculum Unit on Student Learning. Journal of Research in Science Teaching, 48,149-169.

Fowler, M., 2006, GUI Architectures http://martinfowler.com/eaaDev/uiArchs.html.

Gregerson, J.B., Gijsbers, P.J., Westen, S.J.P., 2007, OpenMI: open modeling interface. Journal of Hydroinformatics 9 (3), 175-191.

GWT Project, 2010. MVP Architecture <http://www.gwtproject.org/articles/mvp-architecture.html>.

Hill, C., DeLuca, C., Balaji, V., Suarez M., da Silva, A., 2004, The architecture of the earth system modeling framework. Computing in Sci. & Engin. 6, 18-28.

Ince, D.C., Hatton, L., and Graham-Cumming, J., 2012, The case for open computer programs. *Nature*, 482, 485-488.

Paskin, N., 2005. Digital Object Identifiers for scientific data. Data Science Journal, 4, 12-20.

Peckham, S.D., Hutton, E.W.H., Norris, B., 2013, A component-based approach to integrated modeling in the geosciences: The design of CSDMS. Computers & Geosciences 53, 3-13.

Schwarz, C.V., Reiser, B.J., Davis, E.A., Kenyon, L., Achér, A., Fortus, D., Shwartz, Y., Hug, B., and Krajcik, J.S. 2009, Developing a learning progression for scientific modeling: Making scientific modeling accessible and meaningful for learners. Journal of Research in Science Teaching, 46, 632-654.

Wellman, D., 2008, Google Web Toolkit: Writing Ajax Applications Test First. Better Software Magazine, 26-32.