

1 A Generic Approach for Developing Process-Level Hydrologic Modeling Components

2 Anthony M. Castronova<sup>1</sup> and Jonathan L. Goodall<sup>2</sup>

3 Abstract

4 Component software architectures offer an alternative approach for building large,  
5 complex hydrologic modeling systems. In contrast to more traditional software  
6 paradigms (i.e. procedural or object-oriented approaches), using component-based  
7 approaches allows individuals to construct autonomous modeling units that can be linked  
8 together through shared boundary conditions during a simulation run. One of the  
9 challenges in component-based modeling is designing a simple yet robust means for  
10 authoring model components. We address this challenge by presenting an approach for  
11 efficiently creating standards-based, process-level hydrologic modeling components.  
12 Using this approach, a hydrologic process is implemented as a modeling component by  
13 (1) authoring a configuration file that defines the properties of the component and (2)  
14 creating a class with three methods that define the pre-run, runtime, and post-run  
15 behavior of the modeling component. We present the design and implementation of this  
16 approach, which we call the Simple Model Wrapper (SMW), and demonstrate how it can  
17 be applied to create an Open Modeling Interface (OpenMI)-compliant modeling  
18 component for a basic hydrologic process.

19 **Subject Headings:** Hydrologic modeling; Modeling software architectures;  
20 Component-based modeling; Integrated Modeling; Multi-disciplinary modeling

---

<sup>1</sup> Graduate Research Assistant, Department of Civil and Environmental Engineering, University of South Carolina, 300 Main Street, Columbia, South Carolina 29208, voice: (803) 777-8184, fax: (803) 777-0670, [castrona@cec.sc.edu](mailto:castrona@cec.sc.edu)

<sup>2</sup> Assistant Professor, Department of Civil and Environmental Engineering, University of South Carolina, 300 Main Street, Columbia, South Carolina 29208, voice: (803) 777-8184, fax: (803) 777-0670, [goodall@ceec.sc.edu](mailto:goodall@ceec.sc.edu)

21 **1. Introduction**

22 Hydrologic models are typically built to address isolated parts of the overall  
23 hydrologic cycle, making it challenging to answer science or management questions that  
24 require process representations implemented within different models (e.g.  
25 groundwater/surface water systems or watershed/estuary systems) (Scholten et al. 2007).  
26 This has resulted in attempts to “hard-code” two or more models together, or to extend  
27 models beyond their original scope by including addition subroutines or packages  
28 (Sophocleous and Perkins 2000). Neither of these approaches serves as an adequate long  
29 term solution. In the former approach, the developers’ intent is usually to satisfy a  
30 unique study or application, while in the latter approach, it becomes increasingly difficult  
31 for one model to maintain state-of-the-art algorithms for a multidisciplinary system. A  
32 more generic approach for coupling hydrologic and environmental models across  
33 discipline boundaries is needed.

34 Component-based modeling offers an alternative approach for constructing  
35 hydrologic models that emphasize the decomposition of a system into functional  
36 components that communicate via standard interfaces (Argent 2004). In a component-  
37 based modeling paradigm, processes are designed as computational units that can plug-  
38 and-play with other computational units in a modeling system (Allan et al. 2006). Each  
39 component must have a standard means for communicating with other components  
40 within the modeling system so that the overall modeling system can be re-configured  
41 through the introduction of new components or the “swapping out” of existing ones. The  
42 model developer is able to define the overall composition of model components using

43 unique system representations, which is ideal for modeling complex systems that involve  
44 multi-scale and multidisciplinary components (Kennen et al. 2008).

45         Within the water domain, several component-based modeling systems are in  
46 development. These systems, which are briefly reviewed in the following section, share  
47 the idea of a standard interface definition for modeling components designed to facilitate  
48 interaction between components during simulation runtime. The modeling systems are  
49 often complex pieces of software designed by software engineers, sometimes making it  
50 difficult for modelers to contribute their own process representations in a simple and  
51 straight forward way. Our vision for component-based modeling is as a new way for  
52 authoring open, transparent, and flexible hydrologic modeling systems. In order to  
53 achieve this vision, there is a need to offer modelers an approach for implementing plug-  
54 and-play components that enable them to easily and efficiently incorporate their own  
55 conceptualizations of hydrologic process routines into standards-based modeling  
56 components. This paper presents such an approach that we call the Simple Model  
57 Wrapper (SMW).

58         The following background section presents a brief introduction to component-  
59 based modeling systems under development for environmental systems analysis, focusing  
60 specifically on the Open Modeling Interface (OpenMI) as a standard interface definition  
61 for modeling components. We then present the proposed SMW approach for creating  
62 OpenMI-compliant process level modeling components. We demonstrate how the SMW  
63 can be used for hydrologic modeling through a case study where a hydrologic modeling  
64 routine, the Curve Number Method, is implemented as an OpenMI-compliant component.

65 Finally, we discuss the benefits and costs of using the SMW approach, and component-  
66 based modeling in general, for modeling complex environmental systems.

67

## 68 **2. Background**

69 Component based modeling, has received increased focus in modern  
70 environmental modeling systems. Argent et al. (2006) provides an overview of  
71 component modeling for environmental systems including efforts within the hydrologic  
72 community to redirect modeling focus toward the development of component models, as  
73 well as componentizing existing models. The Community Surface Dynamics Modeling  
74 System (CSDMS), the Earth System Modeling Framework (ESMF), the Object Modeling  
75 System (OMS), and the Open Modeling Interface (OpenMI) are a few examples of  
76 component modeling approaches being developed within the environmental modeling  
77 community.

78 The Community Surface Dynamics Modeling System (CSDMS) is a modeling  
79 environment built from free software modules that focuses on the prediction of sediment  
80 and material transport over various time and space scales (Syvitski et al. 2004). The  
81 objective is to encourage interdisciplinary modeling by coupling components and  
82 enabling developers to create models specific to individual studies. The model developer  
83 interacts with three architectural elements when building models in CSDMS: Standard  
84 Utilities, Modules, and a Toolkit (Syvitski et al. 2004). Standard utilities define data  
85 structure, graphics rendering, module connectors, and a web interface (Anderson et al.  
86 2004). Module components are developed by modelers and represent the actual  
87 sedimentary computations. The Toolkit is supplied to the modelers in order to aid in

88 module development, and consists of numerical solvers, grid generators, and automatic  
89 code generators. The CSDMS uses the Open Modeling Interface (OpenMI) standard to  
90 maintain runtime communication, and the Common Component Architecture (CCA) to  
91 facilitate high performance computing (Slingerland et al. 2008).

