ELSEVIER

Computers & Geosciences 31 (2005) 425–435

COMPUTERS & GEOSCIENCES

www.elsevier.com/locate/cageo

An object oriented approach to the description and simulation of watershed scale hydrologic processes

Jun Wang, James M. Hassett*, Theodore A. Endreny

Faculty of Environmental Resources, Forest Engineering, College of Environmental Science and Forestry, State University of New York, Syracuse, NY, 13210, USA

Received 22 May 2003; received in revised form 25 July 2004; accepted 28 September 2004

Abstract

An object-oriented design (OOD) approach is used to describe watershed scale hydrologic processes. Individual objects (or processes) at multiple levels are described using the 'inheritance' concept, while the interactions of objects (or processes) are described using the 'aggregation' concept. This design methodology is applied to create the new watershed based hydrological model OBJTOP (OBJect oriented, TOPographic based model). In OBJTOP, the watershed (top level object) is composed of five sub-objects: precipitation, evapotranspiration, vegetation, soil and channel. These objects are further subdivided, such as soil into four sub-objects: surface, and root, unsaturated and saturated zones. Each sub-object, designed using the inheritance concept, has its own distinct attributes and behaviors and works independently to represent different individual processes such as soil processes, channel processes, etc. The sub-objects, designed using the aggregation principle, are interconnected and interact with each other within the higher level objects and thus constitute the hydrological process simulation. OBJTOP presents a description of hydrologic processes in a direct and concise manner. OBJTOP's model structure should benefit model coding, maintenance, and future modification efforts, and therefore the program design strategies should be of interest to other program designers.

Keywords: Object oriented design; Hydrologic processes; Hydrologic model; Program design

1. Introduction

Watershed based hydrological models are important tools in operational hydrology and water resources planning and management. A watershed-scale hydrologic model is a simplified description of the hydrological system of a watershed. Different understandings of hydrological processes, varieties of watershed characteristics, and concerns have led to the creation of a wide variety of hydrologic models. These models have proven useful in operational hydrology and water resources management environments, e.g., HSPF (Bicknell et al., 1997), SWMM (Huber et al., 1988) as well as a more science-based, heuristic approach, e.g., TOPMODEL (Beven and Kirkby, 1979).

Procedure-oriented languages, such as FORTRAN and C, have been widely used in hydrological modeling. Popular watershed hydrological models such as HSPF, SWMM and TOPMODEL are designed and coded in FORTRAN. However, the development of the object-oriented programming (OOP) language C++ and object oriented design (OOD) techniques create the

*Corresponding author. Tel.: +315 470 6633; fax: +315 470 6958.

E-mail address: jhassett@esf.edu (J.M. Hassett).

0098-3004/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cageo.2004.09.025

opportunity to explore new means to describe complex hydrologic phenomena. Some important differences between procedural and object oriented approaches are summarized below.

In procedural programming languages, programming tends to be action oriented, with the unit of programming being the function. Procedural language programmers concentrate on writing functions. Groups of actions that perform some common task are formed into functions, and functions are grouped to form programs.

OOP programmers concentrate on creating user-defined types called objects or classes, e.g., class *soil* might be defined in a hydrologic model. Each class contains data as well as the set of functions that manipulate the data. For example, a soil class might contain soil transmissivity T as data and a function to simulate soil moisture using T. The data components of a class are called data members or attributes. The function components of a class are called member functions or behaviors. OOP advocates claim that OOP techniques provide a natural and intuitive view of the programming process, by modeling the attributes and behaviors of real-world objects (Deitel and Deitel, 2003).

Of the object-oriented programming languages, C++ is probably the most widely used. Stroustrup (1997), the C++ creator, explained the difference between C and C++, which help describe some of the benefits of C++.

> A programming language serves two related purposes: it provides a vehicle for the programmer to specify actions to be executed, and it provides a set of concepts for the programmer to use when thinking about what can be done. The first purpose ideally requires a language that is 'close to the machine' so that all important aspects of a machine are handled simply and efficiently in a way that is reasonably obvious to the programmer. The C language was primarily designed with this in mind. The second purpose ideally requires a language 'that is close to the problem to be solved' so that the concepts of a solution can be expressed directly and concisely. The facilities added to C to create C++ were primarily designed with this in mind. The C++ class concept has, in fact, proven itself to be a powerful conceptual tool.

Lafore (1999) cites some advantages of C++ over procedural languages, advantages that can improve the design, structure and reusability of code.

1. *Encapsulation*: Class builds 'firewalls' around objects, forcing all access (data) through member functions (object functions), preventing access to private implementation. This simplifies writing, debugging, and program maintenance. Objects intercept errors before they propagate outward.
2. *Inheritance*: Each subclass shares common characteristics with the class from which it is derived. Inheritance enables the derivation of a new class using all the existing properties of its base classes, and derived classes can add new properties of their own. Inheritance shortens an object-oriented program and clarifies the relationship among program elements.
3. *Reusability*: A new class inherits the capabilities of the class from which it is derived, but new features can be added. This increases software flexibility because reusability makes it simpler and easier to modify and extend the functionality and capabilities of the existing code.
4. *Polymorphism*: This technique allows functions or operators to act in different ways, depending on what they are operating on. Polymorphism simplifies code design and makes programming more efficient in some situations.

