Ubiquitous Distributed Objects with CORBA

Frédéric Achard, Emmanuel Barillot
Gis Infobiogen
7 rue Guy Môquet, BP 8
94801 Villejuif Cedex, FRANCE

Database interoperation is becoming a bottleneck for the research community in biology. In this paper, we first discuss the question of interoperability and give a brief overview of CORBA. Then, an example is explained in some detail: a simple but realistic data bank of STSs is implemented. The Object Request Broker is the media for communication between an object server (the data bank) and a client (possibly a genome center). Since CORBA enables easy development of networked applications, we meant this paper to provide an incentive for the bioinformatics community to develop distributed objects.

1 Introduction

1.1 Why interconnect?

The process of discovery in science greatly relies on the ability of researchers to link previously unrelated data or knowledge. This is true at all levels of research, from the daily task of information gathering to the merging of two theories into one new dogma based on the accumulation of knowledge involving links between different fields [a]. In biology, the problem of linking data is becoming more and more crucial: since the discovery of the genetic code, genetics unifies the biological sciences by offering a common basis for various disciplines such as biochemistry, embryology, taxonomy, cytology and evolution. The 'physics of particles' plays a similar role with regard to 'hard sciences'. However, biology has the advantage that the scale-up between the basic science and the others is much more manageable than in physics. Thus, the community would greatly benefit from an interconnection of the data generated by biologists. This has been particularly true since the explosion of molecular biology: there is a dramatic need to relate mapping data to sequence data and functional data, as well as to relate the genetic data of different species (e.g. model organisms or mammalian comparative mapping).

This need advocates for the interconnection of biological databases. Nevertheless, the biological data is dispersed in a large number of different databases. These databases are often not interconnected in a satisfactory way either for

[a] A good example is the discovery of the double helix [1].

programmers or for users; at the most, they offer reliable hypertext links toward some other databases but very rarely permit one to program easily distributed data gathering. Moreover, biological databases are managed by heterogeneous systems, which may not necessarily follow international standards. Examples of such systems include Relational DBMS, Object-Oriented DBMS, flat files and home-made systems.

The interconnection can be described at several levels:

- *physical interconnection:* the databases should be accessible through the Internet. Unfortunately, there are some highly specialized databases, with interesting and sometimes unique expertise, that can not be accessed via the Internet. This problem is out of the scope of this paper.

- *syntaxic interconnection:* databases are managed by a variety of heterogeneous system that are not interconnected. They use different data models and data manipulation languages. A common language is required.

- *semantic interconnection:* for both data and meta-data levels. At the data level, it can be difficult to reveal a semantic link between related data wich come from distinct databases (or even from one single database). This problem is discussed elsewhere [2]. At the meta-data level, terms are understood by scientists with regard to their experience and culture. For example, molecular biologists, geneticists and epidemiologists have not the same understanding of the term *gene*. This leads to a semantic barrier and prevents the scientists from going outside their specialty.

- *operational interconnection:* having an operational way to manipulate data to and from diverse databases is the purpose of the interconnection. In addition to the types of interconnection described above, adequate user interfaces are required. Such developments would clearly benefit from the use of an architecture-independent language, such as Java™.

1.2 Which interconnection?

Interconnection is a generic term that can be applied to simple references as well as any arbitrary complex system linking distinct databases. Therefore its meaning should be further explained. Basically, the goal is to *interoperate* data from various sources, typically to provide a single end user interface. Therefore, what is needed is more than simple accession number references or hypertext links, which can be considered as the minimum level of interconnection. A productive *interoperation* between databases is required. For example,

biologists expect to be able to query all the databases of interest in one row and to process the data using a tool box of methods such as sequence alignment and EST mapping. In that respect, an interconnection system must:

- provide a unified view of heterogeneous databases,
- ensure the portability of the existing databases and the adaptation of softwares,
- facilitate the evolution of the components of the system,
- be applicable both for large databases as well as for small specialized databases and
- be accessible from heterogeneous architectures.

It follows that one must:

- use an architecture, DBMS and language independent metalanguage to describe the structure of each data to be shared,
- provide a structured description of the data for computation (hypertext is not powerful enough) and
- have an excellent scalability. Data are accumulated at an exponential rate, and the number of areas of knowledge to be connected are increasing too.

The success of projects involving a large number of heterogeneous components is generally achieved through standardization. SQL is a good example of such a success. This standardization can arise de facto as when a product is progressively proved better and adopted by the community, or after a formal consensus as when needs are well specified and the product designed after a general agreement.

