

RECEIVED

JAN 26 1996

OSTI

DATA ANALYSIS IN AN OBJECT REQUEST BROKER ENVIRONMENT\*

DAVID M. MALON, EDWARD N. MAY

*Argonne National Laboratory, 9700 South Cass Avenue,  
Argonne, IL 60439, USA*

ROBERT L. GROSSMAN

*University of Illinois at Chicago  
Chicago, IL, USA*

CHRISTOPHER T. DAY, DAVID R. QUARRIE

*Lawrence Berkeley National Laboratory  
Berkeley, CA, USA*

Computing for the Next Millenium will require software interoperability in heterogeneous, increasingly object-oriented environments. The Common Object Request Broker Architecture<sup>1</sup> (CORBA) is a software industry effort, under the aegis of the Object Management Group (OMG), to standardize mechanisms for software interaction among disparate applications written in a variety of languages and running on a variety of distributed platforms. In this paper, we describe some of the design and performance implications for software that must function in such a brokered environment in a standards-compliant way. We illustrate these implications with a physics data analysis example as a case study.

1 Introduction

1.1 Background

The promise of brokered-request object architectures is alluring—my software will talk to your software, even if I know neither in what language your software is written, nor where it runs. The idea is this: no matter what language you use to implement your software, you describe its *interface* in a single, application-language-neutral Interface Definition Language (IDL), and place an interface description in a repository. You then register your implementation so that it can be found by system utilities.

When I wish to invoke your software, I use standard utilities to find its interface, and pass my request to an Object Request Broker (ORB). The ORB looks for a server capable of handling my request—its location may be transparent to me. The ORB may instantiate such a server if none is already running. The ORB then forwards my request to your software and returns any results, handling the language mapping at both ends.

Services commonly required by many objects—lifecycle services, persistence services, query services, and others—are the subjects of standardization specifications as well.

Work supported by the U.S. Department of Energy, Division of High Energy Physics, Contract No. W-31-109-ENG-38.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

MASTER

Is this environment appropriate for high-performance physics applications? If the physics community ignores these approaches, does it do so at its own peril? Among the questions that must be addressed are these:

- Is the Interface Definition Language rich enough to capture the interfaces required by data-intensive physics applications?
- Is the performance penalty of brokered interactions inherently too great?
- Can we use an ORB simply to *connect* our applications, and then get it out of the way?
- If the ORB does get out of the way, do we lose language-independence, and are we back to home-grown low-level interfaces?
- What is the appropriate level of granularity for brokered interactions?
- The potential location transparency provided by an ORB is appealing, but will performance considerations require that I provide a “smart proxy” to run on your machine when you invoke software on my machine, in order to sustain brokered interactions at a reasonable cost?
- If so, is proxy support a nightmare for providers of general-use software, or can proxy generation be standardized or automated?
- What are the implications of proposed persistence services specifications in this environment?

We have explored these and other issues in a case study, in which we used commercially available request brokers in an examination of a variety of potential implementations of a statistical computation on physics data extracted from a persistent data store. This paper describes a few of our observations based upon this experience.

## 2 Interface Definition Language (IDL)

An ORB architecture such as CORBA can be effective only if its interface definition language is rich enough to support an application’s object interactions. OMG’s IDL was minimally sufficient for our case study; it does, however, lack several desirable features, including templates and method overloading. A catchall type *any* serves as a mechanism to overcome some of these deficiencies.

Type *any* was designed to support passing arbitrary types through interfaces, at the expense of type safety. An *any* manages to be self-describing by inclusion of a TypeCode. While the TypeCode interface allows an arbitrary *any* to be deciphered, construction of an arbitrary TypeCode is potentially highly implementation-dependent; moreover, initial versions of the CORBA specification do not require type *any* to support all constructed data types. Even some of OMG’s own Object Services specifications (e.g., Object Query Services<sup>4</sup>) will fail under minimally CORBA-compliant ORBs.

The lack of template support makes implementation of common interfaces needlessly difficult. A collection of physics Events, for example, provides essentially the same interface as a collection of Muons—only the type of the contained objects is different. There is no way to express this commonality in IDL, and one must either generate an enormous amount of redundant code (collections and iterators for every type of contained object), or build collection interfaces based upon the catchall type *any*. The latter strategy poses difficulties, both because it is not type-safe, and because every client must shoulder the artificial burden of packing or unpacking an *any* with every collection interaction.

### 3 Portable CORBA Compliance

A question that is often obscured in early application efforts is the extent to which one can write pure CORBA-compliant code (as opposed to IONA/Orbix<sup>6</sup>- or IBM/DSOM<sup>5</sup>- or other vendor-complaint code). Thanks to IDL, interfaces are easily portable across ORBs. (Note, though, that with products such as DSOM, it is often nearly essential to include DSOM-specific directives in the interface definition file, albeit enclosed in an *ifdef* block.) Client code, too, is largely portable, except for lifecycle, location, and naming services, but there is hope that these can be largely standardized as well.

