Technical Report
DL-TR-2005-002

A Java GUI and Distributed CORBA Client-Server Interface for a Coastal Ocean Model

R WAIN and M ASHWORTH

JUNE 2005

Council for the Central Laboratory of the Research Councils

Library and Information Services
CCLRC Daresbury Laboratory
Daresbury Warrington
Cheshire WA4 4AD
UK
Tel: +44 (0)1925 603397
Fax: +44 (0)1925 603779
Email: library@dl.ac.uk

ISSN 1362-0207

A Java GUI and Distributed CORBA Client-Server Interface for a Coastal Ocean Model

R. Wain and M. Ashworth

Computational Science and Engineering Department,
CCLRC Daresbury Laboratory,
Daresbury, Warrington WA4 4AD, UK

Abstract

Scientific simulation codes can benefit from a professional Graphical User Interface (GUI) making the use of the program easier and quicker for both technical and non-technical users. We report on recent work on a GUI for the POLCOMS coastal ocean modelling system. In addition to the GUI allowing for parameter setting and the viewing of input and output data, we have implemented a client-server interface allowing real-time monitoring (and potentially computational steering) of the progress of the simulation from a remote workstation. The work utilises Java, Java Swing, VTK and CORBA. We intend that this work should be a stepping stone towards further Grid-related developments for this code.

Keywords

GUI, graphical user interface, CORBA, distributed computing, computational steering, coastal modelling, ocean modelling.

Table of Contents

1 Introduction

This report covers recent work on a Graphical User Interface (GUI) for the Proudman Oceanographic Laboratory Coastal-Ocean Modelling System (POLCOMS). The GUI has been designed to allow the user to view and interact with input and output datasets, to set model parameters, to run the POLCOMS code and to monitor output from the model in real-time.

The work on the GUI focused on the addition of visualisation capabilities as well as the development of a CORBA bridge between the GUI and POLCOMS. This report provides in-depth information on the design, implementation and testing of these additions to the GUI as well as evaluating the work that has been done so far and summarising the possibilities for further work that could be carried out.

A list of useful resources can be found at the end of this report. These resources include on-line tutorials, FAQs, example code, journal articles and books. Most of these have been invaluable references throughout the course of this work and should be used as a first port of call for any information that is not presented here.

2 POLCOMS

The Proudman Oceanographic Laboratory Coastal Ocean Modelling System (POLCOMS) has been developed to tackle multi-disciplinary studies in coastal/shelf environments [1], [3]. The central core is a sophisticated 3-dimensional hydrodynamic model that provides realistic physical forcing to interact with, and transport, environmental parameters. Integrating from ocean to coast, or vice versa, biological production and the fate of contaminants can be determined.

The hydrodynamic model is a 4-dimensional finite difference model based on a latitude-longitude Arakawa B-grid in the horizontal and S-coordinates in the vertical. Conservative monotonic PPM advection routines are used to ensure strong frontal gradients. Vertical mixing is through turbulence closure (Mellor-Yamada level 2.5). The hydrodynamic model can be coupled to an ecosystem model in order to simulate the development of plankton blooms and to study the effects of anthropogenic influences on the marine ecosystem. Such computer models have a wide range of applications, including coastal engineering, offshore industries, fisheries management, marine pollution monitoring, weather forecasting and climate research.

The physical hydrodynamic model and the biological components have been parallelised to take advantage of state-of-the-art high-performance systems. This report concentrates on developments of the model's GUI and its exploitation in a distributed computing environment.

3 Design

At the start of this work there was already a functioning POLCOMS GUI and the major part of the GUI's design had already been implemented. This section of the report provides an overview of the design of communication and visualization components of the GUI. There is then a discussion of the technology decisions that were made as well as a summary of each of the key technologies.

3.1 Overview

In the distributed computing mode of operation, the POLCOMS model runs as a server on a remote machine and the user interacts with a client running on a local PC or workstation. The client-server design was already incorporated into the GUI but had not been fully implemented when this work began. The model can be started in a server mode. In server mode it accepts its input parameters from a connection with the client and returns monitor data by the same route. This was previously implemented by wrapping the Fortran code with a number of C routines, which communicated via TCP/IP sockets. Now the Fortran is wrapped further with C++ and performs the client/server communication using CORBA. Figure 1 shows the structure of the POLCOMS GUI design and the CORBA link to the model.

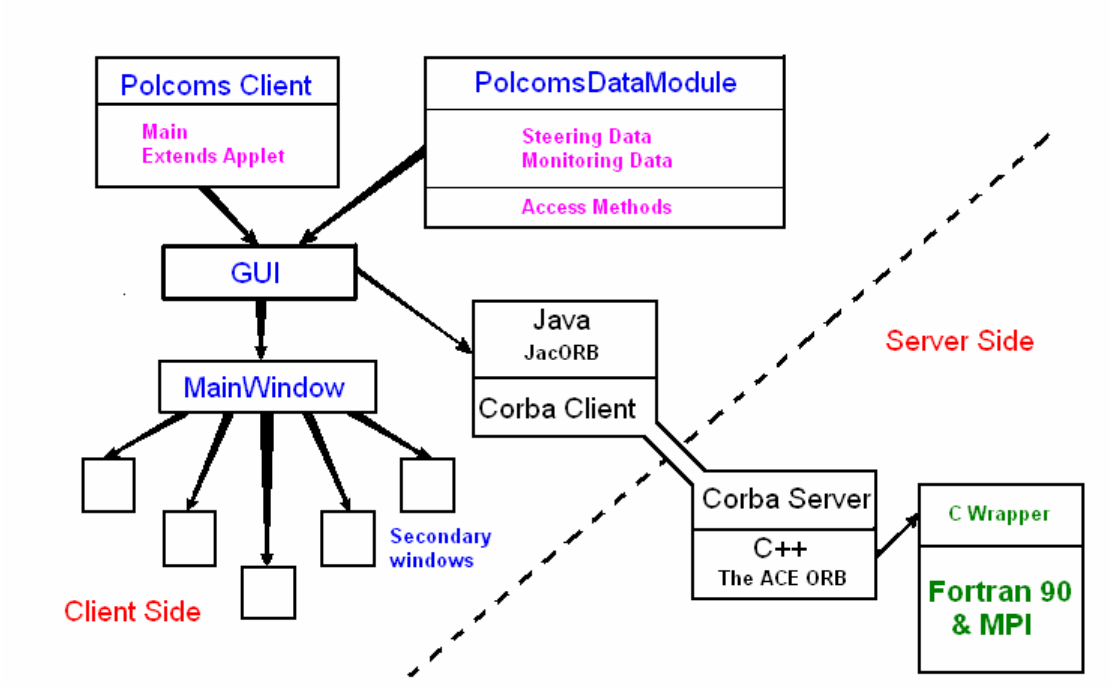

Figure 1: Structure of the POLCOMS Java GUI and CORBA interface.

