available at www.sciencedirect.com

www.elsevier.com/locate/ecolinf

Different modelling tools of aquatic ecosystems: A proposal for a unified approach

António Pereira[a,*], *Pedro Duarte*[a], *Alain Norro*[b]

[a]*CEMAS – UFP, University Fernando Pessoa, Praça 9 de Abril, 349, 4249-004 Porto, Portugal*
[b]*MUMM, Management Unit of the Mathematical Model of the North Sea, Royal Belgian Institute for Natural Science, Gulledelle, 100. B-1200 Brussels, Belgium*

ARTICLE INFO

Article history:
Received 17 February 2006
Received in revised form
14 September 2006
Accepted 16 September 2006

Keywords:
Ecological modelling
Object oriented modelling
Couple physical–biogeochemical modelling

ABSTRACT

Over the last few decades, several modelling tools have been developed for the simulation of hydrodynamic and biogeochemical processes in aquatic ecosystems. Until late 70's, coupling hydrodynamic models to biogeochemical models was not common and today, problems linked to the different scales of interest remain. The time scale of hydrodynamic phenomena in coastal zone (minutes to hours) is much lower than that of biogeochemistry (few days). Over the last years, there has been an increasing tendency to couple hydrodynamic and biogeochemical models in a clear recognition of the importance of incorporating in one model the feedbacks between physical, chemical and biological processes. However, different modelling teams tend to adopt different modelling tools, with the result that benchmarking exercises are sometimes difficult to achieve in projects involving several institutions. Therefore, the objectives of this paper are to provide a quick overview of available modelling approaches for hydrodynamic and biogeochemical modelling, to help people choose among the diversity of available models, as a function of their particular needs, and to propose a unified approach to allow modellers to share software code, based on the object oriented programming potentiality. This approach is based on having object dynamic link libraries that may be linked to different model shells. Each object represents different processes and respective variables, e.g. hydrodynamic, phytoplankton and zooplankton objects. Some simple rules are proposed to link available objects to programs written in different source codes.

1. Introduction and objectives

Models of aquatic ecosystems include biogeochemical processes, such as photosynthesis, nutrient cycling and grazing and transport processes. The former is responsible for local changes of state variables, such as concentration of chemical constituents and biomass of different species or groups of species. The latter are responsible for the transport of pelagic variables across model domain and across model boundaries. As discussed below, there are different approaches to simulate each set of processes.

Many mathematical models used in ecology are based on questionable simplifications, implicit or ambiguous assumptions and not on generally accepted theories. This leads to highly uncertain model results because of the uncertainty associated with model parameters and inputs and, sometimes, the uncertainty in model structure (Scholten and Van der Tol, 1998). In the field of aquatic ecosystem modelling, different models may include different processes, or the same processes described on a different way with more or less detail. The degree of detail may be determined by the importance assumed for the different processes in a particular

* *Corresponding author.*
E-mail addresses: apereira@ufp.pt (A. Pereira), pduarte@ufp.pt (P. Duarte), a.norro@mumm.ac.be (A. Norro).

1574-9541/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.ecoinf.2006.09.004

ecosystem and also by available knowledge. For example, some mathematical models use a simplified description of hydrodynamic transport processes and a very detailed description of benthic biologic processes (e.g. Baretta and Ruardij, 1988), whereas other models include very detailed hydrodynamics and relatively simple ecosystem biological model (e.g. Luyten et al., 1999). Even when such a basic and important process as photosynthesis is considered, it is hard to find many models using exactly the same approach – several different mathematical formulations may be used to describe the relationship between light intensity and productivity (e.g. Duarte, 2005). Recent theoretical developments may help to define more generally applicable modelling "rules" as the Dynamic Energy Budget Theory (Kooijman, 2000).

When an ecological model is built, the uncertainties and variability mentioned above are also reflected on model implementation, reason why there are so many different models developed by different research teams all over the world. Typically, different modelling teams tend to adopt different modelling tools, such as structured programming, the most common approach, (e.g. the EMS Dollard model described by Baretta and Ruardij, 1988; COHERENS model described by Luyten et al., 1999) or object oriented software (e.g. EcoWin, developed by Ferreira (1995), MOHID (on line)).

Models based on structured programming consist of a main program, where some general state variables describing the ecosystem under simulation are defined and calls are made to several sub-routines, at each model time step. These sub-routines calculate all processes represented in the model and the fluxes that affect each state variable. At the end of each simulation cycle, all state variables are updated as a function of the mentioned fluxes. It some cases, sub-routine calculations depend on general scope state variables that are defined and calculated elsewhere in the code. When this happens, it is difficult to reuse these sub-routines from other source codes. Apart from these limitations and depending on the compilers used, it may be difficult to combine sub-routines written in different source codes to build new models.

When object-oriented software is chosen, a possible approach is to define objects[1] representing several groups of variables and processes that may correspond roughly to the sub-routines of the previous approach. However, there is no need to use "general" state variables, because each object should handle its own variables. The advantage here is that by eliminating general scope variables it is easier to avoid programming errors, when a variable is changed at several different sites in a program. Apart from that, objects have several properties that make them very useful in ecological modelling: modularity, inheritance and polymorphism (Borland, 1988).

Modularity implies that objects are self-contained programming code. Inheritance means that it is possible to define "children" and "grandchildren" objects that inherit variables and routines from parent objects. Polymorphism means that objects may have different "shapes". For example, one may define a phytoplankton object that has a set of parameters and procedures to change its state variables (e.g. biomass) by simulating relevant physiologic processes. However, for the implementation of a particular model it may be necessary to include several phytoplankton species, differing in the values of some physiologic parameters, everything else being equal. In this situation, it is not necessary to create code or variables for each new phytoplankton species. Using polymorphism, several dynamic instances of the same object may be created, each with their own parameter values and corresponding behaviour. There is no need to create more state variables, as it would be the case in a structured dynamic model (at least one per species), because each instance of the phytoplankton object has pointers to the corresponding state variables. Therefore, using the same objects, it is possible to build models of different complexity. Furthermore, if it becomes necessary to change some calculation procedure for one of the modelled species, a descendant object may be easily created by overloading only the procedure that has to be changed, everything else being inherited from its ascendants.

Objects may also interact between each other. For example, a zooplankton object may graze a phytoplankton object after inspecting its properties, such as biomass, and "informing" phytoplankton how much will be grazed. This is done by having methods in each object that are responsible to show "public" properties (e.g. biomass) and methods that are responsible to change them as when phytoplankton is grazed by zooplankton (e.g. Ferreira, 1995).

Considering that models are increasingly used in environmental sciences, namely for impact assessment studies, and that in some instances their usage is already predicted in environmental laws as some of the European Union Framework Directives (e.g. Water Framework Directive, European Commission (2000)), it is predictable that in recent future there will be some terms of reference regarding the way different processes are simulated for aquatic ecosystems. Furthermore, authors' experience though participation in different international research projects shows that:

(i) There is not one modelling software suitable for solving all simulation challenges.
(ii) Different research teams, used to work with a specific modelling software, generally do not accept working with other tools for a particular project. This is quite understandable, given the time necessary to handle properly a new modelling tool.

Both (i) and (ii) are a good justification of why it may be so difficult to perform benchmarking exercises with different models.

Following the previous paragraphs, the objectives of this work are:

(i) To provide a quick overview of more common modelling approaches for hydrodynamic and biogeochemical modelling;
(ii) To propose a unified approach to allow modellers to share software code, based on the object oriented programming potentiality.

[1] In the object oriented programming language, the term "class" means the software structure of an "object", which corresponds to the memory allocation of a "class".

2. Historical perspective

2.1. Hydrodynamic modelling with reference to available three-dimensional code

The development of three-dimensional (3D) hydrodynamic models started in the late 70's. Three-dimensional hydrodynamic models are numerical code solving generally the Boussinesq equations. The availability of equations based on physical laws of classic mechanics, know from 18th century, is one of the main differences between hydrodynamic and biogeochemistry models, because equations for biogeochemical phenomena are generally not universally accepted and in most cases empirical or semi-empirical.

