TOWARDS A DISTRIBUTED INFORMATION SYSTEM FOR COASTAL ZONE MANAGEMENT

by

KONSTANTINOS P. MIHANETZIS

B.S. Marine Engineering Hellenic Naval Academy, 1991

Submitted to the Department of Ocean Engineering
In partial fulfillment of the requirements for the degrees of

Naval Engineer
and
Master of Science in Ocean Systems Management
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 1999

Author……………………..……………………………………………………………

Department of Ocean Engineering
May 1, 1999

Certified by...……………………………………………………………………………

Nicholas M. Patrikalakis, Kawasaki Professor of Engineering
Thesis Supervisor, Department of Ocean Engineering

Certified by…….…...……………………………………………………………………

Henry S. Marcus, Professor of Marine Systems
Thesis Reader, Department of Ocean Engineering

Accepted by ……………………………………………………………………………..

Arthur B. Baggeroer, Ford Professor of Engineering
Chairman, Departmental Committee on Graduate Studies

TOWARDS A DISTRIBUTED INFORMATION SYSTEM FOR COASTAL ZONE MANAGEMENT

by

KONSTANTINOS P. MIHANETZIS

Submitted to the Department of Ocean Engineering

on May 1, 1999 in Partial Fulfillment of the Requirements for the Degrees of

Naval Engineer and Master of Science in Ocean Systems Management

Abstract

The need for accurate data retrieval and use has become a very important issue in the field of ocean sciences, engineering and management. The amount of data collected and processed daily is vast and is expected to increase considerably in the following years. Above this, the lack of a globally accepted protocol for storing this data has created a lot of confusion and delays in using the data. The existing tools used to simulate the ocean environment, for reasons of prediction or research, are considered to be highly computationally intensive. Most of them also consist of large monolithic computer programs that are custom made to operate in a specific environment. In order to simulate accurately the ocean environment, either for forecasting or research, a lot of these applications should be able to work together. This could happen either by developing new software that will do so, extending the ones that exist to do so or find a way to link the existing software in a distributed environment as if they were together. In the case of extending or building new software that will handle all the tasks, we would find ourselves to be limited by the computational power of the existing computers.

The need for fast computation and database integration leads to a distributed environment that includes a large number of computational resources, which are able to communicate with each other.

This thesis describes the framework for distributing and linking applications over the World Wide Web (www). This thesis is part of a larger project code named Poseidon, which also involves the implementation of a Metadata Standard for data storage and retrieval.

A tool also named HTMLParser is presented that can be used to automate the procedure of data mining from the www. The HTMLParser automates the procedure of database querying. It is used to extract specific information from the response of a database query and make it available to the user as an object for the Poseidon environment. The HTMLParser can become a very useful tool especially due to the increased rate of database porting on the Internet.

Thesis Supervisor: Nicholas M. Patrikalakis, PhD
Title: Kawasaki Professor of Engineering

Acknowledgments

There are many people that I would like to thank for helping me to reach this point of my career. For me, finishing this thesis reflects the fulfillment of a goal I had long ago set for myself. It was one of my very first dreams since I got into the Hellenic Naval Academy as a first year Naval Cadet, to come to MIT and graduate with a Master's degree. Today I believe my goal was one of the wiser I ever set. The academic environment and the people I met made me think much more differently today. My appreciation of education's importance was always strong but now it has justified its reasoning much more. I believe that educating myself I become a better man, a man that can be much more useful to society and to other people. My goal would have been much more difficult to reach without the people I met here at MIT.

First, I would like to thank Professor Nicholas Patrikalakis who was my thesis advisor for providing me with excellent advice and guidance during the last two years of my academic life. In the last two years I learned from Professor Patrikalakis how important it is to keep your mind open to your visions but if you want to make them real, you have to find a way to organize a schedule for completing them.

Second, I would like to thank Professor Mark Welsh USN for being next to me during the first two years of my academic years as a program advisor. Professor Welsh was an invaluable resource for me and was always available with wise advice when the MIT schedule was getting real tough.

Stephen Abrams, since our first conversation about my thesis topic, provided me with all the help, insights and support in programming languages and for the design and implementation of the Poseidon architecture. He also provided me with very important feedback for the completion of this thesis document.

Other people include, Professor C. N. Nikolaou, Professor C. Houstis, Dr. S. Lalis for the useful interactions that we had during the preliminary studies of the Poseidon architecture development and Dr. Wonjoon Cho for reading and reviewing my thesis

I would also like to thank Sea Grant and the National Oceanographic Partnership Program (NOPP) for their contribution in the initiation of the Poseidon system and subsequently of my thesis, and the Architecture and Infrastructure Working Group of the NOPP LOOPS project headed by Professor Patrikalakis.

Funding of this work was obtained in part from the U.S. Department of Commerce (NOAA, Sea Grant) under the grant NA86RG0074, the U.S. National Ocean Partnership Program via ONR grant N00014-97-1-1018, and the MIT Department of Ocean Engineering.

I have to offer a very big thank you to the Hellenic Navy that allowed me to come to MIT.

Great thanks go to my parents for making every effort to provide me with the best possible education. At last but not least, I thank my fiancée and future wife, Heleni that with infinite love and patience gave me strength to survive my time at MIT.

This thesis is dedicated to the vision for a better world. A world of peace, love and prosperity.

TOWARDS A DISTRIBUTED INFORMATION SYSTEM FOR COASTAL ZONE MANAGEMENT

Table of Contents

List of Figures

1. Introduction

At the end of the millennium the stored information and computational power in computers around the world had increased dramatically when compared with the relative figures just a decade ago.

The main reason for this has been the exponential growth of computer capacity both by means of the numbers of existing machines as well as by means of their computational power and memory capacity. The increased capacity of computers however encouraged the development of large “monolithic” applications that used to run on a single machine. These applications grew in size in almost the same rate as computer power. Their consumption in computational resources was always a matter of detailed study. Also their configuration to run on one machine, or one type of machines running a specific operating system (UNIX, NT, etc), made them dependent on the specific machine architecture. A large number of applications were custom designed to solve specific problems and only their developers had the detailed knowledge of their architecture. The use of old programming languages such as Pascal or Fortran added complexity to the task of updating them[1]. As documentation was in many cases sparse, use of the application and continuation of its development became problematic or impossible. However, in many cases the investment in money and knowledge did not allow the scrapping of such programs. The use of a large number of programs is still very valuable. Characteristic is the code for physical prediction of the ocean (HOPS) created by Harvard University Interdisciplinary Ocean Sciences Group and the Acoustic Tomography code created by MIT Ocean Engineering.

---

[1] Such complexity is mainly derived from the fact that fewer people keep working with these languages and it is much more difficult to find expert people to deal with a problem if it arises. Although the Poseidon system developed in this thesis doesn’t provide for the non-aging of programming languages, the fact that it allows the user to face an application as a “black box” that needs some inputs and provides some outputs, will allow extension of the functionality of the existing application by adding the extended behavior as another object that will be linked to the old object’s output or input.

In the field of ocean monitoring and prediction the amount of information already in existence is vast. Huge databases store historical data of a large number of parameters, such as salinity, sea temperature distribution, sea elevation, etc. A large number of devices (satellites, buoys, AUV's, etc) keep on collecting data daily at high rates. On the other hand the reduction of cost for data sampling and the progress in sciences has allowed the design of new devices that can measure more physical variables for much lower cost. The combined effect of low cost and increased functionality means that more and more devices will be installed to measure more and more parameters of the ocean environment, making the task of data handling even more difficult.

Data such as sea elevation require three variables to be stored for each point of measurement (longitude, latitude and elevation measure), and for a specific time. However, they are usually measured over a large area and for relatively small time intervals. This multiplies the amount of data stored and processed into extremely high values. A sampling of a coastal area of 100 squares kilometers with a grid resolution of 10 meters and for a period of one day, taking samples every 10-sec, would require the storage of 8.64 billion samples.

Although it is not common practice to measure salinity every 10 meters, it is common for the case of sea elevation (satellite images provide much higher accuracy). The fact also that something seems today to be not feasible (or cost effective) doesn't mean that it would be the same in the future. When planning for a system that needs to cope with future requirements, it must be designed to scale gracefully.

What becomes obvious, is that it would be impossible to store all the required data into one single database. This is one of the reasons why most of the data for the ocean environment is distributed around the world. Each of these databases is usually dedicated to collecting and storing a specific piece of information. They are dedicated to regions, type of data or time of collection. Although the ancient wisdom of "dividing and conquering" has worked fine to win the battle of data collecting and preserving, there are still a lot of obstacles in using these data in an effective way. The distribution of this information to different places on different computers with different operating systems has made almost impossible the linking of these databases or the programs that work with them. However, in order to produce accurate predictions of the ocean environment this information has to be used in its totality.

The Poseidon System started as an attempt to integrate all these distributed and heterogeneous valuable resources. New standards for metadata (data about data) have been developed that provide the guidelines for storing and retrieving data [1]. New technologies such as CORBA and Java are used to bring together all these distributed applications and overcoming the problems of increased diversification found in operating systems and programming languages.

In the context of the Poseidon Environment, a user will be able to integrate his custom or proprietary program to work with other programs available in the Poseidon network as well. He will be able to use the utilities provided by the Poseidon System to store and retrieve data in such a way that they will be well described and persistent. Overall the Poseidon System has a versatile and effective architecture that can be extremely valuable in the field of Oceanography and Coastal Zone Management.

This thesis is organized as follows:

- Chapter 2 describes existing work that is similar to the Poseidon Project objectives. It also gives a brief introduction to the technologies used for the development of the Poseidon system.
- Chapter 3 provides a detailed description of the Poseidon architecture and the way it is implemented. It provides information for extending the usability and functionality of the Poseidon environment through the creation of new objects that are compatible with the Poseidon system. It also describes important Poseidon parts such as the Poseidon Repository and the Poseidon Graphical User Interface.
- In Chapter 4 available tools that were developed for extending the Poseidon usability are described. They are tools that are not vital for the operation of the Poseidon system, however their extended features make the integration of existing applications much easier task.
- Chapter 5 describes the application of the Poseidon system in the field of Ocean Systems Management. In this chapter a description of the HTMLParser is included and an example describing the collection of data related to Ocean System

Management. The example uses an online database of a shipbroker to collect data about cargo ship openings at various ports.

- Chapter 6 is a list of conclusions and suggestions for future work related to this thesis work. It includes directions for further study and development of the Poseidon architecture.
- Chapter 7 is a list of the references used during the development of the Poseidon system and the tools described in this thesis.
- Appendix 1 is a list of all the abbreviations used in this thesis.
- Appendix 2 contains the listings of the code of the main part of the Poseidon system including the Graphical User Interface.
- Appendix 3 includes the code listings for the Poseidon Repository and related utilities.
- Appendix 4 includes the code listings for the Generic Object Server and other related utilities. These are parts of the Poseidon system related to the linking and communication of the objects.
- Appendix 5 contains the code listing for two objects that are characterized as local. The first one is a very simple example that receives a single string and displays it on a small separate window.

  The second one is more complicated and receives a two dimensional matrix of floating point numbers and displays a color contour representation of them. In this second case the object requires an interaction panel which is used by the user to set up initial variables before the object can display its results.

  The code listings are presented as basic examples for implementing local objects in the Poseidon environment (a local object is an object that is been executed by the same computer that runs Poseidon applet).
- Appendix 6 contains code listings for the HTMLParser. The appendix contains only the basic code that is required for the operation of the HTMLParser plus the listing for handling the HTML “<input>” tag. This last code is included as an example of how the code for handling HTML tags has been developed.

- Appendix 7 is a list of the Poseidon's Application Programming Interface (API). The Poseidon API is used by anyone that needs to introduce an existing (or new) software program in the Poseidon environment.
- Appendix 8 describes the information required for an object to supply, during the registration procedure with the Poseidon Repository.

