An object-oriented approach to the modelling of free-surface flows

V. Kutija and M. G. Murray

ABSTRACT

Over the past 40 years many hydraulic modelling systems for free-surface flows have been developed and successfully used in research and engineering practice. These systems were, in general, developed using sequential programming techniques while object-oriented programming approaches have only been used in the development of their visual parts. This paper outlines the approach used in the development of the NOAH modelling systems (Newcastle Object-oriented Advanced Hydroinformatics), developed entirely within the object-oriented paradigm. This novel approach has made NOAH modelling systems computationally highly efficient and yet easy to maintain and extend. NOAH 1D and NOAH 2D are designed to model free-surface flows in one and two dimensions, respectively. NOAH 1D is based on the full de Saint-Venant equations while NOAH 2D is based on the Shallow Water equations. Beside the basic ideas behind the development of NOAH modelling systems this paper also presents their main features and discusses general benefits of the application of the object-oriented programming approach in the development of numerical codes.

Key words | finite differences, finite volumes, free-surface flows, object-oriented approach, object-oriented numerics

V. Kutija (corresponding author)
M. G. Murray
Water Resource Systems Research Laboratory,
School of Civil Engineering and Geosciences,
University of Newcastle upon Tyne,
Newcastle upon Tyne NE1 7RU,
UK
E-mail: *Vedrana.Kutija@ncl.ac.uk*;
Michael.Murray1@virgin.net

NOTATIONS

A	Cross-sectional area (m^2)
$A1$	Approximation coefficient
$A2$	Approximation coefficient
Ac	Area of a cell
b_s	Top width/storage width (m)
β	Boussinesq coefficient
$\mathbf{b}(\mathbf{p})$	Source sink terms
$B1$	Approximation coefficient
$B2$	Approximation coefficient
$C1$	Approximation coefficient
$C2$	Approximation coefficient
$D1$	Approximation coefficient
$D2$	Approximation coefficient
d_i	Degree of a node i (number of connected channels)
e	Channel incidence index
$\mathbf{f}(\mathbf{p})$	Flux vector in x
$\mathbf{f}(\mathbf{q}^k)$	Transformed numerical flux vector
g	Gravity acceleration constant (9.81 m/s^2)
$\mathbf{g}(\mathbf{p})$	Flux vector in y
h	Water depth (m)
i	Node number
i_o	Bottom slope
j	Space step index (J-point position)
K	Conveyance (m^3/s)
k	Cell side index
L	Index for the cell on the left side of the interface
L^k	Length of cell side k
λ	Wave speed (m/s)
m	Number of side in a cell
n	Time step index
$\mathbf{n}$	Normal vector
$\mathbf{p}$	Conservative vector

doi: 10.2166/hydro.2007.101

Q	Flow (m^3/s)
Q_e	Discharge in a channel e incidence to the node i, at a Q-grid point closest to the node i
Q_i	External point discharge at a node i
$\mathbf{q}$	Transformed conserved physical vector
q_L	Lateral inflow ($m^3/s/m$)
R	Index for the cell on the right side of the interface
S_o	Bed slope
S_f	Bed friction
t	Time (s)
$\mathbf{T}$	Transformation matrix
T^{-1}	Inverse transformation matrix
τ	Time weighting coefficient
τ^k	Angle between outward normal vector **n** and x axis
u	Local cell normal velocity (m/s)
v	Local cell tangent velocity (m/s)
v_x	X cell velocity (m/s)
v_y	Y cell velocity (m/s)
x	X coordinate (m)
x^n	A local co-ordinate normal to the cell side
y	Y coordinate (m)
$*$	Start region

ABBREVIATIONS

1D	One-Dimensional
2D	Two-Dimensional
GEA	Generalised Elimination Algorithm
GIS	Geographic Information Systems
GUIs	Graphical User Interfaces
HLL	Harten, Lax and van Leer
LAN	Local Area Network
OMT	Object Modelling Technique
OO	Object-Oriented
OOD	Object-Oriented Design
OON	Object-Oriented Numerics
OOSE	Object-Oriented Software Engineering
PC	Personal Computer
NOAH	Newcastle Object-oriented Advanced Hydroinformatics
TCP/IP	Transmission Control Protocol / Internet Protocol
UML	Unified Modelling Language

INTRODUCTION

Most of the hydrodynamic models used currently in engineering practices for simulation of free-surface flows have been developed incrementally over the last four decades. They are mainly based on numerical algorithms developed over the same period and programmed using sequential programming techniques. The object-oriented (OO) approach, adopted in this work, is radically different from the traditional sequential programming approach used in those numerical algorithms. Paradoxically, this 'new' style of problem solving is closer to the ways we address problems in real life than the classical programming techniques ever were (Abbott 1994).

The implementation of an object-oriented paradigm is relatively new within commerce and industry (Budd 1991). Although the object-oriented approach was first conceived in the 1960s it was originally little used (Graham 2001). In the past, programs had to be as short as possible due to the cost and limitations of available memory. That gave the advantage to sequential programming, as object-oriented codes are always longer. In the late 1980s and early 1990s with the advent of the cheap personal computer (PC) hardware and software markets changed. The object-oriented programming approach came to the forefront as it provided the means to code the higher levels of complexity required to produce functional user-friendly software while the sequential programming approach was not able to provide the same level of flexibility (Larsen & Gavranovic 1994).

That change has prompted developments of hydroinformatic modelling systems over the last two decades. Numerical codes for simulation of free-surface flows have been wrapped up into graphical user interfaces (GUIs) that made their use easier. However, the numerical cores have mainly not been redeveloped. Even when new numerical codes have been developed they have not used the object-oriented paradigm, the main reasons being relative satisfaction with the models in current use and the belief that the object-oriented approach is not suitable for coding of numerical algorithms.

However, over the last few years some object-oriented notions have been explored in different modelling systems in the hydroinformatics field. The first class of these developments considers whole models as objects, wrapping the sequentially programmed numerical codes as objects, and coupling them with other models also wrapped up as objects (Gijsbers *et al.* 2002). The second class of developments introduced objects on a conceptual level, considering different parts of the system like objects. Neither of these developments went as far as applying an object-oriented approach to the smaller parts of the systems like discretisation cells or grid points where the basic calculations are performed (Kutija 1998). Work presented in this paper has taken that route, building up completely object-oriented numerical algorithms for free-surface flows.