The main differences between available three-dimensional hydrodynamic models rely mostly on: (i) The representation of the vertical dimension; (ii) the numerical treatment given to turbulence; (iii) the numerical methods used to solve partial differential equations, such as finite difference or finite element methods.

In the so-called "sigma models" the transformation of the vertical dimension developed by Phillips (1957), in meteorology, is used. In this family of "terrain-following" models, the transformed domain is characterised by a flat bottom where it is convenient to apply, for instances, boundary conditions. In such a domain, the vertical sigma velocity computed directly by the model is also the true upwelling velocity that is important for biogeochemistry. When horizontal velocities or other variable of the model need to be known at a given position in the physical space, inverse transformation is needed. The group of 3D models that work without any space transformations are known as "Z coordinate" models.

Because of the difference in scale between horizontal and vertical processes in the ocean, the hydrostatic assumption is currently made in hydrodynamic models. This assumption becomes invalid when coastal models consider very small spatial scales such as less than 50 m on the horizontal dimensions.

It is not the place here to make an intercomparison of all available models. A non-exhaustive list of models is available in Prandle and de Roeck (2005). For the reader focusing on North Sea Shelf applications, Moll and Radach (2003) provide a review of seven 3D models The 3D Marine Coastal COHERENS model was developed within the scope of the European Union Marine Science and Technology Programme during the 90's (Luyten et al., 1999) and made available for the scientific community. COHERENS is a 3D hydrodynamic model solving Boussinesq equation with the hydrostatic assumption on a sigma space. COHERENS is also a model providing great care to solve turbulent phenomena and it purposes numerous different turbulent schemas. Special subroutines cope with the application of boundary conditions. Apart from that, COHERENS includes other facilities, such as particle tracking, sediment movements, biogeochemical sub-models, etc. The subroutine based structure code written in FORTRAN is available with a complete documentation of almost 1000 pages describing all aspect of physics and the code as well as a few examples of uses including input and output files.

As a general matter, COHERENS solves an advection–diffusion equation such as

$$\frac{\partial Y}{\partial t} + \nabla.(uY) + \nabla.(S_Y Y) = [P(Y) - D(Y)] + \nabla.(\nu \nabla Y) \quad (1)$$

where, Y is a given scalar that can be temperature, salinity, or any biogeochemical state variable, u being the velocity vector, S_Y the specific movement of Y and ν the parameter for diffusion. $P(Y)$ and $D(Y)$ are any biogeochemical production and destruction terms, respectively. General discussion as well as all equations can be found in Luyten et al. (1999).

2.2. Ecological modelling with reference to available code

In the next paragraphs, an historical perspective of ecological modelling with reference to aquatic ecosystems will be presented. The authors' objective is not to give an exhaustive review of the work developed over the last decades, but solely to synthesise the most important and common characteristics of available models.

For the purpose of the following discussion, the term "ecological modelling" will be used referring to mathematical modelling of ecosystems including biogeochemistry, with emphasis on nutrient cycling, physiology – individual level processes, such as respiration and feeding – population level processes, such as mortality and recruitment and community level processes, such as predation and competition. Physical processes, such as advection and diffusion, are treated as forcing functions in these models. Ecosystem models consist of compartments, which may be biotic (different biological species or functional groups) or abiotic (e.g. dissolved substances and suspended matter). To build a model it is necessary to define compartments and then specify the equations governing the transfer of material between those (Taylor, 1993). The choice of compartments is guided by the scientific questions that the model is created to address, the available knowledge about the ecosystem and also the subjectivity of the modellers regarding the importance of different processes and variables.

Following the classification scheme of Gertsev and Gertseva (2004) "ecological models" may be classified as homomorphic, since they group biological functional entities and make several simplifying assumptions about real systems. They may also be divided in stationary and time-dependent models. According to the same authors, they may also classify as "models with lumped parameters", when homogeneity is assumed along spatial coordinates (zero-dimensional models (0-D)), or "models with distributed parameters", when heterogeneity along spatial coordinates is considered (one- (1-D), two- (2-D) and three-dimensional (3-D) models). Further, models may be divided in "continuous" and "discrete", according to the way time is represented, deterministic and stochastic, according to the type of mathematical relationships used, analytic and numeric, according to the way model equations are solved. This work will focus only on homomorphic, time-dependent, distributed, discrete and numeric models, although some 0-D models will also be referred, since in some cases these have been integrated in transport models. In Table 1, several examples of ecological models applied to aquatic ecosystems are given, ranging from 0-D to 3-D models, published between 1988 and 2004, with indication

Table 1 – Examples of aquatic ecosystem models published over the last 17 years (see text)

Authors	Model dimensions	Model application
Baretta and Ruardij (1988)	1-D	Ems Estuary (Netherlands) ecosystem simulation (FORTRAN code)
Fasham et al. (1990)	0-D	North Atlantic mixed layer ecosystem model (FORTRAN code)
Bax and Eliassen (1993)	0-D	Modelling multispecies analysis in Balsfjord (Northern Norway) with SKEBUB model
Taylor (1993)	0-D	North Atlantic mixed layer ecosystem model
Ferreira (1995)	1-D examples	EcoWin – object-oriented ecological model for aquatic ecosystems (C++ code in more recent versions), applied to the Tagus estuary (Portugal) and Carlingford Lough (Ireland)
Lenhart et al. (1995)	3-D	ERSEM II – Ecosystem model of the North Sea (FORTRAN code)
Van der Tol and Scholten (1998)	2-D horizontal	Ecosystem model of the Oosterschelde estuary in the Netherlands (SMOES) (SENECA software)
Fulton et al. (2004)	0-D and 2-D applications	Generic bay ecosystem models

of the software or the programming language used to implement some of them. This is solely a small sample of available models. In fact, there are several published revisions of different types of models applied to aquatic ecosystems (Moll and Radach, 2003). For example, these last authors updated a previous revision by Fransz et al. (1991) of models implemented for the North Sea. They also mention several reviewing works on several types of 3D models (e.g. ecological modelling of fjords, by McClimans et al. (1992), general ecosystem models, by Hofmann and Lascara (1998), modelling of contaminant transport, by GESAMP Joint Group of Experts on the Scientific Aspects of Marine Pollution (1991), just to mention a few). Systematic comparisons of different models require much more than the information gathered in Table 1. Details on the variables and processes considered, both on the pelagic and benthic compartments are important, as well as details on turbulence closure schemes for the hydrodynamic processes, the types of vertical coordinates (Cartesian or sigma), the type of grid (e.g. finite differences or finite elements). However, these details are far beyond the scope of this work and will not be further considered.

Ecological models can be represented as a system of differential equations. To account for spatial heterogeneity, the ecosystem may be divided in model boxes. Box size determines the spatial resolution of the model. Typically, box size in coastal ecosystem models has a scale of hundreds to thousands of meters. Hereafter "compartments" will always be used for model state variables, whereas "boxes" will be used for the morphologic divisions of the model domain.

One common feature to box models is the decoupling between transport and biogeochemical processes. Exchange processes of pelagic variables between different boxes and between the ecosystem and its boundaries may be parameterized on the basis of steady-state balances of conservative state variables (e.g. Baretta and Ruardij, 1988) or from simulations with hydrodynamic models (e.g. Bacher, 1989; Raillard and Ménesguen, 1994; Ferreira et al., 1998). In the last case, the results obtained with the fine grid of a hydrodynamic model are averaged over the borders of the ecosystem boxes and over a particular temporal scale to solve the transport equation (see Eq. (1) above) usually in one or two dimensions. The mentioned decoupling implies that a significant part of the system variability is not accounted for in the model due to spatial and temporal averaging. The relative importance of this variability loss should be analyzed on a case-by-case basis. A common argument is that ecosystem models rarely require the spatial and temporal resolution needed for accurate hydrodynamics (Bird and Hall, 1988). However, the more simplified a model is, in terms of spatial and temporal resolution, the more parameterization is needed to keep it realistic.

