ACM SIGPLAN Language Tips

C++ Toolbox

Editor: *G. Bowden Wise, Dept. of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180; wiseb@cs.rpi.edu*

An Object Oriented Framework for Creating Models in Hydrology

Knut Alfredsen and Bjørn Sæther

Computer simulation programs have a wide use in hydrologic and hydraulic applications. We have developed an object-oriented framework that serves as a basis for describing the hydrological system and its processes, and used this as a basis for development of applications. A special consideration is placed on easy reuse and support for further development by the user, thereby solving some of the problems found in existing tools.

The hydrologic system is strongly dynamic, the states are frequently updated as the environment changes. A model of this dynamic behaviour imposes special requirement on the data handling and storage routines regarding speed and efficiency of use.

The framework consists of 4 main parts:

- Structure elements and system topology description, describing the real world hydrological components like lakes and rivers and the transport processes between them.
- Computational methods, describing the processes taking place in the structure elements.
- Components for data transport, storage and administration.
- Simulation control and time handling.

1 Structure Elements and Topology

The structure elements describes the components found in the real hydrological system, describing their internal structure and the relations to the other parts of the system. They are implemented as classes inheriting from a common abstract base class. This approach has two main objectives. It allows for the insertion of different component types into a container, and it provides a foundation for the addition of new components by the users of the framework. A small set of functions in the base class are made virtual and have to be reimplemented depending upon the type of component being derived. The base class also contains the basic functions to describe the transport processes in the system and the connectivity needed to describe downstream and upstream relations. Figure 1 illustrates a part of the structure hierarchy.

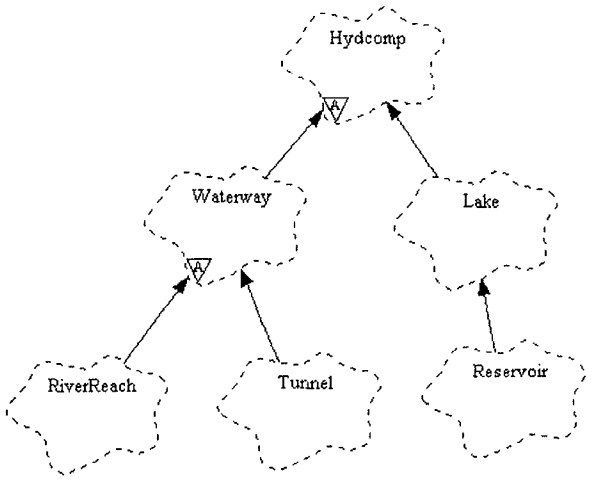

Figure 1: Part of the structure hierarchy of the framework. Notation after Booch [1].

Each of the structure elements contains data that describe the physical structure of the component (like the bathymetry of a lake) and data that describe THE states of the element (like the level of the water surface in the lake). All data are derived from a common base class to facilitate list insertion of objects of different type, they are later converted back to the original type using embedded type information and a type request function. In the future this conversion will be done using Run Time Type Identification (RTTI) when this is available.

Each structural class has two lists, one storing the state information and one storing physical data. The insertion

method has the necessary logic to prevent illegal data objects from being connected to the structural object, it would for instance prevent a lake's bathymetric information from being connected to a tunnel object. The information about which data type can be stored in which component is stored in a global, static object, that is accessible from all structural objects. By using this strategy, there is no need for lists of legal data inside the structural classes, and when new data is defined it is not necessary to update the structural elements. This allows users of the system to define new types of data for their specific needs without accessing the implementation of the structural elements.

To facilitate the construction of structural elements and the necessary data objects, a structure factory has been developed. The factory is based on the abstract factory pattern described by Gamma et.al. [2]. The user of the system can use the factory to request the structure element wanted, and the factory will generate both the structure element and the data objects that the structure element must contain.