The OOP language C++ makes it possible to create a flexible and reusable model structure. The full advantages of C++ become evident when object-oriented design (OOD) principles, such as the open-closed and dependency inversion principles, are incorporated into the model design. Examples of these programming techniques will be provided subsequently.

There are relatively few applications of OOD-OOP principles in hydrologic simulation. Band et al. (2000) described a spatial object-oriented framework to model watershed systems to include hydrological and ecosystem fluxes. Chen and Beschta (1999) developed a 3-dimensional distributed hydrological model-OWLS (the Object Watershed Link Simulation model) for dynamic hydrologic simulation and applied it to the Bear Brook watershed in Maine. Garrote and Becchi (1997) used object oriented techniques with distributed hydrologic models for real-time flood forecasting. Boyer et al. (1996) presented an object-oriented method to simulate a rainfall-discharge relationship with a lumped model. McKim et al. (1993) used object oriented techniques to combine remotely sensed data with hydrologic data to create a dynamic forecast model. Whittaker et al. (1991) introduced an object-oriented approach to simulate hydrologic processes, specifically infiltration excess overland flow.

The above applications used object oriented programming and achieved reasonable results for their hydrologic simulations. However, there is little discussion of OOD principles and how to systematically implement them in program design. In this paper, we discuss model structure and design. Some important OOD principles are discussed. These OOD principles are applied in a later section illustrating OBJTOP design.

2. OOP and OOD principles employed in OBJTOP

2.1. General principles

The class/object and the 'inheritance' concepts are used to study individual parts (processes) of a system, while the 'aggregation' concept is used for simulation of interactions of the individual parts (processes). Objects in a real system can be related in two ways, i.e., in either a 'is a' or 'part of' relationship. For example, rainfall is a kind of precipitation; soil is part of a watershed, etc. Inheritance is used to describe the 'is a' relationship between objects. For example, class 'rainfall', derived from a more general class 'precipitation' using inheritance, shares all the attributes and behaviors from its base class 'precipitation', and in addition can have its own attributes and associated behaviors that make it distinct from its base class. Class 'snow' can be designed the same way. Several sub-objects (precipitation, evapotranspiration, soil and channel) can be aggregated to form a more general object (watershed) thus creating a 'part of' relationship. Aggregation provides a framework to describe the interactions of the sub-objects rainfall, evapotranspiration, soil and channel, in the object watershed.

Two important OOD principles were used in the design of OBJTOP:

1. Open-closed principle: OBJTOP is designed in accordance with this principle (open for extension, closed for modification) in that certain features of the code will not change. When change is required, new code will be added rather than changing old code that already works (Martin, 2003).
2. The Dependency Inversion Principle (DIP). Martin (2003) describes this principle as follows:
   - High-level classes should not depend upon low-level classes. Both should depend on abstractions.
   - Abstractions should not depend upon details. Details should depend on abstractions.
   - If a high level abstraction depends on low-level implementation details, the dependency is inverted from what it should be.

The highest-level classes provide the policy decisions of an application, and are designed not to change with the details of an implementation. Lower level classes incorporate the details of the (current) implementation. OOD provides a mechanism to perform this dependency inversion by using a pure abstract class to design high-level programs. The high level class depends on abstractions and is independent of the details that it controls, and the lower level, detailed programs (or modules) depend on the same abstractions. Thus, the dependency structure of a well designed object oriented program is 'inverted' with respect to the dependency structure that normally results from traditional procedural methods, and by inverting the dependencies, a structure can be created which is flexible and durable (Martin, 1996).

The above OOD principles were applied to OBJTOP design, as described in the following section.

3. Hydrologic principles incorporated in OBJTOP

3.1. TOPMODEL concepts

OBJTOP (OBJect-oriented, TOPographic) is based on TOPMODEL concepts (Beven and Kirkby, 1979). TOPMODEL, a semi-distributed watershed scale hydrologic model, is based on the premise that topography exerts a dominant control on flow routing through upland catchments. It has been widely accepted as it provides a relatively simple framework for the use of DTM (digital terrain model) data and computationally efficient prediction of distributed hydrological responses (Saulnier et al., 1997). The simplicity of the model comes from the use of the topographic index (TI, $\ln(A/\tan\beta)$, where A is contributing area and $\tan\beta$ represents local slope) of hydrological similarity which is derived from topographic data. Beven and Kirkby's original version of TOPMODEL is appropriate in small, humid, homogeneous and shallow soil catchments in which saturation excess overland flow dominates. TOPMODEL also assumes the transmissivity of the soil decreases with soil depth as described by an exponential function, a representation that may not be suitable for all soils (e.g., Ambroise et al., 1996b, a). Several of the TOPMODEL assumptions described above were relaxed in OBJTOP.

The following assumptions and features were incorporated into OBJTOP using C++ class templates technique.

(1) Both saturation and infiltration excess overland flow mechanisms are included, which allows the model to be applied to watersheds with different runoff generation mechanisms.
(2) Hydraulic conductivity can decay with soil depth in either an exponential or a generalized power function form, which allows the model to be suitable for different soil types (Duan and Miller, 1997; Iorgulescu and Musy, 1997).
(3) Both soil topographic index (STI) and topographic index (TI) mechanisms are incorporated, which is a step towards modeling spatially complex watersheds (Ambroise et al., 1996b).
(4) Simulations can be performed with or without channel routing.