Fig. 1a, b and c show the general set ups of three box models presented in Bacher et al. (1991), Van der Tol and Scholten (1998), and Ferreira et al. (1998), respectively. The first model was developed for Marrénnes-Oléron bay (France), the second for the Oosterschelde ecosystem (Netherlands) and the last for Carlingford Lough (Ireland). The first and the last models were developed specifically to estimate carrying capacity for oyster culture. The second model was implemented to evaluate the impact of projected reductions of nitrogen loads on ecosystem carrying capacity for bivalve growth.

There are no specific universal equations that can be used to determine how material is transferred between compartments of an ecosystem model. A general equation of population growth, which can accommodate most of the limiting processes in a closed system, has been formalised by Wiegert (1979).

$$\frac{dX_j}{dt} = \sum_{i=1}^{m} \left(e_{ij}\tau_j p_{ij} f_{ij} X_j\right) - \left(\mu_j + \varphi_j + \rho_j\right) X_j - \sum_{k=1}^{m} \left(\tau_k p_{jk} f_{jk} X_k\right) \quad (2)$$

The first sum represents the assimilated ingestion or uptake by species j from all other modelled species or abiotic sources. The middle term represents losses due to physiologic causes, death or external factors (e.g. grazing) that are not explicitly included in the model. The last summation represents the predation on species j by other species. The coefficients are defined as follows:

e_{ij} is the assimilation efficiency of species j using resource i; τ_{ij} is the maximum specific ingestion/uptake rate of species j; p_{ij} is the preference of species j for resource i; f_{ijn} is the limitation of ingestion/uptake of resource i by species j; μ_j is the specific loss rate due to natural or externally imposed mortality; ϕ_j is the specific loss rate due to excretion; ρ_j is the specific loss rate due to respiration.

These coefficients may depend on a variety of physiologic and behavioural interactions making them non-linear functions of the species or abiotic sources. The equations are not as well-defined as the physical equations of motion, because they are not based on known quantitative laws, as those

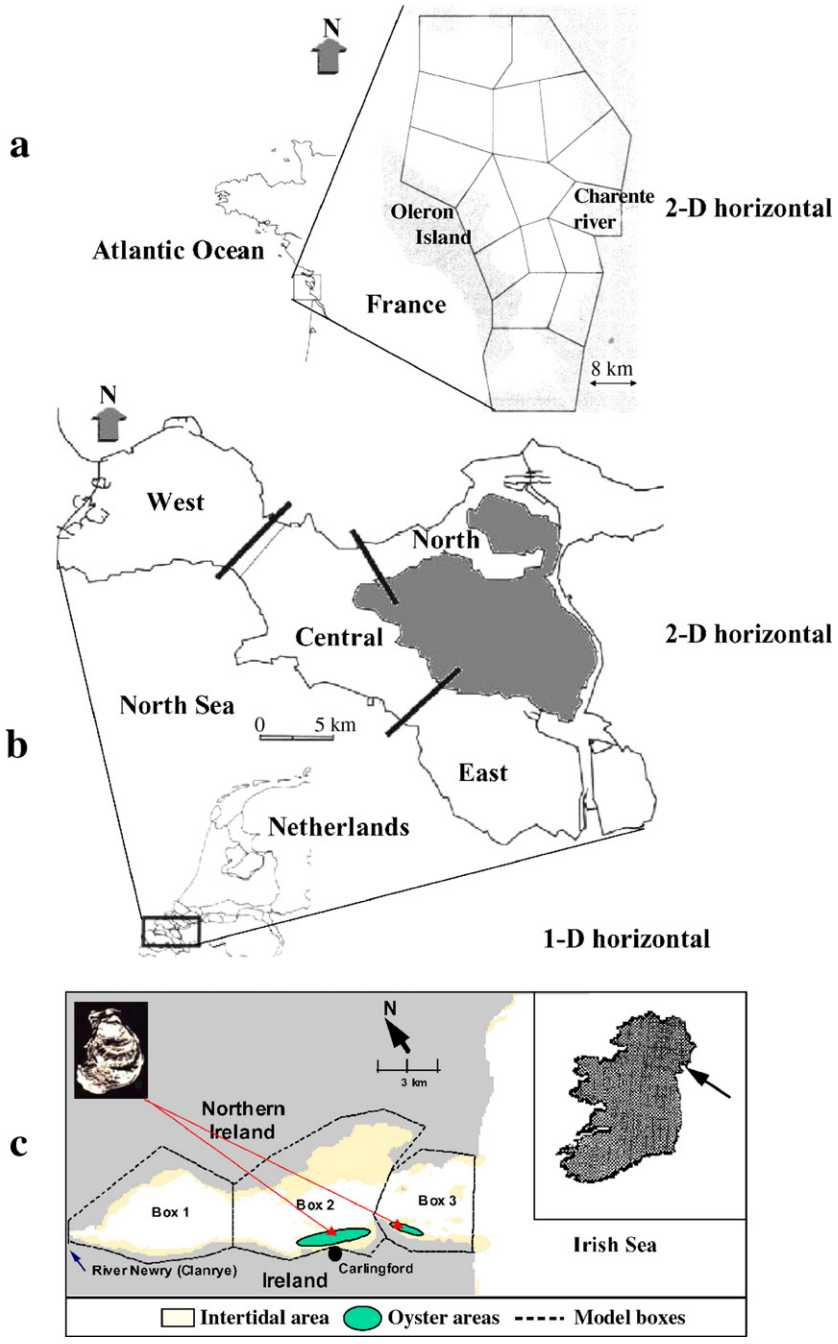

Fig. 1 – (a) Marénnes-Oléron box model (adapted from Bacher et al., 1991), (b) Oosterschelde box model (adapted from Van der Tol and Scholten (1998) and (c) Carlingford box model model (adapted from Ferreira et al., 1998).

available in physics. It is common to simplify most of these coefficients to either constant values, functions of time or space, or functions of the physical forcing (Taylor, 1993). In a distributed model, the above equation must be solved for each model box providing the "sources-sinks" term of the transport equation (see Eq. (1) above).

Some of the models depicted in Table 1 were implemented with software that has been specifically developed for them (e.g. the Ems Estuary Ecosystem Model), whereas, other models were implemented using modelling software designed for box models (e.g. models with SENECA and EcoWin, cf. – Table 1). Typically, FORTRAN models are based on a structured programming approach, with a main program calling subroutines to apply Eq. (2) to model compartments and Eq. (1) (or some simplified form of Eq. (1)) to transport pelagic variables across model boxes. Object oriented modelling software follows a different approach, with objects representing model compartments, encapsulating variables and functions, responsible for the calculation of the terms in Eq. (2). For a review of object properties see Ferreira (1995). The modular nature of objects makes them appropriate for use from different modelling environments.

2.3. *Coupling hydrodynamics and biogeochemistry*

Until late 70's, coupling hydrodynamic models to biogeochemical models was not common and today, problems linked to the different scales of interest remain. The time scale of hydrodynamic phenomena in coastal zone (minutes to hours) is much smaller than those of biogeochemistry (few days). Over the last years, there has been an increasing tendency to couple hydrodynamic and biogeochemical models in a clear recognition of the importance of incorporating in one model the feedbacks between physical, chemical and biological processes. COHERENS model (Luyten et al., 1999) is an example of a coupled model, with subroutines for hydrodynamic calculations and subroutines for biogeochemical processes. EcoDynamo (see below) is an example of an object oriented coupled model.

2.4. *Linking different sub-models and/or different models*

When structured programming is used, different sub-models (consisting of one or several sub-routines) have to be compiled and then linked to the main program. When using object-oriented programming, different objects have to be compiled and then linked to each other. Whilst the compiling process checks for errors in translating the programming code into machine code, the linking process allows the exchange of data at run time among different sub-models, in the former case, objects, in the latter case. Another possible approach would be to have standalone models for various processes and/or variables and let them communicate with each other. In any case, for communication to be possible between different sub-models, objects or standalone models, there must be some communication protocol. When all sub-models or objects are developed by the same person or team, it is normal that there is some a priori agreement about how to interface different parts of the software. However, when several teams develop code independently, things tend to be more difficult, especially when different programming languages are used.