92         The Earth System Modeling Framework (ESMF) is a multidisciplinary effort  
93 between Earth science modeling centers within the United States that aims to increase  
94 software reuse and model interoperability (Hill et al. 2004). Components are defined by  
95 a physical domain, the process which they represent, or their scientific function (Collins  
96 et al. 2005). Each component operates under the Fortran derived concept of “states” in  
97 which all components have one or more import and export states that enable modeling  
98 components to easily exchange data (Collins et al. 2005). Most ESMF components are  
99 “gridded” meaning they simulate physical domains that can be represented by a regular  
100 or irregular computational grid. This framework is most widely used for applications that  
101 require high performance computing, such as the atmospheric sciences for weather and  
102 climate modeling.

103         The Object Modeling System (OMS) is developed by a collaborative effort  
104 between the United States Department of Agriculture- Agriculture Research Service  
105 (USDA-ARS) and partner agencies to resolve the lack of integrated hydrological models  
106 (Kralisch et al. 2005). It uses modern software principals, particularly object-oriented  
107 programming, to provide modelers with the ability to customize their models by  
108 combining individually compiled processes together into a modeling system (Kralisch et  
109 al. 2005). An OMS model consists of multiple, independent modules that are coupled by  
110 a standard software interface. These modules are further divided into system and

111 scientific components, which represent model building tools and physically-based  
112 computations, respectively. The scientific component of a module represents the  
113 physical process that is being calculated, whereas system components are used to manage  
114 the coupling and execution of several scientific components. Models are built from a  
115 combination of “system” and “scientific” components, and are joined together within the  
116 Netbeans (<http://www.netbeans.org>) runtime environment (Ahuja et al. 2005).

117         The Open Modeling Interface (OpenMI) is a component interface standard  
118 developed through the Water Framework Directive (Moore et al. 2005). It differs from  
119 the previous modeling systems in that, while the other systems intend to be complete  
120 modeling environments, the primary aim of the OpenMI is to facilitate interoperability  
121 between otherwise independent environmental models (Gregersen et al. 2007). The  
122 OpenMI, therefore, is designed to serve as a communication standard for model  
123 interoperability (Moore and Tindall 2005) that could be adopted by each of the  
124 aforementioned modeling systems. The CSDMS, in fact, uses the OpenMI in this  
125 capacity. Additionally, the OpenMI Association Technical Committee (OATC) provides  
126 a Software Development Kit (SDK) and component configuration editor application  
127 (OpenMI Configuration Editor GUI) which allows modelers to use the OpenMI as the  
128 basis for a component-based modeling system.

129         The OpenMI communication protocol consists of three fundamental concepts: a  
130 linkable component, an exchange item, and a link (Figure 1) (Brinkman et al. 2005). A  
131 linkable component is an object that implements an OpenMI standard interface (e.g.  
132 ILinkableComponent). Exchange items are objects communicated between components  
133 and are comprised of an element set and a quantity. An element set defines the geospatial

134 objects for an exchange item whereas a quantity describes the variable for an exchange  
135 item including its units and unit dimensions (Gregersen et al. 2007). Components can  
136 wrap databases, models, visualization routines, or any other computational resource  
137 (Gijbers et al. 2005). Links are used to couple components together and define the  
138 source and destination for exchange items transferred between them. A single  
139 component can have multiple input and output exchange items. An interlinked set of  
140 components is called a component composition and can be thought of as a "model" for a  
141 specific system.

142         The OpenMI offers a foundation for component-based modeling within a loosely  
143 coupled structure that can supplement more tightly coupled modeling systems such as  
144 ESMF, CSDMS, and OMS. The OpenMI, however, was primarily designed for  
145 wrapping legacy codes (Moore and Tindall 2005), making it difficult for modelers to use  
146 to create new process-level components in an efficient, straight forward manner.  
147 Modelers must understand OpenMI concepts at a fairly low level to wrap their own codes  
148 as OpenMI-compliant components. The Simple Model Wrapper (SMW) is an attempt to  
149 abstract the details of the OpenMI from model developers in order to encourage the  
150 development of process-level model components that adhere to the OpenMI standard. It  
151 allows these process-level components to be used alongside other OpenMI-compliant  
152 model components to model environmental systems. The following section describes the  
153 SMW design, followed by a section that demonstrates an example application of the  
154 SMW for wrapping a hydrologic process.

155

### 156 **3. The Simple Model Wrapper**

157           The Simple Model Wrapper (SMW) consists of two parts: (1) a configuration file  
158 that defines the metadata for a model component and (2) an abstract class with three  
159 “overridable” methods that a modeler is responsible for implementing. This approach  
160 was selected to minimize the amount of code that must be written in order to create a new  
161 model component. The configuration file for a SMW component, which is an eXtensible  
162 Markup Language (XML) file, defines the metadata properties of that component. These  
163 properties are important because they determine, among other things, the inputs and  
164 outputs for each component. The abstract class inherits from the OpenMI IEngine  
165 interface, making it an OpenMI compliant component. The hierarchical relationship of  
166 the SMW with respect to other OpenMI interfaces is shown in Figure 2. Each tier  
167 provides a level of abstraction from the OpenMI Standard Modeling interfaces. The  
168 IEngine interface is a simplification of the ILinkableComponent interface, and the SMW  
169 abstract class is a further simplification of the IEngine interface that handles many lower-  
170 level operations such as data input and output (I/O) and maintaining state during a model  
171 run.

172

### 173 3.1. The Configuration File

174           To abstract the modeler from manually defining component properties within the  
175 OpenMI IEngine interface, the SMW has been designed to read this information from an  
176 external configuration file with a specific XML schema (Figure 3). The XML schema for  
177 defining the component properties closely follows the OpenMI object model for defining  
178 exchange items, the model description, and time horizons. One SWM component must  
179 be associated with one configuration file. The configuration file must include one or



180 more output exchange item elements and zero or more input exchange item elements.  
181 Each exchange item element must include one element set child element and one quantity  
182 child element. Since component properties may frequently change when prototyping  
183 process-level components, there is an advantage to extracting these properties from the  
184 source code so that components do not have to be recompiled after changes to component  
185 metadata. The configuration file is loaded into memory during the component  
186 initialization phase to populate the component's properties.

