MOHID 2000 - A coastal integrated object oriented model

R. Miranda[1], F. Braunschweig[1], P. Leitão[1], R. Neves[1], F. Martins[2] & A. Santos[1]
[1] *Instituto Superior Técnico, Portugal.*
[2] *Universidade do Algarve, Portugal.*

Abstract

Mathematical modeling, although a recent science, can be considered old when the time scale used to measure its age is the information technology evolution. FORTRAN 77 has been the most popular programming language among hydrodynamic modelers over the last two decades. This language was adequate to the (low) complexity of the models, limited by the exiting computers. The fast increase of the computing capacity allowed the development of more complex and multidisciplinary models. The necessity of new programming techniques became clear. It was started the era of the Object Oriented programming.

Object Oriented programming isolates different parts of the code and allows communication among them using simple and robust interfaces. MOHID 2000 is an integrated modeling tool programmed in FORTRAN 95, using an Object Oriented strategy. It is the evolution of a set of models originally programmed in FORTRAN 77. Taking advantage of the new language possibilities, with a single code it is possible to perform different simulations (1D, 2D or 3D) and use a variety of vertical discretizations. This technique has also turned easy to integrate hydrodynamic, water quality, sediment transport and pollutants dispersion simulations using eulerian or lagrangian formulations.

Having in mind the development of a tool accessible to the final user, the modeler must think about on the algorithms and code, but also on pre and post processing of the model data. MOHID 2000 input data files can be constructed by a graphical user interface. Output files use the standard HDF format making results exploring simple and allows the use of different graphic packages.

1 Introduction

Modeling teams are nowadays faced with several problems due to increasing complexity of their models. MOHID 2000 development team has identified three main classes of problems when developing its programs, pre-processing problems, post-processing problems and development problems.

As a program (model) grows, due to increasing features and options, it becomes increasingly difficult to use and run. Its inputs become less and less user-friendly raising the mentioned pre-processing problems.

On the other hand, the more complex a model gets, the more complex it gets to explore its results. Now the end-user problem faces a post-processing problem.

Development problems are modelers' problems: the management of a growing code with several developers and ever more features and options, must be supported by a solid basis of software engineering [1].

This paper presents solutions found by the MOHID 2000 developing team for each of the problems above mentioned, discussing the reasons leading to them and how they were implemented.

2 Object-Oriented Programming

FORTRAN is one of the most used scientific and engineering programming languages. On one hand it enables the use of reliable, tested and 'already written' software and, on the other hand, the handicap of learning a new language does not exist [2]. Nevertheless from some years on several limitations were becoming more and more clear. Those limitations revealed themselves in an increasing difficulty to design, develop and maintain, clear, reusable and robust scientific software. Some of the most important drawbacks of FORTRAN 77 were the lack of dynamic storage, the inexistence of user-defined types and the lack of polymorphism.

FORTRAN 90, and later FORTRAN 95, made the use of the Object-Oriented (OO) paradigm in FORTRAN software. Although FORTRAN 9x does not have all the features other OO languages have (*e.g.* inheritance), it is nevertheless possible to write good quality OO programs with it.

2.1 What is an Object?

When developing OO software the gravitational center of every object is information. To start thinking in terms of objects one must split the information (data) into elemental and natural pieces. One step forward would lead us to visualize an object as a physical memory area. The object that owns this information becomes form now on responsible for it and this object alone is authorized to modify it, *i.e.*; object data must be *encapsulated* [3]. When it is necessary for other object to access that information it asks for it to the owner creating a *Client/Server* relationship [4].

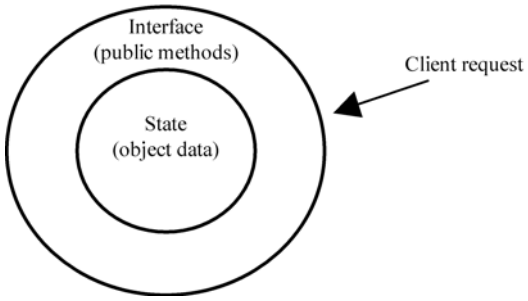

Figure 1: Encapsulation, a client does not have direct access to the server data; the client must request it via a public method.

An object is at the same time the information that it owns and the methods that can operate or access it. Public methods form the interface that client objects use to communicate with the server. An object has well defined boundaries and a state. The *behavior* of an object is the way it reacts to an event, *i.e.*, a client request; the behavior depends on the state of the object [4].

2.2 Object state

Objects from MOHID 2000 system have two kinds of LOCKS: *reading lock* and *writing lock*. When computation is under way the objects lock in writing position; this means that its information is not accessible for clients. When the object is not computing it is idle and clients may request services in the selector area. While a client is accessing data the object is read-lock. An object must wait until no clients are accessing data (it is then read-unlock) to start new computations.