Previous works dealt with the problem of linking different models (e.g. Maxwell and Costanza, 1997; Voinov et al., 2004). The former describes a language for modular spatio-temporal simulation, providing a standard for the development and archiving of simulation models and a spatial modelling environment (SME). The latter authors integrated different Stella (HPS, 1995) models into a landscape model. Stella is a user-friendly modelling environment that does not require any programming skills. Its icon-based module interfaces makes bug-free model development much easier. Prior to integration, Stella models were translated to a format that supported modularity and then incorporated into a spatially-resolved model using the SME, mentioned before, with each model being replicated over a spatial grid.

3. EcoDynamo

EcoDynamo (*Eco*logical *Dyna*mics *Mo*del) is a software application to simulate physical, biogeochemical and anthropogenic processes in aquatic ecosystems. It is an object oriented program application, built in C++, with a shell that manages the graphical user interface, the communications between classes and the output devices, where the simulation results are saved (Pereira and Duarte, 2005).

The simulated processes include:

- hydrodynamics of aquatic systems: water elevations, current speeds and directions;
- thermodynamics: energy balances between water and atmosphere and water temperature;
- biogeochemical: nutrient and biological species dynamics;
- anthropogenic: e.g. biomass harvesting.

The ecosystem characteristic properties are described in a model database with the following files: Morphology file – geometric representation of the model and grid dimensions; Classes file – list of available objects for a particular model, depending on the processes and variables considered; Variables file – list of variable names and initial values for each object; Parameters file – list of parameter names and their values; Loads file – list and location of loads into the model domain. The object hierarchy in EcoDynamo and files used by different objects are depicted in Fig. 2.

The user can choose between file, chart or table to store the simulation results. These output formats are compatible with some commercial software (like MatLab®), enabling their later analysis.

Different objects simulate different variables and processes, with proper parameters and process equations. Objects can be selected or deselected from shell dialogs determining its inclusion or exclusion in each model run.

This application has an interface module that enables communications with other programs for external control. For example, the simulation runs can be controlled by commands like start/stop/pause/restart/step simulation. Simulation activity can be monitored with the help of log files, activated before the simulation run.

The idea of using object oriented programming in ecological modelling goes back at least to Silvert (1993). Ferreira (1995) developed EcoWin and presented a detailed analysis of OOP advantages in ecosystem models. Many aspects of EcoDynamo are very similar to EcoWin. Perhaps, the most important difference is that EcoWin was originally designed for box models, whereas EcoDynamo was designed for coupled hydrodynamic–biogeochemical models. In fact, it is possible to use EcoWin (at least the EcoWin98 version) in this last type of models (cf. – Duarte et al., 2003). However, several changes have to be carried out in the program shell code, mostly related to its box structure. Typically, EcoWin handles a relatively small number of boxes of any size and shape, connected as defined by the user. Therefore, one box may have many connections to a number of other boxes. EcoDynamo domain is defined as a grid, handling up to many thousands of cells of regular size and shape (at present, it handles only Cartesian finite differences grids). Connections between cells are rigidly defined by the matrix grid structure.

EcoDynamo file output is in text format readable by any spreadsheet application, and also in High Density Format (HDF) – useful to handle large volumes of data, such as model

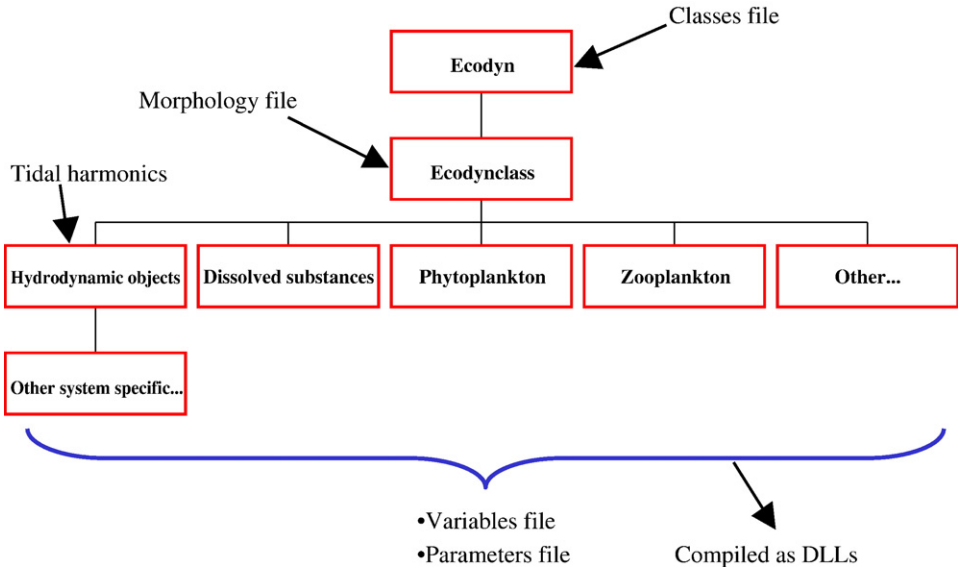

Fig. 2 – EcoDynamo general object structure and files required by different objects (see text).

grid outputs. In EcoDynamo, different objects are controlled by different threads, allowing multiprocessing and increasing simulation speed.

EcoDynamo objects include wrapper functions for easy interface with other programs (see below). Therefore, objects were designed as self-contained code that may be used outside EcoDynamo environment, as external libraries. This makes it possible for people used to different modelling environments to continue using their preferred tools, whilst at the same time they may calculate some processes and variables with EcoDynamo objects.

EcoDynamo allows the user to run parts of model domain (sub-domains). These sub-domains may be defined with a Geographical Information System (GIS) application. In fact, GIS may be used to impose changes in some model grid parameters, such as cell depth. These changes are integrated in the Morphology file (see above) and may then be tested with the model.

3.1. *Description of the structure of EcoDynamo objects*

EcoDynClass is the base simulation object used by EcoDynamo. All the simulation objects inherit from this one. It reads model morphology, initialises relevant fields and implements the default behaviour of the public methods that can be inherited. This object controls the model time step evolution and, as ancestor object of all others present in the simulation, it knows how many objects exist in the simulation and their relationships.

The general outline of the running process is as follows: The shell invokes all active objects using the "Go" method. Hydrodynamic object calculates velocity fields and water elevations, other objects calculate local changes of their variables at each grid cell. These local changes are partial derivatives that are integrated by the Integrate method. After this first round of calculations is completed, the hydrodynamic object transports all "transportable" variables across grid cells. Finally, cell geometry is updated by the Reinitialise method.

The public methods that all objects inherit and must rewrite are:

Go – invoked at each model time step, responsible for all object calculations (as in EcoWin, Ferreira (1995))
Integrate – responsible for time integrations within each grid cell calculations (as in EcoWin, Ferreira (1995))
Update – update one internal variable value, requested by an external object
Inquiry – send to an external object the value of one internal variable
Reinitialize – update velocities, flows and system geometry to the next time step.

EcoDynamo performs the simulation as a cyclic loop. In each cycle:

"Go" is invoked for all the objects
Hydrodynamic object can adjust the model time step
"Integrate" is invoked for all the objects
"Reinitialize" is invoked for the hydrodynamic object.

The constructors of each object inherit from the EcoDynClass (which "knows" the system morphology) and are responsible for the initialisation of the particular variables and parameters from the database files. Each object type is built as a Dynamic Link Library (DLL) and can be integrated at run-time by the EcoDynamo shell.

4. Interfacing code from different modelling software

In the next paragraphs, a methodology is proposed allowing to integrate EcoDynamo objects in other modelling softwares. Since most available modelling code is written in Fortran and

EcoDynamo objects are written in C/C++, emphasis will be in interfacing these programming languages.

