Jul 12th, 8:30 AM - 8:50 AM

From OpenMI to HydroCouple: Advancing OpenMI to Support Experimental Simulations and Standard Geospatial Datasets

Caleb A. Buahin
Utah State University, caleb.buahin@aggiemail.usu.edu

Jeffery S. Horsburgh
Utah State University, jeff.horsburgh@usu.edu

International Environmental Modelling and Software Society (iEMSs)
8th International Congress on Environmental Modelling and Software
Toulouse, France, Sabine Sauvage, José-Miguel Sánchez-Pérez, Andrea Rizzoli (Eds.)
http://www.iemss.org/society/index.php/iemss-2016-proceedings

From OpenMI to HydroCouple: Advancing OpenMI to Support Experimental Simulations and Standard Geospatial Datasets

Caleb A. Buahin[a] **and Jeffery S. Horsburgh**[a]
[a] *Utah Water Research Laboratory, Utah State University (caleb.buahin@aggiemail.usu.edu, jeff.horsburgh@usu.edu)*

Abstract: HydroCouple is a cross-platform, component-based modeling interface definition that largely follows the Open Modeling Interface 2.0 (OpenMI) specification. HydroCouple provides advancements to better facilitate those experimental model investigations that fall into the so-called "embarrassingly parallel" class of simulations, including uncertainty assessment, ensemble forecasting, and optimization simulations. Additionally, HydroCouple explicitly incorporates low level interface definitions for multi-dimensional datasets and geospatial data formats including the Open Geospatial Consortium's Simple Feature Access specification, raster datasets, and meshes that are widely used in the earth systems and environmental modeling field. In this paper, we describe these and other advances provided by the HydroCouple interface definitions. We also illustrate how these advances can be used to facilitate parallelized experimental model simulations that have so far been challenging in OpenMI and other component-based modeling frameworks.

***Keywords**:* Component-Based Modeling; OpenMI; Optimization, Experimental Model Simulations; Cross-Platform.

1 INTRODUCTION

The case for using the component-based modeling approach to address complex earth systems and environmental challenges is compelling because of how it naturally complements the goals of integrated assessment. Integrated assessment seeks to provide relevant information within a decision making context that brings together a broader set of areas, methods, styles of study, or degrees of certainty, than would typically characterize a study of the same issue within the bounds of a single research discipline (Parson, 1995; Laniak *et al.*, 2013). The component-based modeling approach facilitates this goal by prescribing standardized modeling frameworks and interfaces that can be adopted by model developers from different disciplines so that models can be readily linked together to simulate feedbacks between different domains that are often ignored or simulated using simplistic assumptions. The benefit of this is that more holistic evaluations can be performed.

The success of the component-based approach for integrated modeling is predicated on the premise that the wide adoption and acceptance of a component-based modeling framework/standard that fulfils modeling needs across a wide range of disciplines will spur the development of a critical mass of components for that framework/standard. Several of these frameworks and standards with varying degrees of complexity have been proposed for the earth systems and environmental modeling field, including the Earth Systems Modeling Framework (ESMF, Hill *et al.*, 2004), Community Surface Dynamics Modeling System (CSDMS, Peckham *et al.*, 2013), the Object Modeling System (OMS, David *et al.*, 2002), etc. So far, existing frameworks have either used data structures that have an excessively high level of data abstraction, are strongly tied to a specific research discipline, or require complex software stacks for their use.

The Open Modeling Interface (OpenMI; Moore and Tindall, 2005) specification provided a novel path forward by only proposing a set of standardized, programming language neutral interface definitions for how components must be developed so that they can be coupled to exchange data during a simulation.

In the first OpenMI specification (i.e., OpenMI 1.4), components exchange in-memory data directly at runtime through a pull-based data exchange mechanism, where one component requests the data it needs from other components and waits for a response before proceeding with its computations (Gregersen *et al.*, 2005). In the latest OpenMI 2.0 specification, a new control flow has been added called the loop-driven approach where the simulation system loops over all components and each component checks if it needs to update itself based on whether the in-memory data a component needs has been supplied by source components linked to it (Moore, 2010). This direct data exchange between components simplifies the use of OpenMI by removing the need for an underlying complex software framework. The interface definitions provided by OpenMI are object oriented with some support for low level abstractions of datasets that are often used in the modeling field. The OpenMI interfaces are also organized into clear, logical, and well documented inheritance relationships. OpenMI has been one of the more widely used and cited component-based modeling standards (with over 1400 citations on Google Scholar) and has been formally adopted by the Open Geospatial Consortium (OGC) as a standard.