Besides, a realistic solution must leave a large degree of autonomy to each database. It is unrealistic to impose a common DBMS or a common meta-schema. A relational structure can be perfectly suited to describe a laboratory notebook for some precisely standardized biological manipulation, while the complexity of a genome mapping schema clearly requires an object-oriented approach. This means that an upper layer should be used on top of the existing databases to provide a syntaxic interconnection. The problem of semantic interconnection has to be solved separately.

Some pioneering attempts to address the problem of interconnection have been proposed, such as the Object Protocol Model [3] and the Integrated Genomic Database [4]. Those approaches have been enriching the debate, but they are not yet universal standards. One can classify those approaches in two categories:

- data warehouse: data is centralized in an integrated database.
- data interoperation: the queries are distributed over the different databases, and results are grouped back.

Data warehouse is seldom practical because it is costly to curate, maintain and keep large databases evolving. Data interoperation increases the network traffic, but smooths it among the interconnected sites. Each data manager keeps control on his data and is responsible for its quality.

We will now briefly describe a standard that could be used for interconnection, the Common Object Request Broker Architecture, and a first application we have developed on top of it.

2 An overview of CORBA

CORBA [5,6], Common Object Request Broker Architecture, is the result of the work done by major actors of the hardware and software industry to set up a communication framework. The goal is to facilitate the development of distributed applications over a heterogeneous network. In a distributed environment, clients are talking to objects. The services that can be delivered to a client by an object are described in an interface. The means of communication are handled by an Object Request Broker (ORB). The idea behind CORBA is to provide an intermediary layer that handles access requests on data. It frees the developer from most of the portability issues. It enables the development of "distributed objects": objects that are somewhere on a network, some place that will not always remain the same.

CORBA defines a two-fold set of specifications. To be CORBA compliant, a vendor must implement at least the CORBA core: the CORBA Object Model, the CORBA architecture, the Interface Definition Language (IDL) syntax and semantics, an ORB (which supports the Dynamic Invocation Interface, Dynamic Skeleton Interface, the Interface Repository) and one language mapping. In addition, the vendor is free to implement a number of services. Those services, if implemented, are granted a separate compliance point. The main CORBA services are Lifecycle, Events, Naming, Persistent Object Service and Relationships. The points of interest for our work will be further discussed in sections 2 to 4.

CORBA is already a *de facto* industry standard. We strongly believe that it will become a standard in the bioinformatics community. The long term goal is to allow a true and productive interoperation between distributed programs and databases.

CORBA developments can be featured as follows:

- Object Oriented technologies gives the best model to capture the richness of data in biology.
- The data and methods to operate an object are described by means of the Interface Definition Language (IDL), a strongly typed language.
- The IDL compiler generates a stub for the client side and a skeleton for the object implementation side. The ORB must store the IDL definition of each object in the Interface Repository. It means that a client can possibly access a CORBA object with no prior knowledge of its implementation scheme. It is the foundation for interoperability.
- CORBA provides a set of services that prevents one from reinventing the wheel. The Object Management Group, the large industry consortium which accredits CORBA products, guaranties the benefits from an Industry Standards: scalability, robustness, support, long-term future. It is commonly admitted that Bioinformatics developments suffered from a lack of consistent standards.

3 An illustrated view of CORBA

We have sketched out a working example to illustrate how the objects are defined and how to access them. This code was written and compiled on a Sun workstation using the ORBeline^{TM} environment. The scenario models a distributed environment and demonstrates CORBA interoperability. Although we meant it to be quite close to real life issues, we deliberately kept the example simple. Data descriptions are minimal. For this example, we write a simplified "STS bank object".

We chose this example because "Sequence Tag Sites" are essential objects in genome study. They are the skeleton of physical maps, the basis of genetic maps and a keystone in positional cloning. Typically there are pieces of information that need to be shared via a computer media since the biological object can be generated from an electronic description. Moreover, STSs are of interest for any group working in a genomic region.

The object's goal is a management system for STSs. It accepts deposit, deletion and query by name. We first write the IDL interface and then write a

client implementation that accesses this object. Purposely, the object implementation details are not given. We only describe how to register the object. Although we developed the facilities [b] to manage the STSs, the technical implementation is out of the scope of this paper. Our point is to demonstrate a client implementation to an object for which the IDL interface is known. The C++ language is used for this example.