4.1. *Mixing code in C, C++ and Fortran with GNU 'g++', 'gcc' and 'g77' compilers*

The easiest way to mix code written in Fortran 77 and C/C++ languages is to use compilers that are compatible in the object code generated. This could be done with GNU compilers, namely those belonging to the MinGW project (Minimalist GNU for Windows [http://www.mingw.org]). According to the project page, it supplies a collection of freely available and freely distributable Windows specific header files and import libraries, combined with GNU toolsets, allowing producing native Windows programs that do not rely on any 3rd-party C runtime DLLs.

4.1.1. *Call by value/call by reference*

In Fortran language, arguments are passed by reference whereas in C and C++ languages, arguments are passed both by value and by reference. This means that the normal way Fortran subroutines and functions are called allows the modification of their argument variables inside the subroutine/function code, while C and C++ use a slightly different syntax to allow the modification of argument variables. To allow code mixing between Fortran and C/C++ languages all the shared subroutines/functions/methods should pass arguments by reference.

4.1.2. *Splitting code into multiple files*

Normally the program source code is separated into several files:

- One per function or subroutine (Fortran).
- One per function types (C).
- One per class (C++).

Each source code is compiled into an object file (a '.o' file in Unix or a '.obj' file in Windows), and the various object files are linked together into a final single executable.

The advantages of splitting code up this way include:

- The possibility to use different languages for different portions of the program.
- The possibility to have different programmers writing different functions.
- More efficient compilation, since a change to one source file only requires its object file to be recompiled and the object files to be relinked, rather than recompiling the entire body of code from scratch.

The use of the *make* utility to automate the build process of the program is a common practice when the source code exceeds three or four files. Definitely it is the best way, with only one command, to rebuild the necessary object files when one or more changes are made in the source files. In addition, it enables the name control of the object files, independently of the operating system in use.

Nowadays, the Integrated Development Environments (IDEs) and the drag-and-drop facility enable the inclusion or exclusion of source files in the program code very intuitively. When the use of different languages in different parts of the program is intended, it's advisable to manipulate the make utility directly to control program building with more detail.

4.1.3. *Internal symbol names*

When the source code is compiled and turned to object files, the compiler usually change the internal name of the variables, functions or subroutines by appending/prepending underscores or other symbols. The GNU compilers used in this study add a single underscore to all the names used by the code.

The Fortran language is case insensitive: the compiler converts all the symbols to lowercase letters. Additionally, the Fortran compiler appends a single underscore to each symbolic name or, if the name has yet an underscore, appends a double underscore.

For interoperability between C, C++ and Fortran, the interface functions must have lowercase names and use the C-style interface. This means that all the C++ interface functions must have an 'extern "C"' directive.

4.1.4. *Examples*

The next examples will help to understand what is necessary to do in the source files to interface conveniently the three languages. All the examples use three subroutines: one defined in C++ language (cppfunc), another defined in C language (cfunc) and the last one defined in Fortran language (ffunc). Each subroutine accepts one variable as argument and changes its value inside the subroutine.

The main program defines one variable (x) and calls previous subroutines to change that variable's value. Each example has the main program written in a different language.

4.1.4.1. *Example 1: main program in C with subroutines in C, C++ and Fortran.* C requires the subroutines "call by reference" syntax to make the changes in the variable persistent.

The invocation of the Fortran function by the main program is made with 'ffunc_'.

The file with the "cppfunc" function must define its prototype enclosed by the 'extern "C"' directive indicating that it is called from a C-style interface. The source files are shown in Boxes 1 to 4.

Box 1

File: cprog.c (main program)

```
#include <stdio.h>
int main ( )
{
   float x;
   x = 1.0;
   printf ("Before running C function: x = %f ", x) ;
   cfunc (&x) ;
   printf ("AFTER running C function: x = %f ", x);
   cppfunc (&x) ;
   printf ("AFTER running C++ function: x = %f ", x) ;
   ffunc_(&x) ;
   printf ("AFTER running FORTRAN function: x = %f ", x) ;
   return 0;
}
```

Box 2

File: cfunc.c

```
void cfunc(float *a)
{
    *a++;
}
```

Box 3

File: cppfunc.cpp

```
extern "C" {
        void cppfunc (float *a) ;
}
void cppfunc (float *a)
{
    *a += 2;
}
```

Box 4

File: ffunc.f

```
SUBROUTINE FFUNC (A)
A = A + 3
END
```

To compile and link the program the easiest way is to write the file for the *make* utility (Box 5).

Each program file is compiled into an object file using the appropriate compiler and the –c flag. The linker invoked is the default C linker. It's assumed that all files are in the same directory.

Box 5

File: makefile

```
cprog: cprog.o cfunc.o cppfunc.o ffunc.o

    gcc –o cprog.exe cprog.o cfunc.o cppfunc.o ffunc.o
cprog.o: cprog.c
    gcc –c cprog.c –o cprog.o
cfunc.o: cfunc.c
    gcc –c cfunc.c –o cfunc.o
cppfunc.o: cppfunc.cpp
    g++ –c cppfunc.cpp –o cppfunc.o
ffunc.o: ffunc.f
    g77 –c ffunc.f –o ffunc.f
```

Box 6

```
Before running C function: x = 1.000000
AFTER running C function: x =2.000000
AFTER running C++ function: x = 4.000000
AFTER running FORTRAN function: x = 7.000000
```

To compile the program simply invoke *make cprog*. After the generation of the object and executable files run the program with *cprog*. The results are shown in Box 6.

4.1.4.2. Example 2: main program in C++ with subroutines in C, C++ and Fortran. When the main program is written in C++ it must specify to the C++ compiler that the C and Fortran

Box 7

File: cppprog.cpp (main program)

```
#include <iostream.h>

extern "C" {
    void ffunc_(float *a) ;
    void cfunc(float *a) ;
}
void cppfunc (float *a) ;
int main(void)
{
    float x;
    x = 1.0;
    cout << "Before running C function: x=" << x << endl;
    cfunc(&x) ;
    cout << "AFTER running C function: x=" << x << endl;
    cppfunc(&x);
    cout << "AFTER running C++ function: x=" << x
<< endl << endl;
    ffunc_(&x);
 cout << "AFTER running FORTRAN function: x=" << x
 << endl << endl;
    return 0;
}
```

Box 8

File: cppfunc2.cpp

```
void cppfunc(float *a)

{
    *a += 2;
}
```

subroutines will be called with a C-style interface, and also that the Fortran compiler will append one underscore to the Fortran function name.

Box 9

File: makefile

```
cppprog: cppprog.o cfunc.o cppfunc2.o ffunc.o
    g++ -o cppprog.exe cppprog.o cfunc.o cppfunc2.o
ffunc.o
cppprog.o: cppprog.c
    g++ -c cppprog.c -o cppprog.o
cppfunc2.o: cppfunc2.cpp
    g++ -c cppfunc2.cpp -o cppfunc2.o
```

On the other hand, the "cppfunc" function could be called in a native C++ way. The new source files are shown in Boxes 7 and 8.

To compile and link the new program the file for the *make* utility is changed, adding the new files to compile and link (Box 9).

The linker now used is the default C++ linker.

Box 10

```
Before running C function: x = 1
AFTER running C function: x = 2
AFTER running C++ function: x = 4
AFTER running FORTRAN function: x = 7
```

To compile the program simply invoke *make cppprog*. After the generation of the object and executable files run the program with *cppprog*. The results are shown in Box 10.

Box 11

File: fprog.f (main program)

```
PROGRAM FPROG
REAL X
X = 1.0
WRITE (*, *) 'Before running C function: x=' ,X
CALL CFUNC (X)
WRITE (*,*) 'AFTER running C function: x=' ,X
WRITE(*,*) ' '
CALL CPPFUNC (X)
WRITE (*,*) 'AFTER running C++ function: x=' ,X
WRITE(*,*) ' '
CALL FFUNC (X)

WRITE(*,*) 'AFTER running FORTRAN function: x=' ,X
STOP
END
```

Box 12

File: cfunc3.c

```
void cfunc_(float *a)
{
    *a += 1;
}
```

4.1.4.3. Example 3: main program in Fortran with subroutines in C, C++ and Fortran. In the main Fortran program the C and C++ subroutines are called without any underscore but, as

Box 13

File: cppfunc3.cpp

```
extern "C" {
    void cppfunc_(float *a);
}
 void cppfunc_(float *a)
{
    *a += 2;
}
```

the compiler will append one to the function name internally, in both C and C++ files, the prototype must have an underscore after the function name.