The visualization part of the GUI needs to display a range of datasets in a number of different ways. These datasets do not share a common format so it does not make sense to design a class that encapsulates the methods required to read, render and interact with all of them. Instead a top-level visualisation class defines methods for the dataset that requires the broadest set of visualisation capabilities on the basis

that most other datasets would need to inherit some methods from this class. The dataset in question in this case is the initial temperature and salinity data for the model.

Initial temperature and salinity data contains one temperature and one salinity value for every point on a three dimensional grid of size l x m x n. The data needs to be viewed in several ways; slices through the 3D grid at a given l, m, or n and X-Y plots of data. The design requires a class to satisfy all of this functionality in such a way that as many methods as possible can be directly inherited by classes for the other datasets.

3.2 Technology decisions

In order to implement the design effectively a number of technology decisions needed to be made. The chosen technologies and reasons for these choices are presented in this and subsequent sections.

Java is a modern, object-oriented programming language that has been designed from the start to facilitate the provision of services over the Internet. Java offers many advantages including robustness, ease of programming and reusability, but the key features for this project are its portability and the access to a rich environment of existing and developing class libraries in the areas of GUIs, distributed computing and Grid technologies [4].

In the Web-based distributed computing environment, we see major benefits in integrating Java with the traditional high-performance programming environment based on Fortran using a client/server model. This vision emphasises the strengths of each component: Java allows easy access to Web-based computing and facilitates the construction of Graphical User Interfaces (GUI); Fortran allows applications best to exploit high-performance hardware. The difficulty comes in providing the bridge between the Java GUI on the client side and the Fortran on the server side. There are various options but none is particularly straight-forward. Weighing up the various factors of communications overheads, maintainability and efficiency it was decided that the following technologies would be implemented in creating the client-server bridge. Firstly the Fortran would be wrapped in a layer of C to allow easy integration of C++ code. Then a C++ ORB (TAO) would provide the means for client-server communication. On the client-side a Java ORB (JacORB) provides easy integration with the java GUI. This configuration avoids the difficulties associated with using JNI to communicate directly between Java and Fortran. The reason for choosing TAO and JacORB was that these ORBs are both mature in their development and include all the CORBA services required for this project. The ORBs have also been tested for interoperability. Even though the CORBA standard is specified with ORB interoperability in mind, it still cannot be assumed that one ORB will recognise all of the capabilities contained within another.

3.3 Java Swing

The Swing package is part of the Java Foundation Classes (JFC) in the Java platform [5]. The JFC encompasses a group of features to help people build GUIs and Swing provides all the components from buttons to split panes and tables. "Swing" was the code name of the project that developed the new components.

Prior to the introduction of the Swing package, the Abstract Window Toolkit (AWT) components in the Java language provided all the user interface components in the JDK 1.0 and 1.1 platforms. Although it is an unofficial name, it is still frequently used to refer to the new components and related API and is immortalised in the package names. Swing is best described as a layer on top of AWT rather than a replacement for it. The Swing package was first available as an add-on to JDK 1.1. Although the Java 2 Platform still supports the AWT components, most experts strongly encourage the use of Swing components.

Swing is a large set of components ranging from the very simple, such as labels, to the very complex, such as tables, trees, and styled text documents. Almost all Swing components are derived from a single parent called JComponent that extends the AWT Container class. Swing components are recognisable because their names start with "J". For example, the AWT class to implement a checkbox is named Checkbox, whereas the Swing checkbox class is named JCheckBox. In addition, the AWT components are in the java.awt package, whereas the Swing components are in the javax.swing package. As a rule, programmers should not use "heavyweight" AWT components alongside Swing components. Heavyweight components include all the ready-to-use AWT components, such as Menu and ScrollPane, and all components that inherit from the AWT Canvas and Panel classes. When Swing components (and all other "lightweight" components) overlap with heavyweight components, the heavyweight component is always painted on top.

The Swing Component Set provides a set of "lightweight" (all-Java language) components that, to the maximum degree possible, work the same on all platforms. Swing provides many standard GUI components such as buttons, lists, menus, and text areas, which you combine to create your program's GUI. It also includes containers such as windows and tool bars. On the downside, you may find that Swing applications are slower and more memory hungry than their AWT counterparts.

3.4 VTK

The Visualization ToolKit[1] (VTK) [6], [7] is an open source, freely available software system for 3D computer graphics, image processing, and visualization used by a large community of researchers and developers around the world. VTK consists of a C++ class library, and several interpreted interface layers including Tcl/Tk, Java, and Python. Although VTK is freely available, professional support and products for VTK are provided by Kitware, Inc. VTK supports a wide variety of visualization algorithms including scalar, vector, tensor, texture, and volumetric methods; and advanced modelling techniques such as implicit modelling, polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangulation. In addition, a number of imaging algorithms has been directly integrated to allow the user to mix 2D imaging, 3D graphics algorithms and data. The design and implementation of the library has been strongly influenced by object-oriented principles. VTK has been installed and tested on nearly every Unix-based platform, PCs (Windows 98/ME/NT/2000/XP), and Mac OSX Jaguar or later.

[1] http://public.kitware.com/VTK/

3.5 CORBA

The Common Object Request Broker Architecture (CORBA) [8], [9], [10], [11] is a low-level architecture established in 1989 by the Object Management Group[2] (OMG). CORBA is an open, vendor independent architecture and infrastructure that applications can use to work together over networks. Using the standard Internet Inter-ORB Protocol (IIOP), built on top of TCP/IP, a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can inter-operate with any other CORBA-based program.

CORBA provides a specification for the Interface Definition Language (IDL). IDL lets developers define interfaces to their programs and objects in a standardized fashion. With the IDL are mappings that map the IDL definitions and types to programming languages such as C, C++ and Java. CORBA offers developers complete language transparency. Developer and vendor objects interact with one another through an Object Request Broker (ORB).

Using the language mappings, developers can create client-side "stubs" and server-side "skeletons" that their ORBs will understand. CORBA applications are composed of objects. For each object type you define an interface in IDL. The IDL interface defines the syntax for the contract that the server object offers to the clients that invoke it. Any client that wants to invoke an operation on the object *must* use this IDL interface to specify the operation it wants to perform, and to marshal the arguments that it sends. When the invocation reaches the target object, the *same* interface definition is used there to unmarshal the arguments so that the object can perform the requested operation with them. The interface definition is then used to marshal the results for their trip back, and to unmarshal them when they reach their destination. The IDL interface definition is independent of programming language, but maps to all of the popular programming languages via OMG standards. OMG has defined standard mappings from IDL to C, C++, Java (Brose et al, 2001), COBOL, Smalltalk, Ada, Lisp, Python, and IDLscript.

The OMG provide a specification – not an implementation. It is up to other individuals, groups and companies to provide implementations.

Fnorb is a CORBA 2.0 ORB for Python [12].

JacORB is a CORBA 2.5 ORB for Java [13].

TAO (The Ace Orb) is a 2.3 ORB for C++ [14].