Our experience in reviewing and using the OpenMI standard for applications in the hydrologic and hydrodynamic modeling field has, however, revealed some areas where advances can be made. Although OpenMI provides some low level interface definitions for datasets that are routinely used in the modeling field, it does not provide direct support for some of the more standard geospatial dataset formats and their associated topological relationships that are often needed by model developers. Secondly, while OpenMI currently supports the class of simulations we are referring to as experimental simulations that involve running coupled model components hundreds to thousands of times with varied inputs (e.g., optimization, multi-scenario evaluations, ensemble, or parameter estimation and uncertainty assessment simulations), users are limited to two approaches. The first approach involves executing the coupled model components multiple times using new inputs each time in a sequential fashion until a solution is obtained. This sequential approach can lead to excessively long simulation times that are impractical for models that have simulation times on the order of minutes or greater. The second approach that can used is to take advantage of the inherent quality of these types of experimental simulations where each simulation is independent from another to parallelize simulations to reduce simulation times. However, within OpenMI, these type of parallelized simulations can only be done by manually creating clones of the coupled model components and executing each clone in parallel. The challenge with this approach is that for many experimental simulations it is not known in advance how many clones are needed for a solution. Also, in a complex coupled coupled modeling system involving many components, it becomes difficult to track the provenance of cloned components.

Finally, although OpenMI provides interface definitions that may be implemented in any programming language, the interface implementations provided by the OpenMI developers have only been provided in the C# and Java programming languages. There exists, however, a large base of legacy models that have been written using natively compiled programming languages like C, C++, and Fortran that cannot be easily rewritten as OpenMI compliant components using C# or Java. The approach that has been recommended to address this difficulty is to convert an existing model's code into a library that exposes relevant functions and datasets that can be accessed externally from an OpenMI compliant component wrapper library (Gregersen *et al.*, 2007). This approach, nevertheless, has some drawbacks that we have previously illustrated (Buahin and Horsburgh, 2015). C# and Java are interpreted programming languages and are typically compiled into an intermediary bytecode that is interpreted by a virtual machine (VM) infrastructure – e.g., the .NET Framework Common Language Runtime for C# and the Java Virtual Machine (JVM) for Java – before being executed natively on a machine. The intermediary work done by the VM framework can introduce computational penalties, especially when marshalling data between a wrapper component and an underlying legacy model library written using a natively compiled language.

These challenges were the underlying motivation behind the development of HydroCouple. In the following sections, we describe the new interface definitions that have been added to or modified from the OpenMI standard to address these challenges. We also illustrate how these new interface definitions can be used for a hypothetical optimization simulation in a shared memory, parallel computing environment. Ultimately, these advancements are being put forward so that they can be considered for inclusion in the OpenMI standards.

2 HYDROCOUPLE INTERFACE DEFINITIONS

The HydroCouple interface definitions, associated Software Development Kit (SDK), and HydroCoupleComposer component coupling graphical user interface (GUI) and engine were written using the C++ programming language and the Qt framework. The benefit of using C++ is that it can be natively compiled and, therefore, minimizes the data marshalling costs between components and their underlying legacy model codes. Additionally, C++ can be compiled on a majority of operating systems and has bindings to several programming languages including Python, C, Java, and Fortran. The Qt framework provides reflection capabilities that are not available in the standard C++. This allows for object introspection and method invocation at runtime, which facilitates the in-memory model coupling and data exchange process. In the following sections we describe the major interface definitions and changes that have been implemented in the HydroCouple interface specification and the rationale behind them.

