ELSEVIER

Ecological Modelling 79 (1995) 21–34

ECOLOGICAL MODELLING

ECOWIN – an object-oriented ecological model for aquatic ecosystems

J.G. Ferreira

Dept. Environmental Sciences & Engineering, Faculty of Sciences and Technology, New University of Lisbon, 2825 Monte de Caparica, Portugal

Received 23 September 1993; accepted 12 January 1994

Abstract

This paper presents an object-oriented approach to ecological modelling, and develops the underlying concepts into a structure which relates a set of "ecological" objects by means of a server, or shell, which effectively allows these to interact with each other, and displays the results of their interaction. An ecological model, ECOWIN, has been developed using this methodology, and this work examines the framework of the model, and describes its general layout. The basic structure of each object consists of a public section, which interacts with other objects and with the shell, and a private section, which carries out the activities which are characteristic of the object. The forms in which objects interact, and some of the advantages and difficulties of those interactions are discussed. Some "typical" objects are characterised, with the focus on the possibilities of adaptation and extensibility, using for instance, the concepts of inheritance and polymorphism. Some results obtained during the development and testing of the model in two different estuarine ecosystems are also shown, providing a practical application of the concepts and methodologies discussed herein. The virtues of using an object-oriented approach to ecological modelling appear to be, on the one hand, the ease of development and flexibility associated with the modularity and inheritance properties of the objects themselves, and on the other, the much greater conceptual approximation between natural ecosystems and interacting objects, relative to conventional structured programming methods.

Keywords: Aquatic ecosystems; ECOWIN; Object-oriented models

1. Introduction

This paper presents a methodology for simulation of processes in aquatic ecosystems, which employs the object-oriented paradigm in order to represent the different ecosystem compartments and their interactions. The concept of programming with objects is quite old, dating from the development of Simula™ and Smalltalk™ in the late sixties, but it has in recent years come into more widespread use, following the introduction of object extensions into high level languages such as PASCAL and C + +, and permitting the use of objects or classes.

The usage of object-oriented methods to simulate processes in ecological systems aims to simplify model development due to the flexibility associated with the modularity and inheritance

SSDI 0304-3800(94)00033-E

properties of the objects themselves, and to provide much greater conceptual approximation between natural ecosystems and interacting objects, relative to conventional structured programming methods.

Many other approaches have been used to simplify the usage and development of models, and, in all cases, a balance exists between the advantages of a particular methodology and its scope of application. Visual modelling schemes such as STELLA™, which allow a user to pictorially define the system, have been widely used (e.g. Losordo and Piedrahita, 1991; MacIsaac et al., 1991; Chesney, 1993), but the introduction of spatial variability into complex dynamic models may require approaches such as that of Constanza and Maxwell (1991), using parallel processing to consider spatial interactions.

Other approaches range from the rule-based modelling described by Muetzelfeldt et al. (1989), which aims for a more natural linguistic formulation of models using PROLOG, to modelling shells where input/output and other common features for any modelling environment are simplified, but which rely on a conventional structured programming approach for parts of the model inplementation (e.g. Scholten et al., 1990).

Objects have some important characteristics which make them very interesting for programming in general and for ecological modelling in particular. An excellent description of some of the characteristics of objects and their use in ecological modelling has been given by Silvert (1993). Smalltalk™ has become popular in recent years due to the appearance of compiled versions of the language, and Baveco and Lingeman (1992) have described an object-oriented (OOP) application for host–parasitoid relationships.

The most relevant properties of objects in terms of programming and modelling are synthesised below, with examples from aquatic ecosystem modelling where applicable:

(i) Encapsulation. Objects encapsulate attributes (or properties) with methods which act on these. Thus an object can be conceptually thought of as an actor whose actions (methods) are governed by its characteristics (attributes); phytoplankton, for instance, has as one of its attributes a standing crop and as one of its methods growth;

(ii) Inheritance. Objects may have descendants which inherit their attributes and methods, and these may in turn modify the inherited properties; for instance, diatoms will inherit the properties of the phytoplankton ancestor and adapt them in order to require silica for growth;

(iii) Polymorphism. Objects are polymorphic: a program may address a descendant of a base object type without knowing the type, and the methods called will be those of the descendant. The descendant object can be of many forms, and may in fact not have been created yet, but the program will still work properly when a new descendant is addressed.

(iv) Modularity and reliability. Objects are self-contained: a program should not modify an object's attributes directly, but only by calling its methods. On the other hand, an object should not change global variables in a calling program. This approach allows objects to be self-contained, making debugging of complex code straightforward, and avoids propagation of errors. An object can be tested in a separate environment, debugged and plugged into its intended framework. Furthermore, prototyping is made easy, and in the ecological model ECOWIN described below, the objects were "plugged in" to the model initially in a very basic form, and subsequently developed.

(v) Re-usability. Objects are re-usable, since their modularity and degree of abstraction allows them to be inserted in different programs with little or no modification. If an object must be adapted, rather than altering the code a descendant will be created. Debugging is then limited to any new methods, since the base-type will already have been tested.

An in-depth discussion of the merits of objects for general programming applications is given by Rubenking (1992), with examples of application code.

Ecological systems can be described in a much more "natural" form with an OOP approach, and the different objects may interact with each other in different forms, offering great flexibility in the simulation.

Despite the points above, ecological modelling has been slow to integrate the benefits of OOP into the discipline, as can be seen by the relative paucity of papers addressing this topic.

This paper describes an aquatic ecosystem model, ECOWIN, the development of which began in late 1991, using Carlingford Lough, Ireland, and the Tagus Estuary, Portugal as the systems to be modelled (CEC, 1992, 1993; JNICT, 1992). Both systems are macrotidal, but there are significant differences in morphology, salinity distribution, freshwater discharge patterns and rates, catchment usage, ecology, pollution loads and water uses.