3.5.1 OMG-IDL

The OMG's Interface Definition Language is used to define an interface to a distributed object. Compilation of this IDL interface produces a different set of classes depending on the language mapping of the particular ORB in question. The tables below explain the files that are produced by the TAO_IDL compiler and the

[2] http://www.omg.org/

JacORB IDL compiler respectively. The names of the files produced are based on the original IDL file name ‘file.idl’.

3.5.2 Using TAO_IDL

Filename	Function
fileC.cpp fileC.i fileC.h	Contain the client-side interfaces. Notice that the inline functions are in a separate file so you can optionally compile them out-of-line for smaller code. Pure clients only need to link the object file generated from `fileC.cpp`.
fileS.cpp files.i fileS.h	Contain the server-side *skeletons*. Servers must link the object files generated from `fileS.cpp` **and** `fileC.cpp`.
fileS_T.cpp fileS_T.i fileS_T.h	Contain the *TIE* classes. These are the standard (after the CORBA 2.2 spec) skeletons based on composition instead of inheritance. They are in separate files only because some compilers cannot handle mixed template and non-template code in the same source file. You **do not** need to compile these files on any platform. However, the files are required to compile fileS.cpp. Also notice that if your platform does not support namespaces, then you may be unable to use the TIE approach for some IDL interfaces.

Table 1: Files generated by the TAO_IDL compiler (compiled using information from ‘A very simple client’ tutorial [15]).

3.5.3 Using JacORB IDL

When compiling IDL files using JacORB IDL it is important to make sure that the resulting *.java files have the correct package declaration. This can be done by using nested ‘module’ statements in the IDL interface, or simply by adding the correct package name after compilation. For example, to add the resulting files to the Polcoms package you must include the line, ‘package Polcoms’ at the beginning of each file.

Filename	Function
file.java	A java interface.
fileHelper. java	Collection of static methods for type specific operations.
fileHolder. java	To allow for the use of out and inout parameters in Java.
fileOperations. java	Java interface containing mappings of IDL types, constants, attributes, exception definitions and operations defined within the IDL interface.
filePOA. java	Skeleton code used with the POA.
filePOATie. java	Used for server-side tie mechanism.
_fileStub.java	Stub code allowing the creation of a client-side proxy for the object implementation.

Table 2: Files generated by the JacORB IDL compiler.

OMG define specifications for the mapping of IDL to a number of languages (C, C++, Java, COBOL, Ada, Lisp, Python, SmallTalk). Most ORB implementations use the C++ or Java mapping. The mapping specifications define how IDL types, attributes and operations are mapped to programming language constructs. OMG's specifications for the IDL to C++ and IDL to Java mappings can be found on the OMG's website.

N.b. There are a number of languages that share the acronym IDL. Several different Interface Definition Languages other than that defined by OMG exist. There is also Interactive Data Language developed by Research Systems Inc. When searching for information about IDL, make sure you bear this in mind.

4 Implementation

This section of the report focuses on the implementation of the GUI design. Firstly, there is an overview discussing the GUI as a whole. There is then a section on the implementation of visualisation routines. Finally, there is a section on the implementation of CORBA communications.

4.1 Overview

The PolcomsDataModule class contains all the steering and model data for the program and includes methods by which they may be accessed. Other classes extend PolcomsDataModule and thereby inherit all the data and methods. The CORBA client is currently incorporated into the class vtkMonitorWindow which controls the sending of parameters as well as the receiving and displaying of

monitoring data. The class vtkVisWindow is the highest level visualisation class defining a visualisation GUI, visualisation pipelines for xy-plots, 3D and 2D views and a number of methods for reading and accessing data. All other visualisation classes inherit from this class.

The POLCOMS code is naturally highly modular and we can separate its functions into the following: the marine domain, the basic hydrodynamics, tides, open boundaries, initial data, meteorological (surface) forcing, sediments, rivers, output datasets, checkpoint/restart, target system, parallel processing options etc. Each module deserves its own class to hold data, variables and parameters, to contain methods to access and manipulate those data, and to contain methods to display and manage GUI panels to allow those data to be updated.

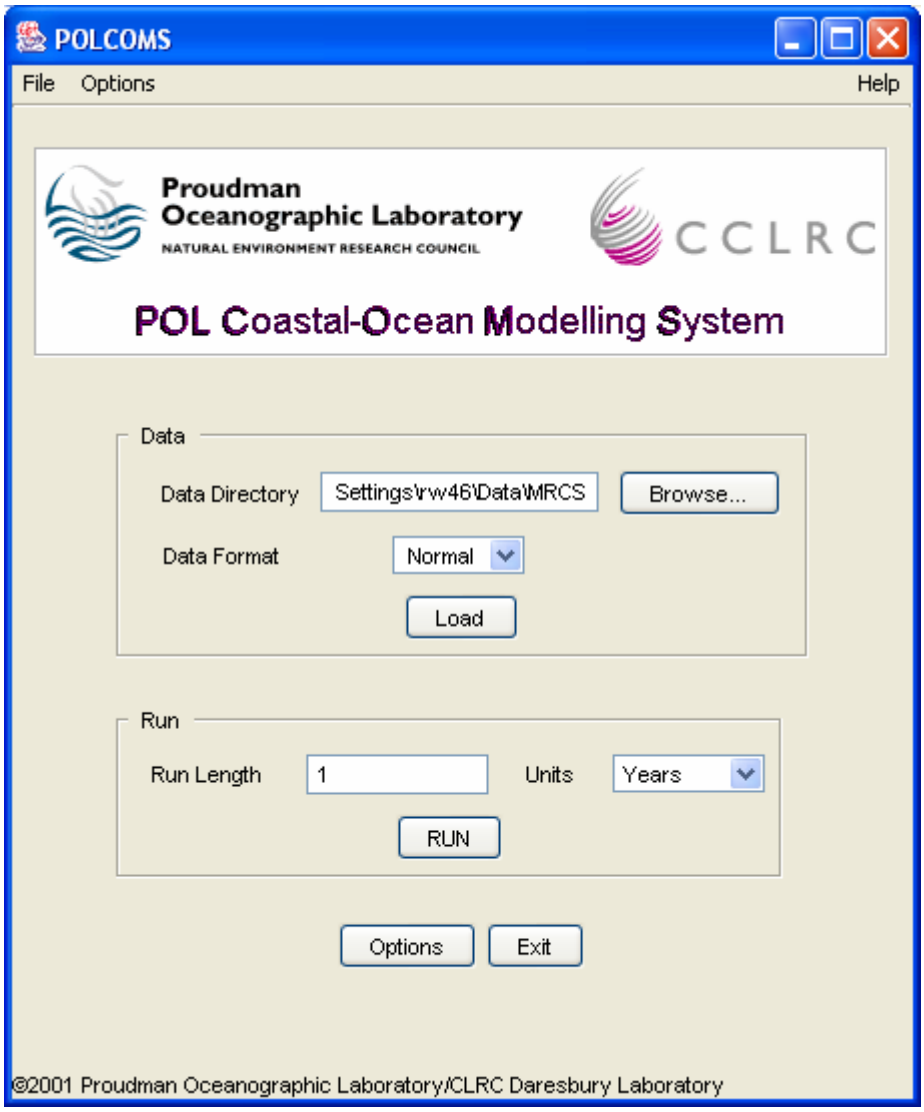

Figure 2: The POLCOMS GUI main window

The main window is shown in Figure 2. The user must select a data directory before loading the input data. A number of options can then be chosen, either through the options menu or by pressing the options button. Most of the options are available via a set of tabbed panes, with one pane for each of the modules referred to above. Many of the options panes contain lists of files that can be visualized by pressing the

'Preferences' button, followed by the 'Visualize' button in the file preferences window. For example, pressing the 'Preferences' button on the 'Domain' pane, followed by the 'Visualize' button in the file preferences window opens the visualization window for bathymetry data shown in figure 3.