2.1 IModelComponent and IAdaptedOutFactoryComponent

In the OpenMI interface specification, the *IBaseLinkableComponent* is the primary interface that defines a model component. It forms the wrapper around a model's computational engine and defines interfaces for functions to initialize a model with arguments, validate a model, prepare a model for a simulation, update a model to its next state, check on the state of a model simulation, and dispose of a model's resources after a simulation. It also specifies the inputs (defined through the *IBaseInput* interface) and the outputs (defined through the *IBaseOutput* interface) that a model can consume and provide to other components respectively. In contrast to OpenMI, HydroCouple specifies two types of components called the *IModelComponent* and the *IAdaptedOutputFactoryComponent*.

The *IModelComponent* interface shown in Figure 1 is equivalent to the *IBaseLinkableComponent* in the OpenMI specification. In addition to all the functions defined by OpenMI for the *IBaseLinkableComponent* interface, the *IModelComponent* interface also specifies a new *clone* function that must be implemented for a component to make a deep clone of itself. This cloning process involves making a copy of the parent *IModelComponent* class and initializing it with the same arguments as the parent while making sure that outputs from the parent and child do not conflict. A parent model component keeps track of all of it child clones, which can be accessed using the *children* function. The *clone* function has been added so that independent copies of a model instance can be made for parallelized simulations. Details of the cloning approach are left up to the model component developer. Linkages with other model components are left up to the caller of the *clone* function.

Figure 1. The HydroCouple *IModelComponent* interface definition that replaces OpenMI's *IBaseLinkableComponent.*

In the OpenMI specification each component can provide a list of *IAdaptedOutputFactory* interface instances which can be used to generate adaptors to mediate data exchange between an output (i.e., *IBaseOutput*) of one component and an input (i.e., *IBaseInput*) of another component. The role of these adaptors is to perform data transformation operations such as spatial and temporal interpolations, data

aggregations and disaggregation, coordinate transformations, etc. that may be needed to supply the correct output from one component to another. To promote the reuse of *IAdaptedOutputFactory* instances, a new component type interface definition that is not bound to any component called an *IAdaptedOutputFactoryComponent* has been introduced in HydroCouple as shown in Figure 2. With this new interface definition, an *IAdaptedOutputFactoryComponent* can be developed and compiled into an independent library that can be loaded into a composition of coupled model components and readily used by any compatible model component to perform the necessary data transformations needed.

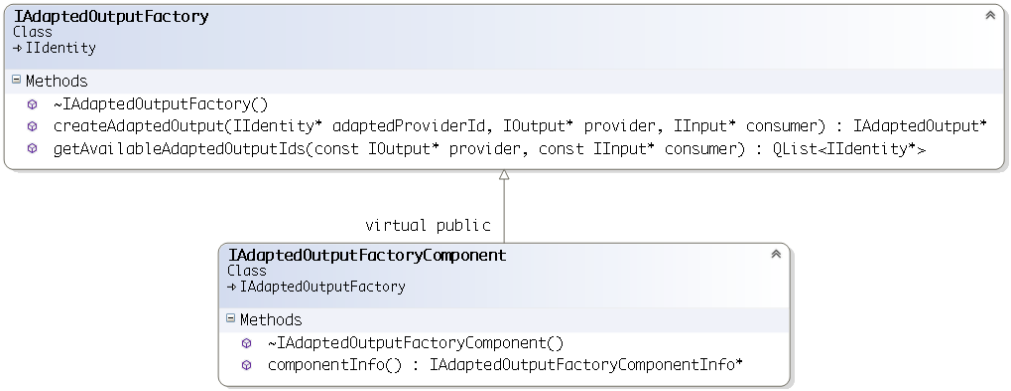

Figure 2. HydroCouple's *IAdaptedOutputFactoryComponent* interface definition inheritance chain.

2.2 IComponentInfo Interface

In the OpenMI specification, the core interface that is instantiated when a component library is loaded is the *IBaseLinkableComponent* interface. In HydroCouple, this has been superseded by the *IComponentInfo* interface, which provides detailed metadata about a component, including the name of the developer of a component, contact information, website for the component, as well as references to publications or documents that describe the inner workings of a model. These details provide model coupling composers with more guidance on the proper context for using a particular component. The *IComponentInfo* interface must be implemented as either an *IModelComponentInfo* interface or an *IAdaptedOutputFactoryComponentInfo* as shown in Figure 3. These two interfaces are responsible for creating new instances of *IModelComponent* and *IAdaptedOutputFactoryComponent* components.

