

# Simulating Watersheds Using Loosely Integrated Model Components: Evaluation of Computational Scaling Using OpenMI

Anthony M. Castronova<sup>a</sup>, Jonathan L. Goodall<sup>a,\*</sup>

<sup>a</sup>*Department of Civil and Environmental Engineering, University of South Carolina, 300 Main Street, Columbia, SC 29208*

---

## Abstract

Complicated research and management questions regarding watershed systems often require the use of more than one simulation model. Therefore, it is necessary to develop a means to integrate multiple simulation models to predict holistic system response. In this paper we explore the use of a component-based approach for the runtime integration of models, implemented as “plug-and-play” software components. The motivation for this work is to quantify performance overhead costs introduced by adopting a component-based paradigm for loosely integrating hydrologic simulation models. We construct a standard rainfall/runoff watershed model using the Open Modeling Interface (OpenMI) Software Development Kit (SDK) where infiltration, surface runoff, and channel routing processes are each implemented as independent model components. We then analyze the performance of this loosely integrated model to quantify computational scaling, using the Hydrologic Engineering Center’s Hydrologic Modeling System (HMS) for comparison. Our results suggest that the overhead introduced by runtime communication of data is not significant when applied for semi-distributed watershed modeling. Our analysis was limited to semi-distributed watershed modeling, however, and future research is needed to understand performance and accuracy for more data demanding hydrologic

---

\*Corresponding author

*Email addresses:* [Castrona@email.sc.edu](mailto:Castrona@email.sc.edu) (Anthony M. Castronova),  
[Goodall@cec.sc.edu](mailto:Goodall@cec.sc.edu) (Jonathan L. Goodall )

models.

*Keywords:* Hydrologic Modeling, Software Architecture, Component-based Modeling, Integrated Modeling, OpenMI

---

## 1. Introduction

Hydrologic models typically focus on a small subset of processes within the overall hydrologic cycle (Singh and Woolhiser, 2002). This approach is adequate if one assumes that the boundary conditions of the problem do not change, and are not changed by, processes happening outside of the model's scope. There is growing acknowledgment, however, that in many cases this is not a correct assumption. Groundwater and surface water provide one example. They are often modeled as separate entities, despite the fact that in many situations modeling water in one domain necessitates modeling water in the other domain as well. This example implies that the coupling between hydrologic subsystems is important. Therefore, we need to consider how to build models that are able to simulate these dynamic system interactions.

One solution to this problem is to simply extend existing models so that they include a larger scope of process representations. This approach has been taken by a number of researchers that have integrated groundwater, surface water, climate, weather, economic, and other models by altering source code so that they are able to exchange boundary conditions during a model simulation (e.g. Ahrends et al., 2008; Anderson et al., 2002; Kunstmann et al., 2008; Yu et al., 2006; Barthel et al., 2008; Mölders and Rühaak, 2002; Maxwell et al., 2007). The general approach taken in these examples can be classified as tight integration (Sui and Maggio, 1999), where components of a system are compiled into a single application. The primary advantage of this approach is that the modeler has full control over all system components, therefore computational performance can be optimized across the entire system.

While there are clear advantages to using a tight integration approach for coupling models, there are also limitations. First, in the case that a model

1 is extended by including new code, adopting a tight integration approach in-  
2 creases code redundancy through a process called “model creep.” Model creep  
3 is where a model originally intended for simulating one set of processes has been  
4 extended to include other processes that were previously considered predeter-  
5 mined boundary conditions. This is problematic because the development team  
6 are likely experts in the field that the model was originally designed to simulate,  
7 not in these related topics. The second case is that two existing models are cou-  
8 pled using a tight integration approach. This requires changes to the model’s  
9 source code, which make it out of sync with core code development efforts. As  
10 a result, it is difficult to maintain a state-of-the-art modeling system with such  
11 an approach.

12 An alternative method for coupling models is to follow a loose integration  
13 paradigm that focuses on the decomposition of a system into a set of interlinked  
14 computational components (Argent et al., 2006). The simplest form of loose  
15 integration is communication through a common file format or by data conver-  
16 sions between model-specific file formats. This simple form of loose integration  
17 can be computationally inefficient if models require runtime communication of  
18 data. To address this limitation, loose integration can also be achieved by cre-  
19 ating models as software libraries instead of stand-alone executables which can  
20 then be integrated into a modeling framework (Löwy, 2005). Furthermore, if  
21 the model libraries follow a standard interface specification, then it is possible  
22 to create a generic modeling framework that facilitates model-to-model data  
23 communication during simulation.

24 Although tightly integrated approaches for modeling hydrologic systems are  
25 likely more common, loose integration is becoming more widely used. Sev-  
26 eral frameworks have been recently developed that offer the option of join-  
27 ing models together in a loosely integrated, plug-and-play manner (e.g., the  
28 Community Surface Dynamics Modeling System (CSDMS), the Earth System  
29 Modeling Framework (ESMF), the Open Modeling Interface (OpenMI), etc.).  
30 These loosely integrated architectures give modelers the freedom to interchange  
31 process representations, and allows them to construct the most representative

1 model possible for a specific system and a specific objective (Morton et al.,  
2 2003). Furthermore, these systems enable modelers to combine the contribu-  
3 tions of legacy models with new process-level models, providing a full suite of  
4 tools to use during model development.

5 Despite the growing popularity of loosely integrated frameworks for modeling  
6 environmental systems, there are few studies that quantify the computational  
7 overhead introduced by the approach. Loose integration has the potential to  
8 introduce performance overheads that may significantly impact model runtime.  
9 For example, consider the Open Modeling Interface (OpenMI) version 1.4 and its  
10 mechanism for data transfer between components. In an OpenMI simulation,  
11 data is transferred between components during runtime using a request and  
12 reply paradigm (Gregersen et al., 2007). Since this can result in thousands or  
13 millions of transactions during a model simulation, we hypothesize that it may  
14 have a considerable effect on execution time. For this reason, it is expected that  
15 model communication will be a source of simulation inefficiency, which will be  
16 amplified as the number of time steps or the size of data transfers increase. We  
17 do not know if the overhead of this communication protocol makes the approach  
18 unattractive for modeling hydrologic systems. This analysis is meant to provide  
19 further insight into this issue.

20 Given this motivation, we investigated the performance of the OpenMI (ver-  
21 sion 1.4) for simulating rainfall/runoff using typical hydrologic engineering cal-  
22 culations. We used the Hydrologic Engineering Center’s Hydrologic Modeling  
23 System (HMS), an industry standard model widely used in engineering practice,  
24 to verify our calculations and to provide a point of comparison for the compu-  
25 tational scaling results. We created a rainfall/runoff model using the OpenMI  
26 and HMS for the same watershed, to simulate the same storm event, using the  
27 same hydrologic process representations. Two metrics were used to evaluate  
28 computational performance: an endurance test in which the number of time  
29 steps was increased and a load test where the number of modeling elements  
30 was increased. The goal of these experiments was to quantify how the OpenMI  
31 scales as the number and the size (i.e., memory) of communications between

1 model components increase.

2 In the following section we provide further background on the distinction  
3 between tight and loose model integration techniques. Additionally, a brief  
4 introduction to the HMS and OpenMI systems is provided to familiarize the  
5 reader with key concepts. We then describe the methodology of this study,  
6 including the implementation of each model and the tests that were conducted  
7 to measure computational scaling. Finally, we present and discuss the results,  
8 and conclude with our findings and suggestions for future research.

## 9 **2. Background**

10 There are distinct differences between the loose integration and tight in-  
11 tegration approaches. In the tight integration approach (Figure 1), the Model  
12 Coupler class directly calls routines defined internally by Models 1 and 2. These  
13 routines must be compiled into the modeling system prior to a model simula-  
14 tion and cannot be discovered at runtime. In contrast, the loose integration  
15 approach (Figure 2) features a Model Engine class that calls Routines 1 and 2  
16 by instantiating an object which implements a standard interface. This stan-  
17 dard interface, denoted as ICoupler, defines simulation methods that must be  
18 implemented in each Routine. Using this approach the Model Engine class re-  
19 ceives data through the standard interface, having no knowledge of how it is  
20 created (Holzworth et al., 2010). This is significant because additional Routines  
21 can be added to the system and the Model Engine automatically knows how  
22 to communicate with them. Hence the system can be easily reconfigured and  
23 extended using a “plug-and-play” approach.

