# CORBA and the CLEO III Data Acquisition System

A. Wolf, K. Honscheid
*The Ohio State University, Columbus, OH 43210*

The new CLEO III DAQ system will be monitored and controlled by a networked system of CPUs. To achieve a dynamical and simple connection method between the various processes we chose the CORBA standard communication method. A port to PowerPC based VME boards was done in close collaboration with Visigenic Corp.

*Key words:* Data Acquisition; Slow Control; C++; Java; CORBA; PowerPC; VxWorks; VisiBroker;

## 1 Introduction

The distributed design of the CLEO III Data Acquisition [1] demands a network embedded monitor and control system ("Slow Control Network"). Since using *standards* is a very useful way of setting up a communication system, we decided to build the Slow Control Network on top of the newly defined CORBA [2] architecture. The general idea is to hide the (hardware) details of the communication in "objects" and "object interfaces" that can be located during *run-time* and called from *anywhere* in the network. Since object–oriented programming languages (OOPL) can hide the complexity of objects and give a clean method/function interface, using a CORBA object is *transparent* for the programmer, meaning it looks like calling any other object in the program.

These technologies were available for UNIX and NT workstations for more than a year. Since parts of the system will run in embedded CPUs with the VxWorks 5.3 [3] real–time operating system, we needed this library on this platform, too. In close collaboration with Visigenic Corp. [4] we ported their CORBA 2.0 implementation (VisiBroker2.0) to this system using the PowerPC 604 processor and the GNU C++ cross compiler. This is the first port of CORBA 2.0 to any real–time system.

## 2 Using CORBA to Send Requests

The CORBA specification define how an "object interface" is declared, bound to a specific language, and which functions are provided to locate and use these interfaces. It is up to the implementor, how this connections are setup, but the protocol on this link is fixed. Other specifications (CORBA 2.0 IIOP) ensure that different ORB implementations can exchange object requests properly. When a link is setup within an IP network, sockets are used to send information to other nodes.

An interface is used to implement a client/server relation between objects. Fig. 1 shows how a client object can locate a server for its request using the VisiBroker implementation of CORBA. This implementation is *agent based*, providing a "directory" of currently available servers, so that a *dynamic picture* of the system is available. Using the `_bind` function of a predefined interface will query this database and setup the best connection method to send requests to the server. Agents in different IP domains can be aware of each other so that request can be forwarded to "anywhere".

Once the "object representation" of the server in the client environment is setup, all the (freely) defined *interface functions* can be called. Calling a function will block the calling thread until the server has executed the function, so that return values can be send back to the calling node. An interface function

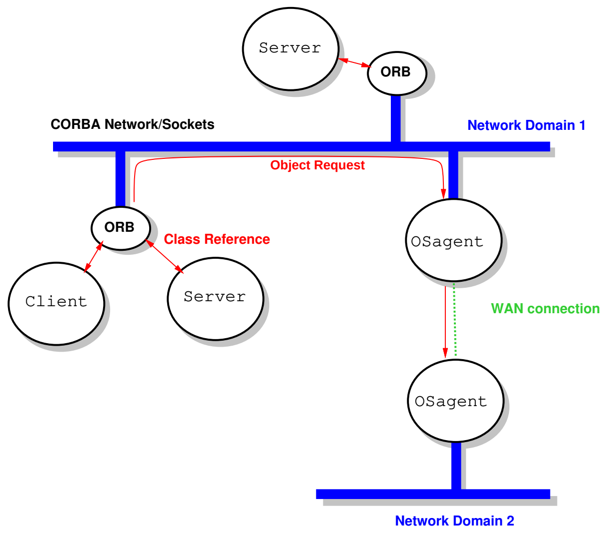


Fig. 1. Global Client/Server structure of VisiBroker

declared as "`oneway`" does not get any return values and completes immediately after the request is sent into the socket.

The interface itself is defined in a definition file and can be "compiled" to any supported language of the used CORBA implementation. VisiBroker supports *C++* and *Java*. We will use this to write simple GUI programs for C++ servers using Java applets, which in turn can run in any Java-enabled web browser.

## 3 Special Considerations for VxWorks

VxWorks matches UNIX concepts best, if a CPU is compared to a UNIX process. Tasks are then the same as "threads" inside a process. We therefore ported the "multi–threaded" version of the VisiBroker implementation. This also has the very nice feature of "optimizing away" the socket connection, if the server object "lives" in the same CPU or UNIX process. But, programs can only work, when all used libraries for implementing the CORBA library and the application code are also "multi–thread safe" or "re-entrant". Because of this, the VxWorks port was done on top of the *Wind Foundation Classes*, giving us re-entrant C++ objects. Additionally, this implementation uses the (new) C++ exception handling. This last porting issue could only be addressed by using special VxWorks OS features, to make the currently non–threaded exception handling inside the GNU compiler re-entrant. But as far as we know, this problem will be solved by the changing the exception handling in the compiler itself, soon.

We were also able to port the agent to the VxWorks platform, using the non–threaded library version. However, we do not anticipate using this process inside VxWorks, since there is always a UNIX/NT workstation with more resources available in the Slow Control Network.

## 4 First Objects In The CPU

The setup of the Slow Control Network will be controlled by a "Configuration Manager". When the VxWorks CPUs reboot, this program will determine what services each of these CPUs should perform, download the correct code into their memory, and start the tasks needed. We programmed CORBA interfaces, so that this can be done via the CORBA connection, initialized shortly after boot time. The interface itself is fixed, but the object name of each CPU is derived from hardware inside its crate.

The Configuration Manager then can map this object name to the correct setup needed, bind to it by using the same name *independent of CPU architecture or IP address* and use the provided interface functions.

Programs started this way can then define their own interfaces and object names, so that a component crate can be found without using the crate mapping list.

## References


[1] CLEO III Data Acquisition System. Technical Design Report; *CLEO CBX-95-104*.

[2] Object Management Group, CORBA 2.0 specifications 96-03-04; (*http://www.omg.org/library/schedule/Technology_Adoption.htm*)

[3] VxWorks 5.3; Tornado 1.0; Wind River Systems; (*http://www.wrs.com*)

[4] VisiBroker 2.0; Visigenic Corp; (*http://www.visigenic.com*)