An object is either ON or OFF. When the construct has been called the object is ON and it has associated memory; otherwise it is OFF. Suppose object Transport (A) requests the velocity field from the Hydrodynamic object (B). The message sent from Transport to Hydrodynamic is an event that receives one of four possible answers, depending on the Hydrodynamic state:

- B's state is OFF, *i.e.*, the object is not created yet. An error message will be send to the client. An error handling code, implemented in the client class, will treat this error. It can either order the B to be constructed or stop execution.
- B's state is write-lock, *i.e.*, the server is performing actions over protected data. The client can either wait till the computation is over or stop execution.
- B's state is read-lock or IDLE, *i.e.*, the object is being accessed by other clients or no client is accessing its information. In both cases, B object will give the client access to the desired information.

2.3 Object structure

Objects communicate via messages. Messages are services that servers provide to clients. Server will behave according to its state. Whenever stimulated by a message (event). A method is a piece of code; when a client requests a server service, sending a message, the server performs an action described in a public method [3]. In MOHID 2000 there are four types of methods as shown in Figure 2 [4]:

- Constructors; these methods create new objects, *i.e.*, new instances of a class. It is possible to have more than one constructor.
- Selectors; these are methods that do not change the state of the object, *i.e.*, they are read-only operations.
- Modifiers: these methods change the state of an object, *i.e.*, they are write operations.
- Destructors: these methods free the memory assigned to an object.

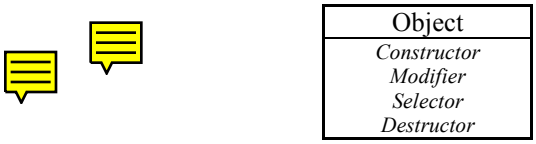

Figure 2: Object's methods organization.

2.4 Difference between Class and Object

A class is an abstraction, it is the software code and it has no associated memory. An object is an instance of a class and it always has associated memory. MOHID 2000 system is OO software so it is possible to create several instances of the same class. One of the obvious advantages of this approach is simplification of nested models; the same code is run as many times as needed, creating the corresponding (nested) model implementation in a straightforward manner.

Objects are dynamically created in run time; it is no longer necessary to compile the program each time it is applied to a new site.

2.5 MOHID 2000 organization

MOHID 2000 uses a layered approach. Table 1 shows MOHID 2000 layer structure. The use of a layered approach ensures that client/server relationship is maintained. A hierarchy is set where lower level subsystems (layers) are servers to upper level systems [4]. Table 1 shows the MOHID 2000 layer structure.

Table 1. Mohid 2000's layer structure.

Level 00	Time, Invert Matrix, LUD, HDF Output, Oxygen Saturation, Triangulation
Level 01	Enter Data, Toga
Level 02	Bathymetry, Water Quality, Discharges, Time Series
Level 03	Horizontal Map, Horizontal Grid, Surface, Wind
Level 04	Geometry, Gauge
Level 05	Map, Open Boundary, Tide Preview
Level 06	Advection Diffusion, Water Quality 3D, Turb1D, BoxDif, Hydrodynamic File
Level 07	Turbulence
Level 08	Hydrodynamic
Level 09	Bottom
Level 10	Free Vertical Movement
Level 11	Water Properties
Level 12	Model

An object placed in a given layer *i* has access to services provided by objects placed in layers bellow, *i.e.*, *i-1* or bellow. As it is shown in Table 1, the object Model is placed at the higher lever; it is this object that controls (synchronizes) the application execution.

Most MOHID 2000 applications use four main objects; most objects in Table 1 are internal classes that act as 'glue' between application objects [4] (*e.g.*, object Model). The most obvious object is Hydrodynamic; there are also WaterProperties, Bathymetry and Time objects.

- The Hydrodynamic class is responsible by the water surface elevation and water fluxes between cells (velocities). Using this class it is possible to choose if computations are to be made (there are several algorithms) or a file to be read from.
- The WaterProperties class is responsible for every intensive property, e.g., salinity, heat (temperature), sediments, phytoplankton, etc. Advection-Diffusion processes and thinks and sources are controlled here.
- The Bathymetry class is responsible by the bathymetry and thus provides essential information concerning the grid formation.
- Time is the class that keeps information about the execution instant. This class is the core of the synchronism system.

3 Pre-processing

The OO strategy used in the development of MOHID 2000 permitted to easily create a Graphical User Interface (GUI) to supply input data to the model. The main goal of this GUI is to make the model easily accessible to end users (others than members of the multidisciplinary development team).

3.1 Development issues

The first question was in which language to write the GUI? FORTRAN always has been criticized for not being the correct programming language for this kind of application.