A fuller description of these ecosystems is beyond the scope of this paper, and may be found elsewhere (Ferreira and Ramos, 1989 and references therein; Douglas, 1992).

Carlingford Lough and the Tagus Estuary are modelled as two very distinct types of applications, providing a wide range of conditions suitable for testing the robustness and versatility of the methodology; the ecosystems have been studied in some depth, providing a good database of information about water quality, pollution loads, and productivity of different ecological compartments.

The focus of the present work is on the conceptualisation and computer implementation of ECOWIN as a methodology for simulation. A full discussion of the application of ECOWIN to each ecosystem will be published shortly, addressing the detailed structure of each model, the approach used in calibration and validation, and the results obtained. This is a current task of the interdisciplinary groups which participate in the two projects.

2. Methodology

ECOWIN uses the concepts of object-oriented programming described previously to implement

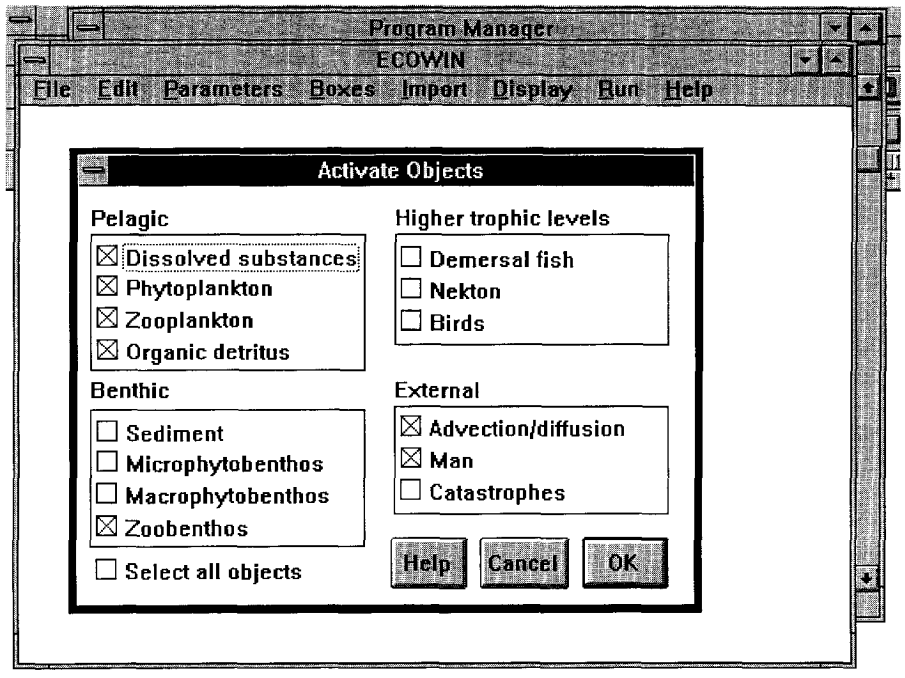

Fig. 1. Select objects dialog box from ECOWIN. The menu structure of the shell is also shown.

an ecological model for aquatic ecosystems. The basic underlying structure is that of a box-model, similar to that of Baretta and Ruardij (1988) for the Ems–Dollard, developed using proprietary software, or MOSES, under development for the Scheldt using SENECA™ (see e.g. Herman, 1993).

Unlike the Ems model, however, ECOWIN is not built around a transport model which supports several different sub-models, pelagic, benthic and epibenthic, but rather consists of a series of self-contained objects. The structure of the model is described below.

The ECOWIN model consists of two basic modules: a shell module and "ecological" objects. The shell module is responsible for communication with the various objects, for interfacing with the user, supplying model outputs, and general housekeeping chores. Both the ECOWIN shell and the different objects have been developed in Turbo Pascal for Windows™ (TPW). The MS-Windows™ environment has numerous advantages for this work, a brief list of which includes:

(i) Access to a potential 64 MB of memory using the Windows API *GlobalAlloc* function in 386 enhanced mode.

(ii) Use of libraries written in any programming language able to compile Dynamic Link Libraries (DLL): different researchers may use C, PASCAL, FORTRAN, etc., to build different parts of the model and this can all be integrated at run-time.

(iii) Expandability (e.g. through Windows NT) to very powerful computing platforms, such as the DEC Alpha AXP.

(iii) Use of communication between applications using Dynamic Data Exchange (DDE).

(iv) Rapid development of a state-of-the-art interface which provides the user with an intuitive "natural" look and feel, consistent through all Graphic User Interface (GUI) applications.

3. ECOWIN objects

3.1. Types of objects

The dialog shown in Fig. 1 illustrates the objects used by ECOWIN. Each object (or class) [1] groups together related attributes, or state variables.

Each object may thus contain one or more state variables. The Dissolved substances object, for instance, contains several state variables making up dissolved inorganic nitrogen (DIN), PO_4^{3-} and silica. It may be extended at any time to contain a new state variable, for instance dissolved oxygen. State variables may thus be added/removed within the object, without affecting the code of any other part of ECOWIN.

Similarly, the methods which control interaction between state variables within the object may be changed easily, due to encapsulation, in a manner identical to that referred by Silvert (1993) for a Gonyaulax–Nitzschia model.

The names of the objects shown in Fig. 1 are generally self-explanatory, but a brief note must be made about external objects. There are three of these:

(i) Advection/diffusion. ECOWIN is presently being used for estuarine work, and the Transport, or advection/diffusion object has as its role the transportation of any other object which requests transportation. It is the strict responsibility of a pelagic object to know that it needs to be transported; pelagic objects will thus request transportation if the Transport object is active, and this will correspond by transporting their state variables, and informing them of the new distribution within the ecosystem.