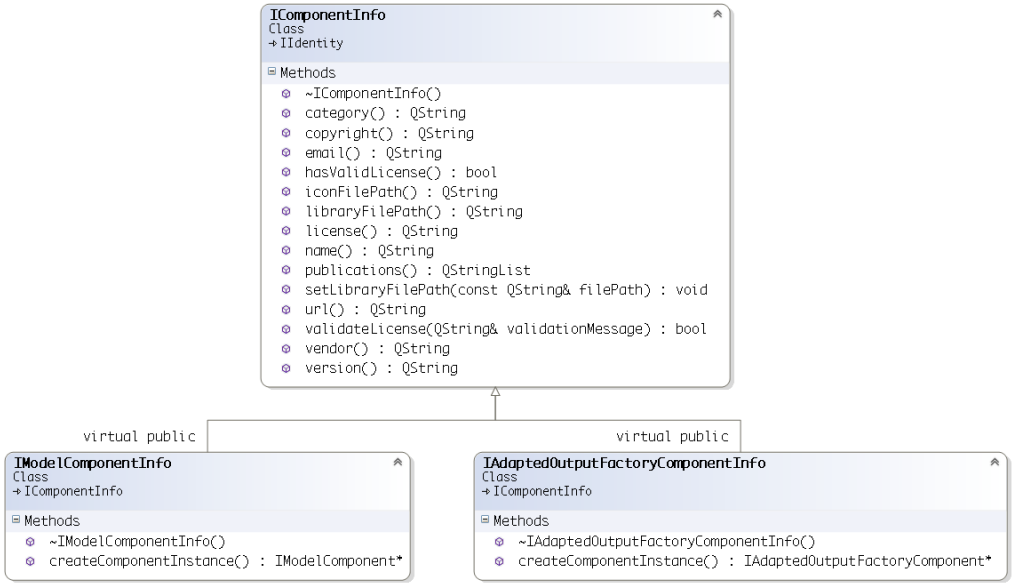

Figure 3. HydroCouple's *IComponentInfo* interface definition inheritance chain.

2.3 The IComponentDataItem Interface

The OpenMI interface specification provides two approaches for providing data to model components. The first is the provision of initialization input data before the start of a simulation through the *IArgument* interface. The second is the provision of the data that can be exchanged at runtime between model components specified through the *IBaseExchangeItem* interface. While OpenMI provides detailed specifications for the types of data provided through the *IBaseExchangeItem* interface through a multi-dimensional array interface called the *IBaseValueSet* and spatio-temporal dataset specializations, no such provisions are made for the *IArgument* interface. The *IArgument* interface's value is an unspecified object type that can be used to provide any kind of data. Providing detailed specifications for the *IArgument* will enable the creation of common tools to create, edit, and visualize standard initialization arguments for different models (e.g., a mesh generator, watershed delineation tools, time-series downloaders, etc.). To provide these specializations in HydroCouple, a new interface has been introduced called the *IComponentDataItem* interface that has various spatio-temporal specializations. This new interface is inherited by both the *IBaseExchangeItem* and *IArgument* interfaces as shown in Figure 4.