2. Literature Survey

2.1 Related work to the Poseidon System.

In the context of the Poseidon System that includes the task of data storing and retrieval, there have been a number of similar projects with related objectives. The most notable of them are funded from large governmental institutions such as NFS, NASA and DARPA and include projects such as the Stanford Integrated Digital Library, the UCSB Alexandria project, the UC Berkeley Digital Library project and many more.

In the context of this thesis, however, which is more focused on application linking, there is less work done. The main reason for this was the lack of architecture technologies that could allow the development in an efficient way.

Today a similar project is been carried on by UCLA's Data Mining Laboratory. The Project code-named OASIS is a similar project that attempts to bring a level of abstraction to application implementation that will provide for transparent linking and communications between distributed objects. However, OASIS is mostly focused on data search and retrieval while Poseidon will also provide functionality for simulations, data analysis and visualization. The second difference is that the level of abstraction used by the OASIS architecture is lower so that it is very difficult to support the complex, cyclic nature of oceanographic simulation workflows or simulation software with multiple inputs and multiple outputs, such as the Harvard Ocean Prediction System (HOPS)

Last, the Poseidon is a system that has been designed to be as user friendly as possible. This is a significant advantage, in that the user does not have to be well aware of the correctness of the workflow at the programming level and any unreasonable assumptions. In Poseidon this allows the user to concentrate on the scientific problem taking away from him the burden of details in the application integration.

2.2 Technologies implemented into the design of the Poseidon System

2.2.1 Common Object Request Broker Architecture (CORBA)

In May 1989 eight US companies (3Com Corporation, American Airlines, Canon, Inc., Data General, Hewlett-Packard, Philips Telecommunications N.V., Sun Microsystems and Unisys Corporation) founded the Object Management Group (OMG). The role of this company was the establishment of industry and detailed object management specifications to provide a common framework for application development. In October 1989 OMG began independent operations as a non-profit corporation. Today OMG has more than 800 members both from the software industry and other institutions. The main product developed by the Object Management Group was a set of specifications code named CORBA (Common Object Request Broker Architecture).

CORBA has been called "the most important (and ambitious) middleware project ever undertaken by our industry" [8].

The Common Object Request Broker Architecture (CORBA) was designed by the OMG to solve problems of interoperability among an increased number of hardware and software products that are available today. At a very abstract level of description, CORBA allows applications to communicate with one other no matter where they are located or what language they are written with.

CORBA is based on the principles of Object Oriented Programming [10][12]. Using CORBA, whole applications can be regarded as objects providing a specific functionality.

The most important part of CORBA is Object Request Broker (ORB). The ORB is responsible for all the required communications between two applications (objects) that need to interact. Although this can be a very complicated operation (especially between heterogeneous platforms and programming languages) the ORB hides all the complexity to its native code, making the deployment of new applications on the network a very simple task.

In a client/server relationship the ORB is responsible for all of the mechanisms required to find the object implementation for the request (server), to prepare the object implementation to receive the request, and to communicate the data making up the request. The client does not have to be aware of where the object is located, its programming language, its operating system, or any other system aspects that are not part of an object's interface.

The ORB provides an extra flexibility that allows the programmers to choose the most appropriate operating system, execution environment and even programming language to use for each component of a system under construction. More importantly, they allow the integration of existing components.

It has to be noted that two more technologies exist which are related to the functionality provided by CORBA. These are the Distributed Component Object Model (DCOM) and Java's Remote Method Invocation (RMI). In the first case DCOM is a similar technology developed by Microsoft. However the late introduction of DCOM as compared to CORBA and the less extensive functionality it provided (especially in its first editions) established CORBA almost as a standard for application development that required transparent communications and object linking.

In the case of RMI, the API provides far less functionality than CORBA. RMI is used as an extension of the Java API to allow applications written in Java and placed on different hosts to communicate. CORBA on the other hand, extends this functionality to allow programs written in different languages to communicate.

2.2.2 Java

Java is a relatively new programming language. It is often described as the World Wide Web (www) programming language. It is created (and updated) by SUN Inc., since 1995[2] and it's widely known due to the extended features it provides for developing applications that run across networks such as the www.

---

[2] Although the first Java Development Kit (which includes the Java API and the Java Runtime Environment) was released at late 1995, the Java language had been into development much earlier with the code name Oak [12][13][14][15][16]. Oak was originally designed to be used for programming the embedded systems in consumer electronics devices. However, the widespread of the World Wide Web changed the

SUN claims that Java is a “write once run anywhere”[13][16] programming language. Although this has proven to be not 100% true, it is very close to accurate. Using the Java Application Programming Interface (API) a developer can write applications that are able to be executed in almost all the available platforms and operating systems without any changes to his code. SUN was able to achieve this by the introduction of the Java Runtime Environment (JRE). By porting platform-specific requirements into the JRE, SUN was able to remove them from the Java API. SUN focused a lot of the Java capabilities on networking and Graphical User Interface (GUI) design. The latest API (1.2) includes an extended set of classes that provided functionality for designing complex 3D graphics and using advanced multimedia capabilities.

Although Java 1.2 is available, the Poseidon System was built using older versions of the Java API. There were two reasons for this:

- The Poseidon System will be mainly available as a Java applet. This means that the Java Development Kit (JDK) supported by the browser (used to download the applet) must be the same or later of the one used to develop the applet. Since the latest JDK (1.2) is not yet widely used it would create a lot of problems in the near future from users using older JDK’s.
- The development environment (INPRISE’s Visibroker 3.2) used for creating CORBA objects doesn’t support the latest JDK.

It is recommended however, that the new JDK will be used, in the near future, for extending the Poseidon capabilities.

---

orientation of Oak design, and when it was renamed to Java, it had already been changed to a programming language for use on the www.

2.3 An early application of the Poseidon System using existing oceanographic applications

In the early stages of the Poseidon system[3] development, a number of tests were conducted to test and estimate the linking capabilities of applications using the CORBA technology. In order to produce a full-scale example of application linking, two different applications were wrapped with CORBA layers to become Poseidon objects:

- Harvard University's, Interdisciplinary Ocean Sciences Group code for physical prediction of the ocean , called HOPS
- Massachusetts Institute of Technology, Department of Ocean Engineering Code for Acoustic Tomography, called OAG, and

These applications were invoked from a remote applet (Figure 2-1) that supplied them with a set of initialization parameters and asked for a two dimensional, color contour representation of the acoustic field. The generated field was representative of the parameters that the user was supplying through the applet's form fields (area of the ocean, time of the year, etc).

In this example, HOPS was a large software application written in Fortran and had a layer of C++. HOPS was active on a Silicon Graphics platform running on a UNIX operating system. OAG was also written in Fortran and utilized a C++ layer and was active on a Silicon Graphics platform running on a UNIX operating system. The applet however that made requests to the two applications was written in Java and was been executed from both UNIX platforms as well as from Windows NT and Windows 95 operating systems.

This example proved the ability of CORBA to link large heterogeneous applications and open the way to the next generation of the Poseidon architecture.

---

[3] The Poseidon system has been under development based on the same philosophy of object interaction and communication since its inception. However during the first stages of the Poseidon system development, tests for linking and communication were performed in a manual way.

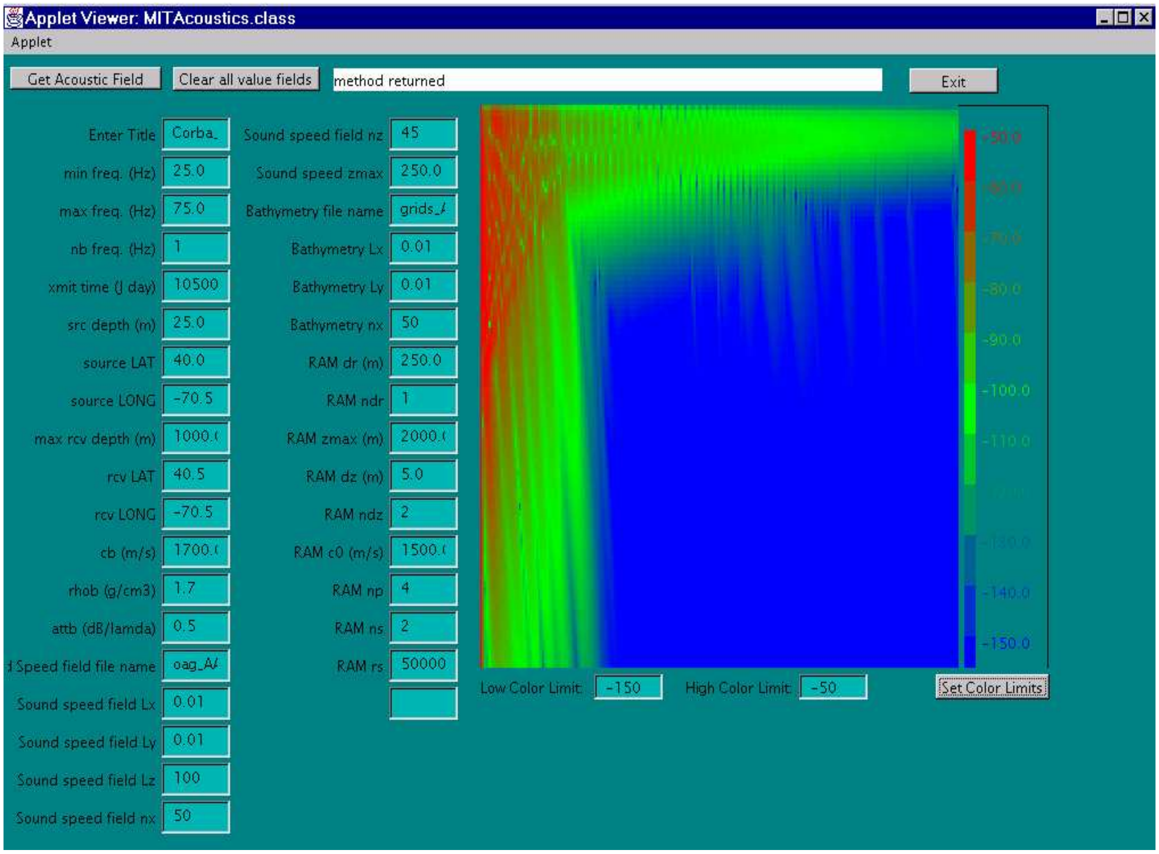


Figure 2-1 Poseidon's applet (early version) with received acoustic field

3. The Poseidon Environment

3.1 The Poseidon Architecture

The basic concept behind the Poseidon system is to provide a framework where a user, employing his personal computer, can connect to the internet and with commonly used software, such as a web browser, can link together applications that run on remote computers and receive the results back to his computer. This idea would help to use large software programs that can not be executed on a personal computer with limited computational power. It will also be very helpful since the user will be able to bring together information from all over the world. To succeed with such a goal, the Poseidon is using CORBA to set a network where communications become transparent to computer platforms and operating systems. In CORBA the functionality offered by the Object Request Broker (ORB) [7] allows the developer of the Poseidon system to add functionality to the system that will provide for easy object discovery, fault tolerance, transparent communications and much more. The architecture of the Poseidon system can be reviewed in Figure 3-1. In the context of this figure, a client contacts the Poseidon server by means of his web browser and downloads the Poseidon applet. The applet executes on the client's computer. The applet uses the ORB to communicate through the Poseidon environment to identify any available resources. Resources can be either other computers that can run specific applications or data sources. Data sources on the other hand can be either databases that satisfy query requests or devices (buoys, AUV's, satellites, etc) that provide real time data.

When the applet has discovered all the available resources it provides all this information to the user. Following the user uses the applet design workspace to link the resources and ask for results. The applet then communicates with the objects, links them according to the user directions, activates them and returns the results of the workflow.

The philosophy of linking objects together is very similar to that found in control theory [2]. Objects act as transfer functions getting a number of inputs and supplying a number of outputs [3]. The outputs are the controlled products of the inputs. In a control system an engineer combines various components (transfer functions) to get a desired output. In the Poseidon system the user combines various applications to produce the desired output. In most cases of control systems the combined components are physically close to each other. In the Poseidon system the components in most cases are far from each other.