The Newcastle Object-oriented Advanced Hydroinformatics (NOAH) modelling systems are NOAH 1D, NOAH 2D and three ancillary tools developed as spin-offs using many of the class structures and techniques developed while researching the two main NOAH modelling systems. NOAH 1D is a modelling system for networks with predominantly free-surface flows while NOAH 2D is a modelling system for two-dimensional free-surface flows. The three ancillary tools are: NOAH Hydraulic Assistant, a steady-state free-surface model for a single channel, NOAH Rainfall Runoff, a hydrological model based on the ARNO model algorithm (Todini 1996) and NOAH Gibberish Controller, a script based model connectivity tool.

In this paper the basis of the object-oriented analysis and design of NOAH modelling systems are presented with the aim of showing how, contrary to adopted opinion, the OO approach is suitable for the development of numerical algorithms for simulation tools. Prior to this, the description of the used numerical algorithms and a short introduction into OO analysis is given.

GOVERNING EQUATIONS AND SOLUTION ALGORITHMS

The NOAH 1D modelling system for flow in networks with predominantly free-surface flows is based on the de Saint-Venant equations for one-dimensional, depth-averaged, nearly horizontal free-surface flows:

Continuity:

$$\frac{\partial h}{\partial t}+\frac{1}{b_s}\frac{\partial Q}{\partial x}=q_L \tag{1}$$

Momentum:

$$\frac{\partial Q}{\partial t}+\frac{\partial}{\partial x}\left(\frac{\beta Q^2}{A}\right)+gA\left(\frac{\partial h}{\partial x}-i_0\right)+gA\frac{Q|Q|}{K^2}=0 \tag{2}$$

for flow in channels. At each node of the network there is a continuity equation:

$$\sum_{di}Q_e+Q_i=0 \tag{3}$$

In NOAH 1D these partial differential equations are solved using the method of finite differences on a staggered discretisation grid (with alternated discharge (Q points) and water depth points (h points) along the channels). At the nodes the water level compatibility condition is implemented which results in a single water depth point situated at each node. Water depth situated at a node is also common for all first discretisation points in channels incident to that node. The numerical scheme used is the Abbott–Ionescu scheme (Cunge *et al.* 1980; Abbott & Minns 1998).

Upon application of the Abbott–Ionescu scheme to the continuity equation (1) we get the following set of linear algebraic equations:

$$A1_j^{n+1}Q_{j-1}^{n+1}+B1_j^{n+1}h_j^{n+1}+C1_j^{n+1}Q_{j+1}^{n+1}=D1_j^{n+1} \tag{4}$$

with

$$A1_j^{n+1}=-\frac{\theta}{2\Delta x},\quad B1_j^{n+1}=-\frac{b_{sj}^{n+1/2}}{\Delta t},\quad C1_j^{n+1}=\frac{\theta}{2\Delta x}$$

and

$$D1_j^{n+1}=\frac{b_{sj}^{n+1/2}}{\Delta t}h_j^n-\frac{(1-\theta)}{2\Delta x}\left(Q_{j+1}^n-Q_{j-1}^n\right)+q^{n+1/2}$$

at all the water depth points (h points). Similarly, upon applying the same finite difference scheme to the momentum equation (2) we get the following set of equations:

$$A2_j^{n+1}h_{j-1}^{n+1}+B2_j^{n+1}Q_j^{n+1}+C2_j^{n+1}h_{j+1}^{n+1}=D2_j^{n+1} \tag{5}$$

with

$$A2_j^{n+1} = -\frac{\beta Q_j^{n+1/2} b_{sj}^{n+1/2}}{\Delta t A_j^{n+1/2}} - \frac{\theta}{2\Delta x}\left(gA_j^{n+1/2} - \frac{\beta\left(Q_j^{n+1/2}\right)^2 b_{Tj}^{n+1/2}}{\left(A_j^{n+1/2}\right)^2}\right),$$

$$B2_j^{n+1} = \frac{1}{\Delta t} + gA_j^{n+1/2}\frac{\left|Q_j^n\right|}{\left(K_j^{n+1/2}\right)^2},$$

$$C2_j^{n+1} = -\frac{\beta Q_j^{n+1/2} b_{sj}^{n+1/2}}{\Delta t A_j^{n+1/2}} + \frac{\theta}{2\Delta x}\left(gA_j^{n+1/2} - \frac{\beta\left(Q_j^{n+1/2}\right)^2 b_{Tj}^{n+1/2}}{\left(A_j^{n+1/2}\right)^2}\right)$$

and

$$D2_j^{n+1} = \frac{Q_j^n}{\Delta t} - \frac{\beta Q_j^{n+1/2} b_{sj}^{n+1/2}}{\Delta t A_j^{n+1/2}}\left(h_{j+1}^n + h_{j-1}^n\right) - \left(gA_j^{n+1/2} - \frac{\beta\left(Q_j^{n+1/2}\right)^2 b_{Tj}^{n+1/2}}{\left(A_j^{n+1/2}\right)^2}\right) \times \frac{(1-\theta)}{2\Delta x}\left(h_{j+1}^n - h_{j-1}^n\right) + gA_j^{n+1/2} i_0$$

at all the discharge points (Q points).

When all these equations obtained for each channel are coupled with the nodal continuity equations and boundary conditions they form a system of equations with equal number of equations as unknowns. So the obtained system of equations is characterised by a nearly banded matrix of the system and it is traditionally solved by a classical looped algorithm (Cunge *et al.* 1980, pp 117–121). However, in NOAH 1D the solution for this system of equations is obtained by the Generalised Elimination Algorithm (GEA) (Kutija 1994) which is an extension of a computationally very efficient branched network algorithm (Cunge *et al.* 1980) developed from the double sweep algorithm for systems of equations characterised by diagonal matrices (Abbott & Minns 1998). The GEA analyses network complexity and adapts its procedures to the network.

According to GEA, for each discretisation point of each channel a localised calculation is performed that transforms coefficients A1, B1, C1 D1, A2, B2, C2 and D2 of Equations (4) and (5) into another set of coefficients that leads to the final solution of the system of equations. Similarly, in the double-sweep algorithm this localised calculation is initiated at one end of a channel and it proceeds according to a predefined algorithmic structure from one discretisation point to another (Abbott & Minns 1998). In order to perform this localised calculation, for each grid point, one only needs to know values of the coefficients at that point and values of the already transformed coefficients in the preceding point. For more details on the GEA please see Kutija (1994).