24 This “plug-and-play” characteristic of loosely integrated systems encourages  
25 modelers to add, remove, or replace components within a model configuration  
26 (Govindaraju et al., 2006). Additionally, computational redundancy is avoided  
27 by designing components to address specific parts of the overall system. This  
28 characteristic is useful in forming unique representations of complex systems,  
29 including multi-disciplinary applications, because components can be built by

1 independent groups (Kennen et al., 2008). Loosely integrated, process-level  
2 modeling offers greater flexibility in model design as well as transparency in  
3 model execution. Using this concept, a hydrologic system can be decomposed  
4 into independent process-level components (infiltration, surface runoff, channel  
5 routine, etc.) that communicate with each other through standard interfaces  
6 (Argent et al., 2006).

7 In our analysis we have elected to use the Hydrologic Modeling System  
8 (HMS), developed by the US Army Corps of Engineers (USACE), as a point of  
9 comparison. HMS is a deterministic watershed model that focuses primarily on  
10 water quantity simulation (Scharffenberg and Fleming, 2009). It was selected  
11 for this study because it is widely used in hydrologic engineering analysis for  
12 simulating rainfall/runoff (e.g. Al-Abed et al., 2005; Knebl et al., 2005). Built  
13 from its predecessor HEC-1, HMS simulates hydrologic systems as an inter-  
14 connected network of hydrologic and hydraulic units (i.e. subbasins, reaches,  
15 reservoirs, junctions, etc.) (Chu and Steinman, 2009). Each of which has a  
16 unique combination of process representations and parameter definitions, that  
17 allow the user to customize the model for a specific watershed study.

18 An HMS model is divided into three categories: watershed physical descrip-  
19 tion, meteorological modeling, and hydrologic simulation (Scharffenberg and  
20 Fleming, 2009). The physical description of a watershed includes a suite of  
21 computational algorithms to simulate infiltration losses, transformation of ex-  
22 cess precipitation, representation of baseflow, and computation of flow in open  
23 channels (Feldman, 2000). The meteorological model contains algorithms for  
24 estimating historic and synthetic precipitation, evapotranspiration, and snow  
25 melt. Lastly, hydrologic simulation is managed by user-defined control speci-  
26 fications, which include the start time, end time, and computational interval  
27 (Scharffenberg and Fleming, 2009).

28 We elected to use the OpenMI version 1.4 as the loosely integrated framework  
29 because it is tailored to the water resources domain. It also allows for the repre-  
30 sentation of discrete geometric features such as the subbasins used in HMS. The  
31 OpenMI was developed in 2005 and was funded through the European Union

1 Water Framework Directive. Its design focuses on establishing an interface stan-  
2 dard to enable model interoperability (Moore et al., 2005). As a result, it is not  
3 meant to be a complete modeling framework, but rather a standard message  
4 passing protocol that is implemented through standard interfaces (Gregersen  
5 et al., 2007). The OpenMI Association Technical Committee (OATC) has also  
6 developed a Software Development Toolkit (SDK) that extends the OpenMI  
7 standard into a modeling framework. We use this SDK along with extensions  
8 designed to more easily create process-level OpenMI components (Castronova  
9 and Goodall, 2010) to implement the loosely integrated watershed simulation.

10 The OpenMI consists of interface standards that are implemented by soft-  
11 ware classes to form the foundation of an integrated model. Four key concepts of  
12 this standard are linkable components, exchange items, links, and model compo-  
13 sitions (Brinkman et al., 2005). Linkable components are objects that perform  
14 calculations and are able to accept input exchange items and produce output  
15 exchange items. Exchange items define the messages that pass between linkable  
16 components during a simulation, and consist of objects that describe the quan-  
17 tity and element set for a data exchange (Gregersen et al., 2007). Lastly, links  
18 connect components together by specifying the pathway through which data  
19 flows from one component the next. Once links between model components  
20 have been established, the system is referred to as a model composition and is  
21 executed with either the command line or GUI software application provided as  
22 part of the OpenMI SDK.

### 23 **3. Methodology**

24 To quantify the computational performance of the OpenMI, we modeled the  
25 same watershed system using the same mathematical relationships, but using a  
26 loosely integrated approach in one model (the OpenMI version) and using an  
27 industry standard model that follows a tight integration approach in the other  
28 (the HMS version). Our goal was to first verify that both models produced the  
29 same results, and second to quantify how each model scaled as the modeling

1 demands increased. To do this, we conducted two tests. In the first, called  
2 the load test, each model executes multiple simulations in which the number of  
3 computational elements is gradually increased to an upper limit. The second,  
4 named the endurance test, evaluates how each model responds to an increasing  
5 simulation duration. During these tests, simulation runtimes were recorded as  
6 model and watershed properties changed. The objective of this experiment was  
7 to understand the computational scaling of a loosely integrated architecture  
8 (i.e. the OpenMI) for semi-distributed rainfall/runoff applications. The HMS  
9 model implemented to provide a basis for comparison because it is a widely  
10 used, industry-standard hydrologic model. In this section we first introduce the  
11 study area, the Smith Branch Watershed near Columbia, South Carolina, and  
12 then describe the model from a mathematical and implementation perspective.  
13 Finally we describe the computational scaling tests.

### 14 *3.1. Study Area*

15 The study area for this work was the Smith Branch watershed located in  
16 Columbia, SC (Figure 3). The watershed has an area of 5.6 mi<sup>2</sup> and consists  
17 mostly of developed land (11% high intensity, 26% medium intensity, 37% low  
18 intensity, and 16% open space). The primary soil type in the watershed is sandy  
19 loam (70%), while 25% of the watershed is loam and 5% is sand. The watershed  
20 is gaged by a US Geological Survey monitoring station 02162093 “Smith Branch  
21 at North Main Street at Columbia, SC”. This gauging station reports both water  
22 surface height and discharge every 15 minutes. We did not calibrate the models  
23 using this observed flow because the goal was to understand computational  
24 performance rather than simulate a particular event.

### 25 *3.2. Model Description*

26 A conceptual view of a rainfall/runoff system is presented in Figure 4. The  
27 models implemented in both HMS and OpenMI contain mathematical approx-  
28 imations for precipitation, infiltration, surface runoff, and streamflow. Precipi-  
29 tation was supplied as model input and obtained from a Next Generation Radar



1 (NEXRAD) precipitation product provided through the National Climatic Data  
2 Center (NCDC). Excess precipitation ( $P_e$ ) is modeled using the Natural Re-  
3 sources Conservation Service (NRCS, formally SCS) Method for Abstraction  
4 (Soil Conservation Service, 1972; Chow et al., 1988). According to this method

$$P_e = P - I_a - F_a \quad (1)$$

5 where  $P$  is precipitation,  $I_a$  is initial abstraction, and  $F_a$  is continuing abstrac-  
6 tion.  $F_a$  is estimated by a conceptual model

$$F_a = \frac{S(P - I_a)}{P - I_a + S} \quad (2)$$

7 where  $S$  is soil water storage. Furthermore,  $S$  is estimated based on an empirical  
8 equation

$$S = \frac{1000}{CN} - 10 \quad (3)$$

9 where  $CN$  is the curve number parameter which is a function of land cover and  
10 soil conditions (Soil Conservation Service, 1975). Lastly,  $I_a$  is estimated using  
11 an empirically derived relationship that states  $I_a = 0.2S$ .

12 These equations are solved to estimate the amount of excess precipitation  
13 ( $P_e$ ) that will occur given a rainfall event. This method for estimating rainfall  
14 abstraction is commonly used in hydrologic engineering practice, which was the  
15 motivation for using it in our study. In addition, because the overall goal of the  
16 study is to test loose integration of hydrologic models model, predictive accuracy  
17 of the model itself was not our primary focus. Curve numbers where derived  
18 using geospatial analysis; spatial data layers of land cover and soil data were  
19 gathered from the Environmental Protection Agency (EPA) and NRCS, respec-  
20 tively, and curve number values were taken from the NRCS Technical Reference  
21 Manual 55: Urban Hydrology for Small Watersheds (Soil Conservation Service,  
22 1975). Weight averaged curve numbers were calculated for each subbasin within  
23 the watershed using this data and approach.