187

### 188 3.2. Method Description

189 Three methods must be implemented when building a new component using the  
190 SMW: *Initialize*, *PerformTimeStep*, and *Finish*. These methods are common to most  
191 component-based modeling systems and represent the pre-run, runtime, and post-run  
192 states of model simulation. A component developer is responsible for implementing  
193 these three methods by overriding them within the SMW abstract class. From the  
194 perspective of the model developer, the SMW reduces the IEngine interface from 19  
195 methods to 3 methods. The simplification of the IEngine interface requires the developer  
196 to implement three public methods and all other methods remain hidden. As a result, this  
197 restricts the control that the developer has over the IEngine interface, which in some  
198 cases can serve as a limitation to component design. While this simplification does limit  
199 the control the developer has when authoring a component, it also greatly increases the  
200 efficiency for creating simple, process-level modeling components by reducing the  
201 amount of code necessary to develop a new component.

202           The *Initialize* method is executed when the component is constructed and prior to  
203 the run-time portion of configuration execution. In this method, modeling units, system  
204 parameters, and initial conditions are loaded into memory from input files. During model  
205 run-time, the *PerformTimeStep* method is called to produce output exchange items for a  
206 specific time step. The developer defines the operations performed on each time step and  
207 is able to use input data supplied by linked components in a composition. The  
208 *PerformTimeStep* method returns the resulting values, making them available to other  
209 components within the component composition. Lastly, the *Finish* method is called when  
210 the model has completed its simulation run. It can be used to close files opened for  
211 reading in the *Initialize* or *PerformTimeStep* methods, or to write out simulation results  
212 produced by a component. This method can also be used to implement post-processing  
213 routines because there is no need to communicate with other model components within  
214 the *Finish* method. Once the *Finish* method has been called, all allocated memory for the  
215 component object is released from the computer.

216

### 217 3.3. Component Composition

218           A model composition defines an interlinked set of components and the exchange  
219 of information between them (Figure 4). A detailed view of a component implementing  
220 the SMW is represented in the shaded region of Figure 4. In this case, three files are  
221 associated with the component: model.dll, config.xml, and elements.shp. The model.dll  
222 file contains a class that performs a process-level computation using the SMW approach  
223 and has been compiled into a dynamic-link library (DLL). The config.xml file is the  
224 configuration file that supplies the component metadata discussed in the previous

225 subsection. Finally, a shapefile is used to spatially define the component's modeling units  
226 and parameters. However, components are not required to use a shapefile for this  
227 purpose; other file formats could be used instead. For example, in the case where it is not  
228 critical to describe the geometries associated with element sets (e.g. if the elements are  
229 points or non-spatial), a simple ASCII text file might be preferred to a shapefile. In this  
230 case, the component developer can write code that reads data from the ASCII file instead  
231 of a shapefile when constructing OpenMI exchange item objects..

232         The interaction between components within a composition is represented by  
233 directional links that join them together into a model composition. Several links exist in  
234 Figure 4 that collectively define how data will flow through the model composition on  
235 each time step of a simulation run. It is important to note that, although values are  
236 calculated using the model.dll file, the transfer of these values is managed completely by  
237 the SMW class. Figure 4 illustrates this concept with the bi-directional link indicating  
238 the passing of input data to the model.dll through the SMW class, and the passing of  
239 calculation results from the model.dll back to the SMW class as output data. This  
240 process is made possible by calling the *GetValues* and *SetValues* methods, respectively,  
241 of the SMW. Together, the SMW and the items within the shaded region comprise one  
242 component in the composition. The modeler will not see the details of the SMW; it will  
243 appear just as any other component within the composition.

244

#### 245 **4. Case Study**

246         In order to provide a practical example implementation of the Simple Model Wrapper  
247 (SMW), this section demonstrates how the SMW can be used to implement a hydrologic

248 process as an OpenMI-compliant component. For context, first consider the overall steps  
249 necessary to create a component-based model for estimating streamflow from a rainfall  
250 event. Model development would start with a conceptualization of the system as a series  
251 of interconnected functional units or components. In this case, the model might be  
252 decomposed into such components as precipitation, infiltration, surface runoff, and  
253 channel routing (Figure 5). Each component in this model, with the exception of the  
254 rainfall component, requires an input exchange item to perform its computation, and each  
255 component will produce an output exchange item that can be used by other components  
256 in the composition. Links between components define the output from one component  
257 that serves as input to another component, thus establishing the data transfers that occur  
258 while the model composition is running. Because each component has no prior  
259 knowledge of the other components within the workflow composition prior to the  
260 component linking step, the components are considered to be loosely-coupled. The  
261 advantage of loose coupling is that it is possible to easily “plug-and-play” different  
262 components within the workflow to create new or different workflow compositions.

263 For the example shown in Figure 5, the rainfall component provides precipitation  
264 values as an output exchange item by reading local data files to populate the exchange  
265 item objects. This demonstrates how components are not limited to process routines, but  
266 can be file readers, visualization tools, or other functional tools. This precipitation  
267 exchange item serves as input for the infiltration component, which is then able to  
268 calculate excess precipitation values for each modeling unit (i.e. subwatershed) in the  
269 watershed. The excess precipitation values are then supplied as input to the surface  
270 runoff component, which calculates the runoff hydrograph that serves as input for the

271 channel routing component. Finally, the channel routing component uses the runoff  
272 hydrographs from the subwatersheds to estimate a streamflow hydrograph at the  
273 watershed outlet.

274

#### 275 4.1. Component Design

276 Although each of these components would be necessary to build an application of  
277 the aforementioned model composition, this section will demonstrate the SMW by  
278 focusing on one of those components: the infiltration component. We choose to  
279 implement this component using the Curve Number (CN) Method because it is a simple,  
280 widely known hydrologic process representation and is therefore appropriate for this  
281 proof-of-concept example. The CN Method uses an empirically derived relationship  
282 between land use, soil type, and antecedent soil conditions, to estimate excess rainfall  
283 from precipitation (Chow et al. 1988). Excess rainfall is defined as the fractional rainfall  
284 that will lead to streamflow.

285 To begin development of the CN component, a new C# .Net class was created that  
286 inherits from the Simple Model Wrapper abstract class. This new class allows the  
287 component developer to implement the three methods, *Initialize*, *PerformTimeStep*, and  
288 *Finish*, discussed earlier. The *Initialize* method is responsible for setting interface  
289 properties that are needed for communication with the component, as well as preparing  
290 internal data structures for the component. Development of this method for the CN  
291 component begins by parsing the configuration file and loading its information into  
292 memory to define the component metadata (e.g. descriptions of the input and output  
293 exchange items associated with the component). The SMW includes utility functions

294 such as *SetVariablesFromConfigFile* that can be used to automate the parsing of the  
295 XML configuration file.