Also the 'extern "C"' directive must be used in the C++ file, indicating that the C++ function is called with a C-style interface.

Box 14

File: cppfunc3.cpp

```
fprog: fprog.o cfunc3.o cppfunc3.o ffunc.o
    g++ -o fprog.exe fprog.o cfunc3.o cppfunc3.o ffunc.o
-lfrtbegin -lg2c
fprog.o: fprog.f
    g77 -c fprog.f -o fprog.o
cfunc3.o: cfunc3.c
    gcc -c cfunc3.c -o cfunc3.o
cppfunc3.o: cppfunc3.cpp
    g++ -c cppfunc3.cpp -o cppfunc3.o
```

The new source files are shown in Boxes 11 to 13.

To compile and link the new program the file for the *make* utility are changed, adding the new files to compile and link (Box 14).

The best way to link this program is with the default C++ linker with 2 special libraries to treat the Fortran symbols (–lfrtbegin and –lg2c).[2]

[2] The program could be generated with the default Fortran compiler (g77) but, in this case, more libraries must be included and the result command will be more complicated.

Box 15

```
Before running C function: x = 1.
AFTER running C function: x = 2.
AFTER running C++ function: x = 4.
AFTER running FORTRAN function: x = 7.
```

To compile the program simply invoke *make fprog*. After the generation of the object and executable files run the program with *fprog*. The results are shown in Box 15.

4.2. *Definition of a "linking protocol" between COHERENS and EcoDynamo*

The next paragraphs describe how to use EcoDynamo objects from Coherens, to provide a real example of the methodology proposed. It is noteworthy that this example may be applicable to other Fortran based models.

4.2.1. *Architectural choice*

To allow Coherens developers to interact with EcoDynamo classes, an architectural choice must be made:

1. Main program in Fortran – Fortran subroutines and functions, calling the EcoDynamo classes using special interface functions, or
2. Main program in C++ – definition of an object-oriented wrapper code in C++ that handles the interaction with Coherens, invoking its functions and passing it relevant data.

The first architecture option seems to be easier to use for Fortran-only programmers.

The idea is to define an interface to EcoDynamo with functions that can be invoked by Fortran, and manipulate the C++ objects (create, use and destroy them, read their properties from permanent storage and call their methods). This is even more challengeable because there is no pointer system in Fortran.

A solution can be found using the concept of "logical units" of Fortran: the C++ interface will generate an integer reference number for all objects manipulated by Fortran. The Fortran code will have to keep a map associating those reference numbers to real objects. The solution proposed associates the address of the object in memory with the reference number in Fortran (the 32-bit integer in Fortran has the same size as a pointer in C).

The compilation phase is straightforward – each source file is compiled with its native compiler. The linking process is a little bit more complex because to have objects instantiated when needed and C++ special mechanisms activated, it is advisable to use the C++ linker. As the main program is written in Fortran, the best way to do this is to link all the files with the C++ linker, adding Fortran libraries as link options, as pointed out in previous example 3.

It is important to notice that this can be very system dependant and caution must be taken before rebuild the executable program in a new platform.

4.2.2. *Coherens using EcoDynamo objects*

4.2.2.1. *Singleton interface class.* For each class from EcoDynamo platform one singleton interface class must be defined. This C-style interface provides a static method that returns the reference address of the object instantiated by the constructor when the first call to that object is performed by the Fortran code.

Every time Fortran code wants to use methods (or read/write data) from that object, the reference must be indicated by Fortran code or, in another way, must be supplied by the singleton interface method.

The singleton interface class must follow the rules:

1. Definition of one public static method that returns the reference address of the class.
2. Definition of one block 'extern "C"' (C-style interface directive) with all the functions that can be called from Fortran:
   a. The names of the functions must be lowercase with underscores appended;
   b. All the parameters must be passed by reference.
3. Changes in the original source code (EcoDynamo C++ sources) must be enclosed by the symbol _PORT_FORTRAN_ to enable compilation in both projects (EcoDynamo application and EcoDynamo/Coherens program).

The makefile that builds EcoDynamo/Coherens program (compile and link) must include the following rules:

1. The source directories of EcoDynamo classes must be added to compilation flags as include directories.
2. The symbol _PORT_FORTRAN_ must be defined in the compilation flags.
3. The Fortran libraries must be added to link command.

The Fortran code must follow the rules:

1. Definition of one integer variable to save the reference address of the class.
2. The first interface function called must be the one that creates the class object.
3. Call the interface functions always with the reference variable.

4.2.2.2. *Example with a light class.* In EcoDynamo there is a class to compute light intensity at sea level and at any depth, as a function of cloud cover, latitude, date and time, using standard formulations described in Brock (1981) and Portela and Neves (1994). Submarine light intensity is computed from the Lambert–Beer law as a function of depth and a water light extinction coefficient. Results from this object are used by other classes, to calculate the water heat budget and photosynthetic rates. The Light Class was chosen to exemplify the usage of EcoDynamo classes from the Coherens code.

4.2.2.2.1. *Header file in C++ code.* To provide an interface to the Light class (TLight symbol), the header file of the class must include the following major changes:

1. Inside the class definition, add one public static method that returns the reference address of the class (Box 16).

2. Outside the class definition, add one 'extern "C"' with all the functions that can be called from Fortran. These functions must reflect all possible interactions between Coherens code and EcoDynamo class. As an example see Box 17.

4.2.2.2.2. *Source file in C++ code.* The source files of Light class must implement the methods referred in the header file.

As an example see Box 18.

4.2.2.2.3. *Invoking C++ code from Fortran.* To use the Light class from Coherens code it is necessary:

1. Declare one integer to store the light class reference in memory and invoke the function that build the Light object[3] (Box 19).

Box 16

```
public:
#ifdef _PORT_FORTRAN_
    static Light* getLight(TLight* plight);
#endif
```

2. At each time step the Light object must update its internal values ("Go" method) to be used by Coherens (Box 20).

When invoking the "build", relevant parameters must be sent, depending on the object in usage. In this case, some of the mentioned parameters are latitude (DLAT) cloud cover (CLOUD2) and the light extinction coefficient (KVALUE)). When function "LIGHT GETVALUES" is invoked, light intensity values are returned by the C++ object for each grid cell (coordinates I, J, K), defined in a 3D Coherens model, as a function of its depth (parameter PDEPTH). The reference of the Light object (in this example) can be used by Fortran to pass it to another C++ object (for instance, the WaterTemperature object) (Box 21).

Box 17

```
/* Functions that can be called from Fortran */
#ifdef _PORT_FORTRAN_
extern "C" {
    void light_(int* plight, int* nc, int* nr, int* nz,
        float* latitude, float* kvalue, float* depth,
        float* sigma, float* cloudcover, float* airtemperat);
    void light_go__(int* plight, float* curtime, float*
julianday);
    void light_getvalues__(int* plight, int* ic, int* ir, int* iz,
        float* totallight, float* parlight,
        float* parhorizontallight, float* hoursofsun,
        float* horizontallight, float* noonpar,
        float* photicdepth, float* subsurfacelight,
        float* parsubsurfacelight, float* atmosphericir);
}
#endif
```

More than that, several objects can be instantiated only in one call from Fortran (all objects needed to the simulation). One interface function can deal with that, accepting all the mandatory parameters and references, and mixing the relationships with the classes internally.

4.2.2.2.4. *Makefile.* In the makefile, the new C++ sources must be appended to the main program with compilation flags and link options changed accordingly. An example is shown in Box 22.

[3] The LIGHTOBJ variable must have the value 0 (zero) when the program starts.