Because the Transport object acts to redistribute the mass of state variables between boxes rather than modify this within a box due to the internal processes (interaction of "biologically active" objects), it has been conceptualised as external.

(ii) Man. Effects such as an increase in the pollutant load to the catchment area, or modifications in the discharge pattern, are methods of the Man object. In this context, the potential attributes of this object are demographic and industrial vari-

[1] An object in TPW corresponds to a class in C++.

ables, coupled with environmental protection factors. In the Tagus, where domestic and industrial wastewater discharge plays an important part in the balance of the estuary (e.g. Ferreira, 1991) the focus of the Man object would be on these sorts of attributes and methods related to them.

In Carlingford Lough, however, the focus of the object is on the role in mariculture for the oyster *Crassostrea gigas*, since the oyster population does not reproduce locally and seeding is carried out annually by importation of spat, and since the main "predator" of the oyster is man.

(iii) Catastrophes. Unexpected events may also be simulated (e.g. floods, oil spills), and this object is designed for that purpose. Events included in this object are characterised by being of large dimensions and by having extensive and profound effects. Since many of these are stochastic, the use of this object will focus on studying system response, rather than on predictability.

3.2. Internal structure of an object

The structure of objects is rigid in its public sections, which are accessed externally by other objects or by the shell, and very flexible in its private sections, which may only be accessed by itself: This is valid both for attributes and methods.

The use of Private declarations within object attributes and methods may be employed, in order to force any external method calls to be made to the Public methods by which the object accesses its attributes. However a limitation for this in TPW is that descendant objects are also limited in access. Instead, in ECOWIN, access is restricted by limiting the methods other objects can access during conceptualisation.

Public methods – Objects that talk

The public sections of an object are used for initialisation and communication, both with the shell and between objects.

Table 1
Public methods common to all ECOWIN objects

Name	Interaction with	Role of the method
Constructor (init)	Shell, common ancestor to all objects	General initialisation of the object, use of inherited methods to read initial conditions from a MS-EXCEL™ spreadsheet file
Destructor (done)	Shell	Destruction of the object, freeing of Windows resources where applicable
Turnon	Shell	This method asks the object for the names of its state variables; these are used by the shell for the user interface (user input), and by the respond method which supplies appropriate values for shell output
Go	Shell, private methods and attributes	This method is invoked at each model timestep, and is responsible for accessing the private section of the object, where all calculations take place for Active [a] methods
Respond	Shell and other objects	This method is used by other objects to enquire about state variable values (e.g. when phytoplankton removes NH_4^+), and by the shell for output and display purposes
Balance	Other objects, private methods and attributes	This method is the contact channel with other objects, whose Active methods alter the properties of an object. In keeping with true OOP, these are not altered directly: this method invokes Passive [b] internal methods which keep track of fluxes
Integrate	Shell, other objects, common ancestor	The integrate method is called by the shell after all go methods of active objects have been called. It in turn calls an integration routine within the ancestor object and requests transportation to the transport object

[a], [b] See Table 2 for a definition of the different types of private methods.

Table 1 shows a list of public methods, common to all objects, which are used for interaction between the different modules, and gives a short description of the function of each.

The shell initialises each object by initialising a pointer variable of the object base-type to subsequently address it by using a syntax such as:

```
P_NewObj := New(P_NewObject,init);
```

where P_NewObject is a pointer type which points to the object. An alteration of P_NewObject to P_SonOfNewObject would be all that is required for the descendant object of the P_NewObject type to be invoked instead, inheriting or overriding methods and attributes where appropriate.

From the object's "perspective", it is irrelevant who asks it for the names of each state variable it possesses (P_NewObj^.turnon), or its current values (P_NewObj^.respond), because these are essentially information channels. The way in which the information supplied to the caller will subsequently be used is also of no concern to the object.

In other cases, namely the go and balance methods, which address the object's private methods, this may sometimes not be irrelevant. The following discussion about the private section of an object examines this in greater detail.

Private methods – Objects that act

Private methods have been conceptually divided into three types, all accessed *only* by the object itself. The defined types are as follows:

(i) Active methods. These correspond to the typical "active" behaviour of an object. Examples could be phytoplankton growth and senescence. Phytoplankton might also accumulate persistent pollutants as well, if these are present in the water, and that method would internally affect its growth method, and would also be classified as active.

(ii) Passive methods. These correspond to the processes which affect the attributes of an object but in which this plays only a passive role. An example might be grazing of phytoplankton by zooplankton, a passive method for the phytoplankton, but active for zooplankton.

(iii) Neutral methods. Advection and diffusion are considered neutral methods because in this context their role is redistribution of concentrations rather than processes of production or destruction of biomass. However, wave or tidal erosion of, e.g., a seaweed community would be a passive method for the seaweed object, rather than a neutral one.

The public go and balance methods invoke all the private methods and attributes of the object, which calculate all fluxes of mass. Because the attributes of each object may only be modified after each time step and a simultaneous solution is required for the system of equations used, a flux array is used for each state variable which is accessed by the object's private methods. This solution is similar to the distributed derivative approach described by Silvert (1993), and appears to be an adequate approach to global integration at each time step.

ECOWIN thus obtains the new concentrations for all state variables of all active objects within a box, by means of an integration method defined in a common ancestor of all objects. This may be an Euler or Runge–Kutta integration, or any other suitable numerical method.

Since, as Silvert (1993) points out, the integration routine need not be coded as an object, an existing routine can be used, and that author has interfaced OOP code to the routine RK4 (Press et al., 1988, 1989).