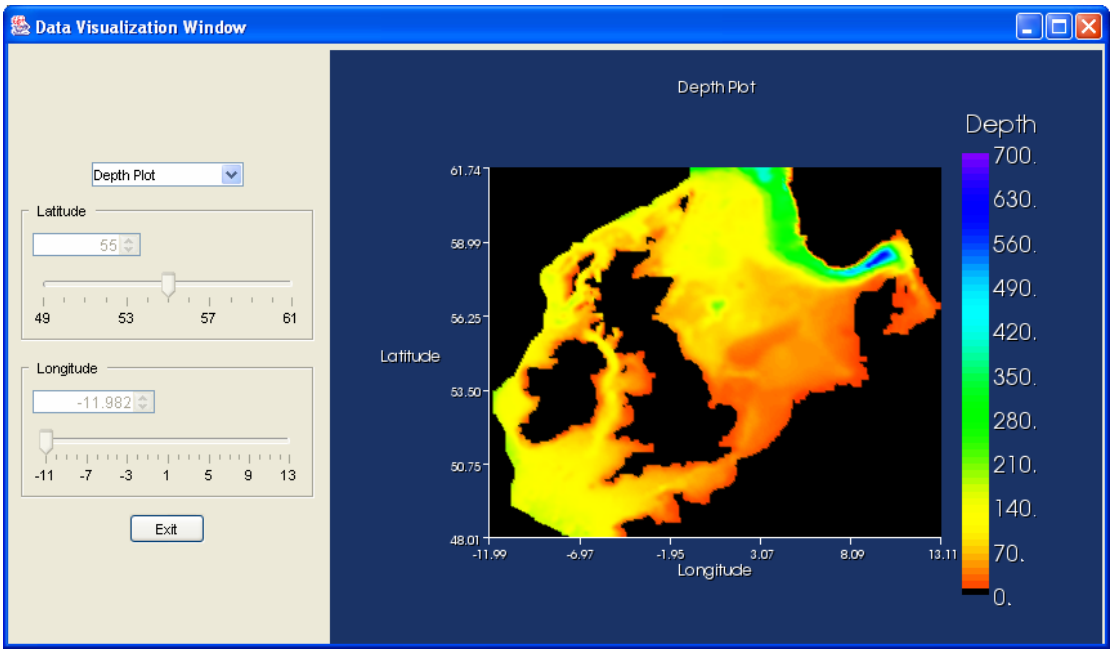

Figure 3: The POLCOMS GUI Visualisation Window, showing ocean depth for a model of the north-west European shelf.

When the model is ready to be run, the user presses the "Run" button, and is presented with the monitor window (Figure 3). In the monitor window the user can set the type and frequency of monitoring data. When the 'Run' button in the monitor window is pressed the GUI communicates the options to the Fortran model code, the model starts and the requested monitor information is sent back via CORBA to the GUI for real-time display.

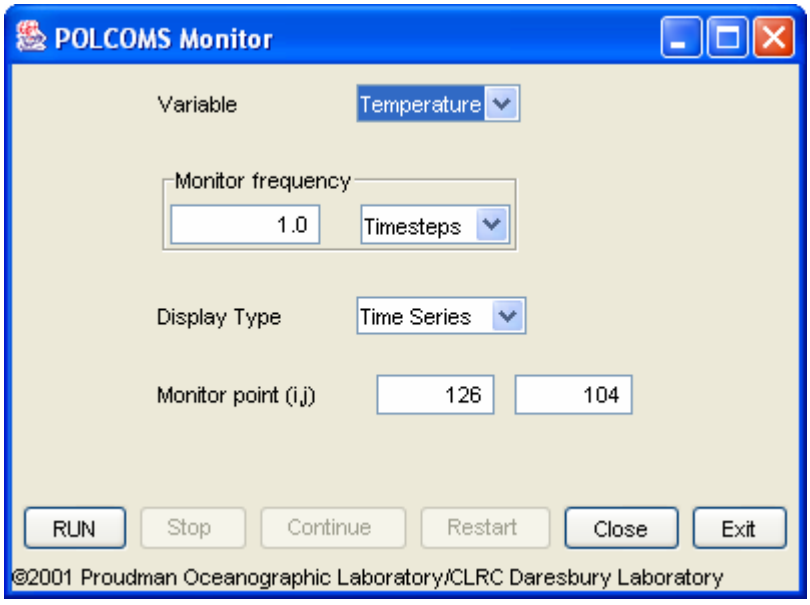

Figure 4: The POLCOMS GUI Monitor Window.

4.2 Visualisation

The input and output data visualisation system within the GUI was developed using the Visualisation Toolkit (VTK). VTK is an open source toolkit written in C with wrappers provided for Java and a number of scripting languages (Python, Tcl/tk). The toolkit provides a good range of visualisation routines and a relatively simple interface to a Java GUI using the 'vtkPanel' class which extends the 'JPanel' JFC Swing class.

The GUI allows a range of data sets to be viewed by the user. Where appropriate the user can also interact with the data, for example, data presented as a 2D map allows for zooming and panning. 3D data sets (Initial temp/salinity) are presented in such a way that the user can choose a 2D slice of the data at a particular latitude, longitude or depth. As the file format and visualisation requirements varied, a separate class was defined to read data files, set up and populate the VTK data structures and layout the GUI for each dataset.

The table below illustrates the classes used to visualise datasets in Polcoms:-

Class Name	**Function**	**Extends**
vtkPolcomsPanel	• Panel containing the render window to which a given visualisation is rendered. Adds to the default window interaction methods defined in vtkPanel.	vtk.vtkPanel
vtkVisWindow	• Provides a GUI for controlling the visualisation of bathymetry and initial temperature and salinity data. • Reads the relevant data files. • Populates VTK data structures. • Sets up VTK actors. • Renders actors.	PolcomsDataSet
vtkRiverVisWindow	• Provides a GUI for visualising river locations on a map as well as xy-plots of flow rate vs time etc. • Reads river data files. • Populates VTK data structures. • Renders data.	vtkVisWindow
vtkSPMVisWindow	• Provides a GUI for visualising SPM source locations on a map. • Reads SPM data files. • Draws map.	vtkRiverVisWindow
vtkOutputVisWindow	• Provides a GUI for visualising Output datasets *(Not yet implemented)*	vtkSPMVisWindow
vtkBoundaryVisWindow	• Provides a GUI for visualising temperature and salinity data on the domain boundary. • Reads temp. and sal. boundary data. • Populates VTK data structures. • Sets up and draws xy-plots.	vtkVisWindow
vtkMetVisWindow	• Currently provides a GUI for displaying	vtkVisWindow

	cloud cover data only.	