The heart of the Poseidon architecture is the Model Management System (MMS) (see Graph 3-1). The MMS consists of a Graphical User Interface (GUI) that allows a user to build an information workflow. An information workflow consists of distributed resources such as databases and applications that are linked together to exchange data. The parts of such a workflow are called nodes or objects. The MMS is responsible for:

- Creating the GUI to be used by the user in order to build the workflow
- Validation of the workflow and
- Management of the execution of the workflow

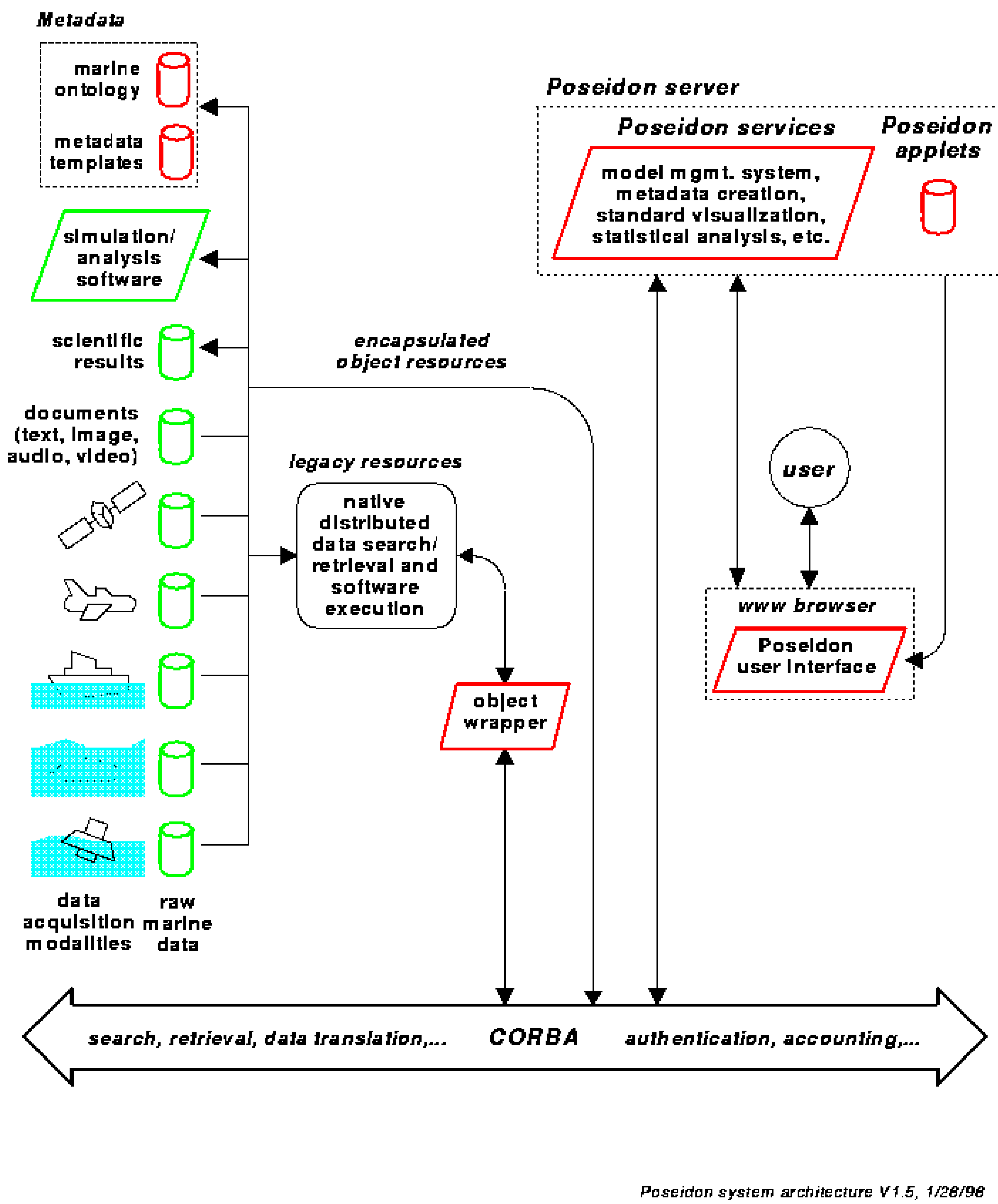
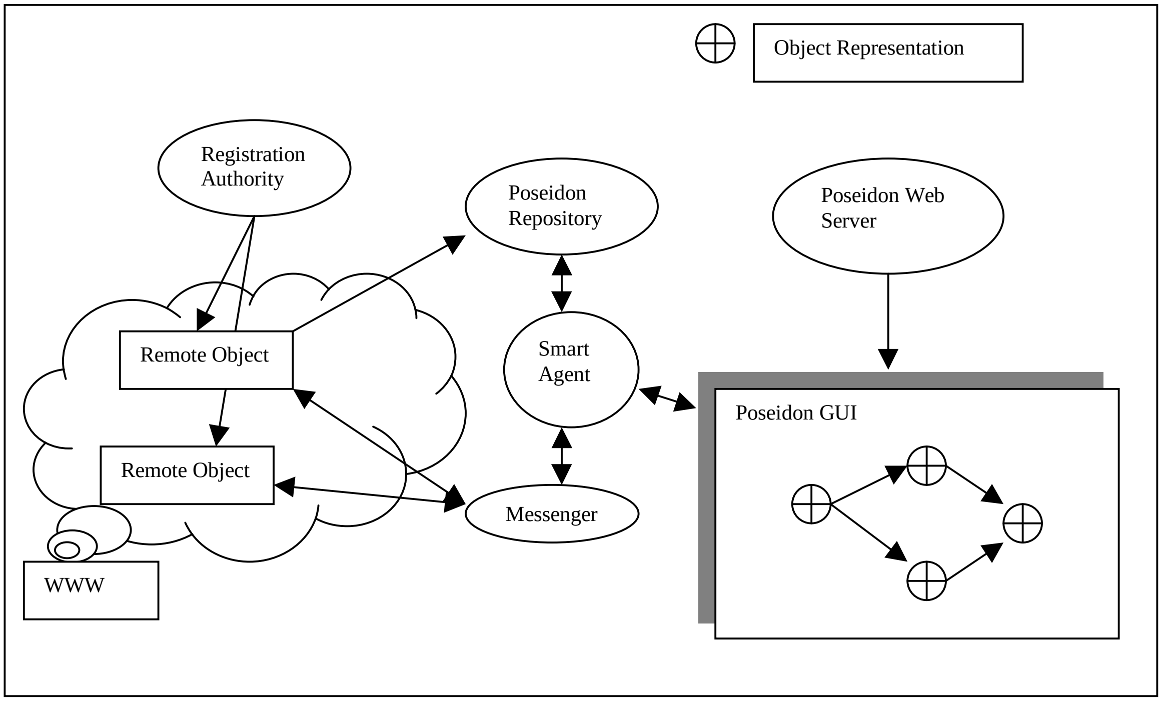


Figure 3-1 – The Poseidon Architecture

The MMS is not a stand-alone application. It requires the execution of other applications before it can be available to operate. First, a Smart Agent[4], that will be responsible for object identification on the network. Second, a GateKeeper[5] that will handle Client (applet) requests from different hosts[6] and finally, the Poseidon Repository (3.3) which is a central registration authority for storing information about the available objects.



Graph 3-1 – Object Communications

---

[4] A Smart Agent is an application provided by Inprise's Visibroker development environment and is used to provide basic networking functionality.

[5] A GateKeeper is also an application provided by Inprise's Visibroker development environment.

[6] A Java applet that is viewed on a browser has a security restriction that forbids it to communicate with servers on a different host. In order to overcome this problem (which is vital for distributed computing) INPRISE's Gatekeeper creates a proxy server on the primary host from where requests on other hosts can be redirected.

The Smart Agent and Gatekeeper[7] are ready-built off-the-shelf applications that perform some important operations related to networking. The Poseidon Repository on the other hand is an application that was built to cover the needs of the Poseidon architecture.

A web server also is required for serving the files of the Poseidon Applet. Such a server can be any of the many available in the market (Netscape Enterprise Server, Microsoft's Exchange Server, Apache Server, etc).

The Poseidon MMS works as follows:

- The Poseidon Central Authority (PCA)[8] starts up at least one Smart Agent and a Gatekeeper on their computers.
- The PCA starts up a Poseidon Repository (PR). The next step for the PR is to register with the Smart Agent and declare its existence.
- Since the PR is up and running, remote objects can be activated. Activation of a remote object is a four-step procedure:
    - I. Execution of the Generic Object Server (GOS). The local server is a generic server that enables the object to respond to requests using the CORBA standard. In this first step it is used just to enable communications with the PR.
    - II. Register the object with the PR, which will respond by issuing a Serial Number[9] for the object.

---

[7] Smart Agent and Gatekeeper are ready applications that have been used as is to quicken the development environment. Future versions of the Poseidon system though, could include custom build applications that perform similar tasks.

[8] The Poseidon Central Authority will include a group of people that will be responsible for maintaining the Poseidon servers and users accounts.

[9] At this stage of the Poseidon development, the Serial Number of each object is made by combining the object's name (the name used to start its object server) and a integer value supplied by a counter in the Poseidon Repository process.

III. The object starts a new instance of the GOS but using the serial number provided. This will uniquely identify the object on the network.

IV. The object's server falls in an infinite loop process waiting for incoming calls.

- A User logs in the Poseidon website and downloads the Poseidon Applet.
- The applet loads from the Poseidon web server.
- During startup the applet contacts the PR and asks for a list of all the available objects.
- The PR replies by sending the list. The list contains basic information about the objects.
- The applet uses the list of objects to create a tree-like representation of the available objects.
- The user creates his workflow by adding and linking the objects that he needs into the design workspace.
- As soon as the workflow is ready, the user can ask for the objects to start operations.
- Before starting the objects (applications) however, the MMS will perform a number of checks. It will check to see if the connections made are valid and then it will communicate with each object to see if there is any special initialization conditions that have to be set. If everything is fine it will start asking the objects to execute and provide results at their outputs. The rule to executing is "any object that has to provide data to an other will be executed first".
- Eventually all the connected objects will be executed and the user will be able to see his results by means of some objects GUI (this is not totally necessary. A user can ask for objects to send the results to a file or email without any intermediate GUI to display the results).

The Messenger that appears in Graph 3-1 is responsible for the communications between the applet and the objects. It is another CORBA object that is been initiated by the applet.

3.2 Converting Existing Applications to Poseidon Objects

The Poseidon System offers a way to link together distributed applications that run on different operating systems and computers. Although this task could be very complex, the goal is to develop a product, which is user friendly and easy to use.

In this manner, and in order to make the Poseidon System successful, it was realized from the very beginning that it was very important to find a way to convert existing applications into Poseidon Objects as easily as possible.

The goal of this problem was achieved by designing a set of top-level classes that extracted all the complexity of the conversion. Every application that needed to become available in the Poseidon System as a Poseidon Object would then require to inherit from this set of classes. In this way the (required) complexity would be added to each new object but the developer would be unaware of it.

Graph 3-1 shows that the user invokes requests on remote objects located on the network. However the MMS provides a set of objects that are local by means that they can be executed on the client machine running the applet. They are objects that provide basic functionality such as basic visualization and are downloaded from the Poseidon web server together with the applet. The following is a more detailed description of local and remote objects and a description of how to create Poseidon objects from existing applications.

3.2.1 Local Objects

Local objects are small applications that are available from the same web server that supplies the Poseidon applet. They are supplied by the Poseidon Development Team (PDT), and are utility applications mainly for visualization and basic mathematical functions (see Appendix 5 for examples of local objects and the way they can be implemented).