OBJTOP can thus provide sixteen schemes for hydrological processes simulation appropriate to various watershed conditions and soil types. This allows flexibility for both model simulation and calibration.

3.2. Features of Class Watershed

In OBJTOP, the class Watershed is composed of five subclasses: Precipitation, Vegetation, Evapotranspiration, Soil, and Channel (Fig. 1). The classes Soil, Channel, Evapotranspiration, Vegetation and Precipitation are thus considered parts of the highest-level class Watershed. As noted earlier, OBJTOP is based on TOPMODEL concepts, in that topography exerts the dominant control on flow. In OBJTOP, topography is described as part of the soil structure and channel geometry so that it can exert influence on all the flow calculations.

Class Precipitation is divided into classes Rainfall and Snowmelt. Class Soil is composed of four subclasses: Surface (mainly for the infiltration process), Root zone, Unsaturated zone and Saturated zone (Fig. 1). The interactions among the above classes are shown in Fig. 2.

Class Rainfall and Snowmelt are components of class Precipitation (Fig. 1) and illustrate an aggregation relationship. The two classes share common characteristics of precipitation and thus represents a generalization relationship; class Precipitation is generalized from classes Rainfall and Snowmelt.

A similar relationship exists in the Soil class. Classes Surface, Root zone, Unsaturated zone, and Saturated zone can be thought parts of Soil. At the same time, the four classes share some common characteristics of soil, so the class Soil can be thought a generalization of the four classes. The two relationships among class Soil and its the lower level Classes Surface, Root zone, Unsaturated zone, and Saturated zone are reflected in the model design to simulate hydrologic processes.

Soil and its derived classes describe and simulate soil processes. The 'is a' and 'part of' relationships coexist in class Soil and its subclasses. The 'is a' relationship is used to simulate individual sub soil processes while 'part of' relationship is used to simulate the interactions of these processes. The inheritance concept is used to simulate 'is a' relationships and to study individual soil processes at multiple levels. The design of 'is a' relationship in Soil and its subclasses is a fundamental feature of OBJTOP design.

3.2.1. Class soil and sub classes

Fig. 3 illustrates the interdependencies and relationships of class Soil and its derived subclasses. Class Soil is divided into two sub classes: upperSoil and lowerSoil (Fig. 3). Class upperSoil is further divided into surface and rootzone classes and lowerSoil is divided into unsatzone and satzone classes. These classes are abstract classes that provide protocol. Classes surfaceI, rrootzone, runsatzone and rsatzone are derived from classes surface, rootzone, unsatzone and satzone, separately. The meanings of the above classes are suggested by their names. For example, upperSoil stands for upper soil, etc. Class surfaceI deals with the surface infiltration process. These fourth layer soil classes are also abstract classes but include function bodies, which are shared by their related fifth layer classes, and perform different tasks for specific soil processes. The fifth and lowest classes surfaceIE, surfaceIN, rrootzoneE, rrootzoneN, runsatzoneE, runsatzoneN, rsatzoneE and rsatzoneN were designed to simulate soil processes in two different ways in OBJTOP. 'E' and 'N' are used to represent two methods to describe change of soil transmissivity with soil depth, exponential and generalized power function decay, respectively.

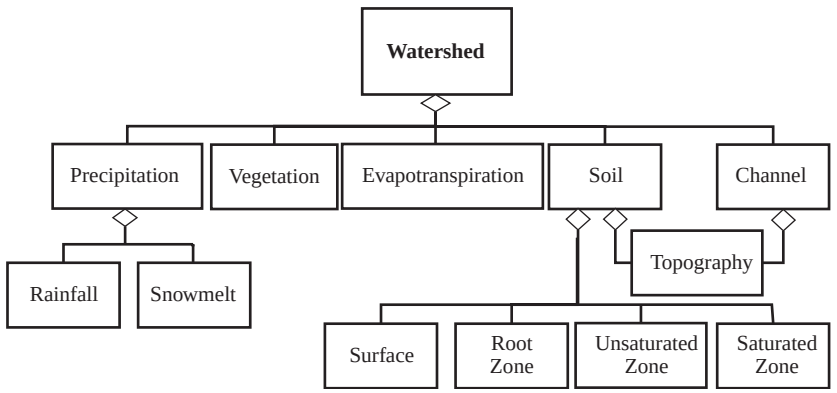

Fig. 1. UML class diagram for Watershed. The diamonds indicate a 'part of' relationship, i.e., classes Rainfall and Snowmelt are a 'part of' the higher level class Precipitation.