Table 3: Visualisation classes for the POLCOMS GUI.

4.3 Communication

The Polcoms GUI utilises CORBA technology to steer and monitor the progress of the Polcoms code. The GUI code includes a CORBA client implemented using JacORB, while the Fortran code has a C wrapper connected to a similar client implemented using TAO (The Ace ORB).

CORBA servers (implemented using TAO) can then be run on any machine that can be seen by both the local machine on which the GUI is running and the remote machine on which the model is running. The configuration used for testing this code had the servers running on the same machine as the model. This was mainly because the TAO libraries were already available on that machine. The client and server sides shown in figure 5 do not correspond to those shown in figure 1. This is because figure 1 related to the client/server design for Polcoms whereas figure 5 relates only to the configuration of CORBA clients and servers.

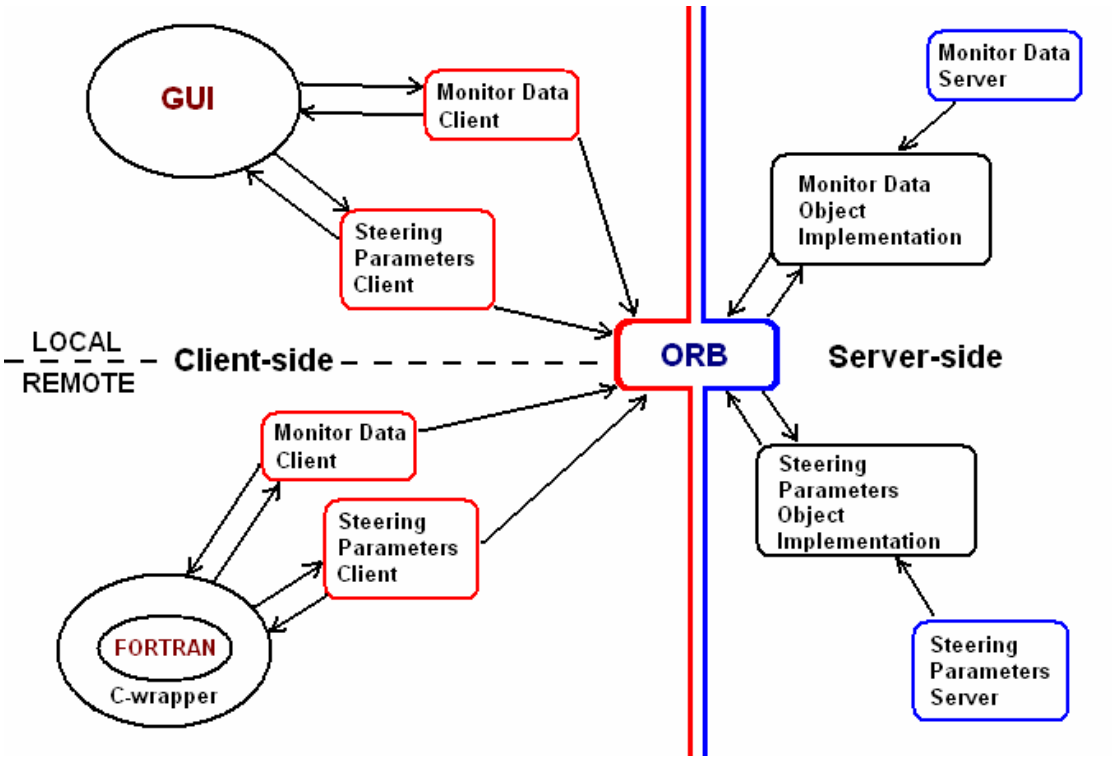

Figure 5: The POLCOMS Client-Server Communication Configuration.

On the client-side steering and monitoring communications are dealt with by a single class called ‘CorbaCommsThread’ that extends the Thread class. The run() method of ‘CorbaCommsThread’ starts a CORBA client that is used for both setting the steering parameters and receiving the monitoring data.

Before the client and server code could be written two IDL definitions were required. One to define the interface to a ‘steering parameters’ object (steerParams) and one to define the interface to a ‘monitoring output’ object (monitorOutput). The following

sections contain a breakdown of these two idl interfaces that form the basis for client/server communications.

4.3.1 Steering

The ‘steerParams’ interface contains no typedefs or attributes. Instead the interface simply defines a list of operations for getting and setting the values of each of the parameters. First the name of the interface is defined.

```
interface steerParams
```

{

Next a list of operations for getting parameters is defined.

```
long getnmonitor();
double getrmonitor();
string getmonvarst();
long getlenmonvarst();
long getmoni();
long getmonj();
long getmonk();
long getldim();
long getmdim();
long getndim();
double gethsmin();
string getbathf();
string getmaskf();
string getts_form();
string getbounts_form();
string getbounzuv_form();
long getlenbathf();
long get……
long get……
double get……
```

…… and so on.

Finally, three operations for setting parameters are defined. One for the model parameters, one for the monitor parameters and one for the input and output filenames (not yet implemented).

```
void set_params(in long l, in long m, in long n, in double
      hsmin, in string bathf, in string maskf, in string ts_form, in
      string bounts_form, in string bounzuv_form, in long lenbathf,
      in long lenmaskf, in long lents_form,in double daldi, in
      double dbedi, in double alat1, in double along1, in double st,
      in double dlt, in long mt, in double tdur, in double avmin, in
```

```
            double dbdzmin, in double q2min, in double ahm, in double cbf,
            in double conv_lim, in double land, in double rstress, in
            double rcloud, in double rsalt, in long ntypes, in double
            bounfreq_TS, in double bounfreq_ZET, in long ig, in long jg,
            in long kg, in double tmp_init, in double sal_init, in long
            noriv);

      void set_monitor(in long nmonitor, in double rmonitor,

            in string monvarst, in long lenmonvarst, in long moni, in long
            monj,  in long monk);

            void set_filenames();

};
```

In this case, the JacORB client attached to the GUI sets the parameters in the 'steerParams' object while the TAO client attached to the model gets them. The object itself simply holds the data and responds to requests from the ORB.

4.3.2 Monitoring

The monitorOutput interface requires a typedef for a sequence of length 50 called LongSeq. Sequences provide one of the easiest ways of representing arrays in IDL and this interface needs to allow for an array of integers to be passed as a parameter.

```
typedef sequence<long,50> LongSeq;
```

Next the interface name is defined.

```
interface monitorOutput

{
```

Three operations are then defined for getting the values of the current timestep, data value at a given array position and count respectively. Count is used as a flag to determine when the GUI client should request the next set of data.

```
      long gettimestep();

      long getdata(in long pos);

      long getcount();
```

Finally, there are three operations for setting the timestep, count and the next array of monitor data.

```
      void settimestep(in long timestep);

      void setcount(in long count);

      void set_monitor(in long timestep, in LongSeq data);

};
```

Here we have the opposite set up to that presented in the steerParams case. It is the TAO client attached to the model that sets values while the JacORB client gets them. This time the interface also requires a typedef for LongSeq which will be mapped to an array in both C++ and Java.