In practice, the rivers system used in the model will consist of several structure elements ordered together in a network structure. To store and control the network, a topology network container has been developed. Traversal of the topology is done through a specialised iterator class. Traversal is usually done by starting at the downstream element and requesting input from the upstream neighbours. If the upstream element is also dependent of its upstream elements, the request is moved up one level and repeated. This goes on until the upstream element has no dependencies, then calculation is performed and the results are transferred downstream. This is normally done for each time step in the simulation. To speed up the process the iterator will find the simulation sequence of the topology during the initialisation, and store this in a list for use by the simulation control system. The actual simulation sequence then traverses the list for each time step, activating the calculations for each element in the list.

After each calculation the internal states are updated and data marked for transfer are sent to the downstream node. This process is done by marking one or more of the items in the list of states as transferable. Each of the transferable objects are then sent to the downstream element when the computations are finished. In a practical application an output state in the upstream element will be transferred to the input state in the downstream element. The mapping of states from one element to the next is controlled by mapping structure, which stores the corresponding states. The transfer process are controlled from the run control unit, and it can be changed if special computational requirements arises.

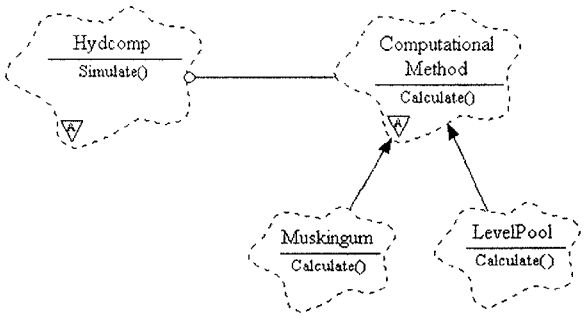

Figure 2: Separation of computational methods and structure.

2 Computational Methods

The choice of which computational method to use is dependent on what to analyse, the detail needed and the availability of data. One of the demands put on the design was that it should give the user several computational methods to select from, and that it should be possible for users to add their own methods without changing the structure elements. A solution with the computational methods as class methods in the structure elements was unusable, so we developed a link between a separate hierarchy of computational methods and the structural hierarchy. Using this solution, the user can add the necessary methods to a list of computational methods found in each structure element. The necessary datalinks are defined in the base class of the computational hierarchy, thereby isolating the users from the details in the data structure. This approach is in many ways similar to the Strategy design pattern found in Gamma et.al [2]. The structure of the solution is shown in Figure 2.

In most cases each method will be made for one structure element, a water transport routine for a river will have other data needs and another solution method than a similar method for a lake. Most of the methods will also need a minimum of physical and state data available in the structural element. To check that a method can be connected and used in a structure object, a verification function is defined and called each time a method is connected to a structural object. This function will throw an exception if the method is found to be unsuitable in combination with the structural element.

Each method will also need a set of data specific to the method such as calibration constants, convergence criteria and others. These are placed in data blocks that belong to each method and then connected to the method at construction time. A specialised factory class is used to create the methods and the necessary data blocks needed.

When a new method is to be added to the system, the user must derive it from the base class of the computational hierarchy and implement functions for calculation, verification and connection to method specific data. These functions must be implemented for each class and they are therefore defined as pure virtual functions in the base class. The new computational method will then be available for insertion into the structural objects.

3 Data Handling Classes

The most important data type in a hydrological model is time series, either in the form of data with a constant time step between each value or in the form of irregularly spaced event data. Specialised classes to store these data types have been developed, providing the user with a consistent interface to time series data.

Input/output from the time series are separated in IO classes. The regular time series class (RegTS) has a RegTSIO base for several different derived classes handling different kinds of IO. Both ASCII, binary and database storage (SQL based) are supported, in addition, similar interfaces exists for screen output. All the IO classes overload the stream operators, so input and output will be handled in a similar fashion. An example shows how to move a time series from the database to screen output:

```
// Define a time series object
RegularTimeSeries reg_ts;

// A database IO class
RegularTSDatabase rts_db;

// A screen plot IO class.
RegularTSPlotter rts_screen;

//
// Some initialization of
// database and screen.

// Transfer the timeseries from
// the database and into the
// timeseries object.
rts_db >> reg_ts;

// Transfer  the regular time
// series to the screen output
// class.
rts_screen << reg_ts;
```

Tools for time series calculations and analysis have been developed that work on the timeseries objects. A timeseries collector has been developed to store several timeseries during a simulation, handling input and output from the timeseries. The timeseries class is part of a model-view-controller structure, thereby allowing for dynamic presentation of results during a simulation.