The Local Objects do not have to be manually activated and registered with the PR. They are automatically registered (but not activated) during the PR startup (see also section 6.1). When a user loads an applet, he can see on the object tree the local objects as well (they won't be distinguished as local though). He can add a local object on his workflow as any other object. The difference will be when the user asks for the objects to start operations. In this case the applet will instantiate any Local Object by starting a Generic Object Server (on the user side) and downloading the necessary files from the web server that was used to download the Poseidon applet.

3.2.1.1 Implementation of a Local Object

The implementation of a local object is very simple. The developer (part of the PDT) creates a small Java file that extends the GenericDefaultObject class.

The complexity of the implementation is kept low by moving to the parent class most of the tasks required for networking and wrapping.

The user has to write one method for the required action that he wants the object to perform. He also has to write a constructor with a call to the parent constructor.

The following example is a very simple one that creates a small frame for visualizing a single line of text (the numbering is not part of the code):

```
package CORBAObjects;
public class LocalSingleDisplayer extends GenericDefaultObject
{
public LocalSingleDisplayer(String name)
   {
    super(name);
   }
public void getBooleanOutput()
   {
            LocalSingleDisplayerFrame frame=new LocalSingleDisplayerFrame();
        frame.resultLabel.setText("Value:"+getfloatInput(0));
    frame.setVisible(true);
    }
  }
```

The LocalSingleDisplayerFrame is another Java code that can be implemented in many ways to show the results (see Appendix 5). The applet however, uses the above class to start the Generic Object Server. The object was already registered when the PR started so the GOS doesn't have to register this object again.

3.2.2 Remote Objects

Remote objects are different from local objects and in most cases they run on different computers from that on which the applet is running[10].

Remote objects are applications that start with the use of the Generic Object Server (Appendix 4).

They are also different from Local Objects in the fact that they can not be identified in the system until they are started (6.3). Their implementation also is generally considered more complicated.

3.2.2.1 Remote Object Implementation

As noted before, in order to make available an application to the MMS and use it as a Poseidon Object, the owner has to make the application compatible with the system. There are two ways to do this for Remote Objects.

The first way needs a sufficient understanding of computer programming and of the language used to write the application. Also it requires a basic understanding of programming with CORBA. Finally it requires the knowledge of the application code in order to make transformations to it.

It is a more elaborate way, which can prove to be useful only for special cases of application integration. It involves the modification of the application code and the use of existing code provided by the PDT. It is recommended only for cases where efficiency (although in most cases this involves only network efficiency) is above all. It will not be

[10] Although the user that started the applet can have started some Remote Objects on his computer before he started the applet, these objects do not count as Local. The applet will make requests to them using the same way as to objects located on remote machines.

covered in the context of this thesis because it requires an extended introduction to CORBA technologies and networking that are beyond the scope of this work.

The second way of making an application compatible with Poseidon is much simpler. In this case, a very basic understanding of a computer language such as Java or C++[11] is required. In this case the approach to the application is done through a system call in the same way, as the owner would execute the application from a shell window. The results are directed to a file and this file is returned as an output. In this example Java is used as the language for linking. The owner of the application has to complete the following steps[12]:

1. Download from the Poseidon Web Site the files named: GenericPEObjectBuildUp.jar and PEObjectTemplate.java and save them on a directory that belongs on the system's classpath.
2. Rename the file PEObjectTemplate.java to a name he prefers. The file extension should be .java.
3. Use a text editor or a Java development environment to open the file.
4. Replace all instances of the PEObjectTemplate word with the same name he used for the file name (without the extension).
5. Add a method that will make a system call to the application he owns. The user must know the number and types (String, int, etc) of inputs and outputs that the application needs and provides. For example, if the application needs an integer and a string as inputs and returns a File as an output, the user should write the following code (without the "Line #:"):

```
Line 1:         public float[]  getFloatArrayOutput()
Line 2:     {
Line 3:                 int parameter1=getIntInput(0);
```

[11] Currently the CORBA specification offers support for programming languages such as Java, C and C++. See [7] for a full list of all supported languages.

[12] Before proceeding with any of the following steps the user should have installed a Java Runtime Environment supporting at least the JDK 1.1 specifications [5].

```
Line 4:            String parameter2=getStringInput(1);
Line 5:            System call (application name parameter1 parameter2);
Line 6:            return file;
Line 7:          }
```

In this case we assume that the owner used to execute his application like: applicationName parameter1 parameter2.

The application takes two inputs and returns a File. The inputs are input#0 and input#1[13]. The user specifies in which port[14] (input) he wants to get the specified argument by specifying the port number to the parenthesis e.g. getStringInput(1). In a similar way getArrayOfFloatsInput(3) would return an array of floats from input#3.

The method names, such as getArrayOfFloatsInput(), are standard and specified by the Poseidon API. A complete list of all the available method calls can be reviewed in (Appendix 7).

The user compiles the file using a Java compiler and saves the .class file (The .class file should be saved on a directory belonging to the classpath).

The next step to making the application a Poseidon Object is to make a profile file that will be used to register the object to the PR.

For this procedure the user starts the Object Profile Creator (OPC) (4.1). The OPC is a Java application that collects all the necessary information required for the Remote Object to register with the Poseidon Repository (PR).

As soon as a profile exists for the Remote Object, it can start as a Poseidon Object.

The Remote Object starts-up by invoking the built-in server (GOS) which is implemented in the GenericDefaultObject class file. The user does this by calling his new compiled file with a Java interpreter. The Generic Object Server (GOS) will create the Generic Object Server Frame (GOSF) (Figure 3-1) which is a Generic GUI for starting

---

[13] Input numbering starts at 0.

[14] A port in this document is refereed to an input or output connection of a Poseidon object. For example an object that takes to numbers and returns its sum will have two input ports (one for each number to be added) and on output port (for the result).

the object. The user employs the GUI to navigate and load the profile file of the newly built object. By clicking on the button “Load Properties”, the GOSF reads the profile file and creates a list with all the information available about the new object. The user then is prompted to give a name for the object, which will be used to identify it on the network (a default name is automatically shown, which is the name used during the profile built-up).

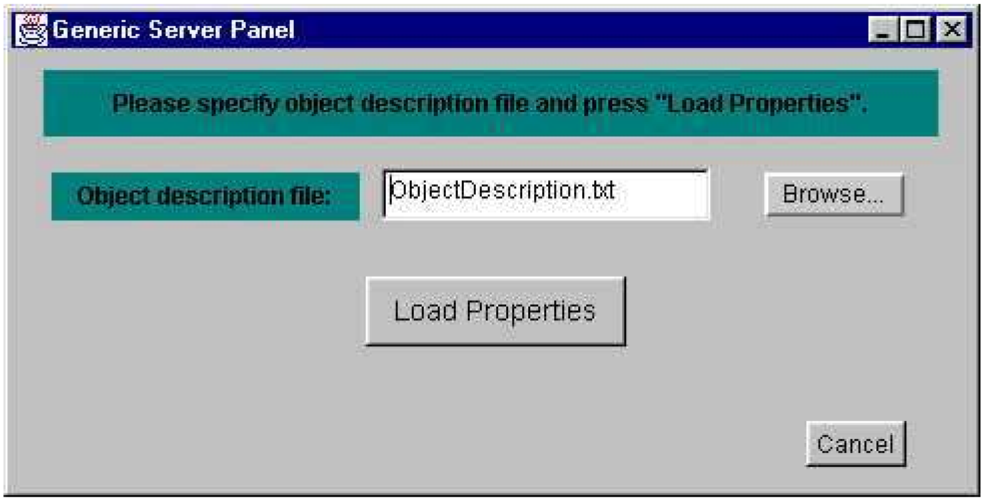


Figure 3-1 – Generic Server Panel

By clicking on the button “Start Object” the user activates to Object. The Generic Object Server uses the information from the profile file and registers the object with the Poseidon Repository. The repository responds with a Serial Number and the GOS uses this Serial Number to start a new server that will handle the incoming requests. After that, the new object server falls into an infinite loop waiting for incoming calls.

The Remote Object could be started again anytime in the future by using the existing profile. The user just starts the GOSF, loads the properties and clicks on “Start Object”.

3.3 The Poseidon Repository (PR)

The Poseidon Repository (PR) is used as a central database for collecting object information. Each object has to register first with the PR in order to provide services for the MMS. During the registration procedure the object supplies the PR with information related to its functionality, specifications and origin (see Appendix 8) and the PR returns to the object a unique Serial Number that is used to identify (the object) on the network.

By supplying a unique Serial Number for each object, it is possible to start numerous instances of the same server and have each one be identified as a different object.

Although a repository implementation with very similar capabilities is natively available from INPRISE's VisiBroker, the PR was built from scratch in order to provide such functionality that would allow it to easily implement standard querying and storing standards such as the latest Metadata standard for data and application indexing.

When a user starts a Poseidon Applet, the applet communicates with the PR and asks for the available objects. The PR in response replies with a list of all the available objects. The applet then builds up the object tree (see 3.4). Whenever the applet asks for any extra information about an object (e.g. a description of the inputs), the PR is contacted to supply the information.

3.4 The Poseidon User Interface (UI).

The user by means of an Internet browser[15] connects to the Poseidon Homepage and navigates to the page that contains the Poseidon main User Interface (Figure 3-2).

The Poseidon main User Interface consists of an applet[16] that is divided into three parts:

- The Main Design Panel
- The Object Tree, and
- The Control Panel.

The Main Design Panel takes the largest piece of the screen. It is located on the upper right side of the applet and it is used to lay down the objects and link them together.

---

[15] In order for the Poseidon Applet to be executed, a browser with support of Java's JDK1.1 should be used.

[16] The Poseidon Environment is also offered as a stand-alone application that a user can download and execute. This allows for better use of system resources since there is no consumption of them for the needs of the browser.

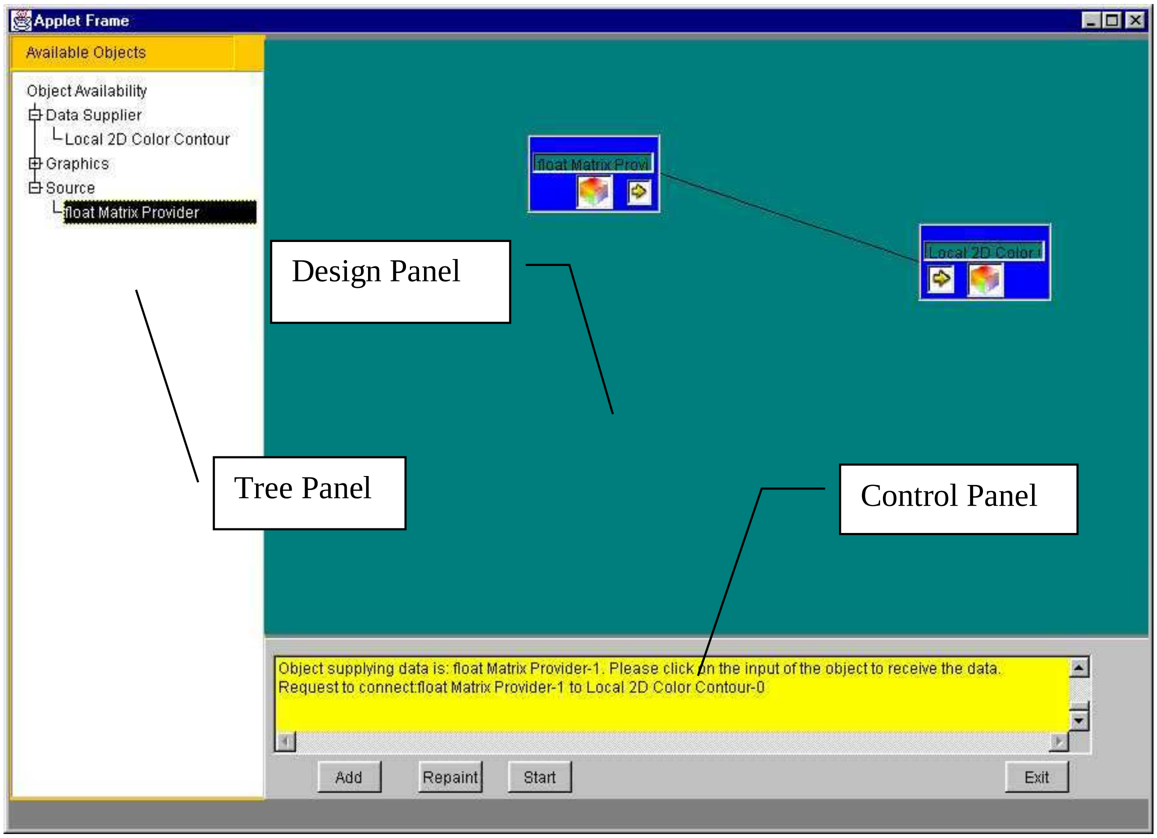


Figure 3-2 –Poseidon GUI

The Object Tree is on the left side of the applet's screen and is used to display the available objects on the network.