24 The watershed response to a rainfall event was estimated using the NRCS  
25 Dimensionless Unit Hydrograph (Chow et al., 1988). This is a deterministic and

1 lumped approach for estimating a streamflow hydrograph at the outlet of a sub-  
 2 basin given an excess precipitation hyetograph. To apply the unit hydrograph  
 3 procedure, several parameters must first be derived: the time at which peak flow  
 4 occurs ( $T_p$ ), the peak flow rate ( $q_p$ ), and the subbasin lag time ( $t_p$ ). Alterna-  
 5 tively, lag time can be represented as  $0.6t_c$ , where  $t_c$  is the time of concentration  
 6 for the subbasin. Peak flow rate was estimated using

$$q_p = \frac{483.4A}{T_p} \quad (4)$$

7 where  $A$  is the watershed area in  $\text{mi}^2$ ,  $T_p$  is in hours, and  $q_p$  has units of  $\text{ft}^3 \text{ s}^{-1}$   
 8 in.  $T_p$  is defined as

$$T_p = \frac{t_r}{2} + t_p \quad (5)$$

9 where  $t_r$  is the rainfall duration in hours. An instantaneous streamflow hy-  
 10 drograph was derived for each subbasin using the unit ordinates of the NRCS  
 11 dimensionless hydrograph and the calculated peak flow rate ( $q_p$ ). Once formed,  
 12 this instantaneous unit hydrograph was used to calculate a direct runoff hydro-  
 13 graph using Equation (6) where  $P_{e,m}$  is the excess precipitation at time  $m$  and  
 14  $U_{n-m+1}$  is the unit hydrograph ordinate of the current index,  $n$ , minus  $m + 1$   
 15 (Chow et al., 1988).

$$Q_n = \sum_{m=1}^{n \leq M} P_{e,m} U_{n-m+1} \quad (6)$$

16 Flow is routed through the channel network using the Muskingum method.  
 17 This method transforms the streamflow calculated at each subbasin outlet through  
 18 a channel network to the watershed outlet. The Muskingum algorithm is derived  
 19 from a variable discharge-storage relationship and considers the total water stor-  
 20 age in a channel as a combination of wedge and prism volumes (Chow et al.,  
 21 1988). The wedge shape accounts for the back water or flood wave effects, and  
 22 is controlled by a weighting factor,  $X$ , where  $0 \leq X \leq 0.5$ . The prism stor-  
 23 age represents the volume of water within a cross-section of the channel, and  
 24 is weighted by the proportionality coefficient,  $K$ , that is approximated as the

1 time it takes for a flood wave to travel through the reach (Mays, 2005). The  
2 total storage,  $S$ , is expressed as a combination of wedge and prism volumes

$$S_{j+1} - S_j = \frac{I_j + I_{j+1}}{2} \Delta t - \frac{Q_j + Q_{j+1}}{2} \Delta t \quad (7)$$

3 where  $I$  is inflow and  $Q$  is outflow at times  $j$  and  $j + 1$ , in units of  $\text{ft}^3 \text{ s}^{-1}$ .  
4 Given that storage can be expressed as a function of the  $K$  and  $X$  parameters,  
5 Equation (7) can be simplified to

$$Q_{(j+1)} = C_1 I_{(j+1)} + C_2 I_j + C_3 Q_j \quad (8)$$

6 where the  $C_1$ ,  $C_2$ , and  $C_3$  coefficients are functions of  $K$ ,  $X$ , and  $\Delta t$  (Chow  
7 et al., 1988).

### 8 *3.3. Model Implementation*

#### 9 *3.3.1. HMS Model*

10 The process of creating the HMS model for Smith Branch is illustrated in  
11 Figure 5. It began with constructing hydrologic elements using the HMS Basin  
12 Model Manager. These elements represent specific parts of the overall hydro-  
13 logic system. Linkages are used to connect elements in series and to define the  
14 order in which the system of equations is solved. To recreate the Smith Branch  
15 watershed, a set of subbasin, reach, and junction elements were connected in se-  
16 ries. Once the watershed was assembled, process representations were chosen for  
17 each hydrologic element. As discussed in Section 3, infiltration was calculated  
18 using the NRCS Curve Number method, surface runoff was calculated using the  
19 NRCS Unit Hydrograph procedure, and channel routing was performed using  
20 the Muskingum method. Lastly, all input parameters were specified for each  
21 mathematical procedure. These input parameters were derived from geospa-  
22 tial data sets and were processed using Geographic Information System (GIS)  
23 software.

24 The HMS Time-Series Data manager was used to apply the measured rain-  
25 fall values to the watershed in a spatially distributed manner. This was achieved

1 by first creating a synthetic precipitation gage for each subbasin within the wa-  
2 tershed. Next, incremental rainfall hyetographs were manually entered from  
3 the recorded NEXRAD precipitation data. The HMS Meteorologic Model man-  
4 ager was used to pair these rainfall hyetographs with their respective subbasins.  
5 Once complete, each subbasin within the watershed was associated with a unique  
6 rainfall hyetograph derived from radar data.

7 After all preparatory data procedures were complete, the HMS Control Spec-  
8 ifications manager was configured to control model simulation. This tool enables  
9 users to specify simulation start and end date-times and also the simulation time  
10 interval. Lastly, a simulation run was created and executed. Simulation results  
11 were stored for each hydrologic element within the system, in the Hydrologic  
12 Engineering Center Data Storage System (HEC-DSS).

### 13 3.3.2. *OpenMI Model*

14 To construct the OpenMI model, components were first developed to per-  
15 form infiltration, surface runoff, and channel flow calculations. Each process was  
16 defined by a specific mathematical relationship, described in the previous sec-  
17 tion: NRCS Curve Number method, NRCS Unit Hydrograph transformation,  
18 and Muskingum channel routing, respectively (Figure 6). A key aspect of the  
19 OpenMI is that components communicate with each other during model simu-  
20 lation on a time step basis. This communication is designated by the “request”  
21 and “reply” loop shown in the right column of Figure 6. This workflow is differ-  
22 ent than most traditional models that tend to execute each process for the entire  
23 simulation duration, sequentially. In contrast, once a simulation is initiated in  
24 the OpenMI model, the “downstream” component in the chain requests data  
25 from its “upstream” neighbor. In this case Channel Flow requests data from  
26 Surface Runoff. The Surface Runoff component is unable to supply values to  
27 Channel Flow because it requires input from another component. Likewise,  
28 the Surface Runoff component will request data from Infiltration which subse-  
29 quently requests data from Precipitation. Since Precipitation does not require  
30 input from any other component, it is able to respond to Infiltration with the

1 requested data. The Infiltration component receives this data, processes it, and  
2 then sends the result to Surface Runoff. This is repeated until Channel Flow  
3 processes its result, at which time the entire sequence of events begins again for  
4 the next time step.

5 Components were developed using an abstract class based on the OpenMI  
6 Association Technical Committee’s (OATC) IEngine interface called the Simple  
7 Model Wrapper (SMW) (Castronova and Goodall, 2010). The OATC IEngine  
8 was created to aid in the process of “wrapping” legacy models into OpenMI-  
9 compliant components. In previous work we showed that the IEngine interface  
10 can be simplified for creating new, process-level model components using the  
11 concept of an abstract class. Therefore, the SMW was developed to enable  
12 model developers to quickly create new process-level components by simplifying  
13 the IEngine interface to an abstract class where the model developer overrides  
14 three methods: Initialize, PerformTimeStep, and Finish. Furthermore, model  
15 attributes are specified in an XML configuration file. The Initialize method is  
16 used to setup a model component prior to model simulation. It can be used  
17 to read input data, parameterize the model, establish initial conditions, or per-  
18 form similar setup operations. The PerformTimeStep method is used to exe-  
19 cute calculations that occur during simulation runtime. This method starts by  
20 requesting input values from “upstream” components, then performing a com-  
21 putation using these values, and finally supplying the results to “downstream”  
22 components. The Finish method is called after a simulation ends and can be  
23 used to write model output files and release allocated memory.