The governing equations for the NOAH 2D modelling system are the shallow water equations for two-dimensional depth averaged flow:

$$\left.\begin{aligned} &\frac{\partial h}{\partial t} + \frac{\partial(hv_x)}{\partial x} + \frac{\partial(hv_y)}{\partial y} = 0 \\ &\frac{\partial(hv_x)}{\partial x} + \frac{\partial(hv_x^2 + gh^2/2)}{\partial x} + \frac{\partial(hv_xv_y)}{\partial y} = gh(So_x - Sf_x) \\ &\frac{\partial(hv_y)}{\partial y} + \frac{\partial(hv_xv_y)}{\partial x} + \frac{\partial\left(hv_y^2 + gh^2/2\right)}{\partial y} = gh(So_y - Sf_y) \end{aligned}\right\}. \tag{6}$$

In NOAH 2D these equations are solved using the method of finite volumes with shock capturing schemes (Erduran *et al.* 2002). For that purpose the domain is subdivided into cells and flow conditions which, within each cell, are given by a vector $[h, hv_x, hv_y]^{\mathrm{T}}$ often denoted by $\mathbf{p} = [p1, p2, p3]^{\mathrm{T}}$. In terms of the conservative vector $\mathbf{p}$ Equations (6) can be rewritten as

$$\frac{\partial \mathbf{p}}{\partial t} + \frac{\partial \mathbf{f}(\mathbf{p})}{\partial x} + \frac{\partial \mathbf{g}(\mathbf{p})}{\partial y} = \mathbf{b}(\mathbf{p}) \tag{7}$$

where

$$\mathbf{f}(\mathbf{p}) = \left[hv_x, hv_x^2 + gh^2/2, hv_xv_y\right]$$

$$\mathbf{g}(\mathbf{p}) = \left[hv_y, hv_xv_y, hv_y^2 + gh^2/2\right]$$

$$\mathbf{b}(\mathbf{p}) = [0, gh(So_x - Sf_y), gh(So_y - Sf_y)]$$

The method of finite volumes is based on the integration of the conservation equations over each cell covering the whole model domain. The result of the integration of the set of Equations (7) over each cell gives us the following set of ordinary differential equations for change of **p** at each cell with respect to time:

$$A_c \frac{d\mathbf{p}}{dt} = -\sum_{k=1}^{m} \mathbf{T}^{-1}(\theta^k)\mathbf{f}^k(\mathbf{q}^k)L^k + \int_V \mathbf{b}(\mathbf{p})dV \tag{8}$$

where $\mathbf{T}(\theta^k)$ is the transformation matrix which can be obtained by rotating the coordinate axes, $\mathbf{T}^{-1}(\theta^k)$ is the inverse transformation matrix, $\mathbf{q}^k$ is the transformed conserved physical vector obtained by multiplying **p** by the transformation matrix and $\mathbf{f}^k(\mathbf{q}^k)$ is the transformed numerical flux vector. We have

$$\mathbf{T}(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \text{ and}$$

$$\mathbf{T}^{-1}(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$\mathbf{q} = \mathbf{T}(\theta)\mathbf{p} = [h,\ hu,\ hv]^{\mathrm{T}},\ \ \mathbf{f}(\mathbf{q}) = [hu,\ hu^2 + gh^2/2,\ huv]^{\mathrm{T}}$$

with u, v local components of velocity in the normal and tangential directions to the cell boundary respectively given by

$$u = v_x\cos\theta + v_y\sin\theta, \quad v = -v_x\sin\theta + v_y\cos\theta.$$

In order to solve Equations (8) one should first solve numerical vector fluxes $\mathbf{f}(\mathbf{q})$ across each cell interface. In the local coordinates (x^n being a local coordinate normal to the cell side), the Riemann problem can be written as

$$\frac{\partial\mathbf{q}}{\partial t} + \frac{\partial\mathbf{f}(\mathbf{q})}{\partial x^n} = \mathbf{0} \tag{9}$$

with the initial state given by

$$\mathbf{q}(x^n, 0) = \begin{cases} \mathbf{q}_L; & x^n < 0 \\ \mathbf{q}_R; & x^n > 0 \end{cases}$$

where $\mathbf{q}_L$ and $\mathbf{q}_R$ are the values of the transformed conserved physical vector to the left and right of the cell interface, respectively. Upon the solution of the Riemann problem for each of the cell interfaces Equations (8) are solved for components of the vector **p**.

The only Riemann solver implemented in NOAH 2D is the HLL scheme introduced by Harten, Lax and van Leer (Toro 1997, p 297). According to this solver the initial step difference between the transformed conservative vectors $\mathbf{q}_L$ and $\mathbf{q}_R$ on the two sides of the cell interface is interfaced by another constant state, often called the star region, separated from the two other initial states by two waves as shown in Figure 1 as $\lambda_{\max}$ and $\lambda_{\min}$.

Firstly flow conditions in the star region have to be found using the method of characteristics:

$$\bar{u}_* = \frac{\bar{u}_L + \bar{u}_R}{2} + \sqrt{gh_L} - \sqrt{gh_R}$$

$$\sqrt{gh_*} = \frac{\bar{u}_L - \bar{u}_R}{4} + \frac{\left(\sqrt{gh_L} + \sqrt{gh_R}\right)}{2}.$$

There, the wave speeds can be given as follows:

$$\lambda_{\min} = \min(\lambda_1, \lambda_{st1}), \quad \lambda_{\max} = \max(\lambda_3, \lambda_{st3})$$

where

$$\lambda_1 = \bar{u}_L - \sqrt{gh_L}, \quad \lambda_3 = \bar{u}_R + \sqrt{gh_R}$$

$$\lambda_{*1} = \bar{u}_{st} - \sqrt{gh_*} \quad \lambda_{*3} = \bar{u}_{st} + \sqrt{gh_*}.$$

And finally, vector flux over the cell interface can be obtained as

$$f(\bar{q}_L, \bar{q}_R) = \begin{cases} f(\bar{q}_L) & if\ \ \lambda_{\min} \geq 0 \\ f(\bar{q}_R) & if\ \ \lambda_{\max} \leq 0 \\ f(\bar{q}_*) & Otherwise \end{cases} \tag{10}$$

with

$$f(\bar{q}_*) = \frac{\lambda_{\max} f(\bar{q}_L) - \lambda_{\min} f(\bar{q}_R)}{\lambda_{\max} - \lambda_{\min}} + \frac{\lambda_{\max}\lambda_{\min}(\bar{q}_R - \bar{q}_L)}{\lambda_{\max} - \lambda_{\min}}.$$

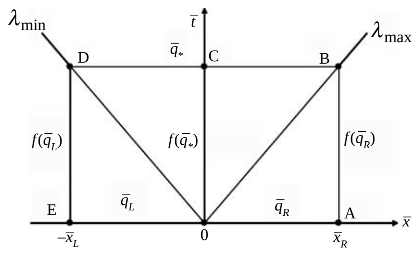


Figure 1 | The structure of the HLL Riemann Solver.