4.3.3 Object Implementation

Once the idl has been compiled an Object Implementation class and a server class must be written. Currently these are coded in C++ using TAO rather than in Java. The servers can be found in the files ‘monitorServer.cpp’ and ‘server.cpp’, while the object implementations can be found in the files ‘steerParams_I.cpp’, ‘steerParams_I.h’, ‘monitorOutput_I.cpp’ and ‘monitorOutput_I.h’.

4.3.4 C-Wrapper for POLCOMS

The final requirement for completing the CORBA bridge is a C-wrapper for the POLCOMS code that can call the TAO/C++ client routines and pass output parameters to the Fortran. There are two C subroutines that provide this functionality. The first is ‘getclient’ which is called by the Fortran routine parm_client. The second is ‘monclient’ which is called by the Fortran routine data_out.

3.3.4.1 Getclient

Getclient takes the model parameters as input and sets pointers to each of the parameters so that their values can be retrieved by the Fortran code. Currently getclient is called once when the model is started.

3.3.4.2 Monclient

Monclient takes a single integer array as input, along with an integer value for the size of the array and an integer value for the current timestep. The routine calls the TAO client monitor_client passing the array and integer values as arguments.

5 Compiling and running

5.1 Compiling and building the Fortran

The Polcoms code was compiled and built using GNU Make [16] under Compaq tru64. In order to compile and build the code under a different OS or with different compilers or options you must edit the makefile, adding a new target that matches your requirements for the build. For information on writing makefiles for GNU make see the GNU make manual at [17].

GNU Make and ACE 5.3 +TAO 1.3 must be installed and the ACE and TAO libraries must be built before the code can be compiled.

To compile and build the code, first ensure that the GNU make executable is in the source directory along with the makefile. Next rename the GNU make executable, calling it ‘gmake’. This distinguishes GNU make from any other make program that may reside on the system and ties in with the commands defined in ‘makefile’. Now type ‘./gmake (Targetname)’, starting GNU make with the target ‘Targetname’. Target names in ‘makefile’ are constructed as follows:

exec-os-buildoptions

Where ‘exec’ is the name of the executable being built, ‘os’ indicates the OS being used and ‘buildoptions’ indicates options associated with the build. An example of a target name is:

shelf-compaq-serial-g

‘shelf' is the executable. ‘-compaq’, indicates Compaq tru64. ‘-serial’ indicates that this is a serial build and ‘-g’ indicates a –g build (i.e. Symbol tables are produced for full symbolic debugging)

The CORBA servers must also be built before the model can be run. The make file ‘servermakefile’ is used to build the servers. To run this make file type the following command: ‘./gmake -f servermakefile’.

A number of environment variables must be set for the makefile to function correctly. ACE_ROOT must contain the path of the ACE installation directory (ACE_wrappers), TAO_ROOT should be set to $(ACE_ROOT)/TAO and lastly, the $(ACE_ROOT)/ace directory should be added to the LD_LIBRARY_PATH environment variable. See TAO and ACE Installation instructions for more details.

5.2 Compiling the GUI

The GUI was developed under Windows XP using the Netbeans IDE (v3.5.1), Java 2 Standard Edition (J2SE v1.4.2), The Visualisation Toolkit (VTK v4.2) and JacORB (v1.4.1). Before trying to compile the code it is important to mount ‘jacorb.jar’ and ‘vtk.jar’ as file systems within the Polcoms GUI project. This will ensure that all VTK and JacORB classes are accessible for inclusion in the GUI classes. Using Netbeans, the compilation of the GUI code is simple. In the ‘Explorer [Filesystems]’ view, select the folder containing the GUI source code. Now go to the ‘Build’ menu and click on ‘Compile all’. Any compilation errors will be reported in the ‘Output Window’. You may encounter errors relating to the VTK class ‘vtkPanel’. If such an error occurs you can work around it by following these steps:

1. In the ‘Explorer [Filesystems]’ view, right-click on the file system icon for ‘vtk.jar’ and click on ‘Unmount Filesystem’.
2. Right-click on the file system icon for the file system containing the Polcoms GUI package folder.
3. Select ‘New’ and then ‘Java Package’ from the resulting menu.
4. Call the new package ‘vtk’ and click on finish.
5. Unpack ‘vtk.jar’ and copy and paste all of the files from the resulting ‘vtk’ directory into the new vtk package folder.

The VTK classes will now be available within the project and any compiler errors relating to vtkPanel should cease.

5.3 Running CORBA services

Before the GUI can be used to pass parameters to and receive monitoring data from the model we must run the CORBA Naming Service as well as servers for the two types of CORBA object.

The following examples assume that services are being run from within the Fortran source directory.

The Naming Service

```
$(ACE_ROOT)/TAO/tao/orbsvcs/Naming_Service/Naming_Service -ORBEndPoint
iiop://machinename:portnumber
```

During testing the Naming service was run on tca18, port 10101. The following command was used to run the service:

```
./../ACE_wrappers/TAO/tao/orbsvcs/Naming_Service/Naming_Service -
ORBEndPoint iiop://tca18:10101
```

The steerParams server

```
server -ORBInitRef NameService=
corbaloc:iiop:machinename:portnumber/NameService
```

During testing the steerParams server was run on tca18, port 10101. The following command was used to run the service:

```
./server -ORBInitRef NameService= corbaloc:iiop:tca18:10101/NameService
```

The monitorOutput server

```
monitorServer -ORBInitRef NameService=
corbaloc:iiop:machinename:portnumber/NameService
```

During testing the monitorOutput server was run on tca18, port 10101. The following command was used to run the service:

```
./monitorServer -ORBInitRef NameService=
corbaloc:iiop:tca18:10101/NameService
```

5.4 Running the GUI

JacORB provides its own batch file (jaco.bat) that can be used for launching clients and servers. ‘jaco.bat’ contains the following commands:

```
@echo off

rem call java interpreter

java -Xbootclasspath:"$(JACORB_HOME)\lib\jacorb.jar;

$(JAVA_HOME)\jre\lib\rt.jar;%CLASSPATH%"

-Dorg.omg.CORBA.ORBClass=org.jacorb.orb.ORB -Dorg.omg.CORBA.

ORBSingletonClass=org.jacorb.orb.ORBSingleton %*
```

The final command calls ‘java’ and sets the ‘Xbootclasspath’ as well as the ORB classes that should be used by JacORB applications. You can run the Polcoms GUI from the command line as follows:

```
Jaco -classpath (path of Polcoms and VTK class
directories);$(JAVA_HOME)\jre\lib\rt.jar Polcoms.Polcoms
```

Alternatively, ‘jaco.bat’ can be updated to run the Polcoms GUI by simply replacing the %* at the end of the file with the following:

```
-classpath (path of Polcoms and VTK class
directories);$(JAVA_HOME)\jre\lib\rt.jar Polcoms.Polcoms
```

Having updated ‘jaco.bat’, save the file with a different name such as ‘polgui.bat’. Now you can simply type ‘polgui’ to launch the GUI from the command line, or alternatively just double-click on the ‘polgui.bat’ icon.