24 Rainfall data was supplied to the model configuration using a component de-  
25 signed to read time-series data stored in Water Markup Language (WaterML)  
26 format; an XML-based standard for storing hydrologic time-series values (Valen-  
27 tine and Zaslavsky, 2009). Prior to simulation, this component creates exchange  
28 items by reading a directory of WaterML files. It uses the information within  
29 them to establish input and output exchange items, element sets, and time  
30 horizons. Values are stored in a buffer (i.e. `Oatc.SmartBuffer`) so they can be  
31 retrieved for specific times, upon request. In the case that a component requests

1 values for a time that does not match those included in the input files, this com-  
2 ponent performs a linear time interpolation from the known values, to respond  
3 to the request.

4 The amount of excess precipitation was determined using the NRCS Curve  
5 Number method. During the initialization phase, curve number values for each  
6 subbasin are read from an input shapefile and stored in memory so they can be  
7 accessed during simulation. Next, input and output exchange items are defined  
8 by reading information contained within the input shapefile. Lastly, potential  
9 infiltration and initial abstraction are calculated for each subbasin using Equa-  
10 tions (1) through (3). During model simulation, the PerformTimeStep method  
11 requests values from the rainfall component. It then calculates how much ex-  
12 cess precipitation will occur, based on these values and subbasin properties.  
13 The component can then supply excess precipitation for each subbasin in the  
14 watershed to the NRCS Unit Hydrograph component. Once model simulation  
15 is complete, the Finish method is called to write results to an output file.

16 Excess precipitation was transformed into subbasin outflow using the NRCS  
17 Unit Hydrograph method. During the Initialize phase, a unique unit hydrograph  
18 is created for each subbasin in the watershed based on their unique physical  
19 properties. This is done by first extracting parameters from the input shapefile,  
20 then calculating the time in which peak flow occurs using Equation (5), and  
21 finally calculating the peak flow rate for this time using Equation (4). Once  
22 these values have been calculated, a unit hydrograph is established using the  
23 dimensionless unit ordinates provided by the National Engineering Handbook  
24 (Soil Conservation Service, 1972). These unit hydrographs are utilized in Per-  
25 formTimeStep method to calculate the outflow at the current time. This is  
26 done using Equation (6), where  $P_{e,m}$  is the excess rainfall that is provided by  
27 the NRCS Curve Number component. In the Finish method, subbasin outflows  
28 are written to disk and internal component objects are released from memory.

29 Lastly, the subbasin outflows were routed through the stream network us-  
30 ing the Muskingum Routing method. During the component's Initialize phase,  
31 channel parameters such as storage and proportionality coefficients, X and K,

1 are read from an input shapefile. A network is also created from the input  
2 shapefile by associating each reach with upstream and downstream neighbors.  
3 During the PerformTimeStep phase, this component retrieves outflow from each  
4 subbasin and routes them through the stream network using Equation (8). The  
5 Finish method is used to save the routed streamflow to a specified output file.

### 6 *3.4. Performance Tests*

7 Performance tests were conducted to evaluate the potential computational  
8 costs introduced by the OpenMI communication paradigm when simulating a  
9 semi-distributed rainfall/runoff system. In the first test, the number of modeling  
10 units per simulation was varied. By increasing the number of modeling units,  
11 we were able to evaluate how each model operates under different data loading  
12 conditions. For the OpenMI model, this increase in modeling units is not only  
13 more computationally demanding, it also results in larger sized (in terms of  
14 memory) data transfers between model components. We termed this test the  
15 “load test” because it increases the size of data that components must pass  
16 between one another during a simulation run (Meier et al., 2007). The second  
17 test was designed to evaluate how each model responds to longer simulation  
18 durations. To do this, we varied the number of time steps performed in a  
19 simulation from 100 to 10,000, while maintaining a constant time step interval.  
20 Since a constant time step was maintained, synthetic rainfall data was required  
21 as input for both models to extend simulation duration. We termed this test the  
22 “endurance test” because it evaluates the ability for models to handle a large  
23 number of data transfers in a single simulation (Meier et al., 2007).

24 Tools were used, such as Arc Hydro (Maidment, 2002) and ArcGIS<sup>®</sup>, to  
25 subdivide the watershed into 1, 50, 100, 150, and 200 subbasins. It was not  
26 practical to subdivide the Smith Branch watershed into more than 200 sub-  
27 basins, but we did extend the load test to a larger number of model elements  
28 into the thousands by considering a synthetic watershed. Several scripts were  
29 written to automate the necessary geoprocessing steps required to create model  
30 inputs. This included the creation of input shapefiles with model parameters

1 such as  $CN$  and  $T_c$  for both the HMS and OpenMI. The OpenMI components  
2 utilized these shapefiles as model input. In contrast, HMS model simulations  
3 necessitated supplementary scripts to extract the element set data from the var-  
4 ious shapefiles, and programmatically write model inputs. The end result were  
5 identical model compositions within each software environment. The model  
6 runtimes used in our analysis were taken from reported values within HMS and  
7 OpenMI simulation output files.

#### 8 **4. Results and Discussion**

9 The first objective of our study was to ensure that both models, the one built  
10 with HMS and the other built with OpenMI, produced the same output pre-  
11 dictions for the watershed system. Figure 7 shows that both models calculated  
12 identical values for cumulative excess precipitation, surface runoff, and stream-  
13 flow. This confirms that no errors were introduced as a result of decoupling  
14 the physical system into individual OpenMI-compliant computational compo-  
15 nents. The second objective was to quantify the overhead introduced during  
16 model simulation by the OpenMI communication standard. This objective is  
17 subdivided into two distinct tests (1) the load test and (2) the endurance test.

18 Results from the load test where the Smith Branch watershed was delineated  
19 into a varying number of subbasins revealed that, as the number of computa-  
20 tional elements increased, the OpenMI's performance remained comparable to  
21 that of the HMS (Figure 8). In fact, the OpenMI model completed its simula-  
22 tion runs faster than the HMS for almost every scenario. The shaded regions  
23 around each line define the range of runtimes for each test case. This vari-  
24 ability in runtime was caused primarily from the limited precision available for  
25 measuring runtimes, in particular for the HMS case, but also variability in com-  
26 puting resources available during each model run. The results suggest that the  
27 HMS runtime is following a linear trend for 100, 150 and 200 subbasins where  
28 the model completed in one second for both the 1 and 50 subbasin cases. The  
29 OpenMI shows some evidence of nonlinearity as the number of subbasins in-



1 creases, and we present a separate test later in this section for extending the  
2 study beyond 200 model elements to further explore this finding. Analysis of  
3 just the test results and not extrapolating beyond 200 subbasins, however, sug-  
4 gests that the overhead introduced by component-to-component data transfers  
5 in the OpenMI are minimal. The small range of variability for the OpenMI  
6 simulations is a result of our ability to precisely measure execution times for  
7 the different simulations. In contrast, the HMS’s variability is much larger, due  
8 primarily to the lack of precision in which execution times could be recorded.  
9 This lack of precision also offers an explanation for the seemingly inconsistent  
10 behavior of the HMS. Since the actual recorded execution times for the HMS  
11 model vary irregularly, we looked at how the shaded region trends as loading  
12 increases.

13 Figure 9 illustrates memory consumption for each system during the loading  
14 test. Values were determined by manually examining CPU resources while si-  
15 multaneously opening, loading, and executing simulations. The “Startup” and  
16 “Model Load” phases of simulation represent the opening of the application it-  
17 self and loading of the various models, respectively. Their values remain fairly  
18 constant for both systems, even as the number of hydrologic units increases. In  
19 contrast, the “Simulation” portion of model execution changed in each scenario.  
20 Both models follow a linear and fairly constant trend with a small slope during  
21 this phase. This suggests that as the number of data points increases beyond  
22 200 elements, we can expect both models to behave in a similar way.

23 Because the loading test suggested that a nonlinear trend may occur in the  
24 OpenMI system as the number of modeling units per simulation continues to  
25 increase, we extended the load test to include a larger number of model ele-  
26 ments. Synthetic element sets were created so that the number of modeling  
27 units could extend beyond the range typically used for semi-distributed mod-  
28 eling. This large-scale loading test was only done for the OpenMI model, as it  
29 was not possible for us to extend the HMS into the range of thousands of sub-  
30 basin units. Figure 10 illustrates how the OpenMI model execution time varies  
31 as the number of modeling elements increases beyond the range of the previ-

1   ous test. The first plot (left) shows the total model simulation time compared  
2   to the component-to-component communication time, which was estimated by  
3   subtracting the sum of the time spent within each component’s *Initialize*, *Per-*  
4   *formTimeStep*, and *Finish* methods from the total simulation time. It is clear  
5   from this analysis that the component-to-component communication represents  
6   only a small fraction of the total model execution time. Also, we did not see  
7   any evidence of nonlinearity in the OpenMI communication itself so that any  
8   nonlinearity present in Figure 8 is attributed to the scaling of the individual  
9   model components and not the OpenMI communication paradigm.