Due to the explicit nature of the used finite volumes solution it is clear how all the calculations can be related to locations. For each cell interface, the localised flux vector across the cell interface is calculated. Then for each cell vector **p** is updated taking into account all the localised flux vectors transformed into original coordinate system.

Although solution algorithms used in NOAH 1D and NOAH 2D originate from different numerical methods they have both been successfully coded using object-oriented techniques. The only requirement for the solution algorithm seems to be its ability to be split into parts of calculations that can be performed locally using local data and no knowledge of the whole algorithm. This was much simpler to implement in the case of NOAH 2D due to the explicit nature of the used solution algorithm which does not require a defined algorithm structure.

OBJECT-ORIENTED APPROACH

Implementation of an object-oriented approach starts with a good analysis of the problem to be solved. There are many different methods available now which can aid object-oriented analysis and design. Most are graphical and conceptual, some are languages in their own right and a few have been converted into development tools that can automate code creation.

During the early 1990s, approximately fifty different object-oriented analysis methods were developed (Muller 1997) and there was an attempt to control and structure the development of OO languages during the 1980s. A number of different systems were proposed: Booch (Booch 1991); Object Modelling Technique (OMT) (Rumbaugh *et al.* 1991). By 1995, the Unified Method version 0.8 using Booch and OMT was developed. In turn this led to the Unified Modelling Language (UML) versions 0.9 and 0.91 with the addition of OOSE and 1997 saw the publication of UML (Jacobson *et al.* 1999) version 1.0 by the Object Management Group (http://www.omg.org). Muller (1997) describes this genesis in four parts: Fragmentation; Unification; Standardisation; Industrialisation. The industrialisation took place with the development of such software as *Rational Rose* (http://www.rational.com/products/rose/index.jsp) which can now generate the object classes and units automatically as a program is designed, leaving the programmer to merely fill in the detail.

Most of these tools are written for standard commerce problems involving large systems, databases and different businesses, and their strength is to promote and standardise known accepted methods for general use and the unknown is not allowed.

The above mentioned tools are well suited for analysis and design of outer layers of hydroinformatics systems like GIS, databases, visualisation, etc., but not for the numerical cores. Standardised tools do not provide the means to capture all the structures and interdependences of a numerical algorithm.

DYNAMIC SYSTEMS AND OBJECT-ORIENTED APPROACH

According to Rumbaugh *et al.* the object-oriented approach is very well suited for modelling time-dependant processes (Rumbaugh *et al.* 1991, pp 84–92). In order to design an object-oriented model of a dynamic system one should:

- identify states that are changing in time (properties or attributes of objects)
- identify continuous and discreet events that can change the states (methods)
- place the whole system in a timing loop at a fine scale.

It is very easy to detect similarities between this approach and the numerical algorithms for the solution of free surface flows.

Identify states

In free-surface flows states are obviously values of the dependent variables (Q – discharge, h – water depth in one-dimensional case; h – water depth, u_x and u_y – velocities in x and y direction, respectively, in two-dimensional case) at the discrete points within the domain. The calculation (grid) points/cells will become objects with their main properties being the dependent variables placed at the grid points/cells. In the case of staggered grids there will be different classes of objects as, in different grid points, we have different dependent variables (discharge and water depth).

Identify events

Events that change the above-defined states are, in principle, all continuous but because we have already defined a numerical method that will be used to approximate real events, our events can easily be defined as discrete calculations of one time step. Methods for each class of objects introduced above will be defined as parts of the chosen solution algorithms at the level of one discretisation point or cell. This decomposure of the solution algorithms to the point/cell level will be straightforward in the cases where explicit numerical schemes are used. However, in the cases of implicit numerical schemes all the dependent variables at one time step are solved simultaneously. Hence, algorithms for the solution of sets of simultaneous equations also have to be decomposed down to the equation/variable level. The GEA in NOAH 1D will be used as an example later in this paper.

Timing loop

The timing loop is not an issue at all as we have already subscribed to it by the choice of the numerical procedure related to the initial-boundary value problems; solution of consecutive time steps on an open domain. At each time step a procedure is triggered that makes all objects, defined as discrete points/cells, update their state. Each of these objects have methods that do basic operations, i.e. calculate the coefficients of the discretised equation, calculate fluxes at the cell interfaces and update flow variables at cell centres. If one was to use an explicit numerical scheme (like NOAH 2D) that would be sufficient, as there is no need for an algorithmic structure (Abbott & Minns 1998). On the other hand, for implicit schemes that require a definite algorithmic structure (like the GEA used in NOAH 1D), a control object has to be defined, which would encapsulate the solution algorithm. The control object has firstly to analyse the network topography and define the solution order in which instances of point/cell objects have to be triggered to perform the basic operations. Then the time loop takes control and this solution order is followed at each time step. This will also be covered later in more detail.

OBJECT-ORIENTED DESIGN OF NUMERICAL ALGORITHMS IN THE NOAH MODELLING SYSTEMS

Within the numerical engine of NOAH systems there is a strict division of work. Objects that control the order in which computation is performed know the network complexity but do not know either the details of the used numerical algorithms or their implementations. On the other hand, calculation is done at the level of localised objects that are not aware of the overall complexity of the system or the order of computation.

This division allows for considerable flexibility in the introduction of new algorithms (or parts of them) and in the reconfiguration of a system. For example, localised calculations at the level of CellCentre objects in NOAH 2D would not be affected by the size of the domain (number of cells). Equally a new type of CellSide object could be introduced based on another Riemann solver and that would not in any way influence the overall solution order.

In both models the top level (overall control) acts as a hub that controls and organises the general calculation solution structure. In NOAH 1D this is the solving order for the channels and nodes and in NOAH 2D the solving order for the cell interfaces and cell centres. This system of delegated control (from the top down) enables different representations to be easily constructed. The top level understands how to control the general calculation structure but does not know how it is really implemented. This reduces the complexity of the main hub and allows for much more efficient implementation.