3.1 Interface Definition

First we define the structure of an STS as it will be stored within the object:

```
// Specification of an STS
struct STS {
    string Name;
    string Species;
    string CA_primer;
    string GT_primer
    long Length;
    float Heterozygosity; // in percentage
};
```

Then we define the services that we are going to offer, related to STS objects. The `interface bank` describes a new CORBA object type. The object can be manipulated by means of the three methods defined within it: `add_sts`, `del_sts`, `get_sts`.

```
// hold the STSs interface bank {

  boolean add_sts ( in STS sts_info);

  boolean del_sts ( in string name);

  boolean get_sts ( in string name ,
                    out STS sts_info);
};
```

Then the Orbeline^{TM} IDL-to-C++ interpreter generates a stub, used to implement the client side, and a skeleton, used to implement the object side. The key points of the client implementation are further explained in the next section.

[b] the object source program is available on request

3.2 Client implementation

First the program makes a connection to the ORB, then asks whether the ORB knows an object called `bank` (which refers to the `interface bank` of the IDL). In case of success, the ORB provides the client with a proxy object. Here the binding is straightforward because the ORB knows where objects are located within a local network. However client may specify the use of an instance of the object that resides on a specific host.

```
// connection to the ORB
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

// bind to the object
bank_var sts_object;
sts_object = bank::_bind();
```

Once the binding is done, methods can be applied to the object.

- Adding an STS to the bank is done by calling the method `add_sts` on the `sts_object`. The method takes an STS description as argument and returns `true` on success.

```
CORBA::Boolean success;
STS a_sts;

a_sts.Name = "AFM115xf2";
a_sts.Specie = "human";
a_sts.CA_primer = "ttcaaaggtggagacccttc";
a_sts.GT_primer = "ttttggtcagaatggagtgg";
a_sts.Length = 107;
a_sts.Heterozygosity = 77;

success = sts_object->add_sts( a_sts);
if( !success) cout << "'add_sts' failed" << endl;
```

- Deleting an STS from the bank is done by calling the method `del_sts` on the `sts_object`. It takes a name as argument (a string in IDL terminology, which maps to a C++ `const char *`) and returns `true` on success.

```
CORBA::Boolean success;

success = sts_object->del_sts( "AFM115xf2");
if( !success) cout << "'del_sts' failed" << endl;
```

- Fetching an STS from the bank is done by calling the method `get_sts` on the `sts_object`. It takes a name as **in** argument (a string in IDL terminology, which maps to a C++ `const char *`), an STS as **out** argument (a pointer to STS, passes as a reference) and returns `true` on success.

```
CORBA::Boolean success; STS *& to_sts = new STS;

success = sts_object->get_sts( "AFM115xf2", to_sts);
if( !success) cout << "'get_sts'" failed << endl;
```

CORBA provides facilities for error checking via the C++ exception mechanism. Each time a method is applied to a CORBA object, we encapsulate it with a try which allows to catch errors such as network failure or wrong argument calls. For example, the `add_sts` method is called as follows:

```
try {
    status = sts_object->add_sts( a_sts);
}
catch( const CORBA::Exception& excep) {
    cout << "Error add_sts " << endl << excep ;
    exit( 2);
}
```

3.3 Object implementation and running the example

The object side implements the facilities described within the IDL interface. The object is registered to the ORB with the following code (note that methods of implementation are not reproduced):

```
int main(int argc, char **argv) {
    // Initialize ORB and the Basic Object Adaptator
    CORBA::ORB_ptr orb = CORBA::ORB_init(argc, argv);
    CORBA::BOA_ptr boa = orb->BOA_init(argc, argv);

    Bank bank_server;
    boa->obj_is_ready( &bank_server);

    // get ready to receive requests
    boa->impl_is_ready();

    return(1);
}
```

We then compile and link our C++ code with the IDL compiler-generated stubs for the client side and the skeleton for the object side. The ORB is started with `osagent`™ (Orbeline smart agent). The object is registered to the ORB by running its code. Then clients can access it via the ORB for each type of method, as described earlier.

By this example we attempt to demonstrate that developing a CORBA client is surprisingly easy and effective. Note that issues such as identification and security should be addressed for a real life program. In addition, a very powerful aspect of CORBA is the use of the Dynamic Invocation Interface which allows one to discover the methods associated to an object at run time. This feature is not demonstrated in this paper.

4 A prospective view on CORBA

The ORB by itself only provides basic mechanisms for brokering object requests. All other services are provided by objects with IDL interfaces that reside on the top of the ORB (see 2). Unfortunately, we were restricted to discussing these services in the "prospective" section because only few of them are currently implemented. Some of these services represent the cutting edge of distributed objects and client/server technologies. However, we believe that most of the basic services will soon be available from ORB vendors. Software developers in the biological community will benefit from some of these services in ways that are briefly discussed in the following sections.