10    The second plot in Figure 10 (right) provides further insight into computa-  
11   tional performance by presenting the simulation runtime for each component,  
12   which equals the sum of the time spent within that component’s *Initialize*,  
13   *PerformTimeStep*, and *Finish* methods. This analysis shows that the major-  
14   ity of the model simulation time is due to the Muskingum component and,  
15   more specifically, to this component’s *PerformTimeStep* method. This finding  
16   illustrates how the computational expense of a single component can have a  
17   profound impact on the overall execution speed of the model. One advantage of  
18   componentizing the model is that it is relatively easy to isolate computational  
19   bottlenecks within the system by tracking the runtime of each component within  
20   the system, and because each component is loosely coupled within the system,  
21   efforts can be made to address computational inefficiencies of a single component  
22   in isolation of the larger system.

23    In the endurance test, the Smith Branch watershed was delineated into 15  
24   subbasins and simulations were executed over varying time durations, all hav-  
25   ing the same time step. The results show that the OpenMI model scales well  
26   in comparison to the HMS as simulation duration increases (Figure 11). It was  
27   predicted that the runtime of the OpenMI model would drastically increase as  
28   the number of time steps per simulation increased. This assumption was made  
29   on the basis that, if a performance lag exists during component communication,  
30   then this lag will become more pronounced as the number of communications  
31   increases. In fact, the OpenMI model completed in approximately half the time

1 as HMS for a simulation that required 10,000 time steps. As previously stated,  
2 the shaded region of the figure represents variability in the simulation runtimes  
3 that resulted primarily from the limited precision in which runtimes could be  
4 measured and, to a lesser extent, variability in computing resources available  
5 when an individual model run was completed. The upper and lower bounds  
6 of these regions are defined by the minimum and maximum recoded execution  
7 times of the respective modeling systems. The HMS scaling follows an irregular  
8 trend that appears to be nonlinear with the exception of the result for 5,000  
9 time steps. In contrast, the OpenMI scaling appears to be linear over the entire  
10 test range. This result was expected because the same calculation is evaluated  
11 on each time step of the model run, therefore 100 more time steps would require  
12 100 more calculations per component. The variability in the OpenMI model's  
13 execution times are very small because it was possible to record simulation run-  
14 times with high accuracy, whereas the HMS variability is much larger because  
15 our method for measuring simulation runtimes of the HMS model lacked pre-  
16 cision. Considering the lower limit of these regions, both models appear to  
17 follow a linear trend, although the HMS runtime increases more rapidly than  
18 that of the OpenMI. From this result we can infer that as the simulation length  
19 increases, the OpenMI model will continue to scale better than the HMS.

20 Figure 12 illustrates the allocated memory for each model as three distinct  
21 phases of simulation: "Startup," "Model Load," and "Simulation." Both models  
22 respond similarly under the "Startup" and "Model Load" phases of the test.  
23 These trends are expected to continue linearly as the number of time steps  
24 exceeds 10,000. The primary concern was how the OpenMI architecture stores  
25 data over long periods of time. As expected, each model consumes more memory  
26 during the "Simulation" phase. Both the HMS and OpenMI models, however,  
27 require additional resources at a similar rate. The outcome of this test suggests  
28 that performance lags do not result from data storage over a long simulation  
29 duration.

30 When interpreting the results from the semi-distributed load and endurance  
31 tests we must consider that HMS is a fully implemented model, meaning it

1 includes additional data logging and post processing routines. The OpenMI  
2 implementation was created for this study and is a very light-weight imple-  
3 mentation that does not include robust error handling, data logging, or user  
4 customization. These differences may influence the scalability of each model.  
5 For this reason, the results should not be interpreted to mean that OpenMI  
6 is faster than HMS, because the two models do not include the exact same  
7 functionality. When designing these experiments, the goal was to quantify how  
8 the two models scaled in a general sense as we increased data communication  
9 loads and frequencies. The most important discovery, therefore, is that the  
10 OpenMI model scaled in a similar way to the HMS. Furthermore, the OpenMI  
11 architecture does not introduce any significant computational overhead.

12 We did not anticipate that both versions of the model would scale simi-  
13 larly, instead we expected that the additional communication introduced by  
14 the loosely integrated approach would result in performance loss. This per-  
15 formance loss, we thought, would stem from inefficiencies in how data is com-  
16 municated between OpenMI model components during runtime. The load and  
17 endurance tests that were performed indicate that the mechanism by which  
18 the OpenMI shares data amongst components introduces insignificant lag for  
19 semi-distributed applications. However, because it is possible that the data  
20 communicated between components was sufficiently small that lags appeared to  
21 be non-existent, we performed another test to evaluate the OpenMI under larger  
22 loading conditions. This second test showed that the OpenMI communication  
23 had a small impact on the overall model runtime and scaled with a linear trend.

## 24 **5. Summary, Conclusions, and Future Work**

25 The primary motivation for this study was to improve understanding of  
26 the computational efficiency of a loose integration paradigm, specifically the  
27 OpenMI version 1.4, for modeling hydrologic systems. Our goal was to first  
28 show that it is possible to construct a process-level loosely integrated hydro-  
29 logic model, and second to quantify the computational cost associated with

1 message passing. To do this, a semi-distributed rainfall/runoff watershed model  
2 was developed as a set of OpenMI components and evaluated in two perfor-  
3 mance tests. For comparison purposes, we constructed a similar representation  
4 using an industry-standard watershed model: HMS. In the first test, the size  
5 of messages passed between components was gradually increased. This test  
6 addressed the OpenMI’s ability to manage the transfer of various sized data  
7 packets between components. The OpenMI model itself was then tested under  
8 larger loading conditions, outside the practical range for semi-distributed water-  
9 shed modeling applications, to further investigate its scalability. In the second  
10 test, the number of messages passed between model components was increased  
11 to test its impact on simulation runtime. This was accomplished by extending  
12 the simulation duration while maintaining a constant time step. This test ad-  
13 dressed the efficiency of the OpenMI’s request and reply mechanism. Together,  
14 these experiments allowed us to quantify how the OpenMI model scaled as the  
15 size and number of messages passed between model components increased.

16 We found that the OpenMI-based model was able to produce identical val-  
17 ues to the HMS model, a result that we expected and that was necessary to  
18 show prior to performance testing. The load test evaluated how efficiently data  
19 is shared among components and quantified the effect that the transfer of in-  
20 creasingly sized data packets has on model performance. We found that the  
21 OpenMI model runtimes scaled well in comparison to the HMS. However, the  
22 shaded regions in Figure 8 suggested that the OpenMI may follow an nonlinear  
23 trend as the number of modeling units is extended beyond what was used in  
24 this study. We were able to further explore this finding by extending the load-  
25 ing conditions beyond the range of semi-distributed applications using synthetic  
26 modeling units. We found from this experiment that the nonlinear trend of the  
27 OpenMI model was caused by the computational expense of the Muskingum  
28 component and not the OpenMI communication mechanism. By decomposing  
29 the system into components with standard interfaces, it is trivial to track the  
30 computational time spent in each component’s key methods during the simu-  
31 lation run. This can provide valuable information for addressing model bottle-

1 necks and improving overall model performance. The endurance test evaluated  
2 how the OpenMI performed for increasing time durations. The results of this  
3 test (Figure 11) again show that the OpenMI scales favorably in comparison to  
4 the HMS. In fact, it appears from our tests to follow a linear trend, suggesting  
5 that as simulation duration continues to increase, runtime will still be within  
6 an acceptable range. Overall, the performance experiments of adjusting the size  
7 and quantity of messages passed within an OpenMI configuration showed no  
8 significant performance issues within ranges typical for semi-distributed water-  
9 shed modeling. Future work should extend these tests to ranges more common  
10 for more computationally demanding hydrologic modeling paradigms.