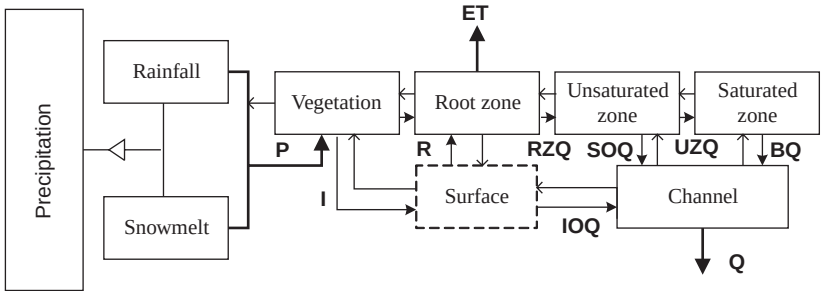

Fig. 2. UML interaction diagram illustrating interactions among classes. P indicates precipitation (snow melt or rainfall); ET is evapotranspiration from the root zone and Q is the total runoff from the watershed. I is throughfall (precipitation after vegetation interception), R is recharge rate to the soil, IOQ is infiltration excess overland flow, RZQ is root zone flow, SOQ is saturation excess overland flow, UZQ is flow from unsaturated zone to saturated zone, and BQ is base flow. The symbols have meanings as follows: —◁— means inheritance (or generalization) and represents "is a" relationship, i.e., rainfall or snow melt is a kind of precipitation. A→ B indicates association, i.e., class A asks class B to do something, B ⟶ A indicates a flow of in formation from B to A. The combination of the two arrows indicates that class B provides data to class A under the request of A. The 'Surface' class simulates the infiltration process for infiltration excess overland flow. For saturation excess overland flow, the recharge rate R goes directly to the root zone after interception of precipitation by vegetation.

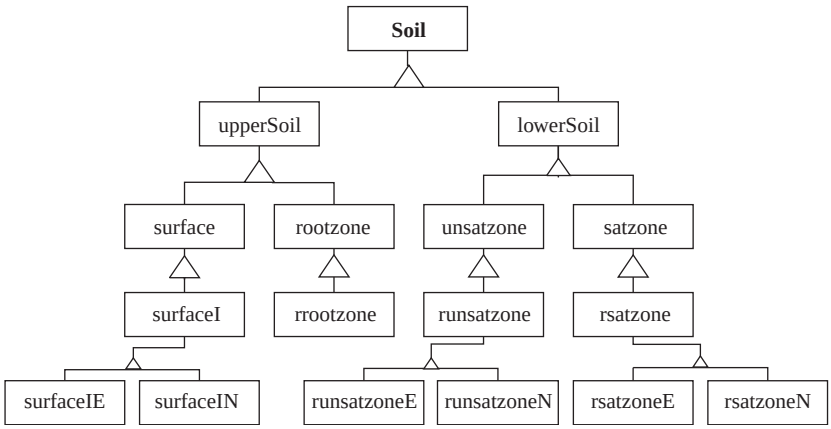

Fig. 3. UML class diagram for soil showing inheritance of soil classes. The Soil class is the highest-level class and provides a protocol upon which all its derived classes must depend. The triangle represents an 'is a' relationship.

The lower level soil classes are derived from higher level soil classes and share common characteristics with the class from which it is derived. In addition, the derived classes have new distinct properties of their own. The lower level detailed class depends on higher-level abstract class and thus illustrates the Dependency Inversion Principle.

If a new mechanism is desired to allow for a new simulation scheme, a new class can be derived from the base class, adding new member data and functions as necessary. For example, if a new mechanism X for soil surface transmissivity decay is desired, a new set of fifth layer classes surface IX, runsatzoneX, rsatzoneX can be derived from their related fourth layer base classes while utilizing the original useful attributes and functions. This illustrates the Open for Extension-Closed for Modification principle, the application of which makes OBJTOP flexible and easy to maintain.

3.2.2. Interactions of Soil Class and Subclasses

Class Soil is composed of four subclasses: Surface, Root zone, Unsaturated zone and Saturated zone (Fig. 1). During program development, the detailed lowest level classes in Fig. 3 were successfully designed and tested for their ability to simulate an individual soil process. They were then incorporated into a new soil class (comSoil) to simulate the interactions of these processes (Fig. 4). Part of the comSoil class declaration is shown in Fig. 5.

Classes comsoilS and comSoilIS share all the member data and member functions of class comsoil (Fig. 6).

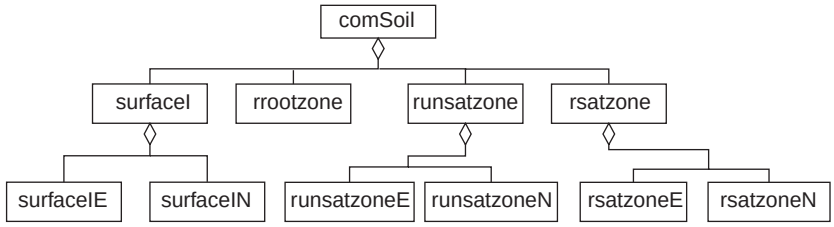

Fig. 4. UML class diagram for comSoil showing aggregation of soil classes . The diamonds indicate a 'part of' relationship. Classes surfaceIE, surfaceIN, runsatzoneE, runsatzoneN, rsatzoneE and rsatzoneN were created to simulate surface infiltration, as well as unsaturated and saturated zone processes in which soil transmissivity changes with soil depth either exponentially or as a power function. Class rrootzone is responsible for simulating root zone process.

```
template <class surInfil, class Unsat, class Sat > class comSoil
{
   protected: // member data
      surInfil _infil;                // the infiltration object
      rrootzone _rtz;                 // the root zone object
      Unsat _unsatz;                  // the unsaturated zone object
      Sat _satz;                      // the saturated zone object

   public:   // member functions
      constructors for initialization ;
      void infiltProcess(…);          // simulate infiltration process
      void rtzProcess(…);             // simulate root zone process
      void unsatzProcess(…);          // simulate unsaturated zone process
      void satzProcess(…);            // simulate saturated zone process
      virtual void soilProcess(…)=0; // pure virtual function: soil
processes
      other member functions;
      destructor for returning resources;
};
```

Fig. 5. There are four member data in class comSoil: objects _infil, _rtz, _unsatz and _satz. These objects together with four related member functions were created to simulate infiltration, root zone, unsaturated zone and saturated zone processes. In this example, (…) means parameters necessary for function operation.