In NOAH 1D objects related to channels and pipes are named edges while ones related to nodes or junctions are called vertices (following the GEA algorithm). Basic calculation objects in NOAH 1D are situated at the discretisation points (grid points). They are named J-points as the letter *j* is used as an index through the discretisation points in the used solution algorithm.

ConnectionEdge
Name Upstream node pointer Downstream node pointer PhysicalEdge pointer Lateral inflow data pointer SolutionEdge pointer Channel type
Check model data Calculate J-point positions Preparation Swap ends

Figure 2 | ConnectionEdge object properties and methods.

Organisation and control

In NOAH 1D organisation and control has, in principle, been split into two levels: one being the overall control executed on the level of the whole system and the other being the local control executed on the level of edges. The overall control in NOAH 1D is performed by the central NetworkControl object that has a list of all the vertices in the network with all the incident edges. This object analyses the network topography and defines the solution order (according to GEA) according to which edges and vertices have to be triggered to perform the localised operations.

The local control level does not know the overall calculation order or anything about how the model is generally connected and structured but does know how to control its own small part. Good example of this localised control is the ConnectionEdge object (see Figure 2). From its limited number of properties it is clear that that object 'has' only a limited amount of information about the overall network and even less about the solution algorithm.

Figure 3 shows objects associated with the ConnectionEdge object during the model set-up. At that point in time ConnectionEdge is related to the PhysicalEdge object, which contains data about edge geometry. Edge geometry is defined by at least two cross sections situated at the ends of the edge but it can also contain many more cross sections at irregular distances along the edge. This information is stored in instances of the CrossSection class of objects.

Basic calculation objects in NOAH 1D are called J-points. Each J-point object is made up of three objects: a Physical point object, an Approximation point object (either of Q-point or h-point type) and a Solution point object (see Figure 4). Due to the use of a staggered grid, at each discretisation point there is either a discharge (Q) or water depth (h) dependent variable. Hence, there are two types of J-point objects: Q-point objects and h-point objects. More details about the roles of the components of J-point objects are presented later.

Figure 5 shows objects associated with the ConnectionEdge object at the run time. The ConnectionEdge is, at that time, linked into the SolutionEdge, which creates the VirtualEdge, and all the J-point objects needs for the calculation of that channel. The SolutionEdge controls the order in which the J-point objects perform their localised calculations.

Although not directly connected to the localised control it is interesting to point out at this point that the Physical point objects in Figure 5, in principle, do not coincide with the CrossSection objects from Figure 3. Physical point objects are situated at calculation points so they are either CrossSection objects defined by the user or interpolated cross sections. All different types of cross sections available in NOAH 1D are classes from the same family, decendants form the TCross-Section class (see Figure 6).

This way of organising cross-sectional object classes reduced the code by deriving all of them as descendants of the same parent class. All the cross-sectional objects need to have

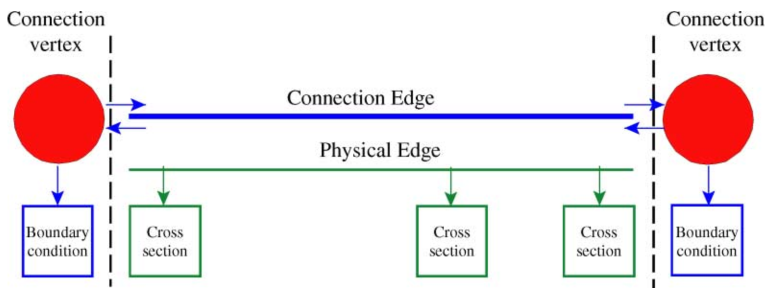


Figure 3 | ConnectionEdge object during the model set-up phase.

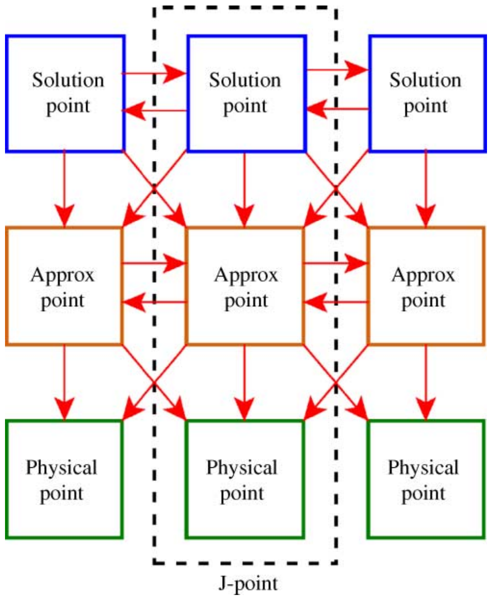


Figure 4 | J-point objects within a channel containing one Approximation point object, one Solution point object and one Physical point object.

their position (chainage) as a property and have methods to calculate their cross-sectional area, top width and conveyance for a given depth as these values are needed in Equations (4) and (5). Even the interpolated cross sections (Physical point objects of Figures 4 and 5) are members of the same family of object classes.

As NOAH 2D uses only an explicit algorithm, organisation and control of the computation is much simpler than in NOAH 1D. In NOAH 2D overall control is very simple and it is done by the object which contains lists of all cells (CellCentre objects) and all cell interfaces (CellSide objects) in the domain. At each time step that object triggers firstly all the CellSide objects to calculate Riemann fluxes across them and then all the CellCentre objects to update their state variables. As the algorithm organisation is very simple there is no need for local control but that could be easily changed if an implicit Riemann solver would be implemented.

Calculation objects in NOAH 1D

J-point objects introduced in Figure 4 consist of three objects: a Physical point object, an Approximation point object and a Solution point object The Physical point object contains data about interpolated cross sections and it is able to return cross-sectional dimensions such as cross-sectional area, top width and conveyance for a given water depth. The Approximation point object contains the appropriate dependent variable (Q or h) as a state variable and the approximation coefficients. For example, the Q-point Approximation object has a discharge state variable and coefficients A2, B2, C2 and D2 from Equation (5). The h-point Approximation object has the