The Control Panel is on the bottom side of the screen and is used to host buttons and message panels for interaction with the user.

When the applet is loaded from the Poseidon Web Server, it contacts the Poseidon Repository and makes a request for the available objects. The repository responds with a list of all the active objects and the applet builds up a hierarchical tree (Object Tree) of all the objects available.

The user navigates through the tree branches and locates the object that he wants to add to his workspace. By pressing the button labeled "ADD OBJECT" on the Control Panel, a new Object GUI (OGUI)(3.4.1) (Figure 3-3) gets drawn on the Design Panel (DP).

Figure 3-3 - Object Representation on Poseidon Applet

The user can drag-and-drop the OGUI anywhere he wants on the DP. The user then adds the rest of the objects he needs to link together. As soon as the user has at least two objects and they are possible to be connected[17] he can click on the output[18] of an OGUI and then on the input of another OGUI to bring up the Port Connection Panel (PCP) (3.4.2). The PCP is used to make connections between the inputs and outputs of the objects.

By clicking on the center image of the OGUI, a user can bring up the Object Activation and Information Center Panel (OAICP) (3.4.3).

When the user has finished linking the objects that he wants, he can click on a termination node's[19] activation image, and after bringing up the OAICP he can ask the object to start operations.

The terminal object will conduct a number of checks to ensure the proper connection and availability of the objects, and if everything is set up correctly, it will ask all the objects connected to its inputs to start operations. These objects in turn will ask all the objects on their inputs to start and this will continue until the request reaches the objects that have no inputs. At this point the direction of operations is reversed. Objects that have ready data on their outputs, inform the objects to which they provide data to start their operations with the supplied data as inputs. The process eventually returns back to the

---

[17] It is obvious that in order to have two objects connected there must be a supply-request relationship between them. This means that two objects, for example, that have no inputs and have only outputs can not be connected.

[18] The input and output of an OGUI is represented by means of a golden arrow. An OGUI has always its inputs located on its left side and outputs on its right side.

[19] A termination node is an object that has only inputs. These usually are graphic or data displayers such as plots and tables.

terminal object that caused the initiation of the objects activation. Terminal objects are usually visualization or storage objects. The data received as inputs are used to create a graph or to be stored in a file. If the object is local and the terminal object is a visualization tool, then a new frame will pop-up with the graph representing the results. If it is a remote object, the graph will pop-up on the machine on which the terminal object was executed[20].

3.4.1 The Object GUI (OGUI)

The Object GUI is a small graphical representation of the object to be used (Figure 3-3). It consists of five parts:

1. The input image
2. The output image
3. The information and activation image
4. The object Name Label, and
5. The GUI frame

The three first images implement a Mouse Listener[21] and response on mouse clicks. The GUI Frame implements a Mouse Motion Listener[22] and is used to drag-and-drop the object GUI around the Design Panel. The Name Label is inactive during any user actions for the time being. The user clicks on the OGUI to interact with the object. By clicking on the input/output images, he brings up the connection panel (3.4.2) to perform object linking. By clicking on the information and activation image, he brings up the OAICP to perform more sophisticated actions.

---

[20] This is a nice way to send results on a remote user without having him performing a single action.

[21] A Mouse Listener is a Special Java Interface used to capture user actions such as mouse clicks and keyboard presses (for more details see [12][13]).

[22] A Mouse Motion Listener is another Java Interface, which is used to capture user actions such as dragging of objects (for more details see [12][13]).

Finally the user can click and drag the OGUI by grabbing it from anywhere on the GUI frame. The OGUI is only used as a utility to simulate the workflow. It has to be mentioned that having an OGUI on the applet doesn't mean that the specified object is been executed on the computer running the applet.

3.4.2 The Port Connection Panel (PCP)

As noted before when a user clicks on the output of an object and then on the input of the object, the PC Panel shows up (see Figure 3-4). The PCP lists on the left side the outputs of the object providing data and on the right side the inputs of the object requiring data. On each side there is a label with a description of the port (a name or a user defined description), and a label indicating the type of data provided or required (String, integer, etc) and a small button.

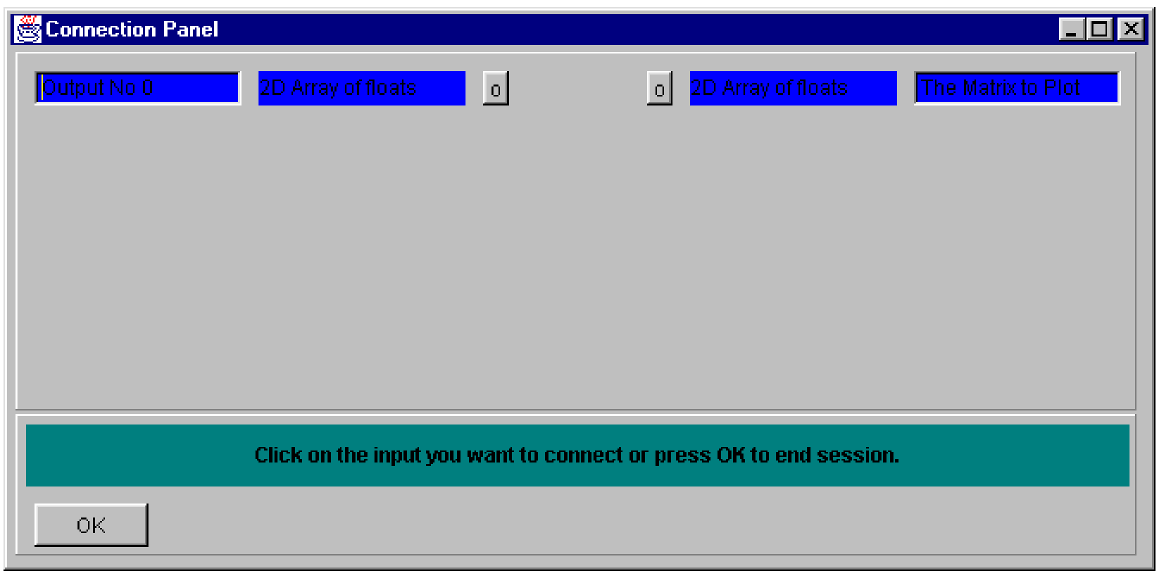


Figure 3-4 – Connection Panel

The user clicks on the button of the output (supplier) to which he wants to make a connection and then the button of the input of the object on the right (requester). If the two types are compatible (e.g. String to String or ArrayOfFloats to ArrayOfFloats) then a connection is recorded and a line is drawn between these two buttons indicating that a connection has been made[23]. If a button shows to be inactive (it has letters in gray and clicking on it causes no action) it means that the specific output or input has already a connection with an other input or output from a previous connection session.

To break a connection between two ports, the user must bring up the PCP that includes as active the two ports. Then by clicking on the output once will break the connection.

3.4.3 The Object Activation and Information Center Panel (OAICP)

The OAICP provides functionality that allows the user to review details about the object. For example when a user clicks on the information and activation image of the object's GUI and brings up the OAICP he can further click on the button labeled "Object Details" and get a list of the object details. For the time being such an action will give the short description of the object as supplied from its creator.

The most use of the OAICP however is for starting the execution of the workflow. As explained in section 3.4, when the user clicks on the "Start Object" button of a terminal object the objects of the workflow will start operations.

The "Show Interactive Panel" is a button that can be seen on the OAIC Panels only for objects that have been registered as having an Interactive Panel section 4.1. It is used to bring up the object-specific interactive panel in order to initialize the object's operations.

The OAICP also is used to remove an unwanted object. The user clicks on the button labeled "Remove Object" and the object is removed from the design panel.

3.4.4 The Object Messenger (OM)

The object messenger is a CORBA object used from the distributed objects and the Poseidon applet to exchange information. It is used during the initial stage of object connectivity and availability check-up as well as during the object operations to provide the applet with feedback about the state of the operations. The OM is responsible for

[23] However the connection is not real. This is just an indication of how the objects will finally communicate to each other and the exchange of data will not start until a specific request is made.

transmitting messages from one object to the other and from objects to the Poseidon applet.

Its main use is to inform the Poseidon applet for any errors that occurred during the workflow activation process.

4. Available Tools for the Poseidon Environment

4.1 The Object Profile Creator (OPC)

The OPC is a utility that is used by developers that need to convert existing applications to Poseidon objects. After the wrapping of their code (as described in 3.2) the user has to create a profile for this object. A profile includes information about the object linking requirements (number of inputs, type of data, etc) and descriptions about the object (see Appendix 8 for a full list of the profile data). The OPC is an application that a user starts up in order to create this profile. In order to start up the OPC, the user calls the Descriptor.class file from a Java virtual machine (at shell prompt: Java Descriptor). The OPC's GUI (Figure 6-1) has the basic (and required) fields for describing the object.

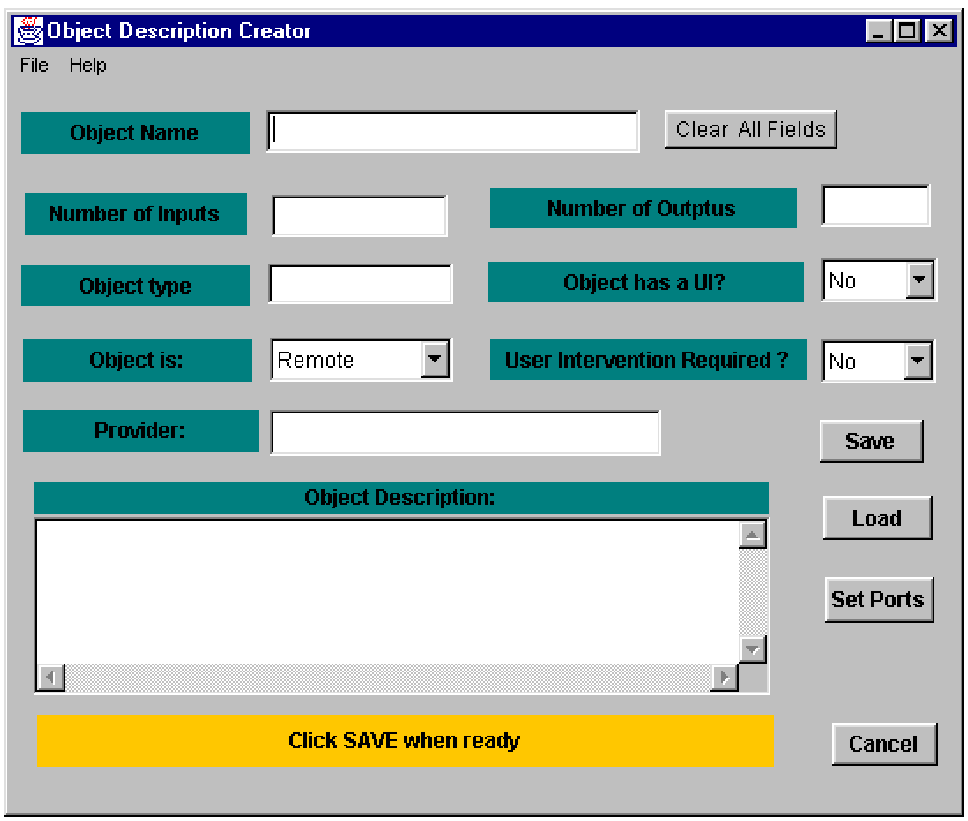


Figure 4-1 – Object Descriptor (Main Panel)

The user fills in the fields and then clicks on the button labeled "Set Ports". This will bring up the "Ports Details Frame" (Figure 6-2) which is built up on the fly and depends on the number of inputs and outputs the user has specified in the first frame (Figure 4-2). Here the user can specify a description for each input or output (a default description is already set) and also specify the type of data the inputs receive and the outputs supply. When done and after returning to the main panel, the information of the object has to be stored in a profile file. The user then presses the button labeled "Save" and navigates the filer to the directory that he wants to save the profile file[24].