They do not have their own member data and member functions. The only difference between comSoilS and comSoilIS is the content of the function describing soil processes since there is no simulation of the infiltration process (surfaceIE, surfaceIN) in comSoilS.

Template class techniques were used in the design of classes comSoil, comSoilS and comSoilIS in order to give user choices for mechanisms of overland flow generation and soil surface transmissivity decay.

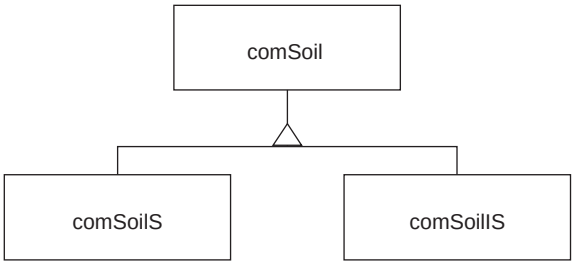

Fig. 6. UML class diagram for comSoil showing inheritance. Two sub classes, comsoilS and comSoilIS are derived from class comSoil for two types of overland flow simulation: saturation excess (comsoilS) and infiltration excess plus saturation excess (comsoilIS) overland flow.

The pure virtual function (which makes class comSoil class an abstract class) provides a protocol for soil process simulation, which is then overridden in derived classes comsoilS and comsoilIS for real, but different soil processes simulation. The soil processes simulated in class comsoilS includes root zone, unsaturated zone and saturated zone processes, while class comsoilIS includes surface infiltration, in addition to the processes in the soil process of comsoilS.

A template class is used to distinguish different mechanisms as follows. In the class comSoil declaration: <class surInfil, class Unsat, class Sat>, where surInfil, Unsat and Unsat are generic variables created to represent different generic classes. Their real names are known only after an object is instantiated. An object can be instantiated as

```
comSoilIS ⟨surfaceIE, runsatzoneE, rsatzoneE⟩ soilIS (...);
                    soilIS.soilProcess (...);
```

A comSoilIS object soilIS is created to perform soil processes using the infiltration and saturation excess overland flow mechanisms with exponential soil transmissivity decay.

When an object is instantiated as

```
comSoilS ⟨surfaceIN, runsatzoneN, rsatzoneN⟩
      soilS (...)   soilIS.soilProcess (...);
```

a comsoilS object soilS is created to perform soil processes using the saturation excess overland flow mechanism and with a power function profile for soil transmissivity decay, while the infiltration process is neglected.

Channel related classes are designed the same way as soil classes to calculate total runoff. Template classes are used to design channel classes to simulate runoff with or without channel routing.

The member function associated with each object is called by using 'object.function (...)'.

3.2.3. Watershed class design and implementation

The final step in the design of class Watershed used template class techniques. The concepts and relationships described in Figs. 1 and 2 are reflected in the Watershed class design. Fig. 7 shows part of the class Watershed declaration.

Class Watershed contains seven member data items, including five objects: a rainfall object (_rain), a vegetation interception object (_intecpt), an evapotranspiration object (_ET), a soil object (_soil), a channel object (_channel), and two simulation parameters: total simulation time steps (Ntsteps) and simulation time interval (DT). The object _rain and _ET provide time series rainfall and potential ET data to the watershed. Object _ intecpr provides throughfall; member functions soilProcess (...) and channelProcess(...) are used to call object _soil's and _channel's related member functions to simulate real soil and channel processes.

Following is the definition of function 'hydroProcess( )':

```
template ⟨class S, class C⟩
void watershed⟨S,C⟩::hydroProcess()
{
   for (int i = 0; i<Ntsteps; i++)
   {
      interceptProcess(_rain[i]);
      soilProcess(i,_intecpt.getThoughQ(),_ET[i]);
      channelProcess(i,_soil.getBaseQ(),_soil.getSur-
      faceQ(),Ntsteps);

   }

}
```

The function hydroProcess ( ) is used to simulate hydrological process in a watershed for a given time period according to the class relationships shown in Fig. 2. The interception process (interceptProcess(...)), soil process (soilProcess(...)) and channel process (channelProcess(...)) are simulated consecutively in hydroProcess ( ). The time series precipitation data is provided by rain object _rain as input to interceptProcess(...) for interception process simulation. The output (throughfall) from interceptProcess(...) and time series potential evapotranspiration provided by object _ET are input to soilProcess(...) for soil process simulation. The outputs (base flow and surface flow) from soilProcess(...) are input into channelProcess(...) to calculate total runoff with the option of including the effects of channel routing or not. The final Watershed class interface is thus concise and conceptually clear.