## 11 **Acknowledgments**

12 This work was supported by the National Science Foundation under the  
13 grant NSF EAR 0622374 “GeoInformatics: Consortium of Universities for the  
14 Advancement of Hydrologic Sciences, Inc. (CUAHSI) Hydrologic Information  
15 Systems.”

## 16 **References**

- 17 Ahrends, H., Mast, M., Rodgers, C., Kunstmann, H., 2008. Coupled hydro-  
18 logicaleconomic modelling for optimised irrigated cultivation in a semi-arid  
19 catchment of west africa. *Environmental Modelling & Software* 23 (4), 385–  
20 395.
- 21 Al-Abed, N., Abdulla, F., Khyarah, A. A., 2005. GIS-hydrological models for  
22 managing water resources in the zarqa river basin. *Environmental Geology*  
23 47 (3), 405–411.
- 24 Anderson, M. L., Chen, Z., Kavvas, M. L., Feldman, A., 2002. Coupling HEC-  
25 HMS with atmospheric models for prediction of watershed runoff. *Journal of*  
26 *Hydrologic Engineering* 7 (4), 312.

- 1 Argent, R. M., Voinov, A., Maxwell, T., Cuddy, S. M., Rahman, J. M., Seaton,  
2 S., Vertessy, R. A., Braddock, R. D., 2006. Comparing modelling frameworks  
3 - a workshop approach. *Environmental Modelling & Software* 21 (7), 895–910.
- 4 Barthel, R., Janisch, S., Schwarz, N., Trifkovic, A., Nickel, D., Schulz, C.,  
5 Mauser, W., 2008. An integrated modelling framework for simulating regional-  
6 scale actor responses to global change in the water domain. *Environmental*  
7 *Modelling & Software* 23 (9), 1095–1121.
- 8 Brinkman, R., Gregersen, J. B., Hummel, S., Westen, S. J., 2005. Part c - the  
9 org.OpenMI.Standard interface specification. *The OpenMI Document Series*,  
10 1–79.
- 11 Castronova, A. M., Goodall, J. L., July 2010. A generic approach for developing  
12 process-level hydrologic modeling components. *Environmental Modelling &*  
13 *Software* 25 (7), 819–825.
- 14 Chow, V. T., Maidment, D. R., Mays, L. W., 1988. *Applied Hydrology*. McGraw-  
15 Hill Inc, Boston, Massachusetts.
- 16 Chu, X. F., Steinman, A., Feb. 2009. Event and continuous hydrologic modeling  
17 with HEC-HMS. *Journal of Irrigation and Drainage Engineering* 135 (1), 119–  
18 124.
- 19 Feldman, A. D., 2000. Hydrologic modeling system HEC-HMS techni-  
20 cal reference manual. *Technical Reference Manual CPD-74B*, U.S.  
21 Army Corps of Engineers, Hydrologic Engineering Center, Davis, CA,  
22 [http://www.hec.usace.army.mil/software/hec-hms/documentation/CPD-](http://www.hec.usace.army.mil/software/hec-hms/documentation/CPD-74B_2000Mar.pdf)  
23 [74B\\_2000Mar.pdf](http://www.hec.usace.army.mil/software/hec-hms/documentation/CPD-74B_2000Mar.pdf).
- 24 Govindaraju, M., Katz, D. S., Kohl, J. A., Krishnan, M., Kumfert, G., Lar-  
25 son, J. W., Lefantzi, S., Lewis, M. J., Malony, A. D., McInnes, L. C., Allan,  
26 B. A., Nieplocha, J., Norris, B., Parker, S. G., Ray, J., Shende, S., Windus,  
27 T. L., Zhou, S. J., Armstrong, R., Bernholdt, D. E., Bertrand, F., Chiu, K.,  
28 Dahlgren, T. L., Damevski, K., Elwasif, W. R., Epperly, T. G. W., 2006. A

- 1 component architecture for high-performance scientific computing. International  
2 Journal of High Performance Computing Applications 20 (2), 163–202.
- 3 Gregersen, J. B., Gijssbers, P. J. A., Westen, S. J. P., 2007. OpenMI: open  
4 modelling interface. Journal of Hydroinformatics 9 (3), 175–191.
- 5 Holzworth, D. P., Huth, N. I., de Voil, P. G., 2010. Simplifying environmental  
6 model reuse. Environmental Modelling & Software 25 (2), 269–275.
- 7 Kennen, J. G., Kauffman, L. J., Ayers, M. A., Wolock, D. M., Colarullo, S. J.,  
8 2008. Use of an integrated flow model to estimate ecologically relevant hydro-  
9 logic characteristics at stream biomonitoring sites. Ecological Modelling  
10 211 (1-2), 57–76.
- 11 Knebl, M., Yang, Z., Hutchison, K., Maidment, D., 2005. Regional scale flood  
12 modeling using NEXRAD rainfall, GIS, and HEC-HMS/RAS: a case study  
13 for the san antonio river basin summer 2002 storm event. Journal of Environ-  
14 mental Management 75 (4), 325–336.
- 15 Kunstmann, H., Jung, G., Wagner, S., Clottey, H., 2008. Integration of atmo-  
16 spheric sciences and hydrology for the development of decision support sys-  
17 tems in sustainable water management. Physics and Chemistry of the Earth  
18 33 (1-2), 165–174.
- 19 Löwy, J., 2005. Programming .NET components, 2nd Edition. O’Reilly Media  
20 Inc.
- 21 Maidment, D., 2002. Arc hydro : GIS for water resources. ESRI Press, Redlands  
22 Calif.
- 23 Maxwell, R., Chow, F., Kollet, S., 2007. The groundwater land-surface atmo-  
24 sphere connection: Soil moisture effects on the atmospheric boundary layer in  
25 fully-coupled simulations. Advances in Water Resources 30 (12), 2447–2466.
- 26 Mays, L. W., 2005. Water Resources Engineering, 2005th Edition. John Wiley  
27 and Sons, Inc, Hoboken.



- 1 Meier, J., Farre, C., Bansode, P., Barber, S., Rea, D., Sep.  
2 2007. Microsoft patterns and practices. [http://msdn.microsoft.com/en-](http://msdn.microsoft.com/en-us/library/bb924359.aspx)  
3 [us/library/bb924359.aspx](http://msdn.microsoft.com/en-us/library/bb924359.aspx).
- 4 Mölders, N., Rühaak, W., 2002. On the impact of explicitly predicted runoff on  
5 the simulated atmospheric response to small-scale land-use changes an inte-  
6 grated modeling approach. *Atmospheric Research* 63 (1-2), 3–38.
- 7 Moore, R., Gijssbers, P., Fortune, D., Gregersen, J., Blind, M., 2005. The  
8 OpenMI document series: Part A - scope for the OpenMI (version 1.0). Har-  
9 monIT.
- 10 Morton, Y., Troy, D., Pizza, G., 2003. A state-based modelling approach to  
11 develop component-based control software for flexible manufacturing systems.  
12 *International Journal of Computer Integrated Manufacturing* 16 (4-5), 292–  
13 306.
- 14 Scharffenberg, W., Fleming, M., 2009. Hydrologic modeling system HEC-HMS  
15 user's manual. Technical Reference Manual CPD-74A, U.S. Army Corps of  
16 Engineers, Hydrologic Engineering Center, Davis, CA.
- 17 Singh, V. P., Woolhiser, D. A., Aug. 2002. Mathematical modeling of watershed  
18 hydrology. *Journal of Hydrologic Engineering* 7 (4), 270–292.
- 19 Soil Conservation Service, 1972. Hydrology. In: *National Engineering Hand-*  
20 *book*. section 4. U.S. Department of Agriculture, Washington, D.C.
- 21 Soil Conservation Service, 1975. Urban hydrology for small watersheds. Techni-  
22 cal Reference 55, U.S. Department of Argiculture, Washington, D.C.
- 23 Sui, D. Z., Maggio, R. C., 1999. Integrating GIS with hydrological modeling:  
24 practices, problems, and prospects. *Computers, Environment and Urban Sys-*  
25 *tems* 23 (1), 33–51.
- 26 Valentine, D., Zaslavsky, I., Jun. 2009. Introduction to WaterML schema. In:  
27 *CUAHSI WaterML 1.0 Specification*. Part 1. CUAHSI universities allied for  
28 water research.

