A COMMON PROGRAMMING FRAMEWORK FOR DISTRIBUTED HYDROLOGIC MODELING RESEARCH: AN OVERVIEW OF THE ARCHITECTURE

Zhengtao Cui, Hydrology Laboratory, Office of Hydrologic Development, National Weather Service, NOAA, Silver Spring, MD, Zhengtao.Cui@noaa.gov;
Victor Koren, Hydrology Laboratory, Office of Hydrologic Development, National Weather Service, NOAA, Silver Spring, MD, Victor.Koren@noaa.gov;
Fekadu Moreda, Hydrology Laboratory, Office of Hydrologic Development, National Weather Service, NOAA, Silver Spring, MD, Fekadu.Moreda@noaa.gov;
Michael Smith, Hydrology Laboratory, Office of Hydrologic Development, National Weather Service, NOAA, Silver Spring, MD, Michael.Smith@noaa.gov;

Abstract

It is a common task for distributed hydrologic modelers to enhance an existing model by adding new components/models or modifying existing components/models. In addition, to test the enhanced model, scientists normally need to 1) run the model with various configurations and, 2) input/output various sets of variables for better understanding. One caution is that models can easily become unmanageable as they are enhanced, leading to solutions that are very expensive to develop and difficult to maintain and extend. This presentation introduces the architectural features of a manageable programming framework designed to facilitate distributed hydrologic modeling research.

A programming framework typically is a hierarchy of node classes and functions that provides services to a theory of the problem domain. It is developed when many applications are going to be developed within a specific problem domain. Application programmers extend the framework by providing leaf classes in the hierarchy and reuse the services provided by the framework. Therefore, both design and code are reused to avoid reinventing the wheel, yet the specific actions should be supplied by the application programmer to solve his/her particular problem. The common programming framework for distributed hydrologic modeling research introduced here is an evolutionary step in the research and development program of the Hydrology Laboratory.

Most commonly available systems use the object-oriented approach to predefined a set of data abstractions (abstract classes). Typically, these systems are then extended via inheritance of these predefined data abstractions. One drawback of this approach is that the extensions are constrained to the predefined set of data abstractions. This paper presents our approach which combines object-oriented design and generic programming techniques to ensure that the framework is modular and scalable. In addition to inheritance, the resulting framework can be extended by templates as well, adding one more degree of freedom when extending the framework. Our approach affords other benefits such as type-safe and performance gains. Other architectural features such as using graph theory to manage distributed data sets, using the object factory design pattern to select simulations at runtime, and the C and FORTRAN programming interface are also discussed. The framework has been implemented in C++ programming language.
The framework is designed to be modular. Each module contains one or more classes. The modules are organized into layers. Upper layers depend on lower layers. Lower layers do not know the existence of upper layers. Modules in the same layer are independent of each other.

Among various classes defined in the modules, there are three interactive core classes: the \textit{HydroDomain}, the \textit{Model}, and the \textit{ModelObjectFactory}. The class \textit{HydroDomain} manages the physical geographic area on which hydrologic simulation will be performed. A \textit{HydroDomain} is a watershed consisting of cells. There are two kinds of \textit{HydroDomains}: connected and unconnected. In a connected domain, a cell is linked to one of its neighboring cells by its flow direction. An unconnected \textit{domain} is the same as a connected \textit{domain} except that the flow connectivity of the grid cells is either unknown or irrelevant. Although rainfall excess and overland flow cannot be routed from cell to cell in un-connected domains, these quantities are nonetheless informative from a water balance standpoint. The \textit{HydroDomain} concept has been implemented using graph theory to solve problems such as visiting each cell in a particular order. The \textit{HydroDomain} is also implemented as a class template. One of its template parameters specifies cell properties. Thus, cell properties could be customized by developers according to their model’s particular needs. For example, a developer might extend the framework by defining triangle cells or sub-basins with irregular shape, etc. to fit a particular modeling requirement.

The class \textit{Model} is designed as an abstract base class template. It acts as the common interface for various concrete hydrologic models. The class \textit{Model} can be extended by both inheritance and by template. The framework uses the object factory design pattern to create model objects at runtime for user’s selection.

The \textit{ModelObjectFactory} class was designed to create instances of various \textit{Model} classes at run-time without modifying existing code when adding new models or components. Because programming languages such as C++ are strongly typed, before an object can be created, its type has to be known. However, in this design, which \textit{Model} to run is selected by users at run-time. What users know is only the model name, normally a string that identifies the \textit{Model} class, not the \textit{Model} class itself designed by the programmer. Using ‘switch’-like statements will not work because the code has to be modified for each newly added model. The \textit{ModelObjectFactory} class delays the creation of class instances until run-time by knowing only the type info such as a string.

The framework also contains C and FORTRAN programming interfaces. We recognize that many scientists are not proficient in object oriented programming and many existing codes were written in C or FORTRAN. Therefore C and FORTRAN programmers can work within this framework without switching to C++ and codes written in C or FORTRAN can be easily incorporated. The investment in C and FORTRAN is protected.

Although the framework was designed for distributed hydrologic research, its design philosophy and data structures described here could help develop an operational model too. For example, the \textit{HydroDomain} class template could also be an underlying structure of an operational model to take advantages of graph theory. The object factory design pattern, \textit{ModelObjectFactory}, can be used to select models at run-time. From the point of view of
maintenance, the object-oriented design and generic programming techniques produce a modular and type-safe framework. Modular means errors are localized and type-safe means errors are caught at compilation-time instead of run-time.

When working within this framework, modelers are afforded the following advantages: 

a) new models can be added with minimal effort,  
b) model data are managed by a graph object, therefore various graph algorithms can be applied to solve problems such as visiting cells within a watershed in a particular order,  
c) a subset of models can be chosen to run in a simulation,  
d) a C and FORTRAN programming interface is provided to protect a modeler’s investment in these languages,  
e) the resulting model is computationally efficient, and  
f) the framework has predefined simulation algorithms therefore no additional programming is needed to input/output grid and time series data after a new model is added.