ECOWIN, however, takes advantage of the Windows™ environment by interfacing the ancestor object to a DLL, or Dynamic Link Library, which is actually linked to the program at run time, not at compile time. By doing so, the numerical method used for integration can not only be written in a programming language different from the source language of the main code, but may be altered at will without need to recompile the program.

The transport of properties due to advection–diffusion is solved simultaneously for all boxes, by means of a matrix used to calculate all advective–diffusive fluxes across box boundaries

and integrate these for all state variables and model boxes. This is achieved by calling the transport object's public methods by the "transportable" objects.

Several methodologies have been implemented for calculation of advection/diffusion: both central differences and upwind transport objects exist, as descendants of a "parent" transport object. The basic objects use an Euler integration, whilst further descendants of these objects use 4th-order Runge–Kutta integration. It is only necessary to change one word of the code in the model shell in order to address any one of these.

Active methods can be obligate or facultative, i.e. they can be called by the object under any circumstances or used only conditionally under particular circumstances. Normally they will be obligate, but an object may, for instance, assess its own importance (role) within the system to further detail calculations of essential properties under certain conditions (e.g. if the biomass of a property is above a certain limit).

The public balance method, which will invoke the passive methods of the object may do nothing more than address one method to maintain the flux balance. In this case the calling object is irrelevant to the passive method of the receiving object (e.g. for grazing, the latter object does not care who eats it).

However, it is possible that the existing mass for a particular property is lumped together as one state variable for public purposes (i.e. in its interaction with other objects) but discriminated internally into more attributes within the object; in this case, it is up to the passive methods to then further sub-divide the way that the consumption affects the different attributes. As an example, grazing by different zooplankton groups may be more or less sloppy, leading to differential losses of dissolved organic carbon from phytoplankton cells.

3.3. Relationships between objects

The inheritance properties of objects are used only to a certain extent, because the different objects all intervene in the cycling of organic matter. Initially, the conceptualisation was developed on the principle of a trophic chain by making each successive (higher) level a descendant of the previous one. Thus, zooplankton would "inherit" appropriate characteristics from the phytoplankton, such as the biomass consumed, and add properties of its own. However, the cycling of

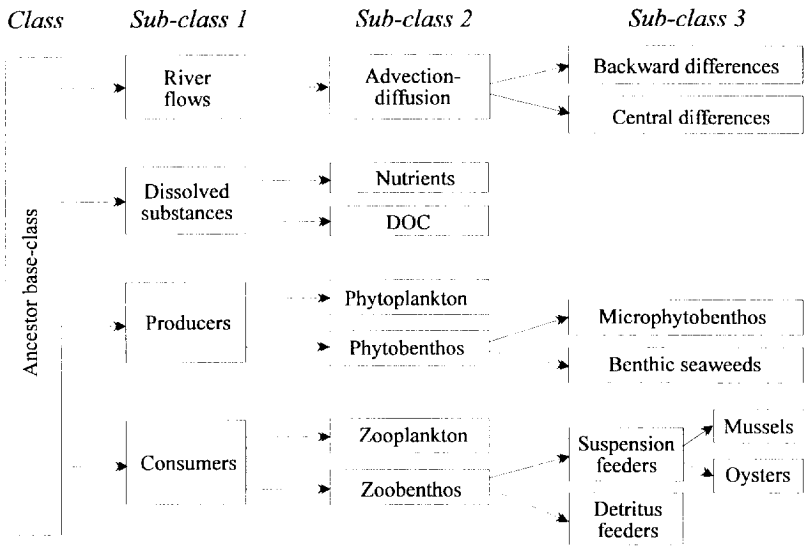

Fig. 2. Simplified scheme of a typical object hierarchy, showing taxonomic relationships.

organic matter through the different biotic and abiotic compartments would result in one of the descendant objects being the parent of the first ancestor, invalidating this approach.

The inheritance structure thus operates at two levels in ECOWIN:

The first is a descendancy of particular types of an object from a basic ancestor, or base-class: this is exemplified, e.g., by the existence of specific filter-feeding and deposit-feeding benthic objects descended from a zoobenthos object, itself a sub-class of a consumer ancestor; the specific filter-feeder object may in turn be developed into further descendants (e.g. oysters and mussels). A simplified scheme based on the Carlingford Lough model is illustrated in Fig. 2.

The second is a descendancy of objects from ancestors which perform functions relevant to them, and whose methods or attributes they must use in order to carry out their own activities. As an example, the Transport object, which is responsible for simulating the salinity distribution (or any conservative property) in an estuarine system, and also for advection–diffusion of non-conservative attributes of other objects, is a descendant of the Flow object, which simulates river flows through a yearly period.

The Flow object is responsible for computing all freshwater flows into the model boxes, and their temporal variation. Brief results for salinity are presented in the next section, and the daily and seasonal variation strongly depend on the ways in which Flow modifies its attributes. As an insight into flow's methods, a continuous variation of the flow pattern might apply Eq. 1, which uses a simple cosine function based on the modal flow, the amplitude variation and a daily stochastic variability in river flow.

$$T_{flow} = T_{modal} + (\cos(\pi^{*}(\tau - \lambda)/180))^{*}(\max h + f_r)) \quad (1)$$

Table 2
Schematic attributes and active and passive methods for some objects (neutral methods are not included: they are all advection–diffusion processes)