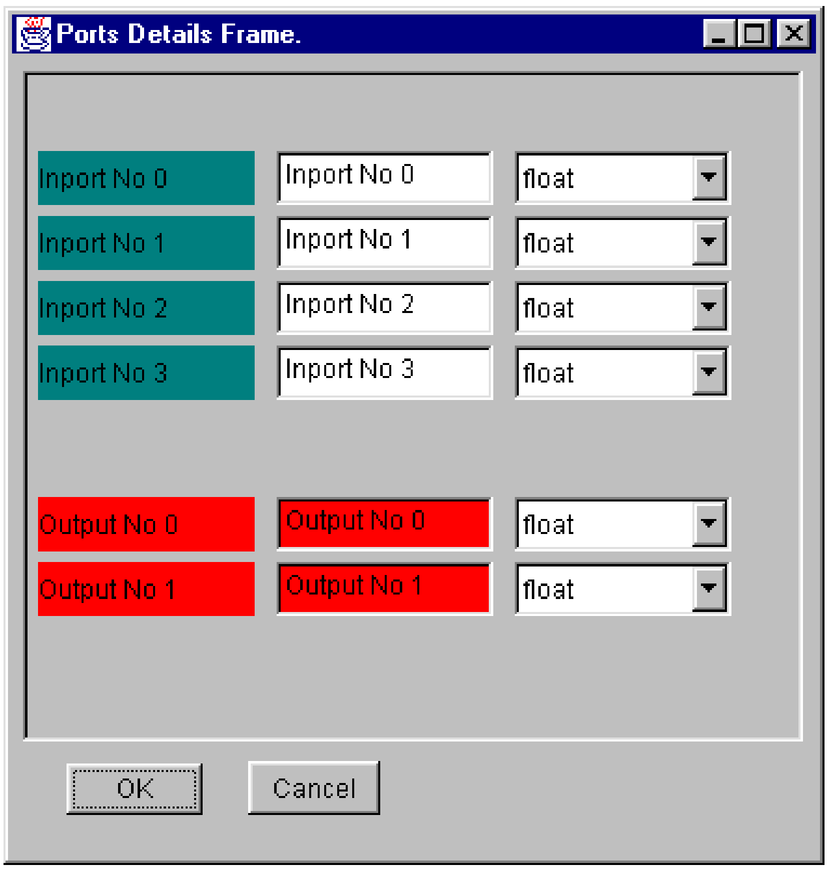


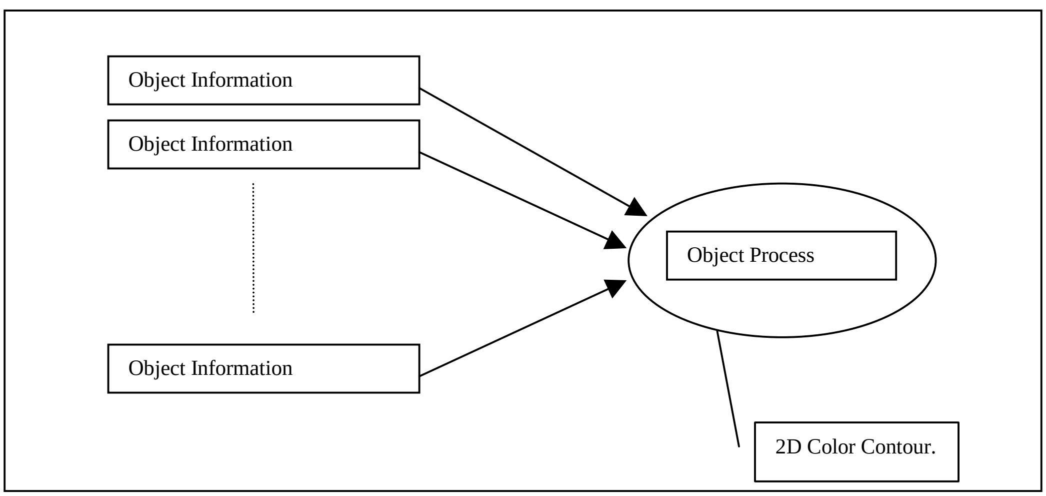
Figure 4-2 – Object's Ports Description Panel

[24] The user should remember where he has saved the profile file in order to retrieve it when it will be necessary for the object to start up.

4.2 *Extended Features for the Objects of the MMS*

4.2.1 Interactive Panels

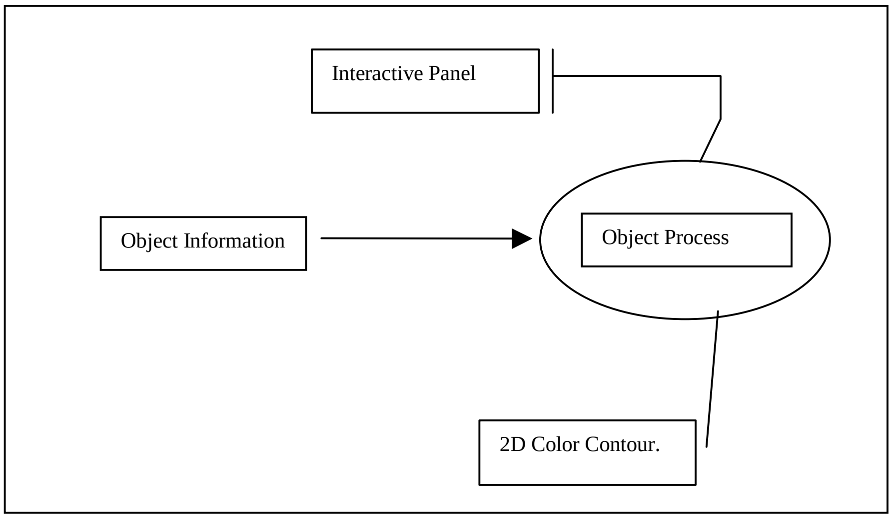
Interactive panels are features that allow the initialization of the objects before any request is made to them. An example is the interactive panel of the 2D Color Contour object. In the case of the 2D Color Contour (2D CC) (an object with one input and no output), a user sends a matrix of numbers and he expects to see a visual representation of them. However, graphs require more information than a matrix of numbers. The units and axis titles are some of this additional information. The 2D CC could be designed to have as many inputs as the number of parameters required to totally describe the graph (including the title) (Graph 4-1). But this would require that a service (usually in the form of a server) should be started for each of them providing the necessary information. It is obvious that such a way of handling the problem is far too inefficient. The solution is to use an Interactive Panel.



Graph 4-1 – Object Initialization (No Initialization Panel)

The Interactive Panels are GUIs that are custom built for each object (in some cases, existing panels can be used for other objects as well). Their purpose is to setup initial parameters so that the object can fulfill a request. In the case of the 2D CC its Interactive Panel would be a GUI asking for all the information such as the title, the x-axis title, the units used, etc. Interactive Panels are called up from the Object Server and not the Poseidon Applet[25].

The Interactive Panels are directly related to the Availability of an object as described above section 3.4. If the user has not used the Interactive Panel to initialize an object and he asks the workflow to execute, an error will show him that the object is not yet available.



Graph 4-2 – Object Initialization with Initialization Panel

Objects that are unavailable are marked with the stop sign when an execution is requested. A user can find out if an object is available or not by asking the workflow to execute. The objects that show the stop sign are unavailable. The user can then use the

OAICP (3.4.3) to bring up the Interactive Panel off the object and initialize it.

[25] The user must be aware that when he asks for objects to show their interactive panel, the panels will popup to the machine screen that the object will be running, not on the machine executing the applet. Although someone could argue that leaving the option to start an Interactive Panel for a remote object has no meaning, this is not true. In some cases a user can be running a remote object in the same machine that runs the applet or from a computer located close to him. Having the option of starting Interactive Panels for remote objects, and from the context of the Poseidon applet can prove to be very useful.

4.2.2 Setting up an Interactive Panel for a New Object

Setting up an interactive panel for a new object is a very simple procedure. The developer that wants to make available an object with the use of an Interactive Panel has to follow the simple rules below:

- In the object constructor methods add the statement “availability=false;”
- Add a method called getInteractivePanel() in which he will call the custom built panel for this object.
- Make a call to the parent object to set the availability parameter to “true” whenever the initialization is complete.

5. Application of the Poseidon System in the Marine Industry

The last years have seen the amount of information available on the World Wide Web grow exponentially. The information is related to almost every single profession and scientific field. Data are provided for free or on a subscription basis. In most cases data are handled from a central databank that responds to user queries.

For the area of marine transportation and trade, a large variety of data exist providing information such as cargo movements, shipping prices, ships schedules, related weather conditions, oil prices and much more ([22][23][24][25][26][27]). Although the information is indeed valuable “as is”, a much more extended use of it could be achieved if it was possible to parse it into existing tools that perform various operations (statistical tools, pattern recognition, etc). Data from various websites could be linked together and piped to a correlation routine that would be able to calculate behavioral patterns in real time. For example oil price data (collected from a related website) for the port of Seattle[26] could be potentially linked with shipping cargo prices for vessels operating from that port and unveil a correlation of the two data sets. The example could be criticized as unimportant since it is well known that there is a positive correlation between these two factors. However, it was used to show the functionality that is desired. In a more extended situation data sets that seem to have no correlation between them could be combined to unveil hidden patterns. An example that could be characterized as “unconventional” could be the linking of the following data:

- Traffic data for ship movements between the Port of Tokyo and the port of Seattle
- Sea temperature distribution for the area of the above route
- Data of fish species for the same area.

---

[26] Although the specific data might not be available at the time of this thesis writing, it is the author’s estimation, that in the near future, the amount of data available on the Internet will be so large, that it will be enough to create valuable patterns for many professional fields.

The linking of such data sets could potentially unveil the effect of ship routes on the distribution of fish populations for the same area. This information could then be used either for establishing more efficient marine protection policies or rearrangement of fishing vessels for better catch.

Many studies have already been done to find patterns in various fields [28][29][30]. However, in many cases working with data was a very tedious process. It had to do with finding the data, acquiring them (making requests from the agencies that own them), formatting them (usually data come in different formats such as tables, lists, etc) and then writing native code or piping them to existing software in order to work with them. Although the data that are available on the Internet do not follow a common standard for storing and retrieving, in most cases they are provided to the user (client) in a table format. This table format is usually the automated response of a Common Gateway Interface (CGI) script running on the server side. Both the request and the response are in an HTML format.

5.1 Generic Web Site Parser. Data Collection and Retrieval

5.1.1 Description

The HTMLParser[27] is an application written totally in Java and allows the user to extract information from the Internet. Although it can not extract any kind of data available, its architecture allows scaling and its capabilities are expandable.

The architecture of the HTMLParser is based on a hierarchical relationship of HTML reserved names (HTML Tags [10]) that can be extended to any kind of descriptive language of this type. Consequently the HTMLParser can easily be modified or extended to parse XML documents [18], which seem to be the next step into web publishing. The

---

[27] The HTMLParser is the main class file used for parsing documents written in an HTML format. However HTMLParser uses a list of other classes written specifically to assist this task. For a complete list with description of the HTMLParser files see Appendix 6.

HTMLParser is not a stand-alone application. It is used in conjunction with applications to provide parsing of documents published in an HTML format.