- 1 Yu, Z., Pollard, D., Cheng, L., 2006. On continental scale hydrologic simulations
- 2 with a coupled hydrologic model. *Journal of Hydrology* 331 (1-2), 110–124.

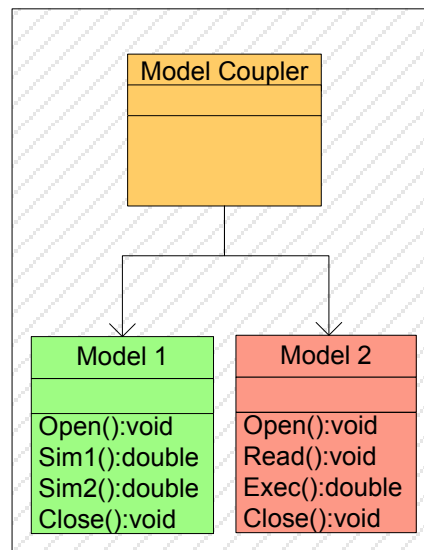


Figure 1: In a tightly integrated system, the Model Coupler class directly points to the methods defined within each Model.

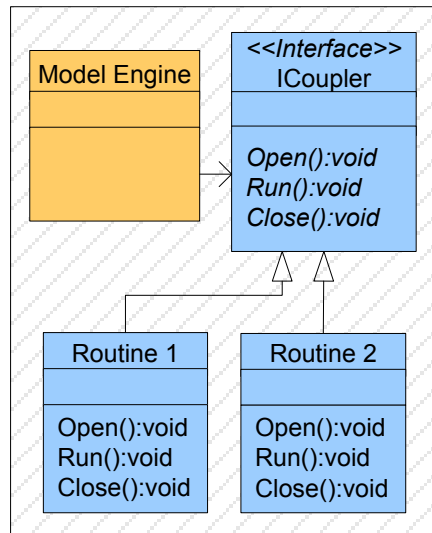


Figure 2: In a loosely integrated system, the ICoupler interface defines the methods required by the Model Engine, but they are implemented by external Routines.

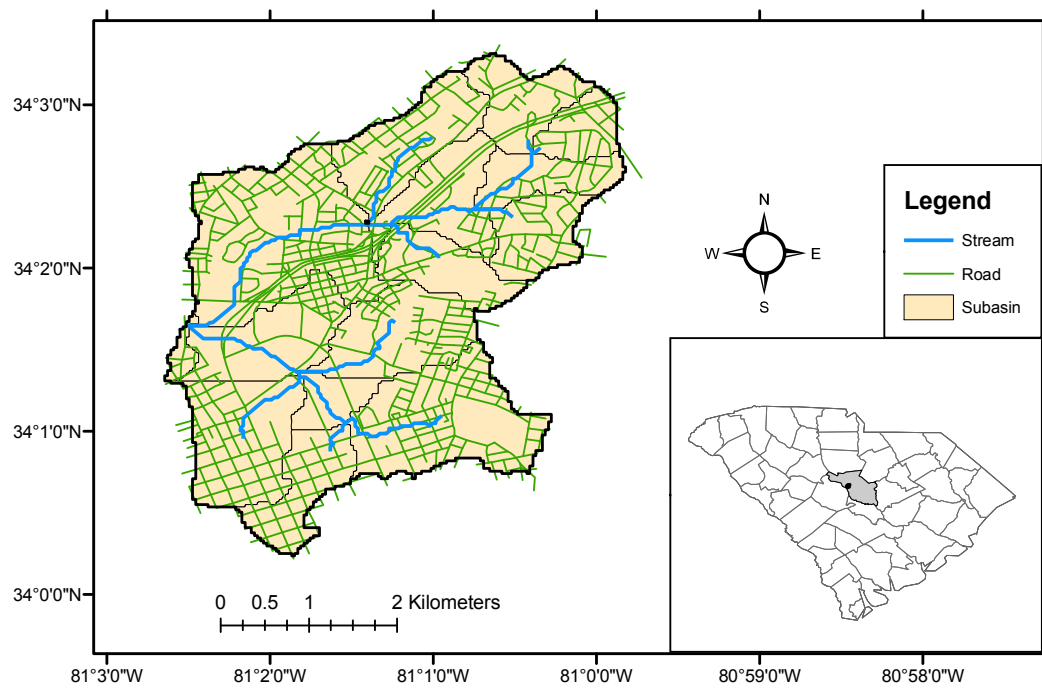


Figure 3: Schematic view of the Smith Branch watershed, including the derived sub-basins and reach network.

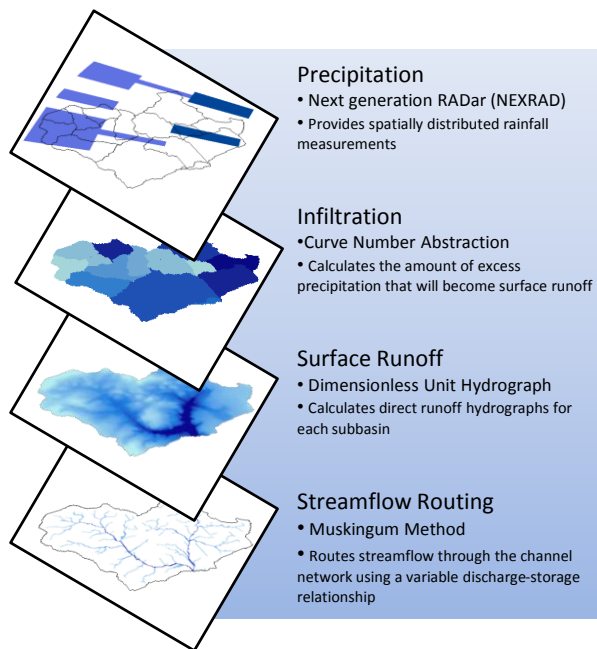


Figure 4: A conceptual representation of a simple rainfall/runoff system

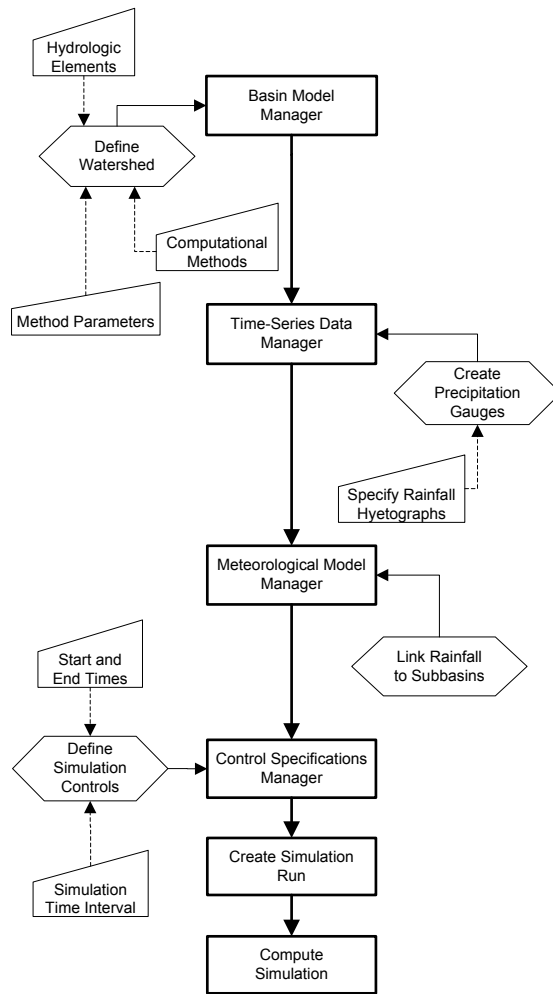


Figure 5: Flow chart showing the model development procedure within the Hydrologic Modeling System.