The ⟨class S, class C⟩ (see Fig. 7) statement is the same as ⟨class surInfil, class Unsat, class Sat⟩ in the declaration of template class ComSoil. S and C are generic class variables representing different soil and

```
template < class S, class C > class Watershed
{
  private: // member data
    int Ntsteps;                // total simulation time steps
    double DT;                  // simulation time interval
    Rainfall _rain;             // rain object
    vegetation _veget;          // vegetation object
    evaporT  _ET;               // ET object
    S _soil;                    // soil object
    C _channel;                 // channel object
  private: // private member functions
    void interceptProcess(…); // perform vegetation interception process
    void soilProcess(…);        // perform soil process
    void channelProcess(…);     // perform channel process
  public: // public member functions
    constructots for initialization ;
    void hydroProcess();        // simulate hydrological processes
    other member functions;
    destructor for returning resources;
};
```

Fig. 7. Watershed class declaration, where ' private ' data and member functions mean they are private to class Watershed and can only be accessed by the class member functions; The three private member functions are used by the public member function hydroProcess() to simulate hydrological processes.

channel classes. The real class names are known after an object is instantiated. Following is an example to implement class Watershed.

Watershed 〈comSoilIS 〈surfaceIE, runsatzoneE, rsatzoneE〉, comChannelR〉 catchmentA(…);
catchmentA.hydroProcess();

In this example, comSoilIS 〈surfaceIE, runsatzoneE, rsatzoneE〉 is the generic class variable S (for soil), and comChannelR is the generic class variable C (for channel). The meaning of soil class comSoilIS 〈surfaceIE, runsatzoneE, rsatzoneE〉 was explained in an earlier discussion, comChannelR represents a channel class that deals with channel routing, and '(…)' represents all parameters necessary for the hydrologic simulation.

When the statement: catchmentA.hydroProcess() is executed, a consecutive hydrological process in catchmentA is simulated with parameters provided through the class watershed constructor as well as time series rainfall and potential ET data provided by objects _rain and _ET. The simulation utilizes infiltration/saturation excess overland flow mechanisms and implements an exponential decay profile and with channel routing for the final output of a time series of flow values.

4. OBJTOP testing using TOPMODEL web page data

TOPMODEL web page data (http://www.es.lancs.ac.uk/hfdg/hfdg.html) were used in OBJTOP to reproduce the simulation result of a TOPMODEL application in the Slapton Wood catchment, a small test site in the

United Kingdom. There is, unfortunately, limited descriptive data about the site on the web page. The time series runoff simulation result of OBJTOP is exactly the same as that of TOPMODEL, which proves the correctness of the internal structure of OBJTOP designed with OOD principles and the OOP language C++. OBJTOP was then applied to the Slapton Wood catchment using a different simulation scheme and different set of parameters than those listed on the TOPMODEL web page. Fig. 8 is the topographic index TI ($\ln(A/\tan\beta)$) spatial pattern showing possible flow paths in the catchment. Fig. 9 is the simulated precipitation—runoff time series using OBJTOP with options for saturation excesses overland flow and channel routing selected.

The parameters used in the simulation of the Slapton Woods catchment are presented in Table 1.

A mean squared model error, and three objective functions CRF1, CRF2, CRF3 are used for model calibration to evaluate model performance.

$$\mathrm{CRF1} = \left(1 - \sum_{i=1}^{n}(Q_{\mathrm{obs},i} - Q_{\mathrm{cal},i})^2 / \sum_{i=1}^{n}(Q_{\mathrm{obs},i} - \bar{Q}_{\mathrm{obs}})^2\right)$$

tends to emphasize calibration with respect to the higher flows (Nash and Sutcliffe, 1970).

$$\mathrm{CRF2} = \left(1 - \sum_{i=1}^{n}|Q_{\mathrm{obs},i} - Q_{\mathrm{cal},i}| / \sum_{i=1}^{n}|Q_{\mathrm{obs},i} - \bar{Q}_{\mathrm{obs}}|\right)$$

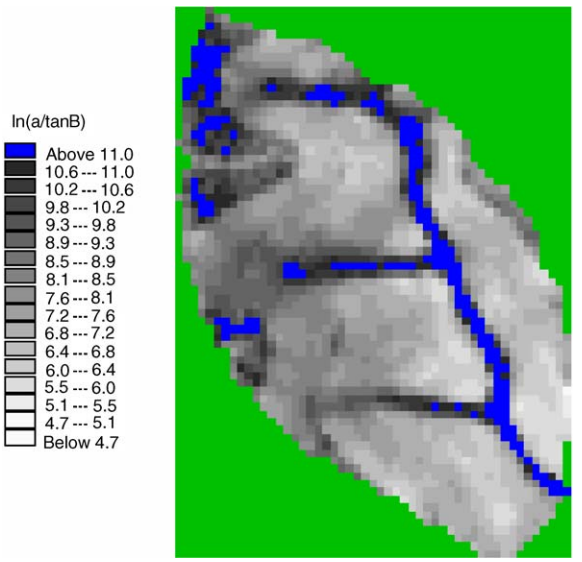

Fig. 8. Topographic Index Spatial Pattern (TOPMODEL Web Page data-Slapton Wood, UK). The topographic index ($\ln(A/\tan\beta)$) incorporates contributing area A and local slope ($\tan\beta$) to identify areas of similar hydrologic behavior.

is potentially useful in a forecasting context. It puts more emphasis on simulations at every time step (Ye et al., 1997).

$$\mathrm{CRF3} = \frac{1 - \sum_{i=1}^{n}(\sqrt{Q}_{\mathrm{obs},i} - \sqrt{Q}_{\mathrm{cal},i})^2}{\sum_{i=1}^{n}(\sqrt{Q}_{\mathrm{obs},i} - \overline{\sqrt{Q}}_{\mathrm{obs}})^2}$$

is used for a more all purpose calibration (Perrin et al., 2001).