296 Next, a second data structure is created within the component to maintain internal  
297 information such as modeling unit IDs, curve number parameters, cumulative infiltration,  
298 cumulative precipitation, and excess rainfall. The shapefile that defines the internal data  
299 structure for the component, that is subwatersheds with CN parameters, is read into  
300 memory to complete this step. Parsing the shapefile is accomplished by utilizing  
301 methods provided by the open source SharpMap library  
302 (<http://www.codeplex.com/sharpmapi>). Again, it is not necessary to use a shapefile for  
303 storing the modeling unit properties and parameters. Any file format is acceptable for  
304 this purpose as the file is read into memory during the modeler-defined *Initialize* method.

305 The *PerformTimeStep* method is responsible for using precipitation values passed  
306 in from the linked precipitation component to calculate cumulative precipitation,  
307 cumulative infiltration, and the resulting runoff for each modeling unit within the study  
308 area. The process begins by calling the *GetValues* method of the CN component (a  
309 method inherited from the SMW abstract class) to retrieve precipitation values from the  
310 SMW's global data dictionary (Figure 6). This incremental precipitation is then  
311 numerically integrated over time to produce a cumulative precipitation value for each  
312 modeling unit. Excess precipitation ( $P_e$ ) for each sub-watershed is then calculated using  
313 Equation 1 that states

314

$$315 \quad P_e = P - I_a + F_a \quad (1)$$

316

317 where  $P$  is the cumulative precipitation,  $I_a$  is the initial abstraction, and  $F_a$  is the  
318 continuing abstraction (Chow et al. 1988).  $I_a$  and  $F_a$  are calculated using the empirical  
319 relationships expressed in Equations 2 and 3, respectively,

320

$$321 \quad I_a = 0.2S \quad (2)$$

$$322 \quad F_a = \frac{S(P - I_a)}{P - I_a + S} \quad (3)$$

323

324 where  $S$  is the maximum potential storage of the soil. Finally,  $S$  is calculated as a  
325 function of the CN parameter using Equation 4.

326

$$327 \quad S = \frac{1000}{CN} - 10 \quad (4)$$

328

329 As shown in Figure 6, these equations for the CN method are solved sequentially,  
330 for each subwatershed within the element set. Once complete, runoff hyetographs are  
331 created by subtracting the previous excess precipitation from the current one. Since  
332 OpenMI components are usually designed to operate on a time step, instead of creating a  
333 full hyetograph, only one value is produced at each subwatershed for each  
334 *PerformTimeStep* call using Equation 5

335

$$336 \quad H(t) = P_e(t) - P_e(t - 1) \quad (5)$$

337

338 where  $H$  is a value from the excess rainfall hyetograph and  $t$  is the current time index.  
339 The final steps of the *PerformTimeStep* method are to write these resulting values to the  
340 global data dictionary within the SWM by calling the *SetValues* method, and to advance  
341 time within the component calling the *AdvanceTime* function included in the SMW utility  
342 class. This enables the surface runoff component in the model composition to retrieve  
343 the values for its own calculations, and to advance time in the CN component to prepare  
344 it for the next time step of the model run.

345         The last method implemented for the CN component is *Finish*. When this  
346 method is called, the CN component has completed the model run and must write out the  
347 calculated excess precipitation values to file. This outputted data can be written to any  
348 file format or data model the component developer chooses for further analysis outside of  
349 the OpenMI runtime environment. Additional functionality can be added within this  
350 method to accomplish such tasks as data post-processing, although it is not necessary for  
351 the CN component developed here.

352         Repeating these basic steps for each of the other three components of the model  
353 composition mentioned in the previous section would allow one to build a component  
354 composition to model rainfall-runoff. Because this paper focuses on the design and  
355 implementation of the SMW, a full modeling example is beyond the scope and will be  
356 part of future work discussed in the final section of this paper. The main difference  
357 between the developments strategies of various components needed to complete the  
358 model composition is the algorithms used for producing each component's output  
359 exchange items. The precipitation component, unlike the other components, would



360 simply use a file to populate its output exchange item. The process of linking the  
361 components into a composition is discussed in the following subsection.

362

#### 363 4.2. Authoring and Executing a Composition

364         Authoring and executing a component composition is mediated by a configuration  
365 editor, which is a user interface application that allows the modeler to define linkages  
366 between components along with other composition parameters. The OpenMI  
367 Configuration Editor is a free and open source editor provided through the OpenMI  
368 Association Technical Committee (OATC). Model components are loaded into the  
369 Configuration Editor via an XML-based OMI file that references a compiled version of  
370 the component's source code in the form of a dynamic-link library. This OMI file acts as  
371 a link between the Configuration Editor and the component itself.

372         Using the OpenMI Configuration Editor, links must be established that define  
373 component to component data transfers. Link properties are edited in order to establish  
374 which output exchange item will be transferred across each link, and what input exchange  
375 item will they be paired with on the receiving end. This is necessary since an individual  
376 component can have multiple input and output exchange items. In the previously  
377 mentioned modeling example, a link must be established to connect the CN component to  
378 a component that can supply a precipitation exchange item. Then a second link must be  
379 established to supply the excess precipitation calculated by the CN component to a  
380 surface runoff component. Once all components are linked in the component  
381 composition, the simulation start and end times must be determined. By default, the  
382 OpenMI Configuration Editor selects the latest overlapping time for all components

383 within the model composition as the simulation end time. Simulation begin time is  
384 established as the earliest overlapping time contained in each component's time horizon.  
385 A component composition can be executed either from the Configuration Editor GUI or  
386 from a console using the command line application.

387

## 388 **5. Summary, Discussion, and Future Work**

389 Modeling of environmental systems is challenging in part because process  
390 interaction often spans several disciplines, making it difficult to model integrated system  
391 response. We argue that no single model can represent all aspects of an environmental  
392 system as accurately as a conglomerate of model components created and maintained by  
393 experts in each field. Specific processes within the hydrologic cycle, for example, can be  
394 linked together using component-based modeling, without having extensive knowledge  
395 of the inner workings of each computational module. Furthermore, componentization of  
396 environmental models also reduces repetitive code because it allows model developers to  
397 share process level components so that unique hydrologic models can be created with  
398 reusable components (Argent et al. 2006). While we believe this abstraction and code  
399 reuse are necessary attributes of a multidisciplinary modeling system, it is nonetheless  
400 important to also consider the increased risks of modelers applying models for which  
401 they do not fully understand the inner workings. Just as with any environmental model, it  
402 is critical for modelers to have an understanding of the theory behind the components'  
403 mathematical representation. Component-based modeling does not remove this basic  
404 need.