Object	Sample attributes [a]	Typical active methods	Typical passive methods
Transport	Salt	Advection–diffusion	–
Dissolved substances	Forms of DIN, PO_4^{3-}, SiO_2, D.O.	Nitrification, formation of particulates	Mineralization of detritus, exsudation
Phytoplankton	Phytoplankton, toxic algae	Production, respiration, senescence, exsudation, production of toxins	Grazing by zooplankton, fish, benthic filter-feeders
Phytobenthos	Microalgae, macroalgae, salt marsh flora	Production, respiration, senescence	Grazing by zooplankton, fish, harvesting of seaweeds
Zooplankton	Zooplankton, copepods	Eat (& slop), grow, reproduce, excrete, natural mortality, swim, settle (for benthic larvae)	Predation by other objects and within the object
Zoobenthos	Filter-feeders, deposit-feeders	Filter, grow, accumulate metals	Fisheries, predation by several other objects
Nekton	Fish, large invertebrates (e.g. Sepia)	Hunt (including select), grow, reproduce, excrete, natural mortality, swim, migrate	Fisheries, hunting by birds, infection

[a] Attributes are normally expressed in concentration units – attributes may be developed into descendant objects based on the level of detail required.

where T_{flow}: river flow; T_{modal}: modal flow; τ: calculation timestep; λ: lag factor; Maxh: maximum amplitude; f_r: stochastic fluctuation.

This equation may be substituted by a time-series, or by a function adjusted to it, where appropriate. For Carlingford Lough, for instance, a time-series of flow data has been used, due to specific characteristics of the local precipitation regime.

Another example of the second type of descendancy is given by the primary producer objects which all descend from an object which calculates global radiation, tidal heights, light extinction coefficients and water temperature, thus providing the basic physical data on light climate and temperature used in the active methods of descendant objects (Duarte and Ferreira, 1993). Whilst the phytoplankton might use the ancestor's methods without change, the macrophytobenthos on tidal flats must override or extend the parent methods in order to calculate the overlapping tide and light window.

The objects described here have been built for particular simulations, and the level of detail differs for the different models. In Carlingford Lough, for instance, the level of detail regarding benthic filter-feeders is much greater than for the Tagus, whereas in the latter system, the methods related to the distribution of pollutants such as heavy metals in different ecosystem compartments must be far more detailed.

Although the model has been applied in two very different systems, the use of the re-usability properties of OOP has led to a significant reduction in development time; major sections of code for different objects are shared, with descendants being written in specific cases, often overriding only a few particular virtual methods, such as those containing ecosystem-specific functions.

Table 2 provides a synthesis of the some of the objects used by ECOWIN, and a summary of important attributes and methods of each. Attributes are usually expressed in concentration units. Some of the attributes are actually developed into descendant objects in particular cases – for example microphytobenthos and macrophytobenthos. Table 2 is only schematic, because one of the fundamental advantages of using object-oriented ecological modelling is the flexibility and extensibility of the objects themselves, tailoring them to the specificity of different ecosystems: it is just as senseless to define a universal set exhaustively as to try to define a universal ecosystem type.

4. ECOWIN shell

The ECOWIN shell is the support structure for existing objects, for descendants of these or for new objects which may be created in the future. It was designed to incorporate the following characteristics:

(i) Provide a GUI consistent with the basic principles set out in the IBM Common User Access (CUA) guide

(ii) Provide dialogs for several user-defined inputs. These include:

- definition of model boxes, in terms of their morphology, diffusion coefficients, etc.;
- choices in the display of results, e.g. which boxes and which variables to display;
- choices in the type of output (graphs, tables, etc.), saving simulation data to spreadsheet files, importing calibration data, etc.;
- definition of the model boxes, in terms of their morphology, diffusion coefficients, etc.;
- switching objects on or off (this is the dialog shown in Fig. 1).

(iii) Define and implement methods for reading in the model initial conditions from EXCEL files. These are actually methods of a common ancestor object of all the model objects, and as such it is the objects who read in the variables, using the names of properties which they define themselves.

(iv) Act on the user's choice of objects, by performing the following tasks:

- destroying any currently active objects;
- initialising the newly selected objects, making them active;
- invoking an Enquire method which addresses each active object in turn, by querying its Turnon method until the object has sent the shell all the state variables it wants to send;

(v) When the user runs the model, interact with the public methods of objects by:

- calling a Go method which invokes the public Go method of all active objects, and then calls the Integrate method of these (see the definitions of public object methods above);
- calling a Respond method which invokes the public Respond method of all active objects, informing the shell of the values for the selected state variables which it must output.

The shell is designed as a series of objects (although not "ecological" ones), so most of the output routines have been developed as descendants of more basic methods, which means that the shell may be quite easily extended to different types of output. Because Windows is not a preemptive operating system, the ECOWIN shell builds in all the necessary features to allow the model to run in the background with other applications and share CPU time.

The shell ensures that all the active objects interact, even though the output displayed may be user-limited to only one state variable or to just one box (e.g. see Fig. 3).

Much of the user input has been routed through EXCEL for Windows spreadsheet files, for simplicity and to provide a familiar user interface. Similarly, the dataset from a simulation may be exported in real time to an EXCEL file, and from there to practically any application, for further post-processing such as statistical treatment. Future developments for the shell include coding a DDE link which will allow the data to be sent directly to an open spreadsheet file, multi-tasking under Windows.

Because of the ease of transporting data in several forms between Windows applications (files, clipboard and DDE), several things need not necessarily be coded in all applications. Presently, in ECOWIN, there is no method to print data; it is simply exported to a spreadsheet, which has a wide range of formatting capabilities, and printed. Similarly, any screen outputs, such as the graphs presented in the next section, are copied onto the clipboard and pasted into a drawing program or word processor, where they may be annotated and subsequently printed.