In the Poseidon environment, the HTMLParser is used with the Web Data Miner (WDM) (5.1.2). With the WDM the user has the ability to start a service that will query databases available on the Internet and extract the data from the server's response.

A user working with the Poseidon applet can then see the WDM as one of the available objects to use in the Object Tree Panel (OTP). The WDM is an object that has a number of inputs equal to the number of query variables and one output, which is an array of strings (or vector of vectors), representing a table of data[28].

In general the HTMLParser has the same functionality as a web browser[29]. However, its use in combination with the WDM is more active since the user can link the operations of the HTMLParser to other applications.

5.1.2 The Web Data Miner (WDM) and its Integration with the Poseidon Environment

The WDM is a stand-alone application that uses the HTMLParser functionality to parse documents written in HTML format. Although the WDM can be used as stand-alone, its main purpose is to serve as a service provider (object) for the Poseidon Environment.

The user initiates the WDM by calling the HTMLDataMiner.class file with the Java interpreter (java HTMLDataMiner) from a shell prompt. This will bring up the WDM main user interface (Figure 7-1).

---

[28] Although the WDM provides only one output (which is an array of strings) it can be easily extended to provide more than one outputs.

[29] Although the HTMLParser is presently restricted to a specific number of HTML Tags, it can easily be extended to the full set of the HTML specifications.

Figure 5-1 – The Web Data Miner GUI

The user has to go through the following steps then, in order to start a service that will start extracting data:

1. Make a sample test and view the results.
2. From the sample test, choose the desired inputs and output
3. Start the service.

The following paragraphs describe the above procedure in more detail.

5.1.2.1 Make Sample Test and View Results

The user specifies the URL of the document containing the query form. He uses the field next to the label “URL to parse for input fields”. For example, in order to parse the shipbrokering website called Internet ShipBrokers ([26]) and find the available open cargoes for a specific time period and area we would write to the URL field: “http://www.netshipbrokers.com/asp/fqcar.asp”. This is the first page that contains the input form for parsing the Internet ShipBroker’s database. Then the user would click on the button labeled “Find Input Forms…”. This action will cause the WDM to initiate an HTMLParser class and start parsing the relative website. When the program has finished with the parsing it will create a number of frames, one for each query form discovered in the specific HTML document[30]. The frames include the viewable[31] components that can also be seen by using a common Internet browser[32]. By clicking on each component, the WDM will print details about it, as discovered in the HTML document[33]. The user specifies which form will be used for the query by clicking the form and then clicking on the "Select Form" of the WDM frame. This will cause the rest of the forms to disappear (Figure 5-2).

The WDM will then show a message "Select the component to be input #0", prompting the user to click on the component[34] to be used as input #0[35].

---

[30] A lot of websites have more than one query-forms in each page.

[31] It is quite often that queries using HTML documents are performed using a number of hidden parameters too. They are usually (key=value) type of information that the servers require in order to perform the specific query. They are included in the HTML document of the query page but are not visible to the user. The HTMLParser reads these hidden pairs and returns them to the application that started it (the HTMLParser, in our case the WDM). When the user has finished with the setting of the input parameters to be used for the query, the WDM will automatically add the hidden values in order to correctly fulfill its task. However, the whole task is transparent to the user, as there is no need for extra complexity.

[32] Although in most cases the components seen in the frames can easily be identified relative to those seen with a regular browser, it is always helpful for the user to have a web browser to show the same site and compare the fields.

In most cases, this will also unveil the websites that have dynamically updated WebPages and can produce misleading the results.

[33] It has to be noted that none of the components on the frame will perform any action to the queried database. They are used for assistance to set up the inputs of the object as will be used in the Poseidon context. As such, clicking on a "Submit" button will only cause the WDM to print the button's info such as name and value.

[34] The components to be used must be components that supply data that the user would fill if he was using a common browser. This means that buttons can not be used as inputs.

[35] In the Poseidon System object inputs and outputs are numbered starting with 0.

Figure 5-2 – Inputs Form created by the WDM

The program will keep asking the user for inputs until all the fields have been specified.

When the user has set all the inputs, the WDM will show the following message:

"All Fields have been set. Please input test values in the fields to perform a test run. Then click on TEST RUN". The user can then fill the fields of the input form with sample values to make a real test. By pressing the button "Test Run" the WDM collects all the information, transforms them into the necessary URL-encoded format and sends them to the server in the same way as if the user had filled the form using his internet browser. The server receives the request and sends back an HTML document with the results in the same way it would send it back if we had used an internet browser. The HTMLParser then takes the HTML document and parses it to find tabulated data. For each set of data encapsulated in a HTML Table format the HTMLParser will create one frame. On each frame a table is created to represent the tabulated data. A large button with a label "Press to select this Table" is also added at the very bottom of each frame. In the case of the Internet ShipBroker query, five frames are created. Four out of the five table frames seem to have data that are not useful for our case[36]. However one of them has exactly the data we need (Figure 5-3). By selecting this table all the table frames disappear and a message is printed on the WDM message panel indicating the index of

[36] It is very often that HTML documents formulate in a tabulated way much more information than just data. It is very common to use HTML Tables to align commercial banners and images.

the table selected (in this case #2)[37]. The next step is to start the WDM as a Poseidon Object.

Cargo ID	Appeared ...	Cargo De...	Load Port/...	Discharg...	Start Date
8953	3/13/99 3...	120000 ...	Izmir/Turk...	Rotterda...	4/9/99
8944	3/13/99 3...	18,000M...	Hualien/T...	Chittagon...	4/10/99
8924	3/13/99 7...	TTL CBM ...	Bombay/I...	Dakar/Se...	4/5/99
8921	3/12/99 7...	2100 MT...	Bremen/...	Sousse/T...	4/25/99
8907	3/12/99 1...	16500 C...	Novoross...	Alexandri...	4/5/99
8917	3/12/99 1...	BAGGED ...	Walvis B...	Dar es S...	4/15/99
8915	3/12/99 1...	rice	Ho Chi Mi...	Dar es S...	4/15/99
8903	3/12/99 1...	T/C - NE...	Mediterra...	Mediterra...	4/15/99
8883	3/12/99 9...	409,3 CB...	Novoross...	Umm Qa...	4/21/99
8896	3/12/99 9...	four piec...	Vigo/Spain	Liverpool/...	6/15/99
8866	3/12/99 8...	5/6000 m...	Santos/B...	Tuxpan/M...	4/15/99
8816	3/11/99 1...	ABOUT 3,...	Mago/Ru...	Busan/S...	4/5/99
8812	3/11/99 1...	12000MT...	Xingang/...	East Afric...	4/10/99
8806	3/11/99 1...	5,000MT ...	Fukuyam...	Keelung/...	4/15/99
8811	3/11/99 9...	bulk carbi...	Izmir/Turk...	Koper/Sl...	4/11/99
8808	3/11/99 9...	STEEL P...	Porto No...	Casabla...	4/11/99
8762	3/11/99 8...	30000 M...	Gresik/In...	Lagos/Ni...	4/15/99
8764	3/11/99 8...	10000/15...	Yantai/Ch...	Caribbea...	4/15/99
8743	3/11/99 7...	20000/5 ...	Visakhap...	Novoross...	4/25/99
8721	3/11/99 7...	ABT 17/1...	Yokoham...	Mexican ...	4/8/99
8705	3/10/99 1...	18,000M...	Hualien/T...	Chittagon...	4/10/99
8703	3/10/99 1...	4,000/5,0...	Xingang/...	Japan Se...	4/10/99
8675	3/10/99 9...	20,000 M...	Banjarm...	Sriracha/...	4/12/99
8659	3/10/99 9...	6000/5 b...	Kandla/In...	Chittagon...	4/10/99
8634	3/10/99 8...	1000 CU...	Novoross...	Lattakia/...	4/10/99
8635	3/10/99 8...	1000 CU...	Novoross...	Alexandri...	4/10/99
8610	3/10/99 8...	22.000 M...	Tocopilla/...	Imbituba/...	4/5/99
8578	3/10/99 7...	10.000 B...	Mexican ...	Mombas...	4/10/99
8571	3/10/99 7...	20.000 B...	Mexican ...	Djibouti/...	4/10/99
8579	3/10/99 7...	10.000 B...	Mexican ...	Lobito/An...	4/10/99
8581	3/10/99 6...	barite in ...	Zhanjang...	Oshawa/...	5/10/99
8559	3/9/99 12...	about 60...	Paranag...	Jakarta/In...	6/1/99
8552	3/9/99 12...	1100 mt...	Taganrog...	Constanz...	4/15/99

Press to select this Table

Figure 5-3 – Results Table from Parsing "Internet Shipbroker's Website"

---

[37] Since the Table component in HTML documents has no "name" attribute which could potentially be used to identify the desired table to work with, the HTMLParser identifies it by an index number which indicates the sequence of the discovery of this table in the HTML document.

The user presses on the button labeled “Start CORBA Object” and a new frame appears (Figure 5-4) that prompts the user to fill it with information about the object.

Figure 5-4 – Description Panel for WDM Created Object

This is the same procedure to create a profile to be used for registering with the PR[38]. When the users fill the fields and press the “OK” button, the WDM follows the same procedure as a remote object (registration of with the PR, etc) and becomes an available service on the network (Figure 7-5).

The next time a user will start a Poseidon applet, he will see in the object tree an object with the same name as the one he used to name the WDM in the description panel. He can then add it to his workflow in the same way he adds any other object and link it with other applications. The use of the WDM (as a Poseidon object) in combination with other available tools can be greatly expanded for additional functionality.

---

[38] In this panel however, there are much fewer fields than those that appear with the OPC. The reason is that many of the fields (number of inputs, data types, etc) have been specified during the testing procedure of the WDM.

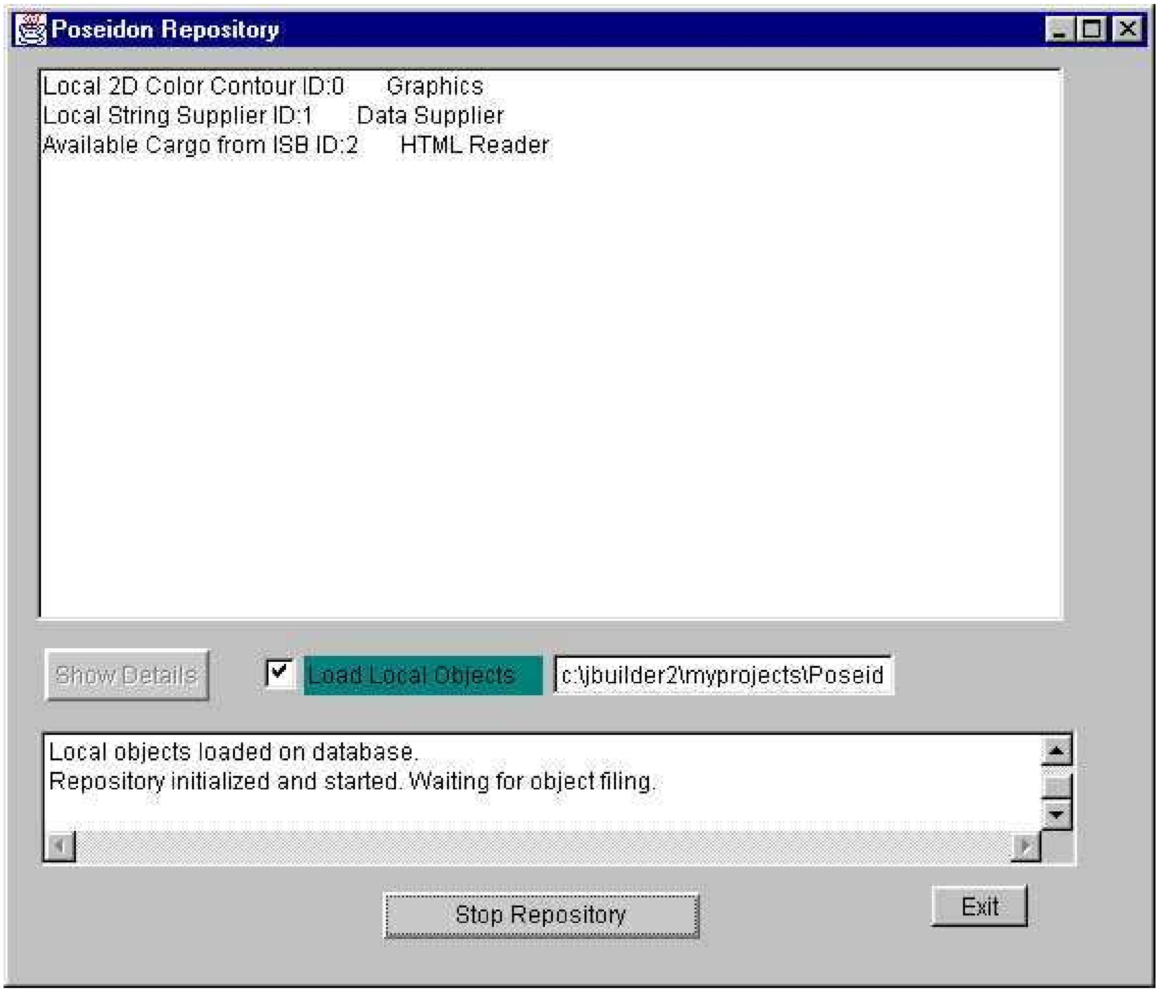


Figure 5-5 – Poseidon Repository GUI

As more and more applications will become available as Poseidon objects, more and more options will become available.