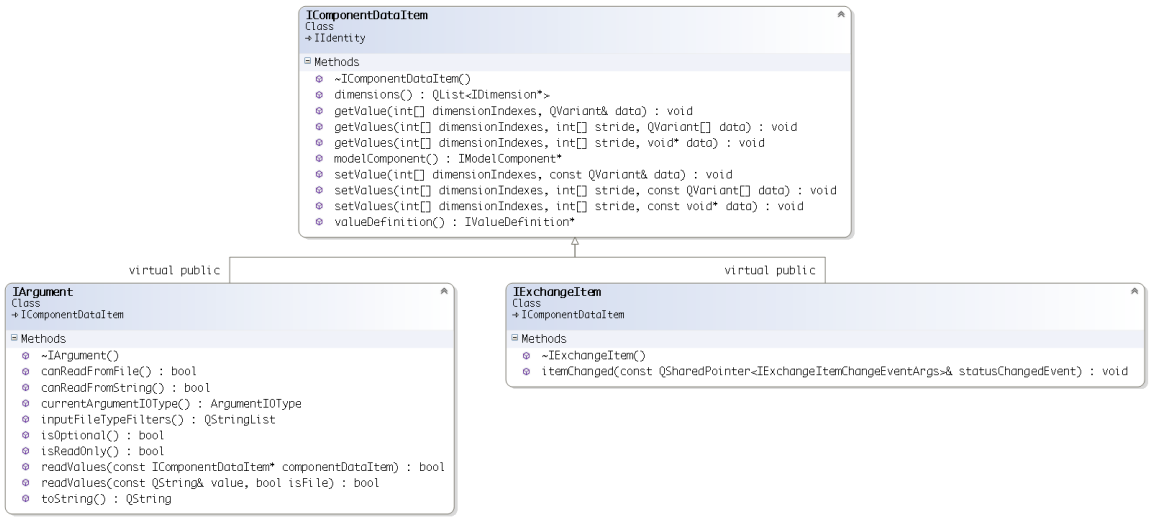

Figure 4. HydroCouple's *IComponentDataItem* interface definition inheritance chain.

In OpenMI, geospatial datasets associated with the *IBaseExchangeItem* interface were specified using an all-purpose interface called the *IElementSet* interface that can be used to represent points, lines, polylines, polygons, and polyhedra. In HydroCouple, the *IElementSet* interface was eliminated in favour of OGC's Simple Feature Access (SFA; Herring, 2011) specification (Figure 4) so that it is aligned with the standard data formats used by several GIS software and models. The additional benefit of implementing the SFA specification is that it defines standard geospatial topological querying and analysis functions that are useful for modeling purposes. The topological information that is missing in the SFA specification as well as OpenMI's *IElementSet* interface for polyhedral and triangular irregular network (TIN) surface interfaces was implemented interface using the quad-edge data structure proposed by Guibas and Stolfi (1985).

In addition to the SFA implementation, HydroCouple also provides *IComponentDataItem* interface specializations for rasters, networks, and cartesian, rectilinear, and curvilinear regular grid types that can be used in a wide variety of hydrologic and hydrodynamic modeling applications. Details about the various specializations of the *IComponentDataItem* and their associated UML diagrams can be found on the HydroCouple website (http://www.hydrocouple.org/hydrocoupledocs).

3 HYDROCOUPLECOMPOSER GUI

To facilitate the Component composition process, we have developed the HydroCoupleComposer GUI (Figure 6), which doubles as an interactive console application that be used to couple models together. The HydroCoupleComposer application is cross-platform (tested on Windows, Mac, and Linux

platforms) and provides a graphical interface for specifying model component initialization arguments, creating linkages between model components, specifying the variables to be exchanged between model components, and managing all the component libraries that are involved in a simulation.