MOHID 2000 development team decided to write the GUI in FORTRAN, having in mind two main goals. The first one was to use existing code of the model, once the GUI must perform many tasks also performed during the initialization of the model. These tasks are the validation of the input data, inquiring default values, loading the bathymetry, etc. This way, using the same code for the model and GUI, turned out to be efficient. The second goal was not to change the language, so the whole development team of MOHID 2000, with a large background of FORTRAN programming, would be able to understand GUI code.

Another important fact, which the development team had in mind, was that the existing high-level languages become more and more interoperable: a requirement for the FORTRAN 2000 standard is its complete interoperability with the C programming language.

Another question was "which libraries to use to write the GUI?" The decision was to use the Software Development Kit (SDK) from the *MS Windows* operating system. On one hand this can seem to be a limitation (Operating systems like *Linux* are becoming more and more popular), but on the other hand projects like *TWIN* offer the possibility to execute *MS Windows* based applications on *UNIX* and *Macintosh* platforms.

3.2 GUI requirements

The GUI must handle several tasks. One task is to receive numerical data input from the user. In the MOHID 2000's GUI numerical data input can be supplied in windows based dialog boxes. After successful numerical data input the data is written into ASCII files which are read by MOHID 2000. In these files, every numeric data field is preceded by a keyword, allowing these files to be organized in any order.

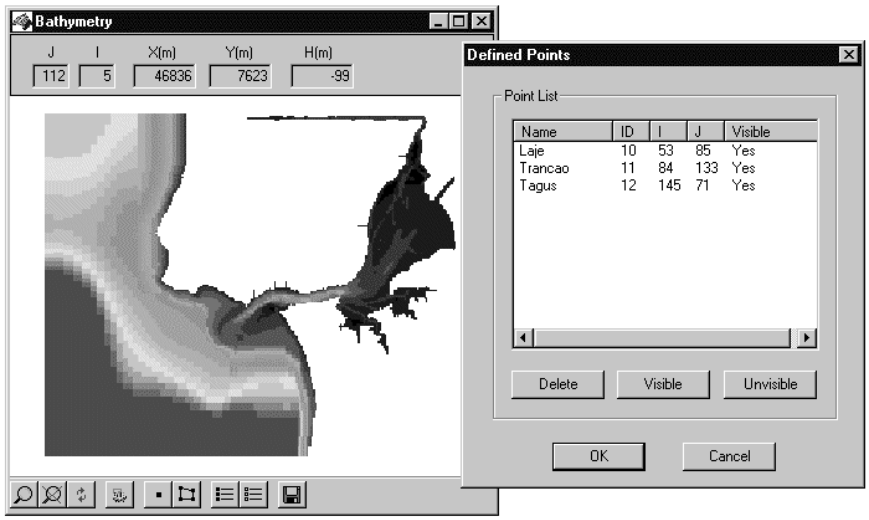

Figure 3: Example of the MOHID 2000's GUI.

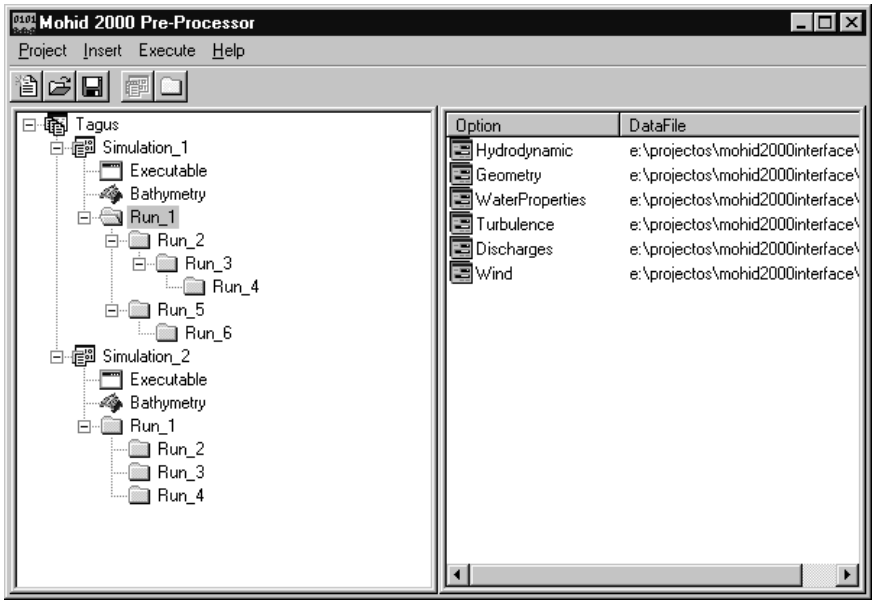

Figure 4. GUI's management module.