Box 18

```
#ifdef _PORT_FORTRAN_

/*
    * Singleton provider –TLight class method
    */
TLight* TLight::getLight(TLight* plight)
{

TLight* Plight=plight;
    if (plight == 0)
        PLight = new TLight();
    return PLight;
}
void light_(int* plight, int* nc, int* nr, int* nz,
float* latitude, float* kvalue, float* depth, float* sigma,
float* cloudcover, float* airtemperat)
{
    TLight* ptr;
    ptr = TLight::getLight((TLight*) *plight);
    *plight = (int)ptr;
    ptr–>SetNumberOfColumns(*nc);
    ptr–>SetNumberOfRows(*nr);
    ptr–>SetNumberOfLayers(*nz);
    ptr–>SetLatitude(*latitude);
    ptr–>SetKValue(*kvalue);
    ptr–>SetDepth(*depth);
    ptr–>SetLayers(*sigma);
    ptr–>SetCloudCover(*cloudcover);
    ptr–>SetAirTemperature(*airtemperat);
}
void light_go__(int* plight, float* curtime, float* julianday)
{
    TLight* ptr = (TLight*) *plight;
    int jd = *julianday;
    if (*plight == 0)
        return;
    ptr–>SetCurrentTime(*curtime);
    ptr–>SetJulianDay(jd);
    ptr–>Go();
}
void light_getvalues__(int* plight, int* ic, int* ir, int* iz,
float* totallight, float* parlight, float* parhorizontallight,
float* hoursofsun, float* horizontallight, float* noonpar,
float* photicdepth, float* subsurfacelight, float*
parsubsurfacelight, float* atmosphericir)
{

    TLight* ptr = (TLight*) *plight;
    char* classname;
    int boxNumber;
    double Value;
    char MyParameter[65];
    if (*plight == 0)
        return;
    classname = ptr–>GetEcoDynClassName();

    /*
```

Box 18 (*continued*)

```
                * the Fortran arrays are indexed by layer,
line and column
                */
            boxNumber = (*iz –1)
                + ptr–>GetNumberOfLayers()* (*ir – 1)
                + (ptr–>GetNumberOfLines()*
ptr–>GetNumberOfLayers())* (*ic – 1);
            strcpy(MyParameter, "Total surface irradiance");
            ptr–>Inquiry(classname, Value,
boxNumber, MyParameter, 0);
            *totallight = Value;
            strcpy(MyParameter, "PAR surface irradiance");

ptr–>Inquiry(classname, Value, boxNumber,
MyParameter, 0);
            *parlight = Value;
            strcpy(MyParameter,
"Mean horizontal water PAR irradiance");
            ptr–>Inquiry(classname, Value, boxNumber,
MyParameter, 0);
            *parhorizontallight = Value;
            strcpy(MyParameter, "Daylight hours");
            ptr–>Inquiry(classname, Value, boxNumber,
MyParameter, 0);
            *hoursofsun = Value;
            strcpy(MyParameter, "Mean horizontal water
irradiance");
            ptr–>Inquiry(classname, Value, boxNumber,
MyParameter, 0);
            *horizontallight = Value;
            strcpy(MyParameter, "Noon surface PAR");
            ptr–>Inquiry(classname, Value, boxNumber,
MyParameter, 0);
            *noonpar = Value;
            strcpy(MyParameter, "Photic depth");
            ptr–>Inquiry(classname, Value, boxNumber,
MyParameter, 0);
            *photicdepth = Value;
            strcpy(MyParameter, "Sub-surface irradiance");
            ptr–>Inquiry(classname, Value, boxNumber,
MyParameter, 0);
            *subsurfacelight = Value;
            strcpy(MyParameter,
"Sub-surface PAR irradiance");
            ptr–>Inquiry(classname, Value, boxNumber,
MyParameter, 0);
            *parsubsurfacelight = Value;
            strcpy(MyParameter, "Atmospheric IR");
            ptr–>Inquiry(classname, Value, boxNumber,
MyParameter, 0);
            *atmosphericir = Value;
}
#endif
```

5. Final remarks

The above methodology may be a practical way to link a modelling library with Fortran or C/C++ code, allowing different modelling teams to continue using their usual modelling environment and still having the advantage of integrating code for specific variables and processes, developed by other research teams. The examples above do not imply that these libraries must be implemented following the object oriented paradigm, although the author's opinion in that this may be advantageous due to the object properties described before. The most important aspect is to keep library routines as autonomous as possible without depending on general scope variables calculated in other parts of the modelling code. If this is absolutely necessary then, those variables should be part of the arguments passed to the subroutines. If such a library is established, models can then be built by linking blocks relevant to a particular problem. These would help bringing some uniformity to the different modelling approaches used for similar problems, and could be useful in defining standard approaches to particular environmental problems related to regulatory issues. The embryo of such a library will be available at http://www.dittyproject.org/, including C++ objects to calculate water temperature, light

Box 19

```
INTEGER LIGHTOBJ

C build Light object – called first time only
            CALL LIGHT(LIGHTOBJ, NC, NR, NZ, DLAT,
KVALUE, H2ATC, GZ0, CLOUD2, SAT2)
```

Box 20

```
C run the Light object
            CALL LIGHT_GO(LIGHTOBJ, HOUR, IDATE)
C get values from Light object
    DO 100 I = 1,NC
    DO 100 J = 1,NR
       IF (NWD(J,I).EQ.1) THEN
DO 101 K = NZ,1,-1
LIGHT_GETVALUES(LIGHTOBJ, I, J, K, TLIGHT,
PLIGHT, PHLIGHT,HSUN, HLIGHT, NOONPAR,
PDEPTH, SSLIGHT, PSSLIGHT, ATMIR)
C do what you want with the values
C code must be included here
101 CONTINUE

    ENDIF
```

Box 21

```
INTEGER WATEROBJ

C build WaterTemperature object with Light reference
CALL WATERT(WATEROBJ, LIGHTOBJ, NC, NR, NZ,
H2ATC, GZ0,SAT2, S, RO, WINDU2, WINDV2, T, HUM2)
```

Box 22

```
## makefile adapted to generate Coherens program
# with TLight C++ object class manipulation

#
SRC_HDR = C:/DITTY/EcoDyn_V6
SRC_EDC = C:/DITTY/EcoDyn_V6/EcoClass
SRC_LIT = C:/DITTY/EcoDyn_V6/Liteobjt
SRC_ECODYN = EcoDyn_sources
CFLAGS = -D_PORT_FORTRAN_ -I$(SRC_HDR)
-I$(SRC_EDC) -I$(SRC_LIT)
FC = g77 -cCPPC = g++ -c $(CFLAGS)
LINK32 = g++ -v -o "$@"
LIBS = -lfrtbegin -lg2c
(…)
OFILES3 = testlight.o LiteObjt.o EcoClass.o
OFILESMAIN = $(OFILES1) $(OFILES2) $(OFILES3)
MAINCOM = coherens.exe
(…)
## creating executable code
# main program
$(MAINCOM): mainprog.o $(OFILESMAIN)
$(LINK32) mainprog.o $(OFILESMAIN) $(LIBS)
mainprog.o: $(IFILES) mainprog.f
$(FC) mainprog.f -o $@
(…)
testlight.o: $(SRC_ECODYN)/testlight.inc
$(SRC_ECODYN)/testlight.f$(FC)
$(SRC_ECODYN)/testlight.f -o $@
LiteObjt.o: $(SRC_LIT)/LiteObjt.cpp $(SRC_LIT)/LiteObjt.h;
 (SRC_EDC)/EcoDynClass.h $(SRC_HDR)/ecodyn.rh
$(CPPC) $(SRC_LIT)/LiteObjt.cpp -o $@
EcoClass.o: $(SRC_EDC)/EcoDynClass.cpp
$(SRC_EDC)/EcoDynClass.h
$(CPPC) $(SRC_EDC)/EcoDynClass.cpp -o $@
```

intensity, phytoplankton growth and hydrodynamics, including documentation explaining which parameters are needed to invoke the various objects. It is noteworthy that other approaches referred before (Maxwell and Costanza, 1997; Voinov et al., 2004, cf. – 2.4) are similar in that they allow linking different models, although they do not seem to provide a direct solution for available Fortran code, used in aquatic ecosystem models, since, on one hand, the linking procedure seems to be quite specific for the type of applications presented by mentioned authors and, on the other hand, available Fortran code is also very specific, justifying the methodology proposed in this work.

