Astronomical Data Analysis Software and Systems X
ASP Conference Series, Vol. 238, 2001
F. R. Harnden Jr., F. A. Primini, and H. E. Payne, eds.

CORBA as an Interoperability Tool for Astronomy

Marc Wenger

Centre de Données astronomiques de Strasbourg, France

Laurent Frisée

Haute Ecole Renequin Sualem, Liège, Belgium

Abstract. Object oriented programming is becoming increasingly important. At the same time, protocols for managing network classes are developing. Independent from any vendor, the CORBA protocol is a major player in this domain, making it a good candidate for managing interoperability between astronomical data providers and other applications. The protocol allows class providers and users to operate in different and independent environments with CORBA managing all communications in a transparent way.

This paper presents the prototype developed at CDS using CORBA for interconnecting different services in a heterogeneous context of several operating systems and different languages. Ease of development and problems encountered are assessed.

1. CORBA

Client/server architecture needs tools to manage communication between the different components. In traditional programming, sockets and Remote Procedure Calls are employed. In object oriented programming, CORBA has become a standard tool that offers remote execution of object methods for specifically designed CORBA classes. It is an interface specification designed by the Object Management Group (OMG[1]), a consortium linking both commercial companies and academic institutions from the object world. It ensures a standard independent from any vendor. Many CORBA environments are available, some commercial like Orbix or Visibroker, others free like omniORB, ORBACUS, and many others.

A CORBA class is defined by its interface, specifying in the Interface Data Language (IDL), the methods, their parameters and return values to be used on the client side and to be implemented on the server side. An IDL compiler produces the *stub* code needed by the client programme to call the object method implemented on its side, and the *skeleton* code interfacing the implemented methods on the server side.

[1] `http://www.omg.org`

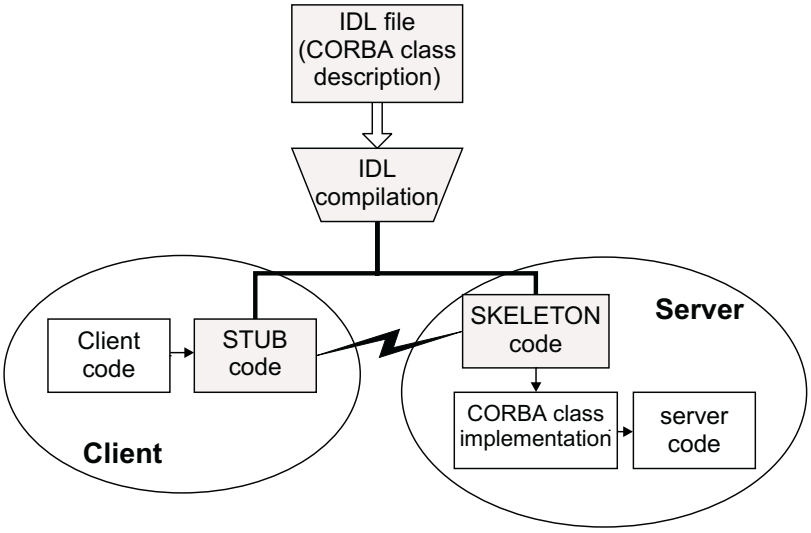

Figure 1. CORBA architecture

Once the stub is available on the client side and the CORBA object is created, the methods are called in the same way as if they were *normal* objects defined and used in an application. The server developer implements the methods, i.e., writes their code as for any class. Figure 1 shows this architecture.

One of the main characteristic of CORBA is its ability to work well in an heterogeneous world: client and server may run on different hardware under different operating systems, have their applications written in different languages and be implemented in different CORBA environments from different providers.

This makes CORBA particularly well suited for astronomy where every data provider and user has their own computing environment. CORBA allows communication in every situation and facilitates the development of interfaces for interoperability between data providers.

2. Interoperability in Astronomy

There are already many interactions between applications in astronomy: services like NED, ADS or CDS/SIMBAD have developed packages allowing client/server queries. Such packages generally require use of a specific language and have a proprietary protocol.

It is obvious that the need for interoperability will increase in the future: Virtual Observatories, Data Mining tools, Remote Observing and Data integration systems are developments that will require interaction between heterogeneous services to improve their functionality and efficiency (Graybeal 2001).

Before using CORBA for interoperability on a large scale, the CDS began by experimenting with it for internal services that have long been interoperating through classical client/server architecture.