The results of the three objective functions: CRF1 = 0.91, CRF2 = 0.72, CRF1 = 0.93. The CRF1 for original TOPMODEL simulation in Slapton Wood using TOPMODEL web page data is 0.81.

OBJTOP was next applied to two small (~60 ha) watersheds, Ward Pound Ridge (WPR), a second growth forest, and B28, an unsewered, urbanizing watershed, within the New York City East-of-Hudson drinking water supply watershed. The WPR simulation used a power function to describe hydraulic conductivity decay with soil depth, as opposed to the exponential decay used in most other TOPMODEL applications. The WPR rainfall-runoff simulation used both infiltration and saturation excess overland flow mechanisms, and achieved a CRF1 value of 0.92. OBJTOP was than updated by adding new components to simulate urban impervious areas and the hydrologic effects of septic systems as found in the B28 watershed. As discussed previously, the OOD-OOP features of OBJTOP make it relatively easy to add new model components to meet new requirements. These results are in preparation.

5. Discussion and conclusions

This paper presents an object-oriented approach using OOD techniques and the OOP language C++ to the description and simulation of watershed based hydrologic processes. OOD and OOP represent a way of organizing programs. The emphasis is on the way programs are designed, not the coding details. In particular, programs are organized around classes or objects, which contain both data and functions that act on that data. In OBJTOP, the 'inheritance' concept is used to simulate the 'is a' relationship and to study individual parts of a system at multiple levels. The 'aggregation' concept is used to simulate the 'part of' relationship and interactions of different parts. This provides a powerful methodology and conceptual tool to describe hydrological processes.

The OOP language—C++ has advantages over procedural languages in describing complicated systems. The OOD principles, such as open-closed and dependency-inversion principles, can help build a more flexible, durable, and mobile structure to meet possible future model modifications. However, the complexity of C++ (more complex than FORTRAN) and the

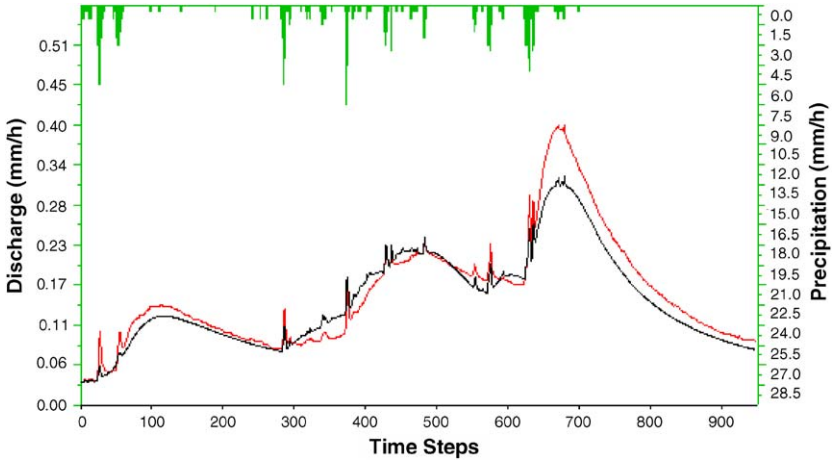

Fig. 9. Precipitation-funoff simulation, time interval: 1 h. Black line: simulated discharge, red line: observed discharge.

Table 1
OBJTOP parameters used to simulate rainfall-runoff in the Slapton Woods, UK watershed

Parameter Name	Parameter value	Comment
m	0.025	m is a scaling parameter
T_0 (m^2/h)	86.41	T_0 is saturated surface soil transmissivity,
T_d (h)	198	T_d is the unsaturated zone time delay
MRZD (m)	0.005	MRZD is the maximum root zone storage deficit
Q_0 (m/h)	3.28E-5	Q_0 is initial discharge
RZD_0 (m)	0.002	RZD_0 is initial root zone storage deficit
MCRV (m/h)	3600	MCRV is main channel routing velocity
ICRV (m/h)	3600	MCRV is the internal channel routing velocity

Data obtained from the TOPMODEL web page (http://www.es.lancs.ac.uk/hfdg/hfdg.html).

difficulties in utilizing OOD and its principles in designing complicated hydrological system may have intimidated hydrologic modelers from applying C++ to the study of watershed processes.

C++ class templates make it possible to write one generic class or function to handle many different data types. C++ class templates were used in OBJTOP to simulate different soil responses to rainfall runoff generation and with or without channel routing, etc. under generic soil, channel and watershed classes. OBJTOP thus provides an efficient framework to incorporate different assumptions and mechanisms that are suitable to different soil types and land use types for hydrological process simulations.

It is true that end users may not be able to tell the difference between models designed with procedure and OOP languages. Model simulation accuracy and efficiency depends on the simulation theory and computational algorithms, and less so on the programming language used for model design and coding. But for a model designer, the picture will be different. Almost all software changes during its lifetime. More time is spent on maintaining, upgrading, and debugging existing code than is spent on creating new work (Oualline, 2003). The biggest advantage claimed for object-oriented technology is reusability and maintainability.

OBJTOP is designed to make the model, to the extent possible, open for extension and closed for modification. Thus, when simulation requirements change in the future, OBJTOP can be extended mostly by adding new code, not by changing old code that already works. Application of this design technique should yield the greatest benefit claimed for object oriented technology; i.e., reusability and maintainability which are important in designing large, complex watershed models.