405           This paper outlines an approach for creating process-level OpenMI-compliant  
406 components called the Simple Model Wrapper (SMW). The SMW abstracts the model  
407 developer from the OpenMI standard by allowing them to specify component parameters  
408 in an XML-based configuration file, and component functionality within a simple class  
409 with three methods that specify pre-run, runtime, and post-run behavior. Furthermore, it  
410 allows modelers to more easily change component parameters, such as exchange items or  
411 time horizon, without the need to re-compile the component's source code, as shown in  
412 Section 3. The end result is the ability to rapidly prototype process-level components  
413 that can be used within OpenMI component compositions.

414           The primary motivation for creating the SMW is to reduce the complexity  
415 associated with creating new, process-level components within an OpenMI-based  
416 modeling framework. OpenMI is a powerful approach for component-based modeling,  
417 however because it was designed for wrapping large legacy models, it can be difficult for  
418 environmental modelers to use to create entirely new process-level components. The  
419 SMW is designed to overcome this limitation. However, because SMW was specifically  
420 designed to aid in the development of process-level components, it limits the control of a  
421 model developer compared with implementing the standard OpenMI interfaces.  
422 Therefore, it is not recommended for wrapping large legacy models, but instead is meant  
423 for the rapid prototyping of process-level model components.

424           It is unclear at this point what computational overhead is introduced by the SMW  
425 because only components with moderate computational demand have been developed and  
426 tested. To date, the size of model compositions using the SMW have been limited to a  
427 handful of components, although this is rarely the case when investigating real-world

428 problems. That said, negligible runtime differences have been observed thus far between  
429 components created using the SWM compared to components created just with the  
430 OpenMI IEngine interface. In order to evaluate the net overhead associated with the  
431 SMW, future research is needed to benchmark a variety of moderately to highly  
432 computationally intensive model compositions. Additionally, the size of the model  
433 compositions being tested, quantified by the number and complexity of components,  
434 must also be varied to determine if inefficiency results from using the SMW to mediate  
435 the communication between components. These two criteria present a large range of  
436 model compositions that must be benchmarked in order to evaluate the overall runtime  
437 efficiency of the SMW.

438         Future work will also focus on applying the SMW to a specific watershed  
439 modeling objective in order to better understand the implementation and computational  
440 aspects of its design. One of the aims of this work will be to understand how different  
441 component compositions, ranging in both level of detail and type of processes  
442 represented, impact predictive capability. Another aim will be to understand how  
443 component compositions can be calibrated when each component has its own internal  
444 parameterization. The end goal of this study will be to show how component modeling  
445 can be used to construct the most representative model, using the simplest process-level  
446 computations necessary for the question at hand. In doing so, we will show how the  
447 SMW can be used to create a library of modeling components that can be swapped in and  
448 out of the model compositions to identify optimal component compositions for specific  
449 environmental systems.

450

451 **Acknowledgments**

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

455

456 **Software Availability**

457 The Simple Model Wrapper source code is available under the MIT license from  
458 <http://code.google.com/p/smw>.

459

460 **References**

461 Ahuja, L. R., Ascough II, J. C., and David, O., 2005. Developing natural resource models  
462 using the object modeling system: feasibility and challenges. *Advances in*  
463 *Geosciences*, 4, 29-36.

464 Allan, B. A., Armstrong, R., Bernholdt, D. E., Bertrand, F., Chiu, K., Dahlgren, T. L.,  
465 Damevski, K., Elwasif, W. R., Epperly, T. G. W., Govindaraju, M., Katz, D. S.,  
466 Kohl, J. A., Krishnan, M., Kumfert, G., Larson, J. W., Lefantzi, S., Lewis, M. J.,  
467 Malony, A. D., McInnes, L. C., Nieplocha, J., Norris, B., Parker, S. G., Ray, J.,  
468 Shende, S., Windus, T. L., and Zhou, S. J., 2006. A component architecture for  
469 high-performance scientific computing. *International Journal of High*  
470 *Performance Computing Applications*, 20(2), 163-202.

471 Anderson, R., Dietrich, W., Furbish, D., Hanes, D., Howard, A., Paola,Chris, Pelletier, J.,  
472 Slingerland, R., Stallard, B., Syvitski, J., Vorosmarty, C., and Wiberg, P., 2004.  
473 *Community Surface Dynamics Modeling System: Science Plan*. <accessed on

474 June 23, 2009 [http://csdms.colorado.edu/mediawiki/images/CSDMS\\_Science](http://csdms.colorado.edu/mediawiki/images/CSDMS_Science)  
475 [\\_Plan\\_Aug04.pdf](#) >

476 Argent, R. M., 2004. An overview of model integration for environmental application -  
477 components, frameworks and semantics. *Environmental Modelling & Software*,  
478 19(3), 219-234.

479 Argent, R. M., Voinov, A., Maxwell, T., Cuddy, S. M., Rahman, J. M., Seaton, S.,  
480 Vertessy, R. A., and Braddock, R. D., 2006. Comparing modelling frameworks -  
481 A workshop approach. *Environmental Modelling & Software*, 21(7), 895-910.

482 Brinkman, R., Gregersen, J. B., Hummel, S., and Westen, S. J. P., 2005. The OpenMI  
483 Document Series: Part C - the org.OpenMI.Standard interface specification.  
484 HarmonIT, 1-79. <accessed on June 23, 2009 [http://www.harmonit.org/docs/](http://www.harmonit.org/docs/partcorg.openmi.standardspecification.pdf)  
485 [partcorg.openmi.standardspecification.pdf](#)>

486 Chow, V. T., Maidment, D. R., and Mays, L. W., 1988. *Applied Hydrology*, McGraw-  
487 Hill Inc, Boston, Massachusetts.

488 Collins, N., Theurich, G., DeLuca, C., Suarez, M., Trayanov, A., Balaji, V., Li, P., Yang,  
489 W. Y., Hill, C., and da Silva, A., 2005. Design and implementation of  
490 components in the Earth system modeling framework. *International Journal of*  
491 *High Performance Computing Applications*, 19(3), 341-350.

492 Gijssbers, P., Gregersen, J., Blind, M., Westen, S., Dirksen, F., and Gavardinas, C., 2005.  
493 OpenMI Document Series: Part B - Guidelines for the OpenMI (version 1.0).  
494 HarmonIT, 1-237. <accessed on June 23, 2009 [http://www.harmonit.org/docs/](http://www.harmonit.org/docs/partcorg.openmi.standardspecification.pdf)  
495 [partcorg.openmi.standardspecification.pdf](#)>

496 Gregersen, J. B., Gijsbers, P. J. A., and Westen, S. J. P., 2007. OpenMI: Open modelling  
497 interface. *Journal of Hydroinformatics*, 9(3), 175-191.