4.1 Persistence Object Service

We expect, from the client side, that an instance of the object is always running and always preserves its state. This requirement represents an obligation on the object side and some coding to do so is necessary. The "Persistent Object Service" (POS) provides the means to standardize this process. In other words, vendors are expected to provide means for easy storage, either by developing their own scheme or supporting bridges to other database systems such as relational ones.

4.2 Object Query Service

This service will allow one to find objects whose attributes meet the search criteria one specifies using a query. Three query languages will be supported: OQL (ODMG-93's Object Query Language), SQL with object extensions or a subset of the two previous ones.

4.3 Collections and Relationships

This service will let you create relationship between objects. It is a dynamic process; one can relate objects without modifying the object. Object itself is not aware that it is part of a relationship. This service, associated with the Object Query Service, will allow queries that are arbitrarily complex.

4.4 Authentification and Authorization

In our example (see 3), the addition and the deletion of an STS are accessible to any client. A real life object server should enforce security of its data. Therefore, when a client connects to an object, the object should be able to check

- if the client is really what it claims to be
- what resources the client can access

Facilities to do so are provided with the Authentification and Authorization Services.

5 Conclusion

A loose system of interoperation that lends autonomy to data managers is mandatory to be widely accepted. In particular, it is not necessary to give an access to the database as a whole, because some data may be redundant or

irrelevant. Another important point is to use a recognized standard; it is clear that too many standards exist in biology. CORBA can play this role, though it is not the answer to every question about interconnection. In particular, it does not address the issue of semantic interconnection. Nevertheless, we think that it will provide the basis for sharing objects in an unified way and that semantic questions will be solved on top of CORBA.

The implementations of CORBA are not very complete at the present time. However as CORBA-compliant ORBs will provide more and more of the services, programming will become easier, both on the object and the client side, and at the same time, distributed objects will become much more dependable. Still, it is difficult to forecast the availability of all the CORBA services. And the network performance is an issue that needs to be addressed.

On the other hand, a very appealing aspect of CORBA lies in the vertical facilities. This term denotes the services that are industry specific. Currently, OMG has established working groups in various domains such as health care or financial services. We believe that the creation of such a group will benefit to the biology community. We hope that such a group will be initiated soon because we believe that the proliferation of biological objects, in the CORBA sense, over the network will be the foundation for a productive interoperation of biological data.

Our middle term project is to offer a CORBA interface to HuGeMap, our local database for human genome data. As a test case, the objects defined will be accessed by the EBI to complement the data of the RHdb database. In that respect, an Object Database Adapter will be developed for the OODBMS we use [c]. The interoperation of this mapping data should facilitate the development of tools to help the cloning of new genes.

Acknowledgments

This paper greatly benefited from Tom Flores and Patricia Rodriguez-Tomé of the EBI for first suggesting the use of CORBA to address the interoperability problems and for profitable discussions [7].

We are thankful to Guy Vaysseix for valuable discussion, Claude Scarpelli and Philippe Gesnouin for providing excellent informatics support.

We also wish to thank Jennifer Fitzpatrick for smart critical reading of the manuscript.

We would like to acknowledge contribution of Visigenic and PostModern Computing for providing their Orbeline™ ORB.

[c] IDB_system by Eric.Viara@infobiogen.fr

This work was supported by the European Community under grant BIO4-CT95-0037 for the BIOTECH Research and Technological Development Program.

References

1. J.D. Watson. *The double helix.* Laffont, 1968.
2. Frédéric Achard. Proposal scheme for storing links among genomic databases. In *Hugo's Human Genome Meeting*, Heidelberg, March 1996.
3. I-Min A. Chen and Victor M. Markowitz. An overview of the object-protocol model (OPM) and OPM data management tools. *Information Systems*, 20(5), 1995.
4. Otto Ritter, P. Kocab, M. Senger, D. Wolf, and S. Suhai. Prototype implementation of the integrated genomic database. *Computers and Biomedical Research*, 27:97–115, 1994.
5. Robert Orfali, Dan Harkey, and Jery Edwards. *The Essential Distributed Objects Survival Guide.* Wiley Computer Publishing Group, 1996.
6. Jon Siegel. *CORBA, Fundamentals and Programming.* Wiley Computer Publishing Group, 1996.
7. European Bioinformatics Institute. *Linking Biological Databases workshop*, Hinxton Hall, November 1995.