The object-oriented design approach will become more popular in the future. Even for the traditional procedural language FORTRAN, the object-oriented methods were gradually introduced starting from FORTRAN 90 and 95. However, the inheritance principle will not be fully supported until FORTRAN 200X (Akin, 2003).

OBJTOP tries to apply systematically OOD principles and OOP language C++ in the design and simulation

of watershed scale hydrologic processes. The design philosophy, methodology, and principles described in OBJTOP could be of some help in bridging C++ and hydrologic modeling and for FORTRAN programmers who are interested in using OOP methodologies in new versions of FORTRAN.

Acknowledgements

Parts of this work were funded by the New York City Department of Environmental Protection, for which the authors are grateful. The authors also thank the two anonymous reviewers who provided useful comments on an earlier draft of this manuscript.

References

Akin, E., 2003. Object-oriented Programming via Fortran 90/95, first ed. Cambridge University Press, Cambridge, UK 360pp.

Ambroise, B., Beven, K., Freer, J., 1996a. Application of a generalized TOPMODEL to the small Ringelbach catchment, Vosges, France. Water Resources Research 32, 2147–2159.

Ambroise, B., Beven, K., Freer, J., 1996b. Toward a generalization of the TOPMODEL concepts: topographic indices of hydrological similarity. Water Resources Research 32, 2135–2145.

Band, L.E., Tenenbaum, D.E., Brun, S.E., Fernandes, R.A., 2000. Modelling watersheds as spatial object hierarchies: structure and dynamics. Transactions in GIS 4, 181–196.

Beven, K.J., Kirkby, M.J., 1979. A physically based, variable contributing area model of basin hydrology. Hydrological Sciences Bulletin 24, 43–69.

Bicknell, B.R., Imhoff, J.C., Kittle, J.L., Donigian, A.S., Johanson, R.C., 1997. Hydrological Simulation Program FORTRAN: User's Manual for Version 11. EPA/600/R-97/080, US, AQUA TERRA Consultants, Mountain View, CA.; University of the Pacific, Stockton, CA.; Environmental Research Lab., Athens, GA.; Geological Survey, Reston, VA. Office of Surface Water.; Corps of Engineers, Washington, DC, 755pp.

Boyer, J.F., Berkhoff, C., Servat, E., 1996. Object-oriented programming for a simulation of the rainfall-discharge relationship. In: Proceedings of Hydroinformatics '96. Balkema, A.A. Zurich, Switzerland, pp. 299–305.

Chen, H., Beschta, R., 1999. Dynamic hydrologic simulation of the Bear Brook Watershed in Maine (BBWM). Environmental Monitoring and Assessment 55, 53–96.

Deitel, H.M., Deitel, P.J., 2003. C++ how to program, 4th ed. Prentice Hall, Upper Saddle River, NJ 1408pp.

Duan, J., Miller, N.L., 1997. A generalized power function for the subsurface transmissivity profile in TOPMODEL. Water Resources Research 11, 2559–2562.

Garrote, L., Becchi, I., 1997. Object-oriented software for distributed rainfall-runoff models. Journal of Computing in Civil Engineering 11, 190–194.

Huber, W.C., Dickinson, R.E., Barnwell, T.O., 1988. Storm Water Management Model, version 4 user's manual. EPA 600/3-88-001a, US Environmental Protection Agency, Athens, GA 568pp.

Iorgulescu, I., Musy, A., 1997. Generalization of TOPMODEL for a power law transmissivity profile. Hydrological Processes 11, 1353–1355.

Lafore, R.W., 1999. The Waite Group's Object-Oriented Programming in C++, third edition. Sams, Indianapolis, IN 850pp.

Martin, R.C., 1996. The dependency principle. C++ Report 8, 61–66.

Martin, R.C., 2003. Agile software development: Principles, Patterns, and Practices, 1st ed. Pearson Education, Upper Saddle River, NJ 552pp.

McKim, H.L., Cassell, E.A., Lapotin, P.J., 1993. Water resource modelling using remote sensing and object-oriented simulation. Hydrological Processes 7, 153–165.

Nash, J.E., Sutcliffe, J.V., 1970. River flow forecasting through conceptual models; part I—A discussion of principles I. Journal of Hydrology 10, 282–290.

Oualline, S., 2003. Practical C++ programming, 2nd ed. O'Reilly, Sebastopol, CA 574pp.

Perrin, C., Michel, C., Andreassian, V., 2001. Does a large number of parameters enhance model performance? Comparative assessment of common catchment model structures on 429 catchments. Journal of Hydrology 242, 275–301.

Saulnier, G.-M., Beven, K., Obled, C., 1997. Including spatially variable effective soil depths in TOPMODEL. Journal of Hydrology 202, 158–172.

Stroustrup, B., 1997. The C++ Programming Language, 3rd ed. Addison-Wesley, Reading, MA 911pp.

Whittaker, A.D., Wolfe, M.L., Godbole, R., Van Alem, G.J., 1991. Object–oriented modeling of hydrologic processes. AI Applications in Natural Resource Management 5 (4), 49–58.

Ye, W., Bates, B.C., Viney, N.R., Sivapalan, M., Jakeman, A.J., 1997. Performance of conceptual rainfall-runoff models in low-yielding ephemeral catchments. Water Resources Research 33, 153–166.