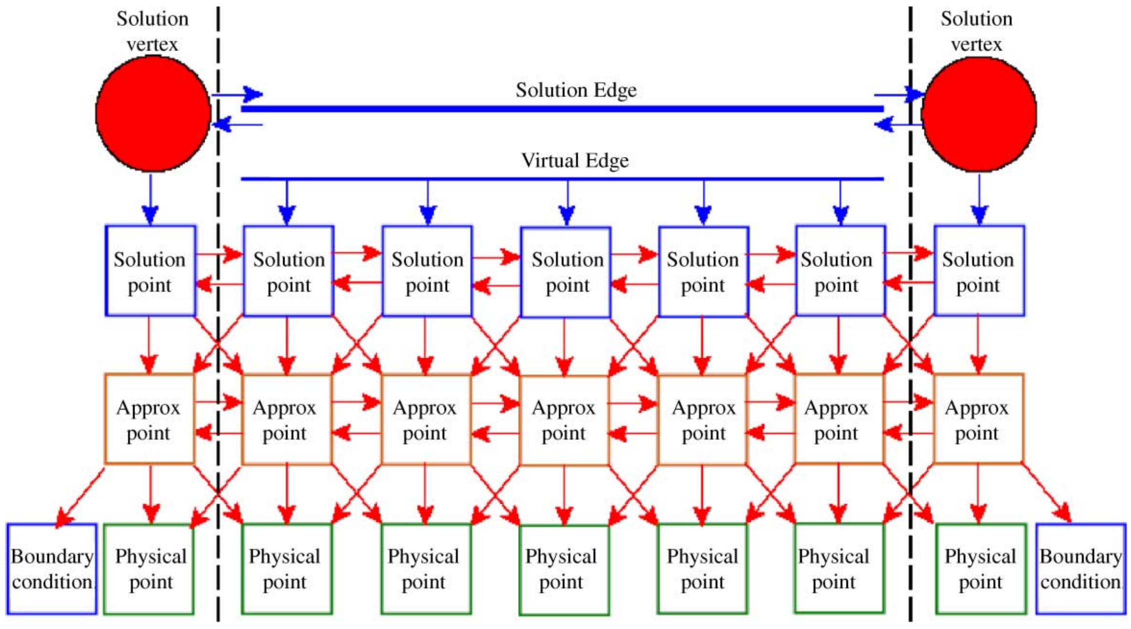


Figure 5 | SolutionEdge, VirtualEdge and J-point objects.

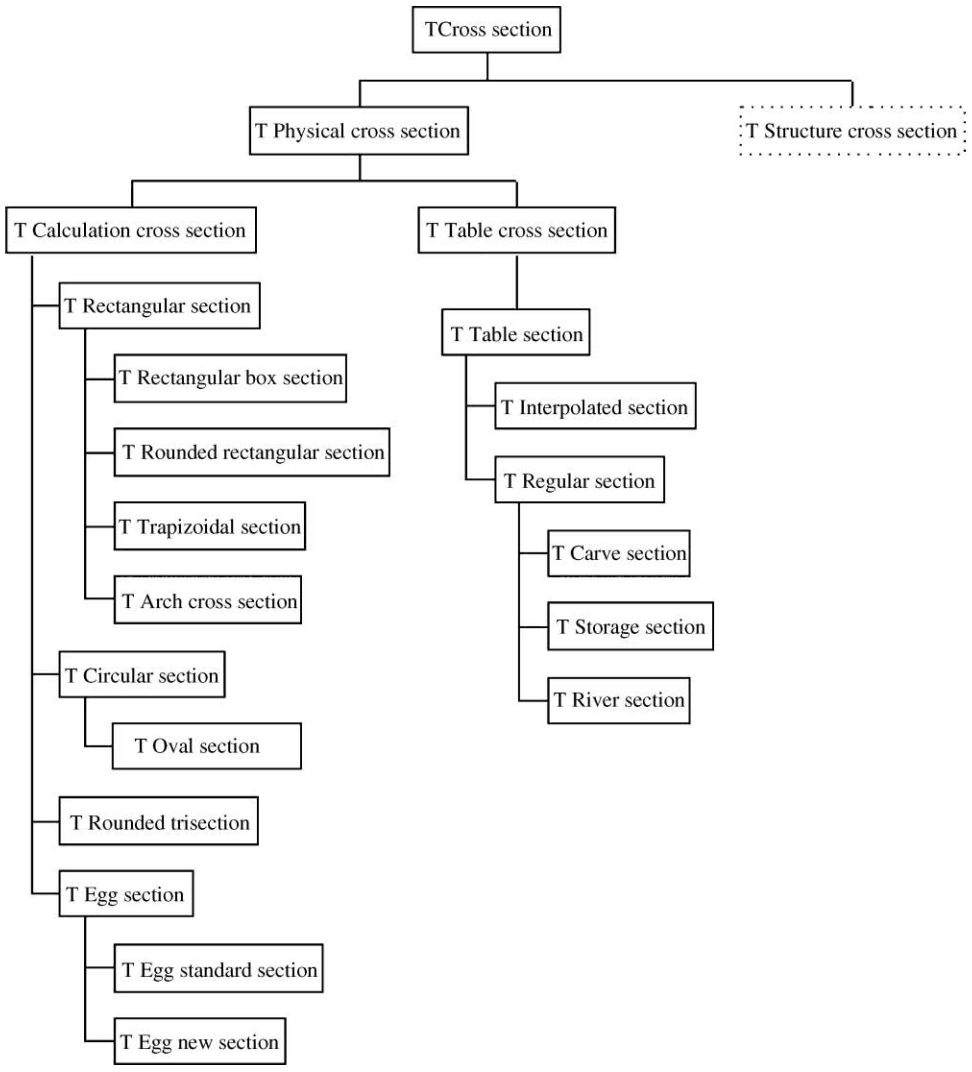


Figure 6 | TCross-Section object class diagram.

water depth state variable and coefficients A1, B1, C1 and D1 from Equation (4). The Solution point object contains the transformation equations by which coefficients A1 to D2 from Equations (4) and (5) are transformed locally as a part of the GEA algorithm. As seen from Figure 6 these three objects are closely linked together using pointers. They also have pointers to the appropriate objects in the neighbouring J-points. These pointers enable the calculation object to access data in the neighbouring points.

The split of a J-point object into three separate objects enables the replacement of the numerical scheme or the solution algorithm for the system of simultaneous equations with the minimal disturbance to the rest of the code. Each J-point object performs a small, local, part of the overall computation and it is not aware of the whole algorithm. It only performs its calculations when prompted by the control objects.

At the vertices of the network SolutionVertex objects are situated (see Figure 7). They can be seen as modified *h*-point objects as they have one *h*-point Approximation object and one Solution point object and as many Physical point objects as there are incident edges.