N.B. In the example commands above, $(JAVA_HOME) represents the root directory of java sdk and $(JACORB_HOME) represents the root directory of JacORB.

5.5 Using the GUI to start a run

This section provides a simple step by step tutorial for starting the GUI, passing parameters, starting the model and monitoring data.

1. Run CORBA Naming Service and servers as instructed in the previous section.

2. Open a Command Prompt and cd to the location of the ‘polgui.bat’ file.

3. Type ‘polgui’ to launch the GUI.

4. Click on the ‘Browse’ button and select the directory containing ‘parameters.dat’ and ‘filenames.dat’.

5. Now click on the ‘Load’ button to load the parameters and filenames into the GUI. Then click on ‘OK’ to remove the info message confirming the load.

6. Go to the ‘Run Length’ textfield and enter ‘1’. Alter the units to years by clicking on the arrow next to the units menu and selecting ‘Years’.

7. If you wish to alter any other options at this stage, click on the ‘Options’ button, select the appropriate pane, change the option and click on ‘OK’.

8. Click on the ‘Run’ button to enter the ‘Monitor Window’.

9. Go to the ‘Monitor Frequency’ textfield and enter ‘1’. Alter the units to days by clicking on the arrow next to the units menu and selecting ‘days’.

10. Select an appropriate ‘i’ and ‘j’ so that the point being monitored is not on the land.

11. Click on the ‘Run’ button and wait for the empty axes to be displayed in the display window.

12. Run the model by typing ‘./shelf’. A run of length one year will begin and monitoring data will be returned for each day of the year (i.e. 365 times per run).

6 Testing

For all tests the GUI was running under Windows XP on a 1.59GHz PC with 256MB of RAM. The model was running under Compaq Tru64 Unix. Testing was carried out using a domain with dimensions 251x206x20.

All current visualisation capabilities have been tested. The GUI can display bathymetry, initial temperature/salinity, boundary temperature/salinity, river data, SPM locations, time series locations and cloud cover data.

It has been verified that the GUI can send steering parameters to the model running on a remote machine. These can then be used as initial parameters when the model starts. It is not yet possible to pause the model's execution and alter the steering parameters, although the design of the model does allow for this through the ‘server loop’ incorporated into the routine ‘b3drun’.

Monitoring of data has been tested for a range of short runs (up to a week) with different monitoring frequencies (from once an hour to once a day). Monitoring of data has also been tested for a run length of 6 months with data monitored daily. These runs generated good results, but occasional problems with the visualisation of the data and the CORBA connection have been encountered.

Tests revealed a bug associated with the annotation in the monitor display window. Lower case ‘i’ and ‘j’ characters do not display correctly. This bug has not yet been resolved. The relevant code can be found in ‘vtkMonitorPanel.java’. The CORBA connection has also failed during some tests. Such a failure produces an error message window containing the message ‘CORBA connection error’. All monitoring data received prior to the failure is then lost when the run is restarted. Such an error will occur if CORBA services and servers have been started incorrectly or in the wrong order, or when the remote machine on which the services are running is shut down.

One test with a run length of one year resulted in a ‘java.lang.outofmemory’ exception. This is a concern and attempts have been made to identify and correct any memory management problems in the code. The error has not been repeated in subsequent tests. The class ‘memory’ has been included in the GUI source. This class provides a useful means of monitoring memory usage using the ‘memory.report()’ method.

When running the code with large data sets (> 200 x 200 x 10) a ‘memory exceeded’ or ‘stack overflow’ error message may be encountered. If such an error occurs the amount of memory allocated for the stack and/or data storage may need to be manually increased. If you are using ksh this can be done using the following commands:

ulimit –s bytes (sets stack size to ‘bytes’)

ulimit –d bytes (sets data storage size to ‘bytes’)

or using csh:

limit stacksize bytes (sets stack size to ‘bytes’)

limit datasize bytes (sets data storage size to ‘bytes’)

where ‘bytes’ is the number of bytes that you wish to allocate. 1048576 is the value used during testing.

7 Evaluation

This project has been successful in that the Java GUI can now be linked in real-time to the Fortran application code and visualisation capabilities have been integrated into the GUI using VTK. After a short learning period Java was found to be quick,

easy and robust to use. Working with CORBA proved more difficult, but the relatively simple interfaces defined here will provide a good foundation for developing communications between GUI and model in the future. At the very least the work demonstrates that real-time monitoring of data can be achieved via CORBA. Working with VTK also proved to be a challenge, but this was in part down to the lack of example code and documentation for VTK with Java. The majority of examples that are available cover 3D visualisation techniques which are not central to the work that has been done on the GUI. It has been difficult to find examples of techniques for producing xy-plots and the VTK help files are sometimes vague in their descriptions of methods. Despite these problems a range of visualisations for different data sets have been produced. This code also provides a set of examples from which any further work can inherit.

Problems encountered during the work include dealing with memory management in a mixed language application, especially, where VTK is incorporated into the GUI code. Efforts have been made to track memory usage (using the 'memory' class). There are also some difficulties involved with maintaining the CORBA code due to the number of stages that any update has to go through. Updating the idl interface will result in a knock on alteration being required in several other files (Object implementation classes, clients, etc…). Because of this it is vital to keep track of changes as they are made.

8 Future improvements

The POLCOMS GUI still requires a number of refinements and additions that either fall outside the requirements for this work or could not be included because of time constraints. This section summarises the future work that needs to be done as well as indicating some of the possibilities for expanding the GUI's capabilities.

8.1 Visualisation

Currently there are a number of datasets that cannot be visualised. These include tidal data, meteorological data and all output datasets. As long as the format for each of these datasets can be established it should be relatively straightforward to write classes to visualise them by extending the current visualisation classes. There is also room for improvement of some of the less developed visualisation classes such as 'vtkMetVisWindow' which currently controls the display of cloud cover data, but needs to be expanded to cover all meteorological datasets.

The functionality of the monitor window is currently limited to displaying only temperature data on an xy-plot. In the future the intention is for this to be updated to allow the user to choose all parameter types and to allow other forms of visualisation such as slices through the data.

A possible addition to the GUI's visualisation capabilities might be to incorporate a method of viewing and saving animations made up of images of datasets changing as the simulation progresses.

8.2 Communication

CORBA communication can be enhanced by using the ‘server loop’ built in to the Fortran (see ‘b3drun.F’) to allow ‘scanning’ for parameter changes while the code is running.

The 'steerParams' idl interface currently makes no provision for the passing of input and output filenames from the GUI to the model. This is a relatively simple operation but was not thought to be vital to this work as the new capabilities of the GUI can be demonstrated without it.