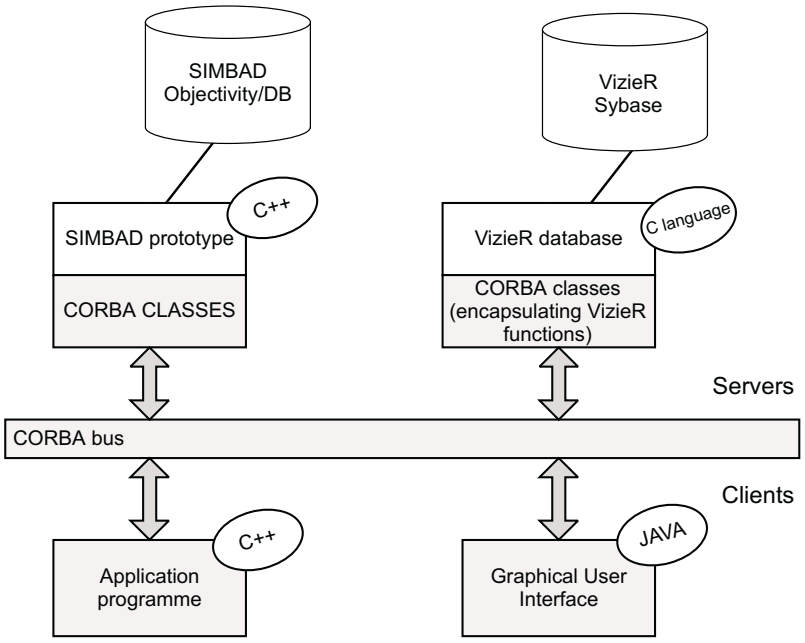

Figure 2. CORBA prototype at CDS

3. The CORBA Experiment at CDS

The experiment consisted of two server extensions (see Figure 2): (1) addition of CORBA classes to the SIMBAD prototype using Objectivity/DB (written in C++ so that the required CORBA classes were easy to develop; Wenger et al. 2000) and (2) encapsulation of the VizieR interface (written in C and using the Sybase relational DBMS; (Ochsenbein et al. 2000)) for querying catalogues by identifier.

Two clients were developed in this experiment: a simple server package written in C++ for querying SIMBAD through the CORBA class, and a graphical user interface written in JAVA to query both SIMBAD and VizieR through the CORBA classes.

This prototype incorporated several key features:

- Use of legacy code in different languages, demonstrating the ease of encapsulation via CORBA classes.
- Connection with database management systems of different DBMS technologies – Sybase as relational and Objectivity/DB as object oriented. (This revealed interaction problems between a DBMS and CORBA.)
- Development of different clients using different languages, namely C, C++ and JAVA, to assess the heterogeneous implementation of CORBA.

4. Lessons Learned

The entire system was developed and implemented within a few months. Use of different subsystems in the prototype uncovered three interesting technical prob-

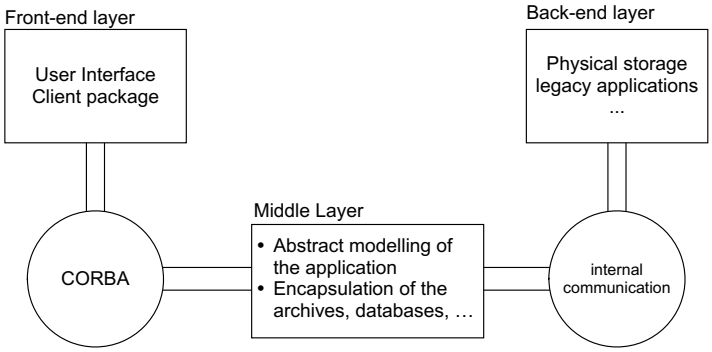

Figure 3. Three-tier architecture

lems: (1) an incompatibility between the compilers required by a CORBA tool and Objectivity/DB, (2) an interference between the Standard Template Library (STL: template classes provided only as C++ header files) and a mini-STL used by one CORBA environment (MICO) and (3) a "buggy" interaction between the threading mechanisms used by the CORBA tools and by Objectivity/DB.

Bypassing these problems required moving from one CORBA tool to another and also implementing a three-tier architecture – this also allowed better independence between the server components (see Figure 3), leaving CORBA classes between fully mastered pieces of code and helping to solve some incompatibilities with pre-existing packages.

5. Next Steps

A follow-up to this experiment could consist of defining some standard classes for querying each concerned service. Query language, parameter syntax and return values build the standard. The XML language could be an important element of this standard for exchanging data through CORBA classes. Distributing an IDL file is enough to allow users to access the CORBA services.

Generalization to more applications would be a first step towards the definition of business classes in Astronomy like those that already exist in other disciplines such as Medicine (CORBAmed) and Biology (Life Science Research).

References

Ochsenbein, F. et al. 2000, A&AS, 143, 23

Wenger, M. et al. 2000, in ASP Conf. Ser., Vol. 216, Astronomical Data Analysis Software and Systems IX, ed. N. Manset, C. Veillet, & D. Crabtree (San Francisco: ASP), 247

Graybeal, J. 2001, this volume, 189