A Generic Approach for Developing Process-Level Hydrologic Modeling Components

Anthony M. Castronova[1] and Jonathan L. Goodall[2]

Abstract

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

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

---

[1] 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

[2] 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@ecec.sc.edu

1. Introduction

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

Component-based modeling offers an alternative approach for constructing hydrologic models that emphasize the decomposition of a system into functional components that communicate via standard interfaces (Argent 2004). In a component-based modeling paradigm, processes are designed as computational units that can plug-and-play with other computational units in a modeling system (Allan et al. 2006). Each component must have a standard means for communicating with other components within the modeling system so that the overall modeling system can be re-configured through the introduction of new components or the “swapping out” of existing ones. The model developer is able to define the overall composition of model components using unique system representations, which is ideal for modeling complex systems that involve multi-scale and multidisciplinary components (Kennen et al. 2008).

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

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

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

2. Background

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

The Community Surface Dynamics Modeling System (CSDMS) is a modeling environment built from free software modules that focuses on the prediction of sediment and material transport over various time and space scales (Syvitski et al. 2004). The objective is to encourage interdisciplinary modeling by coupling components and enabling developers to create models specific to individual studies. The model developer interacts with three architectural elements when building models in CSDMS: Standard Utilities, Modules, and a Toolkit (Syvitski et al. 2004). Standard utilities define data structure, graphics rendering, module connectors, and a web interface (Anderson et al. 2004). Module components are developed by modelers and represent the actual sedimentary computations. The Toolkit is supplied to the modelers in order to aid in module development, and consists of numerical solvers, grid generators, and automatic code generators. The CSDMS uses the Open Modeling Interface (OpenMI) standard to maintain runtime communication, and the Common Component Architecture (CCA) to facilitate high performance computing (Slingerland et al. 2008).

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

The Object Modeling System (OMS) is developed by a collaborative effort between the United States Department of Agriculture- Agriculture Research Service (USDA-ARS) and partner agencies to resolve the lack of integrated hydrological models (Kralisch et al. 2005). It uses modern software principals, particularly object-oriented programming, to provide modelers with the ability to customize their models by combining individually compiled processes together into a modeling system (Kralisch et al. 2005). An OMS model consists of multiple, independent modules that are coupled by a standard software interface. These modules are further divided into system and scientific components, which represent model building tools and physically-based computations, respectively. The scientific component of a module represents the physical process that is being calculated, whereas system components are used to manage the coupling and execution of several scientific components. Models are built from a combination of “system” and “scientific” components, and are joined together within the Netbeans (http://www.netbeans.org) runtime environment (Ahuja et al. 2005).

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

The OpenMI communication protocol consists of three fundamental concepts: a linkable component, an exchange item, and a link (Figure 1) (Brinkman et al. 2005). A linkable component is an object that implements an OpenMI standard interface (e.g. ILinkableComponent). Exchange items are objects communicated between components and are comprised of an element set and a quantity. An element set defines the geospatial objects for an exchange item whereas a quantity describes the variable for an exchange item including its units and unit dimensions (Gregersen et al. 2007). Components can wrap databases, models, visualization routines, or any other computational resource (Gijsbers et al. 2005). Links are used to couple components together and define the source and destination for exchange items transferred between them. A single component can have multiple input and output exchange items. An interlinked set of components is called a component composition and can be thought of as a "model" for a specific system.

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

3. The Simple Model Wrapper

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

3.1. <u>The Configuration File</u>

To abstract the modeler from manually defining component properties within the OpenMI IEngine interface, the SMW has been designed to read this information from an external configuration file with a specific XML schema (Figure 3). The XML schema for defining the component properties closely follows the OpenMI object model for defining exchange items, the model description, and time horizons. One SWM component must be associated with one configuration file. The configuration file must include one or more output exchange item elements and zero or more input exchange item elements. Each exchange item element must include one element set child element and one quantity child element. Since component properties may frequently change when prototyping process-level components, there is an advantage to extracting these properties from the source code so that components do not have to be recompiled after changes to component metadata. The configuration file is loaded into memory during the component initialization phase to populate the component's properties.