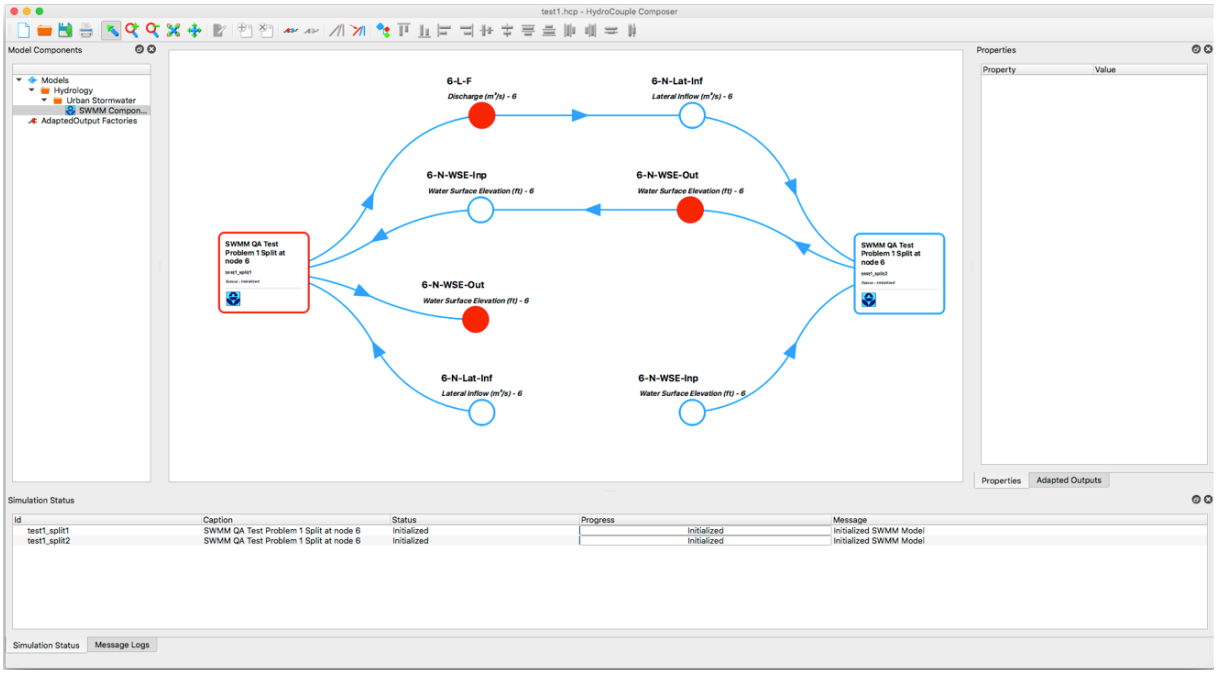

Figure 5. HydroCoupleComposer GUI.

4 OPTIMIZATION SIMULATION USING HYDROCOUPLE

To illustrate how the new cloning interface provided by HydroCouple can be used in practice, the following is a hypothetical optimization simulation that involves the coupling of an optimization model component (i.e., *Optimization Component*) to the United States Environmental Protection Agency's Stormwater Water Management Model (i.e., *SWMM Component*) that has been wrapped as a HydroCouple component. This hypothetical modeling exercise could represent an effort to properly size conveyance conduits and other hydraulic infrastructure in a stormwater system for a design storm. This optimization simulation can be accomplished in HydroCouple using the steps illustrated in the sequence diagram shown in Figure 7. In this example, the *Optimization Component* serves as the trigger component that interacts with the driver executable.

At the beginning of the simulation, the main function in the executable calls the *update* function of the trigger *Optimization Component* to update itself to its next state. The *Optimization Component* then repeatedly calls the clone function on the *SWMM Component* to make as many clones of itself as needed. Two clones are created in the example illustrated in Figure 7. After the cloning is completed, the *Optimization Component* enters the parallelized region (labelled "Par" in Figure 6), where it asks for the values it needs from each of the cloned SWMM components so that it can calculate respective objective function values that are to be minimized or maximized for each *SWMM Component*. For this particular example, the objective function might be the degree of overtopping of the various conduits in the SWMM model. The parallelization can be done using a simple "*# pragma omp parallel for*" pre-processor directive for the loop that iterates over all the cloned *Model Components* when using the OpenMP shared-memory parallel programming library. Before each of the *SWMM Components* computes the values needed to evaluate their objective functions, they request the variables for which the objective function is to be estimated from the *Optimization Component*. For our example, this might be the size of pipes, detention ponds, pumps etc., that are generated by the *Optimization Component* through some sort of intelligent search algorithm e.g., a multi-objective evolutionary algorithm. After the variables are returned, each *Model Component* updates their states based on the variables they have received. The *Optimization Component* then uses the returned values from each of the *Model Components* to compute respective objective function values. If the optimization criteria of minimizing or eliminating the overtopping in the conveyance system is met at some specified threshold, the

Optimization Component finalizes its simulation and terminates. Otherwise, the update function is called on the *Optimization Component* to start the whole process again.