ECOWIN runs under Windows 3.1 in a 386 PC or better. It was developed in Borland's TPW (version 1.5), using a 486DX at 33MHz, and a machine of similar characteristics is recommended. A normal yearly run on a 486, using a one-day timestep, with several active objects, takes about 30 seconds.

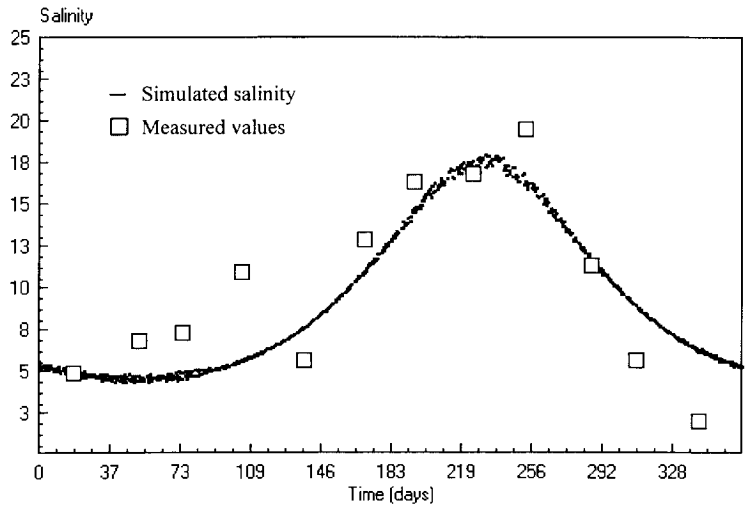

Fig. 3. Salinity (‰) in the upstream channel (box 1) of the Tagus Estuary.

5. Sample model outputs

Since this paper focuses on the conceptualisation of ECOWIN, only a few results are shown, illustrating applications of the methodology. These are model runs carried out using some of the objects, both for the Tagus Estuary and for Carlingford Lough.

Fig. 3 shows a run of ECOWIN for the Tagus Estuary with only the Transport object active. The plot is for salinity in the most upstream box of the model, where the salinity fluctuation is greatest, and datapoints are shown, which have been used to validate the model. The river flow is varying continuously according to Eq. 1, between limits of 150 and 650 $m^3\ s^{-1}$.

Fig. 4 shows an identical graph for Carlingford Lough, showing the salinity variation in the three model boxes. The river discharge is quite different from that of the Tagus, because of climatic differences, and during the months of January and February flows are depressed due to frosting. The type of flow algorithm used by the Flow object is different, and the salinity distribution reflects this reduction in flow in the winter months. As would be expected, the oscillation in salinity decreases in amplitude towards the estuary mouth.

The graphs in Fig. 5 (a–c) show the evolution of NH_4^+, phytoplankton and zooplankton in Carlingford Lough, with different active objects. Fig. 5a shows the evolution of these parameters in box 1 (the upstream box), with only the Dissolved substances and Phytoplankton objects active; the ammonia is rapidly depleted because there is no exchange between the model boxes and with the system end-members, and the phytoplankton standing stock reaches a plateau, which remains constant because no grazing objects are acting on it.

Fig 5b shows the effect of the Transport object on the system. Again, only box 1 is shown, for simplicity, and the graph is of a stable yearly pattern of variation, achieved after the model has run for a year. There is a peak of NH_4^+ in the beginning of the year, caused by increased river flow in the beginning of winter and depressed phytoplankton productivity due to low solar radiation. Subsequently the phytoplankton biomass increases during the spring, accompanied by nutrient depletion. The reduction of the algal standing crop during the summer months is due to exchange with other boxes, because no grazing object is active. In the autumn, there is a smaller phytoplankton peak with a subsequent decline at the end of the year.

Fig. 5c shows the additional effect of the Zooplankton object, for box 1, which grazes practically all of the phytoplankton biomass produced, resulting in a very low phytoplankton standing

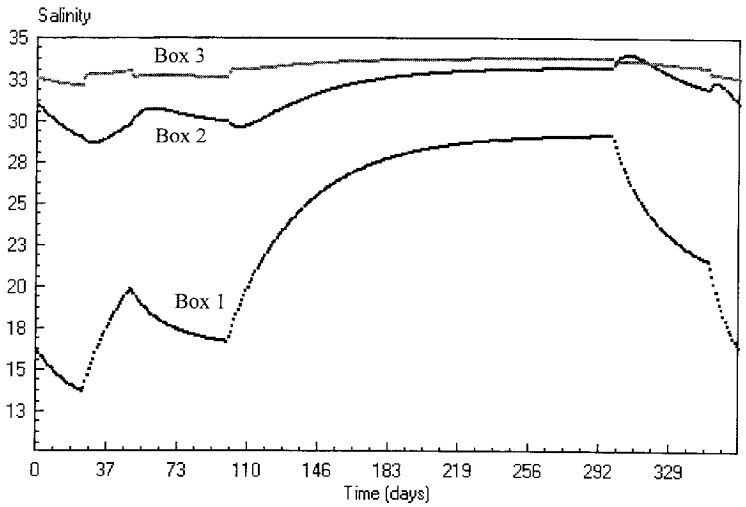

Fig. 4. Salinity (‰) in the three model boxes of Carlingford Lough (box 1 – head, box 3 – mouth).