3.2. Method Description

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

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

3.3. Component Composition

A model composition defines an interlinked set of components and the exchange of information between them (Figure 4). A detailed view of a component implementing the SMW is represented in the shaded region of Figure 4. In this case, three files are associated with the component: model.dll, config.xml, and elements.shp. The model.dll file contains a class that performs a process-level computation using the SMW approach and has been compiled into a dynamic-link library (DLL). The config.xml file is the configuration file that supplies the component metadata discussed in the previous subsection. Finally, a shapefile is used to spatially define the component's modeling units and parameters. However, components are not required to use a shapefile for this purpose; other file formats could be used instead. For example, in the case where it is not critical to describe the geometries associated with element sets (e.g. if the elements are points or non-spatial), a simple ASCII text file might be preferred to a shapefile. In this case, the component developer can write code that reads data from the ASCII file instead of a shapefile when constructing OpenMI exchange item objects..

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

4. Case Study

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

For the example shown in Figure 5, the rainfall component provides precipitation values as an output exchange item by reading local data files to populate the exchange item objects. This demonstrates how components are not limited to process routines, but can be file readers, visualization tools, or other functional tools. This precipitation exchange item serves as input for the infiltration component, which is then able to calculate excess precipitation values for each modeling unit (i.e. subwatershed) in the watershed. The excess precipitation values are then supplied as input to the surface runoff component, which calculates the runoff hydrograph that serves as input for the channel routing component. Finally, the channel routing component uses the runoff hydrographs from the subwatersheds to estimate a streamflow hydrograph at the watershed outlet.

4.1. Component Design

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

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

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

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

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

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

$$I_a = 0.2S \qquad (2)$$

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

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

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

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

$$H(t) = P_e(t) - P_e(t-1) \qquad (5)$$

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

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

Repeating these basic steps for each of the other three components of the model composition mentioned in the previous section would allow one to build a component composition to model rainfall-runoff. Because this paper focuses on the design and implementation of the SMW, a full modeling example is beyond the scope and will be part of future work discussed in the final section of this paper. The main difference between the developments strategies of various components needed to complete the model composition is the algorithms used for producing each component's output exchange items. The precipitation component, unlike the other components, would simply use a file to populate its output exchange item. The process of linking the components into a composition is discussed in the following subsection.

4.2. Authoring and Executing a Composition

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

Using the OpenMI Configuration Editor, links must be established that define component to component data transfers. Link properties are edited in order to establish which output exchange item will be transferred across each link, and what input exchange item will they be paired with on the receiving end. This is necessary since an individual component can have multiple input and output exchange items. In the previously mentioned modeling example, a link must be established to connect the CN component to a component that can supply a precipitation exchange item. Then a second link must be established to supply the excess precipitation calculated by the CN component to a surface runoff component. Once all components are linked in the component composition, the simulation start and end times must be determined. By default, the OpenMI Configuration Editor selects the latest overlapping time for all components within the model composition as the simulation end time. Simulation begin time is established as the earliest overlapping time contained in each component's time horizon. A component composition can be executed either from the Configuration Editor GUI or from a console using the command line application.

5. Summary, Discussion, and Future Work

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

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

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

It is unclear at this point what computational overhead is introduced by the SMW because only components with moderate computational demand have been developed and tested. To date, the size of model compositions using the SMW have been limited to a handful of components, although this is rarely the case when investigating real-world problems. That said, negligible runtime differences have been observed thus far between components created using the SWM compared to components created just with the OpenMI IEngine interface. In order to evaluate the net overhead associated with the SMW, future research is needed to benchmark a variety of moderately to highly computationally intensive model compositions. Additionally, the size of the model compositions being tested, quantified by the number and complexity of components, must also be varied to determine if inefficiency results from using the SMW to mediate the communication between components. These two criteria present a large range of model compositions that must be benchmarked in order to evaluate the overall runtime efficiency of the SMW.