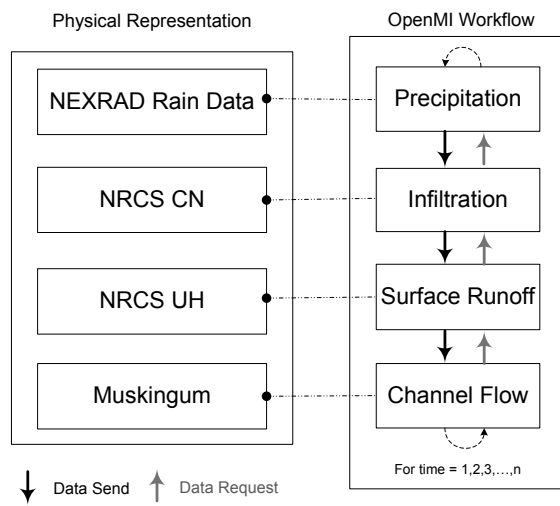


Figure 6: The work flow of the loosely integrated model, showing the interaction of each component with a single physical process representation.



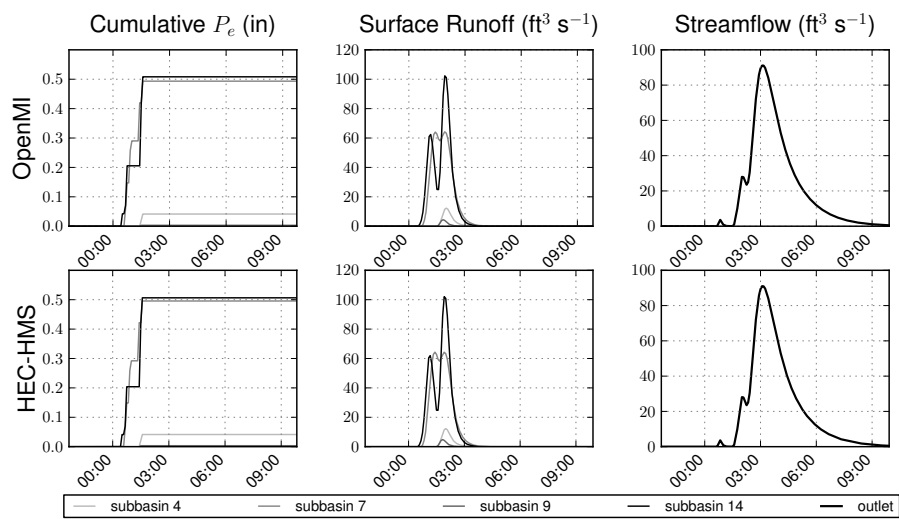


Figure 7: The results produced by each modeling system. The OpenMI values are given in the first row and HMS values are in the second. Cumulative  $P_e$  and surface runoff are shown for the contributing subbasins, and streamflow is at the outlet of the watershed.

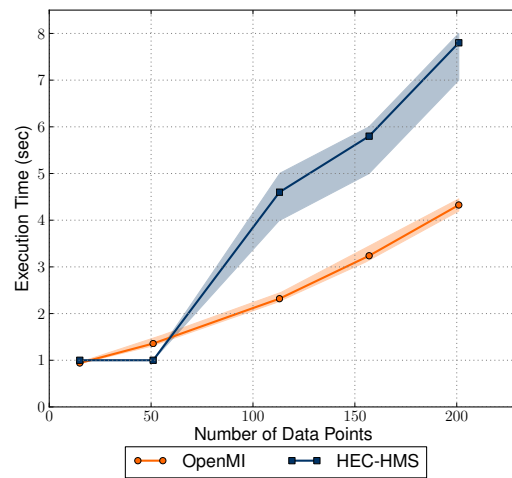


Figure 8: The load test shows how model run time is effected as the number of computational elements increases. The shaded regions indicate variability in simulation run time, bounded by the minimum to maximum recorded values.

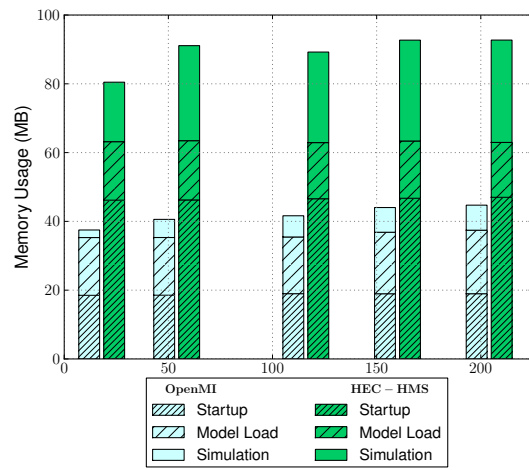


Figure 9: The memory allocated during each phase of model simulation, under various loading conditions.

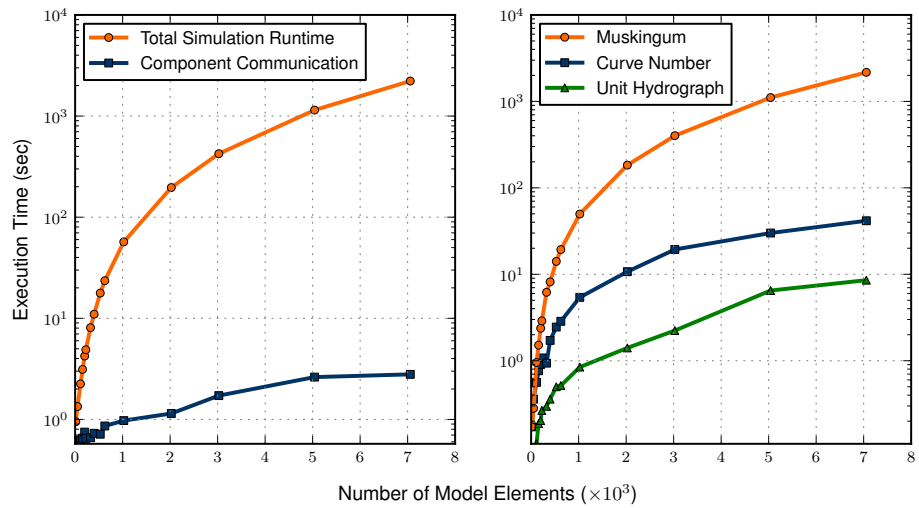


Figure 10: The large-scale loading test illustrates how the OpenMI model scales when the number of model elements extends into the thousands. The plot on the left gives the total simulation time for the composition and the fraction of time spent outside of the component's *Initialize*, *PerformTimeStep*, and *Finish* methods and therefore attributed to component-to-component communication. The plot of the right presents the total time spent within each component's *Initialize*, *PerformTimeStep*, and *Finish* methods.

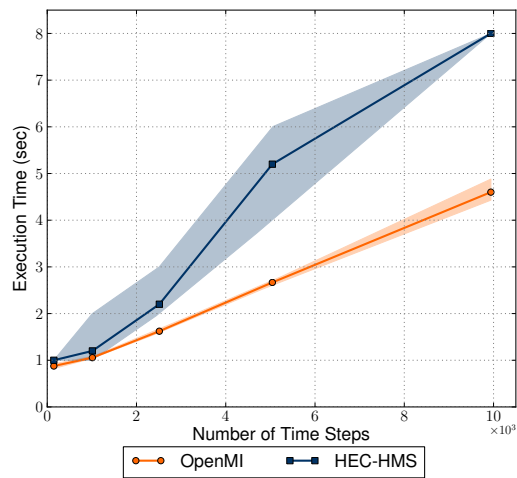


Figure 11: The endurance test shows how model run time is effected by an increasing number of simulation time steps. The shaded regions indicate variability in simulation run time, bounded by the minimum and maximum recorded values.

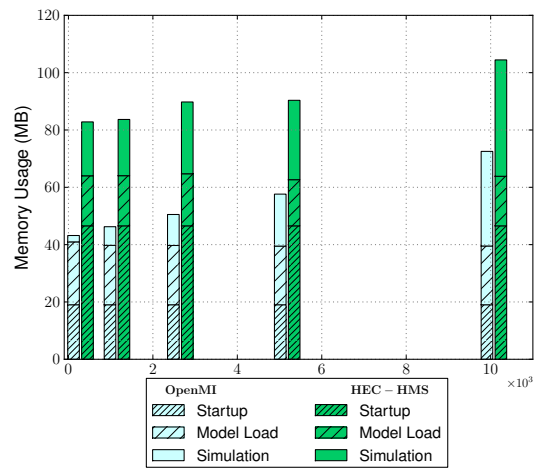


Figure 12: The memory allocated during each phase of model execution, for various simulation durations.