Similar classes with the same IO structure have been developed for storing x-y based data (curves) and three dimensional surface data.

In many hydraulic calculations, different kinds of grids are used in the calculation. A special grid container has been developed to handle a variety of 2- or 3- dimensional grid structures. An iterator allows the user to traverse the grid independent from the type of grid used. The grid container is a template class, which enables users to create a custom designed grid element that reflects the computational method used.

```
// A finite volume cell grid,
// grid data in class Cell.
GridContainer <Cell> cellgrid;

// A FEM grid,
// class Node represent the nodes.
GridContainer <Node> nodegrid;
```

A common way of describing a river reach is by a set of cross sections taken at locations along the river. Cross sections are stored in a cross section class that in addition to holding the cross section data (coordinates) also has methods to retrieve and calculate information about the cross section. Several cross sections can be grouped in a cross section list. The cross section list also has a set of methods e.g., to find the slope between two cross sections or the water covered area between cross sections.

4 Simulation Control and Time Handling

To handle time steps in the model a special time handler class has been developed. This class is instantiated as a

singleton object (Gamma et.al [2], globally available to all the other objects in the model. The methods can inquire the time handler for the current time step, start time, remaining time and other information about the simulation setup.

The simulation control system is the overlaying control structure of the entire model. The control system will build the model structure, assign the methods, initialise the time handler and start the simulation. All user interaction (in the form of user interfaces) with the system is isolated to the simulation control, thereby avoiding user interaction embedded in the structural, computational or data storage classes.

The simulation control is derived from a common base defining the interface to the simulation control unit, and it is possible to redefine it to provide the model with a custom designed control unit. This provides the user with the possibility to build specialised simulation procedures based on the predefined interface. This functionality is necessary in many hydraulic and hydrologic application when a sequential traversal and simulation of the system must be changed to an iterative or optimising scheme.

5 Conclusion

One of the main considerations of the framework was that it should be possible for users to incorporate their own designs by extending the framework. This is possible by using inheritance from the base classes defined for structure elements, computational classes, physical data blocks and state data blocks. It is also possible to reimplement key functions in the run control system to accommodate special simulation strategies.

In most cases the user of the system will only extend parts of the system, by usually adding computational methods and new state data. The structural elements and physical data will for most applications already be available through the framework. In many cases it will be necessary to redefine the simulation control unit to accommodate special simulation strategies.

The reusability of the system components have proven itself to be good. The data storage and IO classes have been applied to several projects, both as a part of using the entire framework and as data containers in other developments.

We have so far applied the entire framework in the development of a flood routing application and as a base for a series of models for assessing the impacts of changed flow regime on fish habitat. Compared to the traditional design and development used in our environment, the framework has proved itself to be a very effective tool for fast prototyping and testing of new modeling strategies.

References

[1] BOOCH, G. *Object-Oriented Analysis and Design with Applications.* Benjamin-Cummings Publishing Company, Inc., 1994.

[2] GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. *Design Patterns: Elements of Reusable Object-Oriented Software.* Professional Computing Series. Addison-Wesley, 1994.

Knut Alfredsen (Knut.Alfredsen@bygg.ntnu.no) received a MSc degree in Civil Engineering from The Norwegian Institute of Technology in 1988. Currently doctoral student at the Department of Hydraulic and Environmental Engineering, The Norwegian University of Science and Technology. The main working area is the use of object-oriented analysis and design methods in creation of models in hydrology and hydraulics. Other interests include models for impact analysis of river regulations and fish habitat modelling.

Bjørn Sæther (Bjorn.Sather@civil.sintef.no) received a MSc degree in Civil Engineering from The Norwegian Institute of Technology in 1980. Currently Research Engineer at SINTEF Civil and Environmental Engineering, Department of Water Resources. The main working areas is application of object-oriented methods in hydrological modelling, simulation of river systems and hydrological database systems.