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

Acknowledgments

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

Software Availability

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

References

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

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

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

June 23, 2009 http://csdms.colorado.edu/mediawiki/images/CSDMS_Science _Plan_Aug04.pdf >

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

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

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

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

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

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

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

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

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

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

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

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

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

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

System: Annual Report to the National Science Foundation. <accessed on June 23, 2009 http://csdms.colorado.edu/mediawiki/images/2008CSDMS_AnnualReport.pdf>

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

Syvitski, J., Paola, C., Slingerland, R., Furbish, D., Wiberg, P., and Tucker, G., 2004. Building a Community Surface Dynamics Modeling Sytem: Rationale and Strategy. <accessed on June 23, 2009 http://csdms.colorado.edu/mediawiki/images/CSDMS_Rational_and_Strategy_Apr04.pdf >

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

Figure 2: The abstraction of the Simple Model Wrapper from the OpenMI Standard Modeling interfaces.

Figure 3: Graphical representation of the configuration file showing the relationships between parent and child elements in the XML schema.

Figure 4: The role of the Simple Model Wrapper within a component composition containing four components

Figure 5: Model composition showing the exchange of boundary conditions between hydrological components.

Figure 6: Algorithm for the computation of infiltration using the CN method, implemented within the *PerformTimeStep* method.

Figure 1:

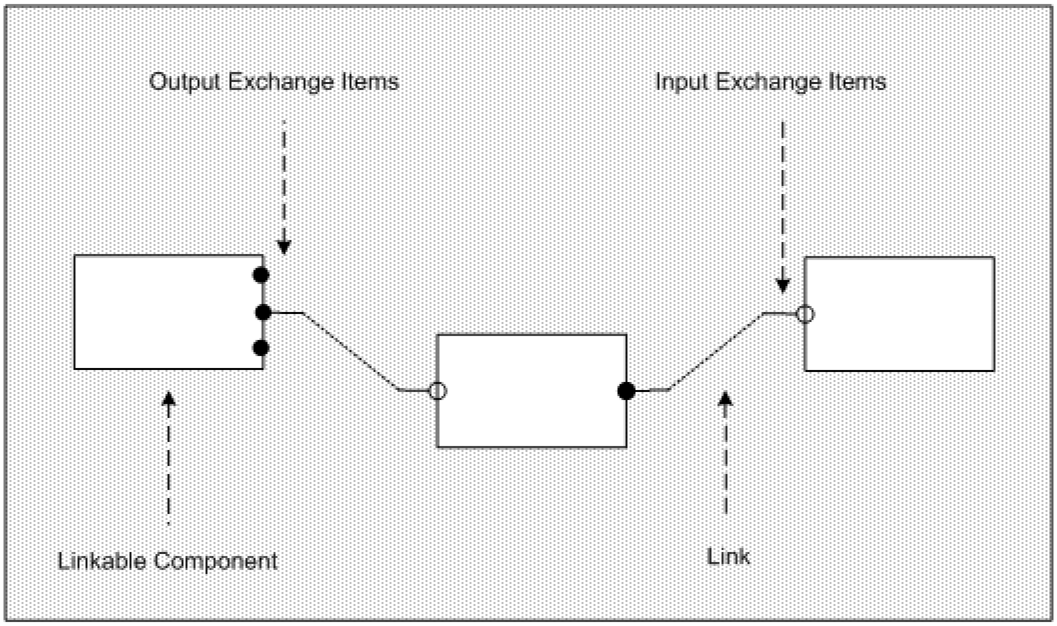
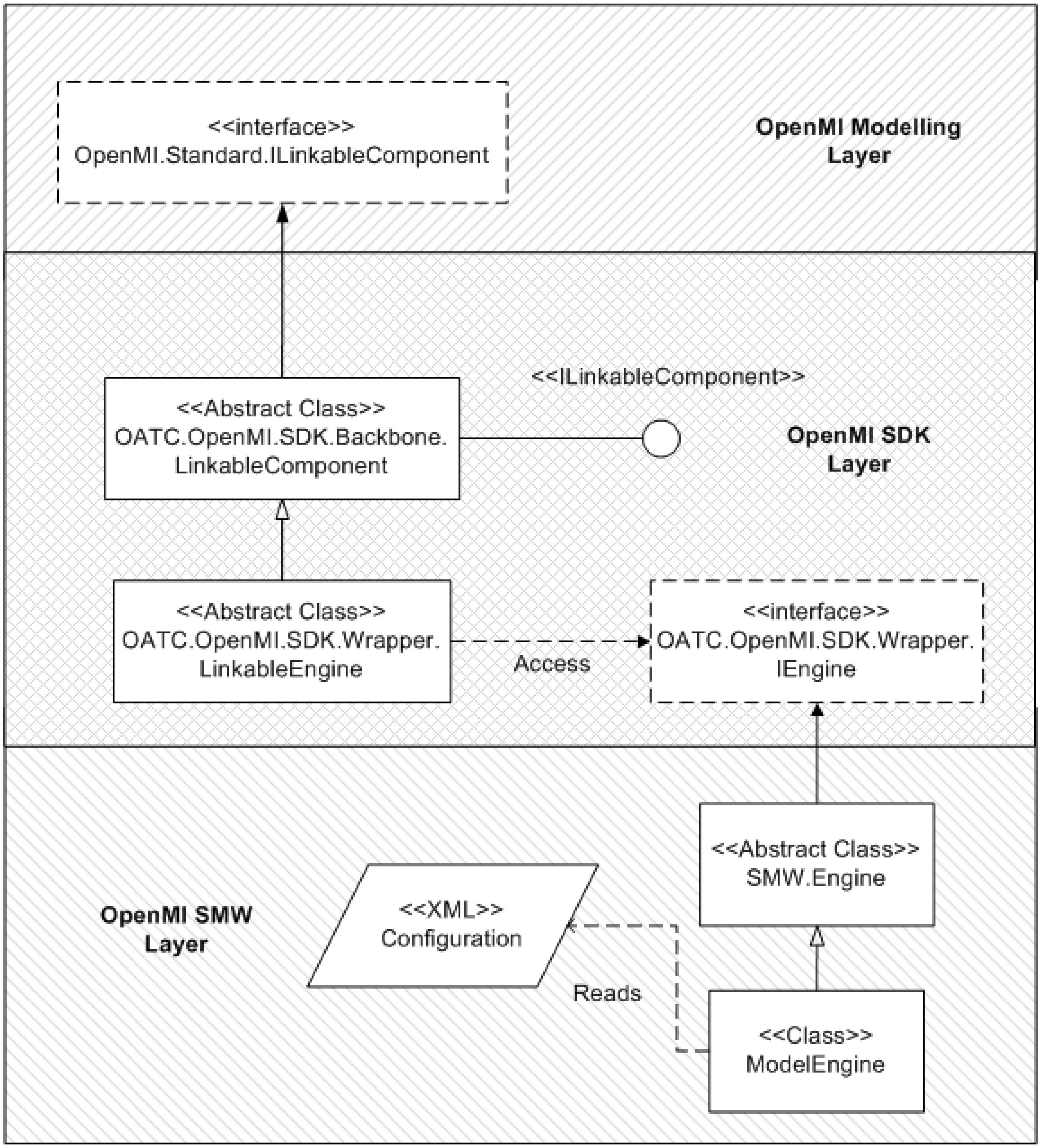

Output Exchange Items
Input Exchange Items
Linkable Component
Link

Figure 2:



<<interface>>
OpenMI.Standard.ILinkableComponent
OpenMI Modelling Layer
<<ILinkableComponent>>
<<Abstract Class>>
OATC.OpenMI.SDK.Backbone.LinkableComponent
OpenMI SDK Layer
<<Abstract Class>>
OATC.OpenMI.SDK.Wrapper.LinkableEngine
Access
<<interface>>
OATC.OpenMI.SDK.Wrapper.IEngine
<<Abstract Class>>
SMW.Engine
OpenMI SMW Layer
<<XML>>
Configuration
Reads
<<Class>>
ModelEngine

Figure 3:

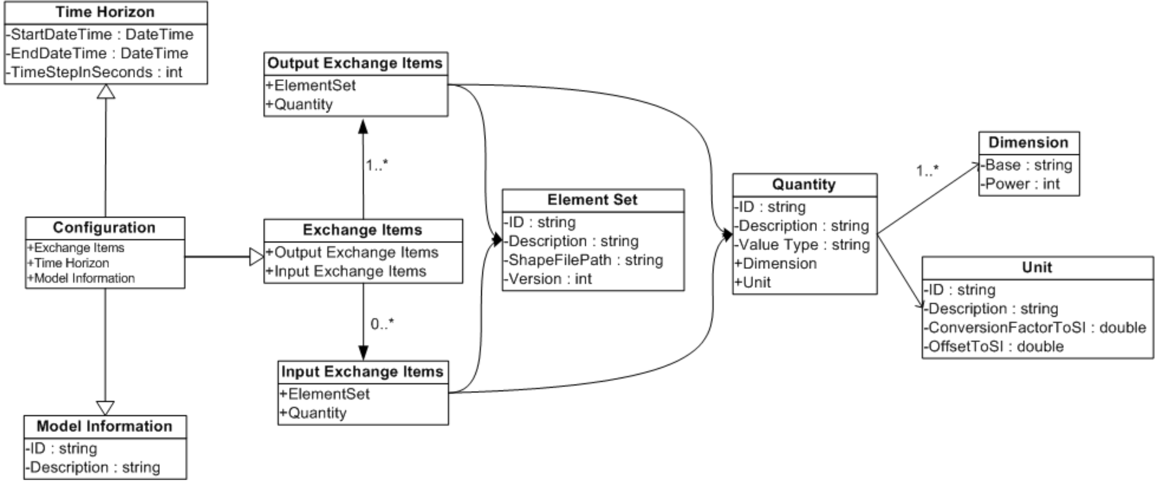
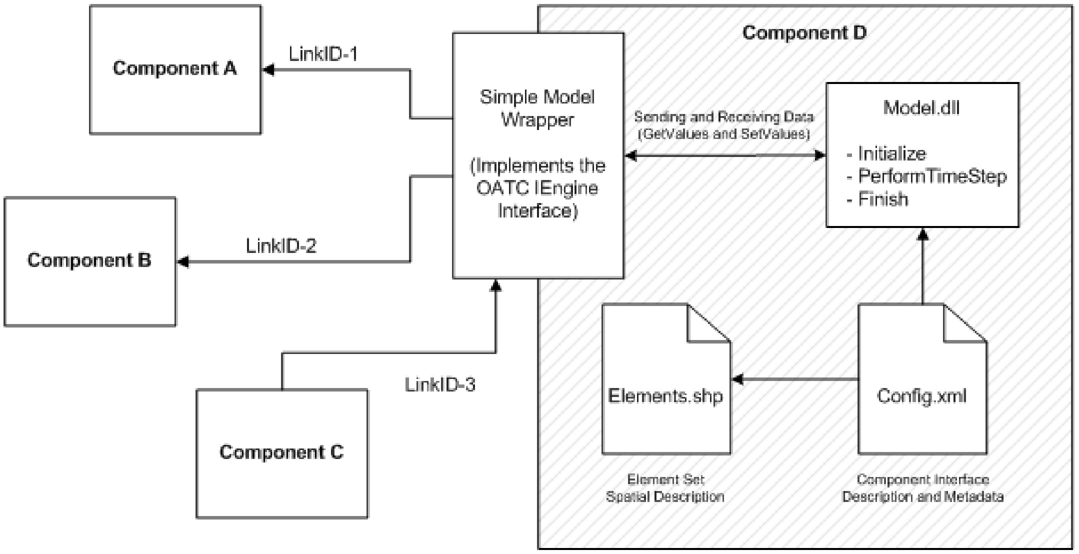