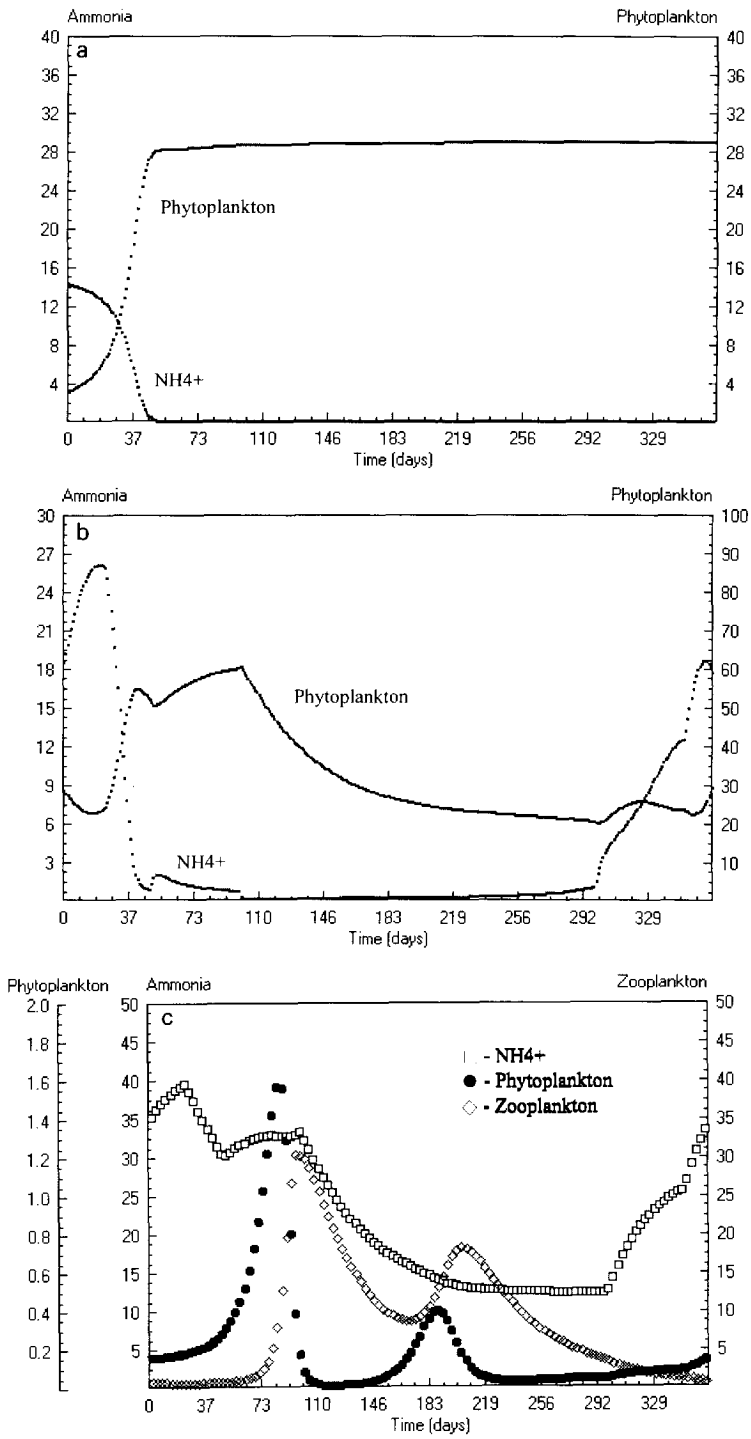

Fig. 5. (a) NH_4^+ (μmol l^{-1}), and Phytoplankton (mg chl. *a* m^{-3}) in Carlingford Lough (box 1); active objects are: dissolved substances and phytoplankton. (b) NH_4^+ (μmol l^{-1}) and Phytoplankton (mg chl. *a* m^{-3}) in Carlingford Lough (box 1); active objects are: transport, dissolved substances and phytoplankton. (c) NH_4^+(μmol l^{-1}), Phytoplankton (mg chl. *a* m^{-3}) and Zooplankton (mg d.w. m^{-3}) in Carlingford Lough (box 1); active objects are: transport, dissolved substances, phytoplankton and zooplankton.

crop. The graph shows peaks for NH_4^+, phytoplankton and zooplankton which succeed each other in time reflecting the processes involved.

These graphs show how the modular approach using objects may provide information about the responses of different parts of the system. The activation of other objects would alter the dynamics of the state variables shown in these figures, and switching them off selectively helps in analysing the relative importance of different processes in the ecosystem.

6. Conclusions

An object-oriented ecosystem modelling approach has great advantages in terms of model construction, extensibility, and adaptation to particular circumstances. Objects are tailor-made to fit a particular situation, and the focus of the model can be placed on metal pollution, benthic productivity, red tides, etc., by extending or adapting different objects.

Since objects not only provide encapsulation, but also have inheritance properties, it is easy and convenient to develop descendants, who can modify or extend them. For instance, the phytoplankton object may be further sub-divided into diatoms and dinoflagellates, each of which share certain common attributes, and define new attributes of their own. An object can therefore be extended according to its role in the ecosystem.

Maintenance of a particular model is thus also greatly simplified because of the nature of OOP: new developments can be coded as descendants of existing objects, and the usage of polymorphism means that it may be only necessary to change one word of code in the shell (the name of the new instance of the class) to address it throughout the whole system.

As was shown previously, objects may be switched on or off, which parallels their existence (or importance) within a system. The analysis of the model's sensitivity to different components may be much more easily studied, even "on the fly", by altering the characteristics of the ecosystem.

ECOWIN does not provide a possibility to construct objects from within the shell, and some programming is necessary. However, it takes full advantage of OOP, allowing the user to interface his objects to the shell quickly and easily.

ECOWIN thus strikes a balance between a complete "packaged software" approach, which will not easily lend itself to extensions to code, and a first or second generation modelling approach. In the first case, usage is simplified, but limitations arise as the complexity of the simulated system increases. In the latter case, many changes can be effected, but interaction with the user is normally tedious, and modifications usually require a profound understanding of the code, accompanied by extensive debugging.