The amount of direct connections (pointers) between various objects in Figures 4 and 7 might look abundant but that choice was made at the design stage in order to ensure efficient calculation. It is important to take into account

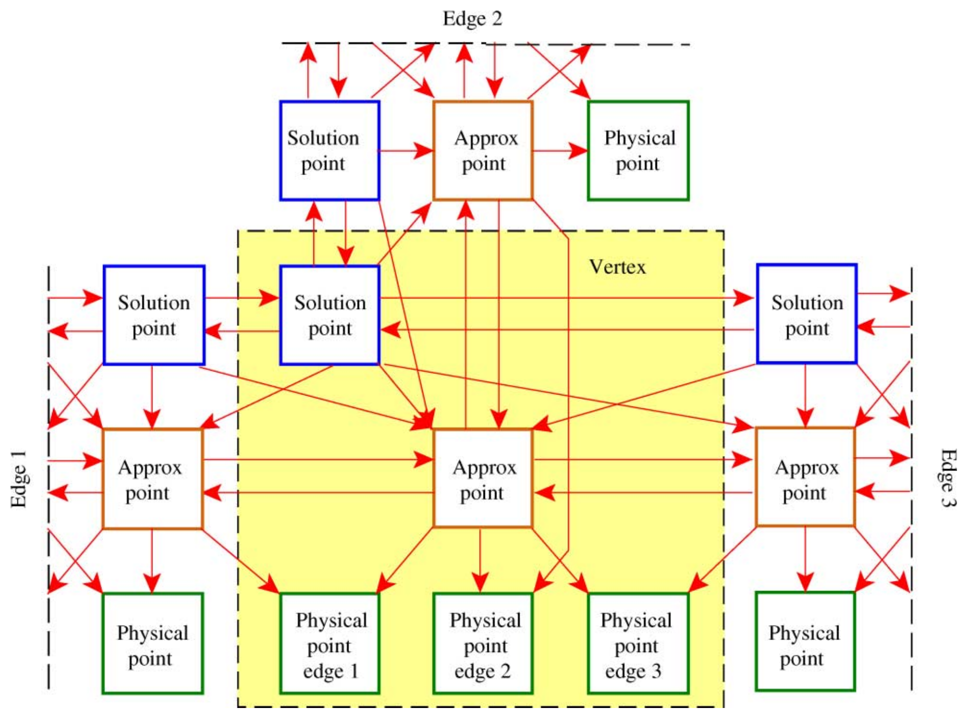


Figure 7 | SolutionVertex object. An example of a SolutionVertex object at a vertex where three edges meet containing one Approximation point object, one Solution point object and three Physical point objects.

that all the methods are executed on the level of individual object instances and if there is no connection provided to another object then no information from that object can be received. Following indirect connections through pointers to the SolutionEdge object would not be so computationally efficient.

Calculation objects in NOAH 2D

Figure 8 shows a few cells within a NOAH 2D domain. Each cell has links to its neighbouring cells and, if appropriate, external boundaries. Neighbouring cells share cell sides. At the calculation level cells are organised in such a way that each cell contains a CellCentre object and a number of CellSide objects. While the CellCentre object is associated only with one cell the CellSide objects are shared between two neighbouring cells. In that way fluxes across the cell interfaces (CellSides) have to be calculated only once and used in two cells.

Within a CellSide object the vector flux across the cell interface in local coordinates $f(\bar{q}_L, \bar{q}_R)$ is calculated using Equation (10). Within the CellCentre objects the state variables are components of the conservative vector $\mathbf{p}$; water depth h, momentum per unit width hv_x and hv_y in x and y direction, respectively. At each time step they are being updated using the explicit solution of Equation (8).

FEATURES OF NOAH MODELS

The NOAH models are completely coded in Delphi 5. Their numerical parts were developed together with the

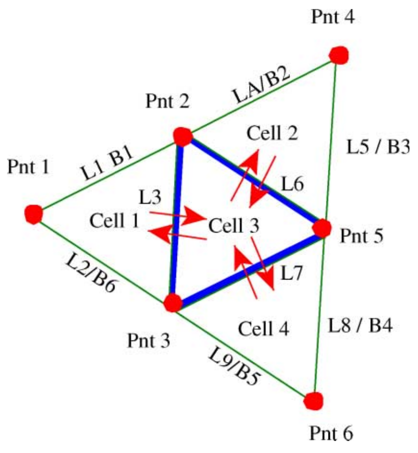


Figure 8 | NOAH 2D cells.

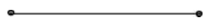
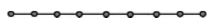
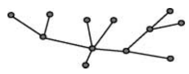
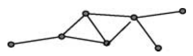
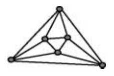
GUIs but care was taken so that the numerical parts could also be run separately in order to facilitate possible inclusion into other modelling systems. A distinctive feature of a NOAH GUI is the object inspector on the left-hand side of the screen that allows the user to browse through the object instances describing the physical entity within the network. This feature offers an overview of the physical model and enables easy inspection and editing of the model structure.

In general, NOAH 1D's simulation results correspond well with commercial modelling systems built using the same governing equation and similar numerical methods. Results of NOAH 2D are the same as published results for the HLL scheme (e.g. Erduran *et al.* 2002).

The main consequence of NOAH's OO background is its unrivalled speed of calculation. A small example was set to illustrate this using NOAH 1D. A series of networks, involving a simple channel, a series of concatenated channels, a branched network, a looped network with only a few loops and a strongly looped network were modelled. All the networks have 12 ten-kilometre long channels and a space step of 100 m which in total gives 1200 discretisation points. A flow event of the duration of one day was simulated for all these networks. The time step of 100 s was used which gives 864 time steps. All the simulations were performed on two desktop computers: first, a Pentium 3 (Windows NT, 512MB/1 GHz) and second, a Pentium 4 (Windows XP, 512MB/2.8 GHz). The times needed for these simulations are given in Table 1.

It must be pointed out here that NOAH 1D computational efficiency results from the use of a very efficient numerical algorithm and from its object-oriented structure. Due to the use of a computationally more demanding numerical solution in NOAH 2D this modelling system does not show such computational efficiency.

Table 1 | Computational times for the Example networks

	Pentium 3 (1 GHz)	**Pentium 4 (2.8 GHz)**
	25 s	5 s
	25 s	6 s
	27 s	6 s
	30 s	6 s
	38 s	8 s

NOAH 1D allows a variety of computational speeds to be used (the above results are achieved with the fastest option). When interested in tracing some event one can run just one time step at a time or choose some intermediate speed. All the runs, no matter what the chosen speed, are accompanied with the real-time graphing of the time series of discharges and water levels at chosen points. It is also possible to animate water level changes in longitudinal and cross-section views during a model run. This is made possible by the OO structure of the numerical algorithm that allows interaction with the individual objects. The same feature allows the setting up of watches on the chosen objects and records when their state variable (discharge or water depth) achieves its maximum or exceeds a defined value.