498 Hill, C., DeLuca, C., Balaji, Suarez, M., and da Silva, A. 2004. The architecture of the  
499 earth system modeling framework. *Computing in Science & Engineering*, 6(1),  
500 18-28.

501 Kennen, J. G., Kauffman, L. J., Ayers, M. A., Wolock, D. M., and Colarullo, S. J., 2008.  
502 Use of an integrated flow model to estimate ecologically relevant hydrologic  
503 characteristics at stream biomonitoring sites. *Ecological Modelling*, 211(1-2), 57-  
504 76.

505 Kralisch, S., Krause, P., and David, O., 2005. Using the object modeling system for  
506 hydrological model development and application. *Advances in Geosciences*, 75-  
507 81.

508 Moore, R., Gijsbers, P., Fortune, D., Gregersen, J., and Blind, M. 2005. The OpenMI  
509 Document Series: Part A - Scope for the OpenMI (version 1.0). HarmonIT, 1-17.  
510 <accessed on June 23, 2009 [http://www.harmonit.org/docs/partaorg.openmi.  
standardspecification.pdf](http://www.harmonit.org/docs/partaorg.openmi.<br/>511 standardspecification.pdf)>

512 Moore, R. V., and Tindall, C. I., 2005. An overview of the open modelling interface and  
513 environment (the OpenMI). *Environmental Science & Policy*, 8(3), 279-286.

514 Scholten, H., Kassahun, A., Refsgaard, J. C., Kargas, T., Gavardinas, C., and Beulens, A.  
515 J. M., 2007. A methodology to support multidisciplinary model-based water  
516 management. *Environmental Modelling & Software*, 22(5), 743-759.

517 Slingerland, R., Murray, B., Wiberg, P., Tucker, G., Sun, T., Campbell, K., Pratson, L.,  
518 Syvitski, J., and Peckham, S., 2008. Community Surface Dynamics Modeling

519 System: Annual Report to the National Science Foundation. <accessed on June  
520 23, 2009 [http://csdms.colorado.edu/mediawiki/images/2008CSDMS\\_](http://csdms.colorado.edu/mediawiki/images/2008CSDMS_)  
521 [AnnualReport.pdf](http://csdms.colorado.edu/mediawiki/images/2008CSDMS_AnnualReport.pdf)>

522 Sophocleous, M., and Perkins, S. P., 2000. Methodology and application of combined  
523 watershed and ground-water models in Kansas. *Journal of Hydrology*, 236(3-4),  
524 185-201.

525 Syvitski, J., Paola, C., Slingerland, R., Furbish, D., Wiberg, P., and Tucker, G., 2004.  
526 Building a Community Surface Dynamics Modeling Sytem: Rationale and  
527 Strategy. <accessed on June 23, 2009 [http://csdms.colorado.edu/mediawiki/](http://csdms.colorado.edu/mediawiki/images/CSDMS_Rational_and_Strategy_Apr04.pdf)  
528 [images/CSDMS\\_Rational\\_and\\_Strategy\\_Apr04.pdf](http://csdms.colorado.edu/mediawiki/images/CSDMS_Rational_and_Strategy_Apr04.pdf) >

529



530

531 **Figure 1:** The fundamental OpenMI concepts shown within a system of three

532 components

533 **Figure 2:** The abstraction of the Simple Model Wrapper from the OpenMI Standard

534 Modeling interfaces.

535 **Figure 3:** Graphical representation of the configuration file showing the relationships

536 between parent and child elements in the XML schema.

537 **Figure 4:** The role of the Simple Model Wrapper within a component composition

538 containing four components

539 **Figure 5:** Model composition showing the exchange of boundary conditions between

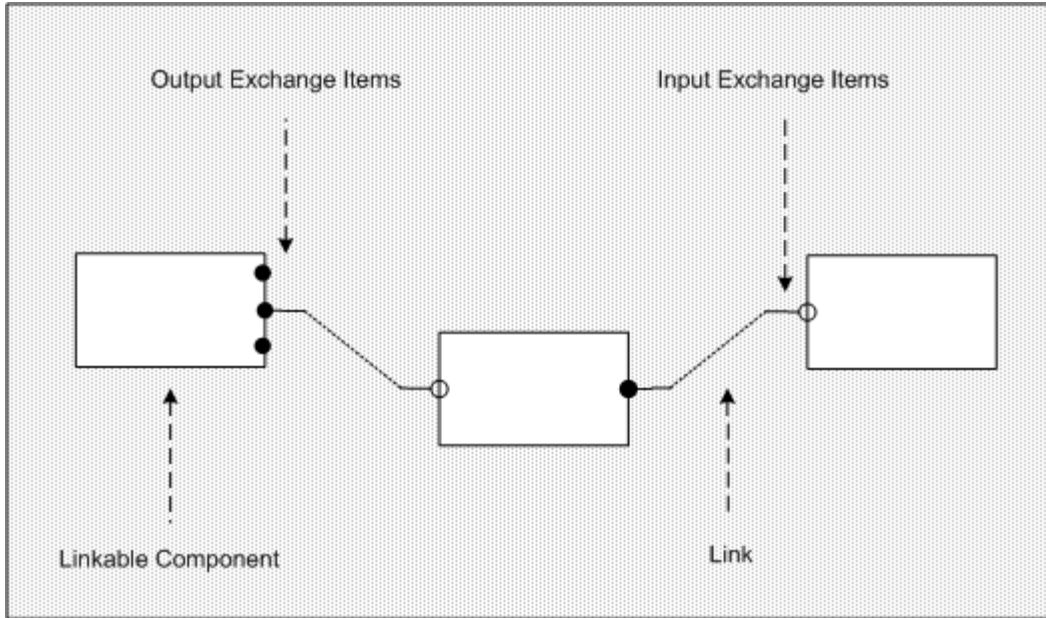
540 hydrological components.

541 **Figure 6:** Algorithm for the computation of infiltration using the CN method,

542 implemented within the *PerformTimeStep* method.