What may be substantially more difficult is portability of implementations. Declaration of and access to instance variables is problematic, especially with inheritance. “IDL interface inheritance does not require that the corresponding C++ classes are related, though that is certainly one possible implementation.”<sup>3</sup> The problem is especially vexing today, when C++ bindings are still essentially the extended C bindings that vendors provided while waiting for approval of the C++ specification. Even “wrapper” interfaces, which merely forward method invocations to an underlying language-specific object instance, while trivial in many cases, can be quite difficult to support (even non-portably!) for cooperating objects.

Another dilemma facing implementors is that particular vendors’ ORBs offer features that are extremely valuable, but non-standard. Some examples in DSOM include cooperative metaclasses, the ability to describe the inheritance behavior of proxies, and a number of additional classes and frameworks. Smart proxies, for example, may be essential to high-performance applications, but it is unclear how one might provide them without sacrificing portability among ORB products.

### 4 Proxies Everywhere

When a client invokes a method on a CORBA object, he is, in practice, talking to an object *proxy*, which dispatches the request via an ORB to the actual object, which may in turn reside on a different machine. This location transparency is an appealing part of the ORB architecture, and well suited to objects that interact at a coarse level of granularity. Consider, though, what happens when a user wishes to reach a `Set_of_Events` object referred to by a `Run` object, in one standards-compliant implementation.<sup>2</sup>

The user talks to a Run proxy, which talks to a Run. The Run talks to its data via an interface, and hence, via a RunData proxy, which talks to a RunData object. The RunData object must instantiate and connect a Set\_of\_Events and a Set\_of\_EventsData object, whose data location of described by a PID (persistent ID) object. (This work may be accomplished by means of a Factory object; details, which may be quite complex, are omitted here.) In any case, the RunData object acquires and returns up the chain a proxy to the Set\_of\_Events object.

At the end of this process, how many objects have proxies to the Set\_of\_Events object? (How many ought to?) Proxy management is not automatic, and problems akin to memory leaks in C++ may be distributed across a number of processors. From a user's point of view, though, the primary problem may be the performance implications of so many proxied interactions. Note that it is entirely plausible that the user code, the Run, the RunData, the PID, and the Set\_of\_EventsData are all on different processors, and it is not far-fetched to think that other objects in this example may be on still other processors.

One can certainly improve performance by the use of smart proxies to reduce the number or proxied interactions; for example, a Run proxy could maintain a local copy of its data. This is a perilous practice today—ORB vendors generally support smart proxies, but they are highly vendor-dependent; moreover, it is not clear that one would want to write and maintain separate proxy objects for each type of object in a physics database.

## 5 Software Maintenance and the Code Explosion

A conspicuous aspect of ORB-based programming is the code explosion. In DSOM, for example, the IDL for a Muon interface with 5 data attributes and no other methods generates 842 lines of code, with client bindings for only one language and *no* implementation-specific code. More significantly, supporting this Muon interface requires approximately 30 files, even with only one client language binding. This count includes files corresponding to the Muon object, a MuonData object, a MuonCollection object, a MuonCollectionData object, and a MuonIterator object; it does not include link libraries and interface repositories. Admittedly, many of these files may not require human intervention (though such intervention turns out to be necessary more often than one might expect), but even management of the files themselves is a burden.

There is a serious need for more software tools if applications development for ORB architectures is to become widely viable. Even when our conceptual model for implementing other particle interfaces is identical, for example, to that for Muons, there are no tools to automate equivalent, tedious, error-prone work.

## 6 Conclusions

Distributed object architectures are here to stay, and the high energy physics community cannot afford to ignore them. Some evolution in current ORB environments is required to address the deficiencies cited in this paper; nonetheless, it is possible today to use such architectures successfully for applications with sufficiently large

interaction granularities. Current developers who wish to write CORBA-compliant implementations that are portable across ORBs are in a bit of a predicament, although their situation will improve in several respects when implementations of the OMG's Common Object Services Specifications become more generally available. Fine-grained applications may wish to wait for additional refinements in ORB environments. For all potential developers, the availability of appropriate software tools may be the key to successful implementation and support of large-scale ORB-based distributed object applications.

### Acknowledgments

The submitted manuscript has been authored by a contractor of the U.S. Government under contract No. W-31-109-Eng-38. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

### References

1. Object Management Group, *The Common Object Request Broker: Architecture and Specification, Revision 1.2* (OMG, Draft 29 December 1993).
2. Jon Siegel et al, *Persistent Object Service Specification*, OMG Document Numbers 94-1-1 and 94-10-7 (Object Management Group, 1994).
3. Digital Equipment Corporation et al, *IDL C++ Language Mapping Specification*, OMG Document 94-9-14 (Object Management Group, 1994).
4. IBM et al, *Joint Submission: Object Query Service Specification*, OMG TC Document 95-1-1 (Object Management Group, 1995).
5. *SOMobjects Developer Toolkit Users Guide, Version 2.0* (IBM, 1993).
6. *Orbix Programmer's Guide 1.3* (IONA Technologies Ltd, 1995).