Time Horizon
-StartDateTime : DateTime
-EndDateTime : DateTime
-TimeStepInSeconds : int
Output Exchange Items
+ElementSet
+Quantity
1..*
Configuration
+Exchange Items
+Time Horizon
+Model Information
Exchange Items
+Output Exchange Items
+Input Exchange Items
0..*
Input Exchange Items
+ElementSet
+Quantity
Model Information
-ID : string
-Description : string
Element Set
-ID : string
-Description : string
-ShapeFilePath : string
-Version : int
Quantity
-ID : string
-Description : string
-Value Type : string
+Dimension
+Unit
1..*
Dimension
-Base : string
-Power : int
Unit
-ID : string
-Description : string
-ConversionFactorToSI : double
-OffsetToSI : double

Figure 4:



A process-level component built using the Simple Model Wrapper

Figure 5:

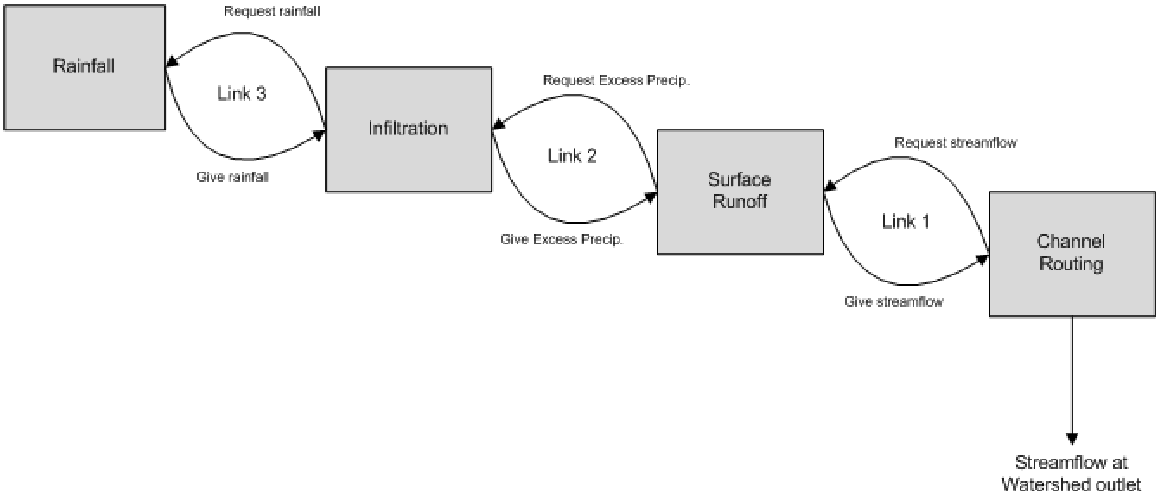
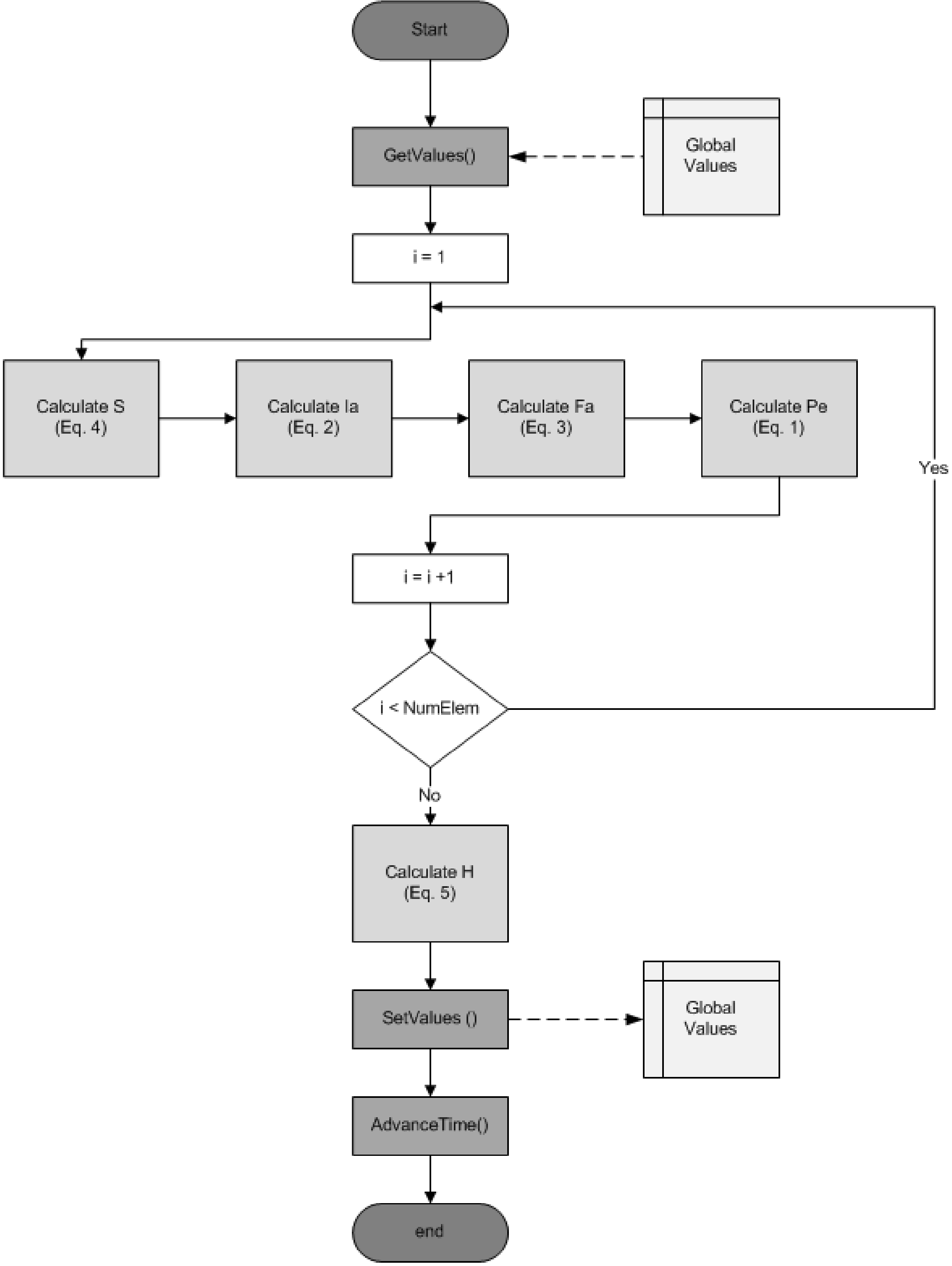

Rainfall
Request rainfall
Link 3
Give rainfall
Infiltration
Request Excess Precip.
Link 2
Give Excess Precip.
Surface Runoff
Request streamflow
Link 1
Give streamflow
Channel Routing
Streamflow at Watershed outlet

Figure 6:


Start
GetValues()
Global Values
i = 1
Calculate S (Eq. 4)
Calculate Ia (Eq. 2)
Calculate Fa (Eq. 3)
Calculate Pe (Eq. 1)
Yes
i = i +1
i < NumElem
No
Calculate H (Eq. 5)
SetValues ()
Global Values
AdvanceTime()
end