543

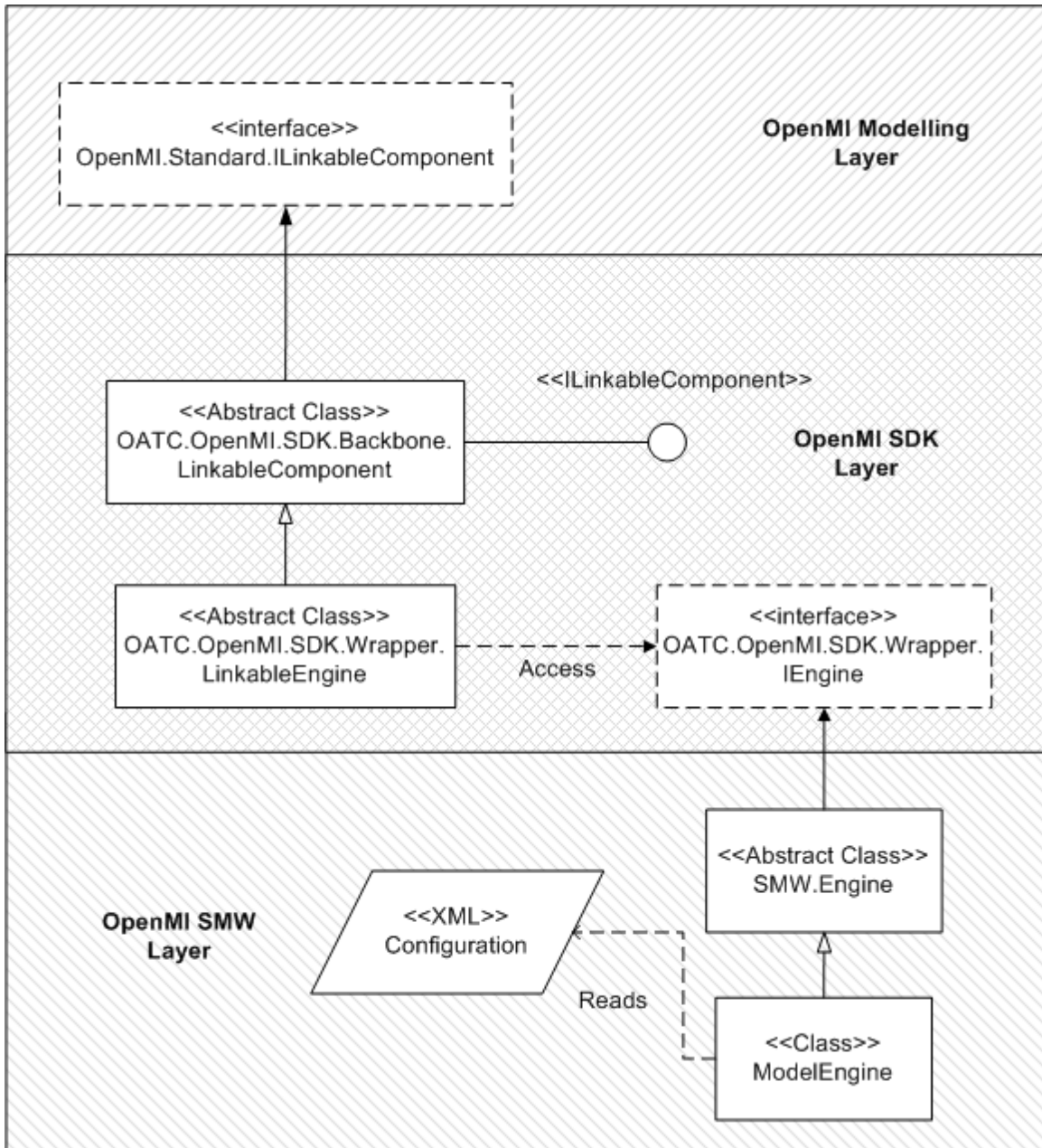
544 Figure 1:  
545



546  
547

548  
549  
550

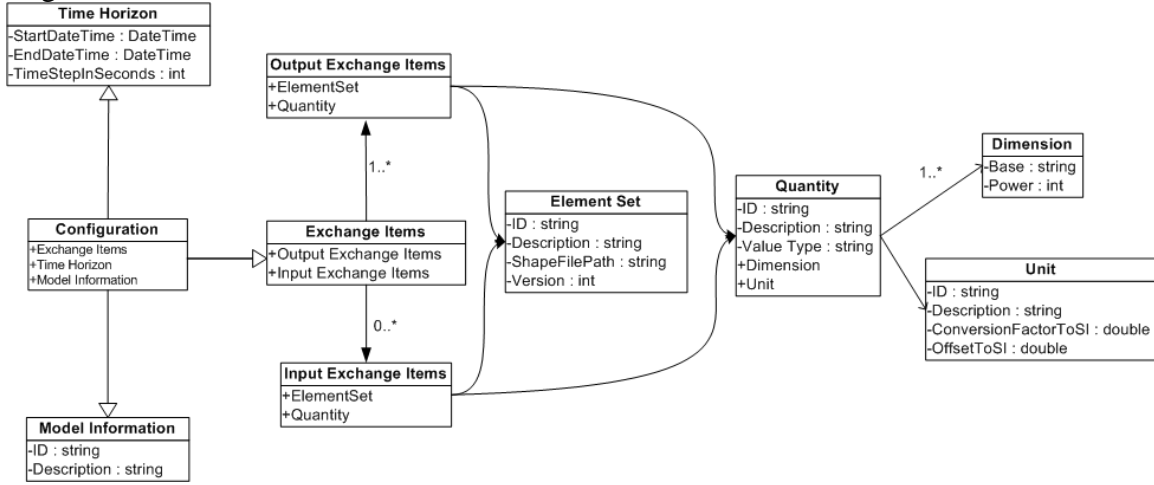
Figure 2:



551

552  
553

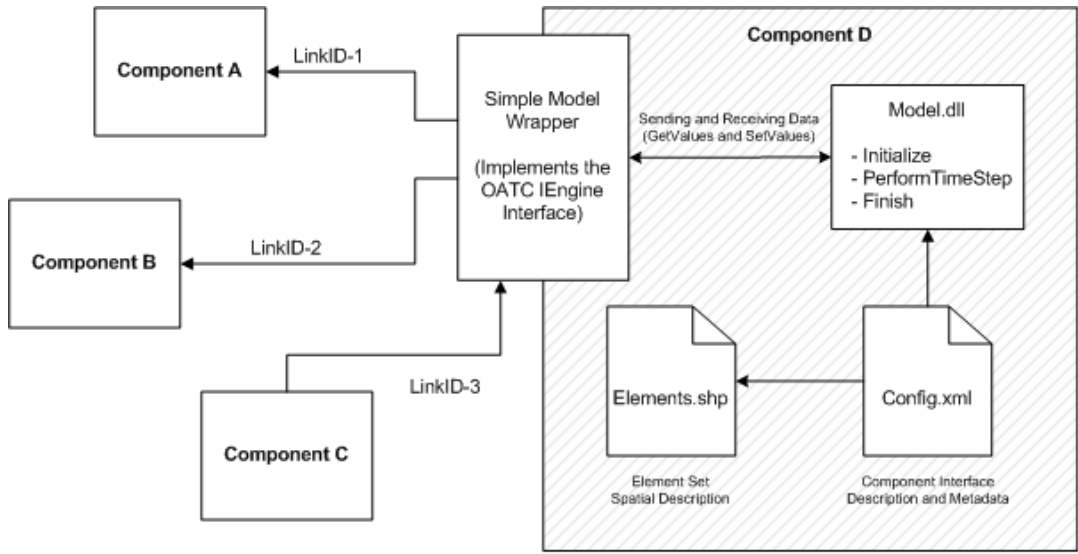
Figure 3:



554  
555  
556

557  
558

Figure 4:

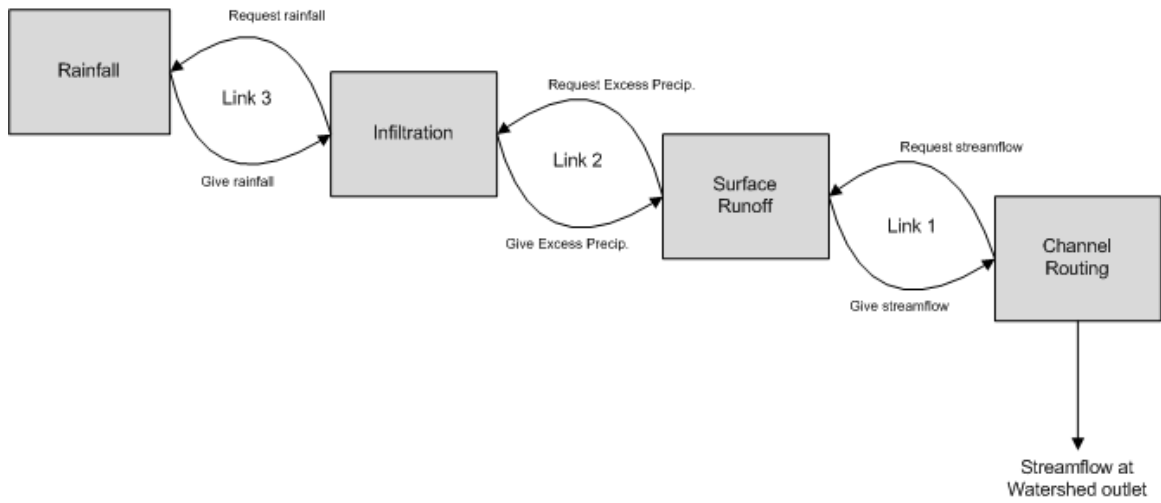


A process-level component built using the Simple Model Wrapper

559  
560

561  
562  
563

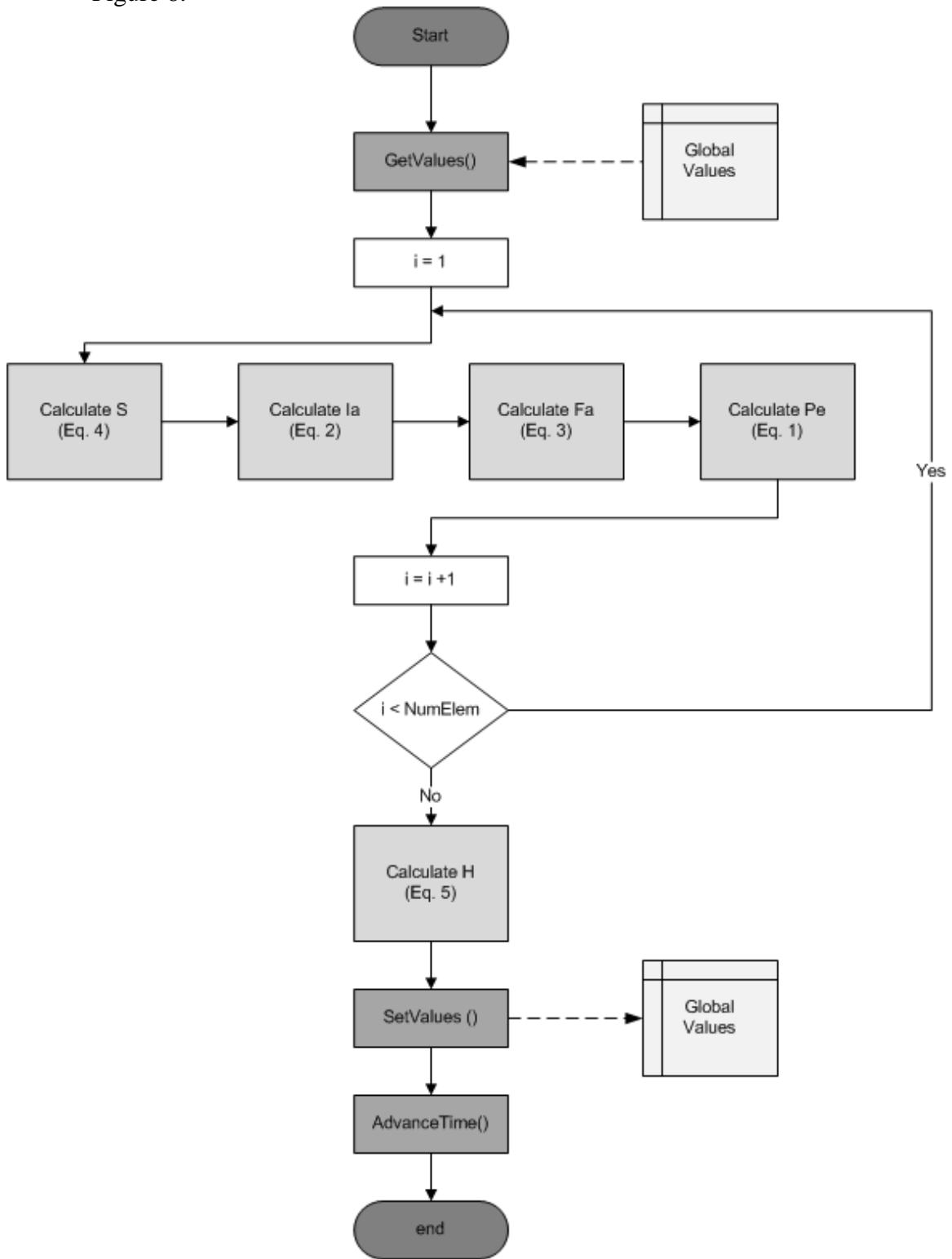
Figure 5:



564  
565

566  
567

Figure 6:



568  
569