Acknowledgements

This work was supported by DITTY project (Development of an information technology tool for the management of Southern European lagoons under the influence of river-basin runoff) (EESD Project EVK3-CT-2002-00084). The authors wish to thank two anonymous reviewers for their comments that were useful in improving a first version of the manuscript.

REFERENCES

Bacher, C., 1989. Capacité trophique du bassin de Marennes-Oléron: couplage d'un modèle de transport particulaire et d'un modèle de croi ssance de l'huître Crassostrea gigas. Aquatic Living Resources 2, 199–214.

Bacher, C., Héral, M., Deslous-Paoli, J.-M., Razet, D., 1991. Modèle énergétique uniboite de la croissance des huîtres (Crassostrea gigas) dans le bassin de Marennes-Oléron. Canadian Journal of Fisheries and Aquatic Sciences 48, 391–404.

Baretta, J., Ruardij, P. (Eds.), 1988. Tidal Flat Estuaries. Simulation and Analysis of the Ems Estuary. Springer-Verlag, Berlin, p. 353.

Bax, N., Eliassen, J.-E., 1993. Multispecies analysis in Balsfjord, northern Norway: solution and sensitivity analysis of a simple ecosystem model. Journal du Conseil - Conseil International pour l'exploration de la mer 47, 175–204.

Bird, S., Hall, R., 1988. Coupling hydrodynamics to a multiple box water quality model. Tech. Rep. EL-88-7. Waterways Experiment Station, Corps of Engineers, Vicksburg, MS.

Borland, 1988. Turbo Pascal. Object-oriented programming guide. Borland International, Inc., p. 124.

Brock, T.D., 1981. Calculating solar radiation for ecological studies. Ecological Modelling 14, 1–9.

Duarte, P., 2005. In: Subba Rao, D.V. (Ed.), Photosynthesis-Irradiance Relationships in Marine Microalgae. Algal Cultures, Analogues of Blooms and Applications, vol. 2. Science Publishers Enfield (NH), USA, pp. 639–670. Chapter 17.

Duarte, P., Meneses, R., Hawkins, A.J.S., Zhu, M., Fang, J., Grant, J., 2003. Mathematical modelling to assess the carrying capacity for multi-species culture within coastal water. Ecological Modelling 168, 109–143.

European Commission, 2000. Directive 2000/60/CE of the European Parliament and of the Council of 23 October 2003 establishing a framework for Community actin in the field of water policy. Official Journal of the European Communities L327, 1 (22.12.2000).

Fasham, M.J.R., Ducklow, H.W., McKelvie, S.M., 1990. A nitrogen-based model of plankton dynamics in the oceanic mixed layer. Journal of Marine Research 48, 591–639.

Ferreira, J.G., 1995. EcoWin – an object-oriented ecological model for aquatic ecosystems. Ecological Modelling 79, 21–34.

Ferreira, J.G., Duarte, P., Ball, B., 1998. Trophic capacity of Carlingford Lough for oyster culture – analysis by ecological modelling. Aquatic Ecology 31, 361–378.

Fransz, H.G., Mommaerts, J.P., Radach, G., 1991. Ecological modelling of the North Sea. Netherlands Journal of Sea Research 28, 67–140.

Fulton, E.A., Smith, A.D.M., Johnson, C.R., 2004. Effects of spatial resolution on the performance and interpretation of marine ecosystem models. Ecological Modelling 176, 27–42.

GESAMP Joint Group of Experts on the Scientific Aspects of Marine Pollution, 1991. Coastal modelling. Reports and Studies - GSAMP 43, 187.

Gertsev, V.I., Gertseva, V.V., 2004. Classification of mathematical models in ecology. Ecological Modelling 178, 329–334.

Hofmann, E.E., Lascara, C.M., 1998. Overview of interdisciplinary modelling for marine ecosystems. In: Brink, K.H., Robinson, A.R. (Eds.), The Sea. The Global Coastal Ocean. John Wiley & Sons, Inc., New York, pp. 507–540.

HPS, 1995. Stella: High Performance Systems. http://www.iseesystems.com/softwares/Education/StellaSoftware.aspx.

Kooijman, S.A.L.M., 2000. Dynamic Energy and Mass Budgets in Biological Systems. Cambridge University Press, Cambridge, p. 424.

Lenhart, H.-J., Radach, G., Backhaus, J.O., Pohlmann, T., 1995. Simulations of the North Sea circulation, its variability, and its implementation as hydrodynamical forcing in ERSEM. Netherlands Journal of Sea Research 33, 271–299.

Luyten, P.J., Jones, J.E., Proctor, R., Tabor, A., Tette, P., Wild-Allen, K. (Eds.), 1999. COHERENS – A Coupled Hydrodynamic–Ecological Model for Regional and Shelf Seas. Users Documentation. Mumm Report, Management Unit of the Mathematical Models of the North Sea, p. 914.

Maxwell, T., Costanza, R., 1997. A language for modular spatio-temporal simulation. Ecological Modelling 103, 105–113.

McClimans, T.A., Loed, L.P., Thendrup, A., 1992. Programme on Marine Pollution – Fjord Water Quality and Ecological Modelling State-of-the-Art and Needs. Royal Norwegian Council for Scientific and Industrial Research, Oslo.

MOHID homepage. Available at http://www.mohid.com/. [Visited at December 5th, 2005].

Moll, A., Radach, G., 2003. Review of three-dimensional ecological modelling related to the North Sea shelf system. Part 1: models and their results. Progress in Oceanography 57, 175–217.

Phillips, N.A., 1957. A coordinate system having some special advantages for numerical forecasting. Journal of Meteorology 14, 184–185.

Pereira, A., Duarte, P., 2005. EcoDynamo: Ecological Dynamics Model Application. DITTY Report. Available at http://www.dittyproject.org/Reports.asp.

Portela, L.I., Neves, R., 1994. Modelling temperature distribution in the shallow Tejo estuary. In: Tsakiris, G., Santos, M.A. (Eds.), Advances in Water Resources Technology and Management. Balkema, Rotterdam, pp. 457–463.

Prandle, D., de Roeck, Y.H., 2005. Modelling in Coastal and Shelf Seas-European Challenges. European Science Fondation Marine Board Position Paper, vol. 7. 28 pp.

Raillard, O., Ménesguen, A., 1994. An ecosystem model for the estimating the carrying capacity of a macrotidal shellfish system. Marine Ecology. Progress Series 115, 117–130.

Scholten, H., Van der Tol, M.W.M., 1998. Quantitative validation of deterministic models: when is a model acceptable? The Proceedings of the Summer Computer Simulation Conference, SCS, San Diego, CA, USA, pp. 404–409.

Silvert, W., 1993. Object-oriented ecosystem modelling. Ecological Modelling 68, 91–118.

Taylor, A.H., 1993. Modelling climatic interactions of the marine biota. In: Willebrand, J., Anderson, D.L.T. (Eds.), Modelling Oceanic Climate Interactions. NATO ASI Series, Series I: Global Environmental Change, vol. I. Springer-Verlag, pp. 373–413.

Van der Tol, M.W.M., Scholten, H., 1998. A model analysis on the effects of decreasing nutrient loads on the biomass of benthic suspension feeders in the Oosterschelde ecosystem (SW Netherlands). Aquatic Ecology 31, 395–408.

Voinov, A., Fitz, C., Boumans, R., Costanza, R., 2004. Modular ecosystem modeling. Environmental Modelling and Software 19, 285–304.

Wiegert, R.G., 1979. Population models: experimental tools for the analysis of ecosystems. In: Horn, D.J., Mitchell, R., Stairs, G.R. (Eds.), Proceeding of Colloquium on Analysis of Ecosystems. Ohio State University Press, pp. 239–275.