4 Post-processing

Like pre processing, post processing of results produced by large computer programs should be, as much as possible, user friendly. Results produced by MOHID 2000, are mainly of two types: results stored as column data (*e.g.* time series) and results stored in matrix format (*e.g.* velocity fields). The first type of results is stored in ASCII files, portable and easy to use by spreadsheet applications (like *MS Excel*). In case of matrix data, the amount of information to store is much bigger and the organization of matrixes inside ASCII files is complex. MOHID 2000 uses the Hierarchical Data Format (HDF) standard data format to write matrix data [5].

HDF software is developed and supported by the National Center for Supercomputing Applications (NCSA) and is freely available. It is used worldwide in many fields, including Environmental Science, Neutron Scattering, Non-Destructive Testing, and Aerospace, to name a few. Scientific projects that use HDF include NASA's Earth Science Enterprise. The main advantages of using HDF are that they are self-describing, very efficient storage, and platform independent. Many visualization tools are available to browse HDF files. The results are usually represented in from of a tree, like that represented in the Figure 5. In Figure 5 a particular array included in the output file (free surface) is also represented.

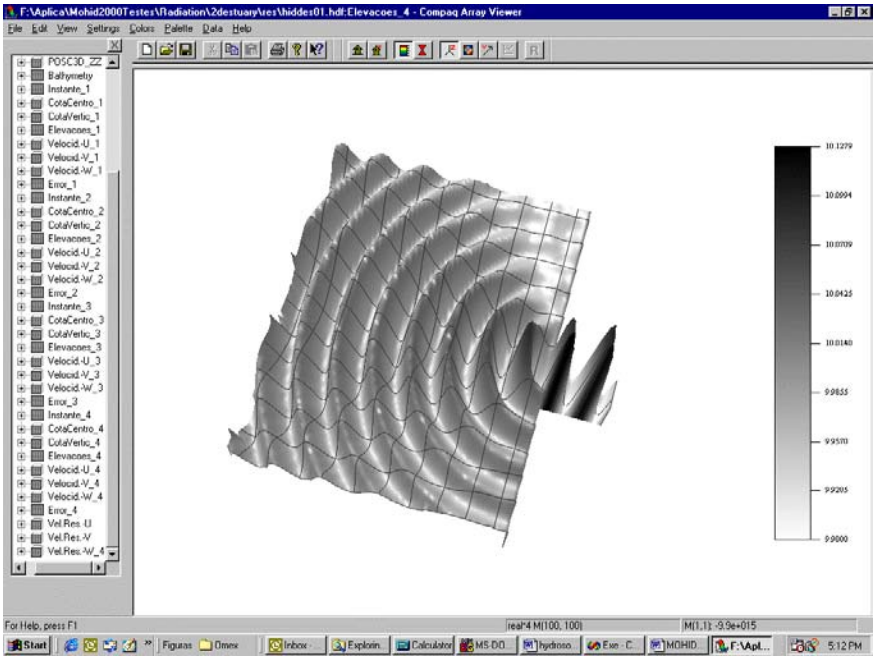

Figure 5. Explore a HDF results file (Compaq Array Visualizer).

5 Conclusions

Reliability and maintainability are the most important features of a well-engineered software system. There is always a trade-off between efficiency and reliability. Inefficiency is predictable and measurable; reliability is not. It is possible to improve the program efficiency hence it should not be the main concern during the development phase. On the other hand, if a program proves unreliable, it is not easy (it may even prove virtually impossible) to enhance it. The best way to assert software quality is reliability rather than efficiency.

The use of OO paradigm to develop large software systems like MOHID 2000 is the most powerful software-engineering available techniques to ensure reliability. Encapsulation, responsibility and client/server relationships between objects prove to be important features assuring that unpredicted software behavior is both minimized and, whenever present, easily located. When an object is found not to function properly, the problematic functionality is either replaced or repaired without interfering with the rest of the program.

Also taking full advantage from the use of FORTRAN 9x a GUI was developed. This choice allowed the use of a programming language already known by the team simplifying Mohid System second generation GUI development.

To solve the output problem the team uses the HDF format. This choice proved to be extremely useful because it allows output file browsing, it is machine architecture independent and there is a broad range of software that can read it.

6 Acknowledgements

This work was developed in the framework of OPCOM project (MAS3-CT97-0089) founded by EC DGXII, which is acknowledged.

7 References

[1] Sommerville, I. *Software Engineering*, Addison-Wesley, London, 1995.

[2] Nyhoff, L.R. & Leestma, S.C., *Fortran 90 for Engineers & Scientists*, Prentice-Hal: Englewood Cliff NJ, 1997.

[3] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. & Lorensen, W. *Object-Oriented Modeling and Design*, Prentice-Hal: Englewood Cliff NJ, 1991.

[4] Duffy, D. *From Chaos to Classes*, McGraw-Hill: London, 1995.

[5] Folk, M. *HDF as an Archive Format: Issues and Recommendations, White* Paper, http://hdf.ncsa.uiuc.edu/archive/hdfasarchivefmt.htm, 1998.