5.1.3 Notes on using the WDM and HTMLParser.

Although the HTMLParser was built using the specification of the HTML language, there are some issues that could cause the HTMLParser to produce wrong results or not work at all. Following is a list of a few that have been discovered during the testing face:

1. The HTMLParser would not be able to handle errors in the HTML document if not specifically programmed to do so. This was the case of closing tags

been used more times than it should[39]. Although the HTMLParser now handles this specific error, other errors in documents could create problems with unpredictable results.

2. Some websites continuously change their context and should be avoided. In some cases websites add or remove HTML components (input fields, tables, etc) dynamically so the user can not be sure what will be the parsing results in a future run of the WDM.
3. The HTMLParser does not support the HTTPS (secure HTTP) protocol at this point. The HTMLParser will not parse Websites that use this protocol.
4. Users should be aware that query forms that are part of a Java applet[40], could not be parsed by the HTMLParser.

6. Conclusions – Suggestions for Future Research and Development

Poseidon is a software system, which can be applied to multiple application areas. The development of the code took many paths and changed a lot from the first version. If it was to be written again from scratch, many things would have been designed with a totally different approach. The reason for this is that during the design phase, the developer faces a lot of problems to solve or discovers some new exciting functionality that he would like to include. However changes can not always be done for a number of reasons (development delay, planning of new strategy for application functionality, etc).

---

[39] During the testing period it was discovered that some web sites including well-known sites such as www.Altavista.com had a number of errors in their websites. For example the use of double </td> closing tags instead of single. This used to mislead the creation of the HTMLParser hierarchy list with totally wrong results.

[40] For example with the oceanographic *in situ* data provided by EPIC, see: http://www.epic.noaa.gov/epic/dbsummary/dbsummary.html

Another reason for reconsidering of changing parts of the code is that during the development period, a number of new technologies have become available that could be used to overcome existing problems (treated with a less optimal solution) or enhance the existing code. Since the initiation of the design of the Poseidon code, there has been considerable progress in the fields of Java and CORBA technologies. The latest Java API includes new extended functionality (especially in the field of 2D and 3D graphics creation and handling) that could be very interesting for the case of visualization applications for the Poseidon Environment. Also the latest CORBA specifications [7], include new enhanced functionality such as determination of method return type during runtime, which would be very useful for the design of the remote object's generic functionality[41].

The following are a number of suggestions for future work. They are listed according to the applications mentioned in the previous paragraphs.

6.1 Suggestions for the Poseidon Repository

- Change the object storing procedure to store different object descriptions and active objects. Currently each new object asking for registration with the repository, stores all the information (name, type, number of inputs, etc) associated with it. This means that two identical objects started with different names will have two different sets of data stored in the repository database. However almost all of the data in this case are identical and a new approach can be designed to optimize object registration.
- Make the PR to read and write data using Metadata specifications.
- Move the startup of the local objects from the Repository functionality to a stand-alone application that could be started automatically from the web server serving the Poseidon files. At this point the registration of the local

[41] It has to be noted that one of the most difficult parts in the design of the code for the remote object wrapping was to deal with the return types of the method calls. The technique used to overcome this problem is quite complex and a little non-conventional (Appendix 7).

objects is handled from the PR during the initialization phase. This limits the start up of the PR only in the environment (path) of the web server.

- Allow the existence of more than one PR's in order to provide load balancing during the registration or information request process. Such a scheme could also be extended to provide fault tolerance by migrating PR information between them, in case that one goes down.

6.2 Suggestions for the Poseidon Applet

- The Poseidon Applet (or application) can handle up to 50 objects at the same time. This is controlled by the MAXIMUMNUMBEROFOBJECTS parameter in the Poseidon.java file. A number had to be set due to the fact that the objects are stored in an array (the array GCO). In Java a creation of an array has to be done with a final[42] value. Although the limit could easily be extended to larger numbers, this could become a threat for computers with low RAM capacities. The reason is that the array will reserve a piece of memory proportional to its size. The suggestion for future work is to change the array to be a vector. This will allow for reallocation of memory according to the number of objects. It requires though a considerable amount of work and attention in order to perform the casting operations during the retrieval of data from the vector.
- Have one messenger object for each Applet started. Currently each applet will start a messenger with the same name, and pass it to the objects during the "connect"[43] phase. Although it should not be a problem to have only one

---

[42] A final value in Java is a value that is set during the code creation and can not be changed during runtime.

[43] The "connect" phase is when the user clicks on a terminal object and asks for object activation. Then after checking if all the connections have been specified by the user, the applet sends a message to the terminal object to start linking with the objects at its inputs.

messenger, the communications can become congested if more than one applet is started with a negative effect on performance. A solution could be to have each applet get an ID (e.g. from the repository or web server) and then initiate a messenger using a unique name by means of this ID.

6.3 Suggestions for the Generic Default Object

- Extend the functionality of the GDO to be able to register objects to the PR even if they are not started. The functionality provided with Visibroker's API by means of the Object Activation Daemon (OAD) could be used to automatically activate registered objects that are capable to start-up and provide services.

6.4 Suggestions for the HTMLParser.

- Extension of the HTMLParser to read ordered and unordered lists[10].
- Creation of a Generic Class File for Tag Identification (GCFTI). By means of a generic class file I mean a class file that is not dedicated to identifying, storing and presenting the information of a specific tag. For example the HTMLFormComponent class is responsible for the above actions related to a FORM HTML tag. A GCFTI could increase performance by reducing the compiled code, and provide the means for more efficient parsing of new tags (tags that show up by HTML or XML extension [18]).

This will eventually create an iterative process that will make requests to all the objects to link to each other as specified in the FOI.

7. References

[1] P. Wariyapola, N. M. Patrikalakis, S. L. Abrams, P. Elisseeff, A. R. Robinson, H. Schmidt, K. Streitlien, Ontology and Metadata Creation for the Poseidon Distributed Coastal Zone Management System, IEEE Advances in Digital Libraries Conference, ADL ’99, NY: IEEE, 1999.

[2] S. Shinners, Modern Control System Theory and Design, J. Wiley, New York 1998.

[3] D. Towel, Transfer Function Techniques for Control Engineers, Iliffe Books, London 1970.

[4] Object Linking and Embedding (OLE), The Object Linking and Embedding Developers Network, http://www.microsoft.com/oledev/.

[5] Java JDK 1.2 (Java 2).

[6] The Object Management Group (OMG) background information, web site address http://www.omg.org/omg/background.html.

[7] The Object Management Group (OMG), CORBA 2.2/IIOP Specification, web site address http://www.omg.org/library/c2indx.html.

[8] Robert Orfali and Dan Harkey, Client Server Programming with Java and CORBA Second Edition,Wiley Computer Publishing,1998.

[9] Andreas Vogel and Keith Duddy, Object Management Group, Java Programming with CORBA, Wiley Computer Publishing, 1997.

[10] Richard Johnsonbaugh & Martin Kalin, Object Oriented Programming in C++, Prentice Hall, 1995.

[11] Peter T. Davis, Securing Client/Server Computer Networks, Mc Graw-Hill, 1996.

[12] John Hunt, Java and Object Orientation: An Introduction, Springer-Verlag, 1998.

[13] Gary Cornell & Cay S. Horstmann, Core Java – Second Edition, SunSoft Press, 1997.

[14] David Geary & Alan L. McClennan, Graphic Java – Mastering the AWT, SunSoft Press, 1997.

[15] David Flanagan, Java Examples in a Nutshell: A Tutorial Companion to Java in a Nutshell (Nutshell Handbooks), O'Reilly & Associates, 1997.

[16] Mary Campione & Kathy Walrath,The Java Tutorial: Object- Oriented Programming for the Internet (Java Series), Addison-Wesley (also available at the www at http://java.sun.com/docs/books/tutorial/index.html), 1998.

[17] The World Wide Web Consortium, HyperText Markup Language, http://www.w3.org/.

[18] The World Wide Web Consortium, Extensible Markup Language, http://www.w3.org/XML/.

[19] INPRISE Corporation, Inc., Programmers Guide for "Visibroker for Java 3.3", http://www.inprise.com/techpubs/books/visibroker/vbj33/index.html.

[20] INPRISE Corporation, Inc., Reference Guide for "Visibroker for Java 3.3", http://www.inprise.com/techpubs/books/visibroker/vbj33/index.htm.l

[21] INPRISE Corporation, Inc., Gatekeeper Guide for "Visibroker for Java 3.3", http://www.inprise.com/techpubs/books/visibroker/vbj33/index.html.

[22] Containership Databank and Capacity Forecasts, http://www.mdst.co.uk/infoprod/databank.html.

[23] Oil World, http://oilworld.com/.

[24] Waterborne Commerce Statistics Center, http://www.wrsc.usace.army.mil/ndc/wcsc.htm.

[25] National Transit Database, http://www.bts.gov/ntda/ntdb/.

[26] Internet Shipbrokers, http://www.netshipbrokers.com./

[27] Bloomberg, http://www.bloomberg.com/welcome.html.

[28] Mike Field & Martin Golubitsky, Symmetry in Chaos: A Search for Pattern in Mathematics, Art and Nature, Oxford University Press, 1996.

[29] Ian Stewart & Martin Golubitsky, Fearful Symmetry: Is God a Geometer?, Blakwell Publications, 1992.

[30] George M. Hall, The Ingenious Mind of Nature: Deciphering the Patterns of Man, Society and the Universe, Plenum Press, 1997.

Appendices

Appendix 1 – Abbreviations

Abbreviation	Description
API	Application Programming Interface
BOA	Basic Object Adapter
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
FOI	Flow Of Information
GCFTI	Generic Class File for Tag Identification
GOS	Generic Object Server
GUI	Graphical User Interface
HTML	HyperText Markup Language
JDK	Java Development Kit
JRE	Java Runtime Environment
MMS	Model Management System
OAICP	Object Activation and Information Center Panel
OGUI	Object GUI
OM	Object Messenger
OMG	Object Management Group
OPC	Object Profile Creator
ORB	Object Request Broker
OTP	Object Tree Panel
PCA	Poseidon Central Authority
PCP	Port Connection Panel
PDT	Poseidon Development Team
PE	Poseidon Environment
PMS	Poseidon Metadata Standard

Abbreviation	Description
PR	Poseidon Repository
PUI	Poseidon User Interface
RMI	Remote Method Invocation
UI	User Interface
WDM	Web Data Miner