Unfortunately, the latter approach tends to mean that the user-base of models is frequently very restricted, which does little to diversify the knowledge sources of different processes involved in ecological modelling, which must by nature be interdisciplinary.

Limitations on the size of models can be linked to three factors: computer memory, model complexity, and execution speed. The use of pointers and large memory models in Windows, the advent of native 32-bit code for Windows NT and the constant increase in RAM on personal computers means that the limitations will focus more on complexity and speed.

In ECOWIN, size limitations are essentially those presently imposed by the operating system and hardware: The compiler produces fast executable code, allowing a complex set of state variables and interrelationships to be considered. The approach used means that this ecosystem complexity may be built up gradually and with limited scope for errors, without leading to a tangle of equations and variables, and easily allows several people to work simultaneously on different objects of a model (a sort of human parallel processing).

The design specifications of ECOWIN imposed a degree of abstraction of the shell from the model objects, which gives the objects scope to define almost totally how the system behaves. The shell is comparable to an ecosystem which may contain objects. If it contains water, then this will exhibit typical "water" behaviour, such as

currents and resuspension of deposited material, encapsulated in a Water object. The model system may then be built upon by adding different components, to correspond to existing ecosystem compartments, such as phytobenthos or fish.

For the user of the model, the objects and shell appear to be seamlessly integrated, but in fact the two parts are very loosely coupled. They have a type of client–server relationship: the shell will run with no objects present, although it will do very little, and as objects are plugged in the system, it will respond appropriately.

For the programmer, the only thing that ECOWIN requires is conformism for the public structure of a typical object. The rest is up to the individual user. If any objects are not to his liking, he can throw them away and write new ones: better still, write a descendant.

Acknowledgments

The author wishes to thank the EEC FAR programme (AQ2516) and JNICT (PMCT 687/MAR), which have supported this research. He is also indebted to B. Keegan, B. Ball and D. Douglas for use of data from Carlingford Lough, and to R. Neves and P. Duarte for stimulating discussions and suggestions through the course of this work. Helpful comments of two anonymous reviewers on an earlier draft are also acknowledged.

References

Baretta, J. and Ruardij, P. (Editors), 1988. Tidal Flat Estuaries. Simulation and Analysis of the Ems Estuary. Springer-Verlag, Berlin.

Baveco, J.M. and Lingeman, R., 1992. An object-oriented tool for individual-oriented simulation: host–parasitoid system application. Ecol. Model., 61: 267–286.

Chesney, E.J., 1993. A model of survival and growth of striped bass larvae *Morone saxatilis* in the Potomac River, 1987. Mar. Ecol. Prog. Ser., 92: 15–25.

Commission of the European Communities (CEC), 1992. Interim report to the FAR program "Development of an Ecological Model for Mollusc Rearing Areas in Ireland and Greece". CEC, Brussels, December, 1992.

Commission of the European Communities (CEC), 1993. Interim report to the FAR program "Development of an Ecological Model for Mollusc Rearing Areas in Ireland and Greece". CEC, Brussels, July, 1993.

Constanza, R. and Maxwell, T., 1991. Spatial ecosystem modelling using parallel processors. Ecol. Model., 58: 159–183.

Douglas, D.J., 1992. Environment and Mariculture (A study of Carlingford Lough). Ryland Research Ltd., Ireland.

Duarte, P. and Ferreira, J.G., 1993. A methodology for parameter estimation in seaweed productivity modelling. Hydrobiologia, 260/261: 183–189.

Ferreira, J.G., 1991. Factors governing mercury accumulation in three species of marine macroalgae. Aquat. Bot., 39: 335–343.

Ferreira, J.G. and Ramos, L., 1989. A model for the estimation of annual production rates of macrophyte algae. Aquat. Bot., 33: 53–70.

Herman, P., 1993. Estimating estuarine residence times in the Westerschelde (the Netherlands) using a simple box model. Joint European Estuarine Project (JEEP92) final report. Commission of the European Communities, Brussels, in press.

Junta Nacional de Investigação Científica e Tecnológica (JNICT), 1992. Relatório anual do projecto "Desenvolvimento de um Modelo Ecológico para o Estuário do Tejo". JNICT, Lisboa.

Losordo, T.M. and Piedrahita, R.H., 1991. Modelling temperature variation and thermal stratification in shallow aquaculture ponds. Ecol. Model., 54: 189–226.

MacIsaac, H.J., Sprules, W.G. and Leach, J.H., 1991. Ingestion of small-bodied zooplankton by zebra mussels (*Dreissena polymorpha*): can cannibalism on larvae influence population dynamics? Can. J. Fish. Aquat. Sci., 48: 2051–2060.

Muetzelfeldt, R., Robertson, D., Bundy, A. and Uschold, M., 1989. The use of PROLOG for improving the rigour and accessibility of ecological modelling. Ecol. Model., 46: 9–34.

Press, W.H., Flannery, B.P., Teukolsky, S.A. and Vettering, W.T., 1988. Numerical Recipes in C. Cambridge University Press, Cambridge.

Press, W.H., Flannery, B.P., Teukolsky, S.A. and Vettering, W.T., 1989. Numerical Recipes in FORTRAN. Cambridge University Press, Cambridge.

Rubenking, N., 1992. Turbo Pascal for Windows: Techniques and Utilities. Ziff-Davis Press, California.

Scholten, H., Hoop, B. and Herman, P., 1990. SENECA 1.2 user manual. DIHO, RWS/DGW, Netherlands. Ecolmod Rapport EM-4.

Silvert, W., 1993. Object-oriented ecosystem modelling. Ecol. Model., 68: 91–118.