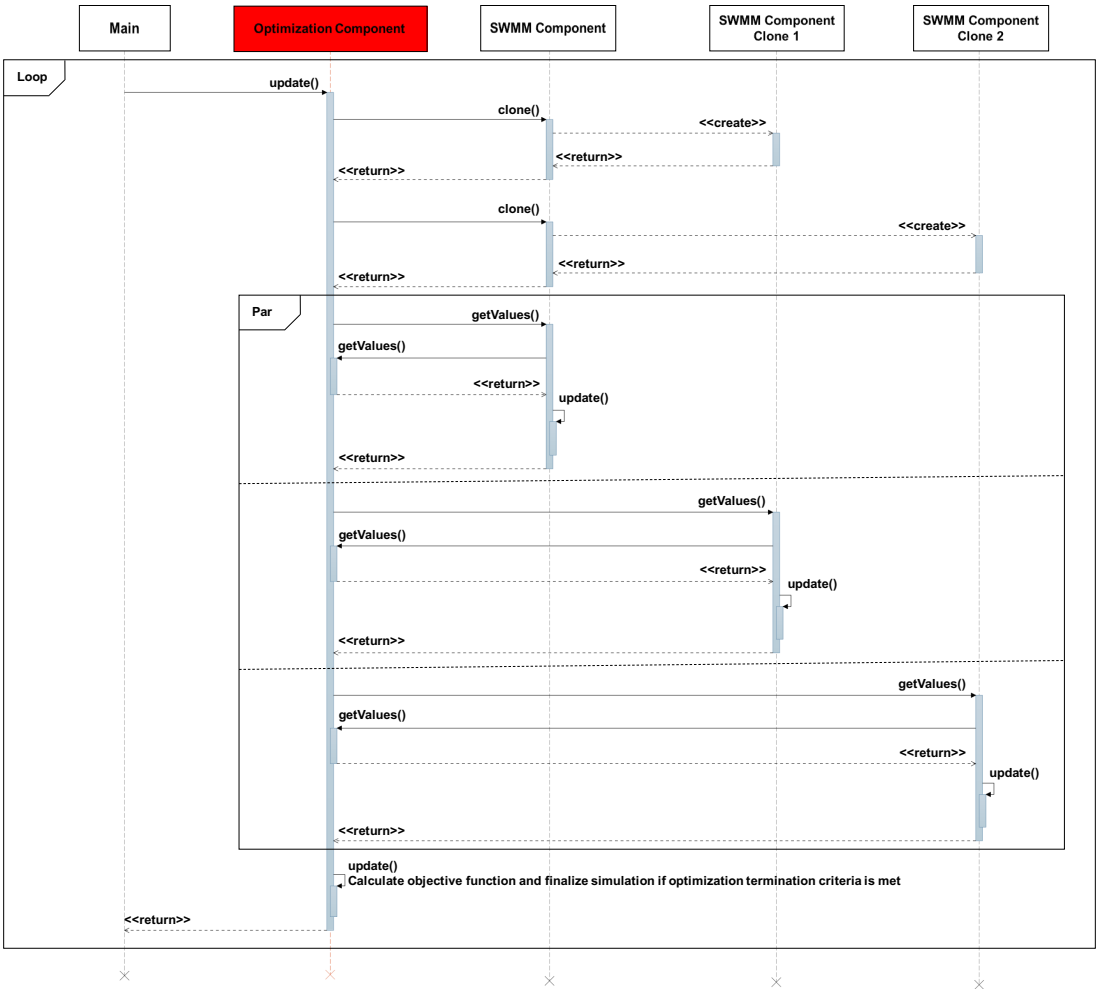

Figure 7. Sequence diagram for optimization simulation using HydroCouple.

5 CONCLUSIONS

In this paper we have described the HydroCouple interface definitions, which expand on the capabilities of the OpenMI specification to provide support for several standard geospatial dataset formats and their associated topological relationships. The benefits of these new interfaces are to increase compatibility with existing GIS formats used in several models and reduce the computation time needed to convert data from these standard formats into formats needed for modeling and estimating topological relationships. Further, HydroCouple provides new interface definitions to facilitate parallelized simulations for the class of experimental model simulations that include model calibration, sensitivity and uncertainty analysis, Monte Carlo analysis etc., that have heretofore been possible in OpenMI using the sequential approach, which may lead to excessively long simulation times. We have demonstrated how the new interface definitions may be implemented using a hypothetical optimization example that involves the coupling of an optimization model component to a SWMM model to minimize overtopping in a stormwater system.

HydroCouple was developed using C++ and the Qt framework, which provides bindings to several programming languages and can be compiled on many of the major operating systems. To facilitate the use of the HydroCouple interface, a HydroCouple Software Development Kit (SDK) that provides implementations of the core interface classes has been developed. So far, HydroCouple and its associated tools have been compiled for Windows, Mac OSX, and the Ubuntu Linux operating systems. The HydroCouple Composer GUI has been developed to facilitate the graphical composition of model

components together. The HydroCouple Composer software also doubles as a console application that can be used with command line arguments to couple model components and run compositions.