The machine name and port number used for CORBA communication is currently ‘hard-wired’ into the code at line 172 of ‘CorbaCommsThread’ as well as in ‘tao_client’ and ‘monitor_client’ that are used for communication by the model. Ideally the machine name and port number should be entered into the ‘Target’ options pane in the GUI. This pane needs to be updated to allow for this. These parameters could then be passed to a ‘CorbaCommsThread’ object so that the location of distributed objects could be determined by the GUI. The machine name and port number defined in ‘tao_client’ and ‘monitor_client’ must match those defined in the GUI, but there is no mechanism for passing these parameters as they are required by the model before a CORBA connection can be established. Currently ‘tao_client’ and ‘monitor_client’ must be updated manually to ensure that this is the case. It would be useful to include an option for altering the machine name and port number in the Fortran code so that the code did not need to be altered whenever it is required to run on a different machine or use a different port.

8.3 GUI Selection Panes

Currently the ‘Options window’ features a debugging pane which has not yet been implemented. The tidal data options pane is not fully implemented.

As discussed above, the target pane could be used to specify a machine and port number as a target for CORBA communication. This pane needs updating before this is possible.

A data format choice JComboBox has been included on the main menu. There is currently only provision for one data format. As new data formats are developed it is important that this should be kept up to date. This also applies to the ‘readdata’ methods in each of the vtk classes which must be able to read all possible data formats.

8.4 Grid Computing

We intend that the work described here should be extended to allow the model to make use of current and future developments in Grid computing. This will probably entail the replacement of the CORBA code by an interface built around Globus or Web Services. Remote access to input and output datasets could usefully utilise the Storage Resource Broker (SRB).

9 Web Resources

Java

APIs –

http://java.sun.com/j2se/1.4.2/docs/api/

Java Swing

APIs –

http://java.sun.com/j2se/1.4.2/docs/api/

The JFC/Swing Tutorial –

http://java.sun.com/docs/books/tutorial/uiswing/

VTK

VTK Mailing list –

http://www.vtk.org/mailman/listinfo/vtkusers

Java - VTK examples –

http://ij-plugins.sourceforge.net/vtk-examples/

VTK can be downloaded from –

http://www.vtk.org/get-software.php

CORBA

OMGs CORBA website –

http://www.corba.org/

OMGs CORBA FAQ –

http://www.omg.org/gettingstarted/corbafaq.htm

Download the OMG IDL to C++ Mapping specification from –

http://www.omg.org/cgi-bin/doc?formal/03-06-03

Download OMG IDL to Java Mapping specification from –

http://www.omg.org/cgi-bin/doc?formal/03-09-04

Corba Papers on the web –

Wrapping ADPAC CFD Code

http://accl.grc.nasa.gov/IPG/CORBA/wrap_fortran.scott.html

Developing CORBA-Based Distributed Scientific Applications From Legacy Fortran Programs

http://accl.grc.nasa.gov/IPG/CORBA/CAS_corba.pdf

NPSS on NASA's IPG: Using CORBA and Globus to Coordinate Multidisciplinary Aeroscience Applications

http://www.ipg.nasa.gov/workshops/papers/NPSS_CAS_paper.html

A Framework for Distributed Mixed Language Scientific Applications

http://www.hep.net/chep95/html/papers/p18/p18.pdf

JacORB

JacORB 1.4.1 Programmers Guide –

http://www.jacorb.org/docs/ProgrammingGuide_1_4_1.pdf

JacORB website –

http://www.jacorb.org/

JacORB can be downloaded from –

http://www.jacorb.org/download.html

ACE + TAO

Installation instructions ACE –

http://www.cs.wustl.edu/~schmidt/ACE_wrappers/ACE-INSTALL.html

TAO Documents –

http://www.cs.wustl.edu/~schmidt/ACE_wrappers/TAO/docs/index.html

TAO FAQ –

http://www.theaceorb.com/faq/faq.html

Installation instructions TAO –

http://www.cs.wustl.edu/~schmidt/ACE_wrappers/TAO/TAO-INSTALL.html

Download ACE and TAO from –

http://deuce.doc.wustl.edu/Download.html

GNU Make

GNU Make manual –

http://www.gnu.org/software/make/manual/html_mono/make.html

Download version 3.80 of GNU Make from -

ftp://ftp.gnu.org/pub/gnu/make/

References

[1] M. Ashworth, R.J. Allan, C.J. Müller, H.J.J. van Dam, W. Smith, D. Hanlon, B.G. Searle and A.G. Sunderland, *Graphical User Environments for Scientific Computing*, UKHEC Technical Report, http://www.ukhec.ac.uk/publications/reports/guienv.pdf

[2] J.T. Holt, R. Proctor, M. Ashworth, J.I. Allen and J.C. Blackford, *Eddy Resolved Ecosystem Modelling in the Irish Sea*, in Realizing Teracomputing: Proceedings of the Tenth ECMWF Workshop on the Use of High Performance Computing in Meteorology, eds. W. Zwieflhofer and N. Kreitz, 2004, pp.268-278, (World Scientific).

[3] J.T. Holt, R. Proctor, J.C. Blackford, J.I. Allen and M. Ashworth, *Advective controls on primary production in the stratified western Irish Sea: An eddy resolving model study*, 2004, in press, *Journal of Geophysical Research.*

[4] Daniel J. Berg and J. Steven Fritzberger, *Advanced Techniques for Java Developers*, 1999, (John Wiley & Sons).

[5] Sun Microsystems Inc., *The Swing Tutorial: Creating a GUI with JFC/Swing,* http://java.sun.com/docs/books/tutorial/uiswing/

[6] William Schroeder, Ken Martin, Bill Lorensen, *The Visualisation Toolkit An Object-Oriented Approach To 3D Graphics 3rd Edition*, Kitware Inc.

[7] The Visualisation Toolkit Users Guide, Kitware Inc.

[8] Th. Mowbray and R. Zahari, *Essential Corba: System Integration with Distributed Objects*, 1995, (John Wiley & Sons) ISBN 0-471-10611-9

[9] J. Siegel, *CORBA 3: Fundamentals and Programming*, Second edition., (John Wiley & Sons), 2000

[10] Michi Henning, Steve Vinoski, *Advanced Corba Programming with C++*, Addison Wesley. 1999.

[11] Gerald Brose, Andreas Vogel Keith Duddy, *Java Programming With Corba*, Third Edition, John Wiley and Sons Inc. 2001.

[12] Fnorb Homepage, http://www.fnorb.org

[13] Jacorb Homepage, http://www.jacorb.org

[14] The ACE ORB (TAO) Homepage, http://www.theaceorb.com

[15] Carlos O'Ryan, *A Very Simple Client Tutorial*, http://www.cs.wustl.edu/~schmidt/ACE_wrappers/TAO/docs/tutorials/Quoter/Simple/Client/index.html

[16] GNU make Manual http://www.gnu.org/software/make/manual/html_mono/make.html.

[17] http://www.gnu.org/software/make/manual/html_mono/make.html