Another feature facilitated by the object-oriented structure is the ability to add new cross-sectional types (see Figure 5) with the minimum extension of the existing code. A special consequence of this is NOAH 1D's ability to deal with drainage systems and the receiving waters at the same time as it allows an extensive variety of cross sections. NOAH 1D's computational efficiency together with the extensive batching facilities enabled our colleagues to undertake research into effects of channel geometry and roughness on the flood frequency curves using a synthetically generated series of 10 000 years of rainfall records.

During the development period a series of ancillary tools and technologies emerged. Hydraulic Assistant, a steady state and backwater curve generator, was developed, originally as a test program to validate the object model for different cross-section types (see Figure 5). However, it soon became apparent that this test code was a useful teaching model and hydraulic design aid. Hydraulic Assistant has subsequently become a project in its own right. NOAH Rainfall Runoff is another spin-off using the object-oriented numerical (OON) approaches developed for the main NOAH 1D and 2D models. Unlike the Hydraulic Assistant code that reuses other NOAH 1D the rainfall runoff model uses the same numerical techniques and ideas to build a generic object framework capable of implementing a range of different rainfall–runoff algorithms.

NOAH Gibberish is again different from the previous two tools and is made up of two separate parts. The first is the NOAH Gibberish scripting language which is a text based command type instruction set that allows data and instructions to be passed between models and *via* a central controlling hub. The NOAH Gibberish Controller acts as the central hub between different modelling systems allowing data to be collated and the simulation progress controlled. The text scripts are sent across the LAN or Internet using TCP/IP. All that is required for any model to join the NOAH Gibberish system is for the modelling system to have a TCP/IP socket (available in most modern compilers) and the ability to interpret the scripts. NOAH Gibberish was originally designed to enable NOAH 1D and NOAH 2D to communicate and interact with each other at a sub-time step. NOAH Gibberish is currently implemented in NOAH 1D using OON techniques that couple directly into the solver and approximation objects that make up the J-points.

DISCUSSION

The main reason why NOAH 1D is computationally efficient is the absence of decisions during run time (e.g. no time consuming 'if' statements). All decisions are made during the initial set-up and objects are assigned direct pointers to the objects that will later supply them with their required information. For example, a discretisation point has to be able to calculate its cross-sectional area, top width and conveyance for any given water level. Classically, there would have been a subroutine that would calculate these values but each time it would have to check the type of cross section.

In a NOAH 1D set-up the physical point object (within a J-point) contains all the cross-sectional information. Appropriate physical point objects are created at the start of the computation. Once within the time loop the other parts of the J-point, or neighbouring J-points, just take the required information from the physical point objects without needing to check the cross-section type.

The other reason for the computational efficiency is the avoidance of need to search for values as members of the arrays. All the procedures are executed along the linked lists and any member of the list has direct pointers to the other objects from which it needs information during the run-time.

CONCLUSION

A use of object-oriented approach in design and implementation of numerical codes within the NOAH modelling system has resulted in unexpected benefits. These are, in the first place, easier maintenance and extendibility, and in the case of NOAH 1D, exceptional computational efficiency. The adopted approach has enabled inclusion of a variety of unusual user-friendly features and it has also opened up a possibility to build complex modelling systems by integration of self-standing systems on the level of their internal objects.

ACKNOWLEDGEMENTS

Initial development of the NOAH 1D modelling system from April 1999 to December 2000 was supported by the EPSRC, grant no GR/M44378, while the initial development of the NOAH 2D Modelling System was supported by the EU grant EVK1-CT-1999-000021 from January 2001 to December 2001.

REFERENCES

Abbott, M. B. & Minns, A. W. 1998 *Computational Hydraulics*, 2nd edn. Ashgate Publishing, Aldershot, UK.

Abbott, M. B. 1994 The question concerning ethics, or: The metamorphosis of the object. In *Hydroinformatics '94* (ed. A. Vervey, T. Minns, V. Babovic & C. Maksimovic), pp. 3–8. Balkema, Rotterdam.

Booch, G. 1991 *Object-Oriented Design with Applications*. Benjamin/Cummings, New York.

Budd, T. 1991 *An Introduction to Object-Oriented Programming*. Addison-Wesley, Reading, MA.

Cunge, J. A., Holly, F. M. & Verwey, A. 1980 *Practical Aspects of Computational River Hydraulics*. Pitman, London.

Erduran, K. S., Kutija, V. & Hewett, C. J. M. 2002 Performance of finite volume solutions to the shallow water equations with shock-capturing schemes. *Int. J. Numer. Methods Fluids* **40**, 1237–1273.

Gijsbers, P. J. A., Moore, R. V. & Tindall, C. I. 2002 HarmonI T: towards OMI, an open modelling interface and environment to harmonise European developments in water related simulation software. In *Proceedings of the 5th International Conference on Hydroinformatics, Cardiff, UK*, vol 2 (ed. I. D. Cluckie, D. Han, J. P. Davis & S. Heslop). IWA Publishing, London, pp. 1268–1275.

Graham, I. 2001 *Object-Oriented Methods Principles & Practice*, 3rd edn. Addison-Wesley, Reading, MA.

Jacobson, I., Booch, G. & Rumbaugh, J. 1999 *The Unified Software Development Process*. Addison-Wesley, Reading, MA.

Kutija, V. 1998 Use of object-oriented programming in modelling of flow in open channel networks. In *Proceedings of Hydroinformatics 98 Conference, Copenhagen*, vol. 1. Balkema, Rotterdam, pp. 633–640.

Kutija, V. 1994 A generalised method for the solution of flows in networks. *J. Hydraul. Res.* **33** (4), 535–554.

Larsen, L. C. & Gavranovic, N. 1994 Hydroinformatics: further steps into object orientation. *J. Hydraul. Res.* **32** (extra issue), 195–202.

Muller, P. A. 1997 *Instant UML*. Wrox Press, Birmingham, UK.

Rumbaugh, J. *et al*. 1991 *Object-Oriented Modelling and Design*. Prentice Hall, Englewood Cliffs, NJ.

Todini, E. 1996 The ARNO rainfall-runoff model. *J. Hydrol.* **175**, 339–382.

Toro, E. F. 1997 *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer, Berlin.