While the interfaces we have advanced in HydroCouple were developed primarily to facilitate our particular modeling applications in the hydrologic and hydrodynamic modeling field, they are applicable to other environmental and earth systems modeling fields and we have put them forward so that they will be considered for adoption in future versions of the OpenMI standard.

SOFTWARE AVAILABILITY

The HydroCouple interface definitions, the HydroCouple SDK, HydroCouple Composer, and example HydroCouple components can be downloaded from the GitHub organization page at https://github.com/hydrocouple. The HydroCouple interface definitions can be reviewed at http://www.hydrocouple.org/hydrocoupledocs/index.html.

ACKNOWLEDGMENTS

This research was supported by National Science Foundation EPSCoR Grant IIA 1208732 awarded to Utah State University as part of the State of Utah EPSCoR Research Infrastructure Improvement Award. Any opinions, findings, and conclusions or recommendations expressed are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

Buahin, C.A. and J.S. Horsburgh, 2015. Evaluating the Simulation Times and Mass Balance Errors of Component-Based Models: An Application of OpenMI 2.0 to an Urban Stormwater System. Environmental Modelling & Software 72:92–109. DOI:10.1016/j.envsoft.2015.07.003

David, O., S. Markstrom, K. Rojas, L. Ahuja, and I. Schneider, 2002. The Object Modeling System. L. Ahuja, L. Ma, and T. Howell (Editors). Agricultural System Models in Field Research and Technology Transfer. CRC Press. DOI:10.1201/9781420032413.ch15

Gregersen, J.B., P.J.A. Gijsbers, S.J.P. Westen, and M. Blind, 2005. OpenMI: The Essential Concepts and Their Implications for Legacy Software. Advances in Geosciences 4:37–44. DOI:10.5194/adgeo-4-37-2005

Gregersen, J.B., P. J. A. Gijsbers, and S. J. P. Westen, 2007. OpenMI: Open Modelling Interface. Journal of Hydroinformatics 9:175. DOI:10.2166/hydro.2007.023

Guibas, L. and J. Stolfi, 1985. Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi. ACM Trans. Graph. 4:74–123. DOI:10.1145/282918.282923

Herring, J.R., 2011. OpenGIS® Implementation Standard for Geographic Information - Simple Feature Access - Part 1: Common Architecture. http://portal.opengeospatial.org/files/?artifact_id=25355 (last accessed 03/29/2016)

Hill, C., C. DeLuca, M. Suarez, and A. Da Silva, 2004. The Architecture of the Earth System Modeling Framework. Computing in Science & Engineering 6:18–28. http://dx.doi.org/10.1109/MCISE.2004.1255817

Laniak, G.F., G. Olchin, J. Goodall, A. Voinov, M. Hill, P. Glynn, G. Whelan, G. Geller, N. Quinn, M. Blind, S. Peckham, S. Reaney, N. Gaber, R. Kennedy, and A. Hughes, 2013. Integrated Environmental Modeling: A Vision and Roadmap for the Future. Environmental Modelling & Software 39:3–23. DOI:10.1016/j.envsoft.2012.09.006

Moore, R. (Editor)., 2010. The OpenMI Document Series: OpenMI Standard 2 Specification. https://publicwiki.deltares.nl/download/attachments/41549981/OpenMI+Standard+2+Specification.pdf

Moore, R.V. and C.I. Tindall, 2005. An Overview of the Open Modelling Interface and Environment (the OpenMI). Environmental Science & Policy 8:279–286. DOI:10.1016/j.envsci.2005.03.009

Parson, E.A., 1995. Integrated Assessment and Environmental Policy Making: In Pursuit of Usefulness. Energy Policy 23:463–475. DOI:10.1016/0301-4215(95)90170-C

Peckham, S.D., E.W.H. Hutton, and B. Norris, 2013. A Component-Based Approach to Integrated Modeling in the Geosciences: The Design of CSDMS. Computers & Geosciences 53:3–12. DOI:10.1016/j.cageo.2012.04.002