37

Designing Scaleable Applications Using CORBA

R.B. Whitner
Hewlett-Packard Company
Network & System Management Division
3404 E. Harmony Rd.
Fort Collins, CO 80525 USA
Phone: 970-229-3821 FAX: 970-229-2038
E-mail: rick_whitner@hp.com

Abstract

The Object Management Group's Common Object Request Broker Architecture (CORBA) is quickly becoming an industry-leading approach to building distributed applications. CORBA was investigated for its use to build applications that must deal with very large numbers (up to millions) of managed objects. This paper presents a set of challenges to scaleability encountered during that investigation and a set of techniques that can be used to address those challenges.

Keywords

CORBA, scaleability, object identity, object granularity, partitioning, Entity-Warehouse Model

1 INTRODUCTION

CORBA is an architecture—defined by an industry consortium called the Object Management Group (OMG)—for building distributed applications. CORBA supports a computation model based on communication among objects. An object is a computational entity that supports a well-defined set of services, called its interface. Implementations of object interfaces are deployed for use by object users. Object users (clients) issue requests to objects by means of object references which encapsulate object location information. An object request broker (ORB) manages the dispatch of requests between object users and objects.

A. A. Lazar et al. (eds.), *Integrated Network Management V*

CORBA was investigated for use in a distribution infrastructure on which applications that must manage very large numbers of objects will be built. The objects in these applications typically represent real resources in the environment, such as workstations, printers, network elements, etc. These objects are referred to here as entities. The applications manage the entities in a variety of ways. For example, one application maintains a topology of relationships that exist among entities. Another renders graphical representations (maps) of various topologies. Others provide report generation, trend analysis, fault management, etc. Various protocols—of which CORBA is just one—can be used to operate on these entities.

The distribution infrastructure had to support environments scaling up to millions of entities. Additional constraints were also imposed. Constraints worth mentioning were that the infrastructure could not impose the use of proprietary extensions to CORBA, nor make the applications dependent on any particular vendor's ORB. Additionally, it was anticipated that there would be significant costs associated with transitioning legacy environments to CORBA. A model that would enable developers to easily integrate legacy applications and choose how much of CORBA to use was desired.

The investigation examined various ways of applying CORBA such that the requirements could be reasonably satisfied. The results of the investigation are presented here. Section 2 presents a list of issues that application developers will face when using CORBA for large-scale application development. Section 3 presents a model for using CORBA to build applications that will scale to support very large numbers of objects. Section 4 presents some conclusions.

2 CORBA SCALEABILITY ISSUES

This section describes key issues that must be addressed when building CORBA applications that have very high scaleability requirements.

2.1 Appropriate Level of CORBA Object Granularity

In an object-oriented environment, an object represents an abstraction of some 'real' thing that the developer defines so that a computational representation of the thing can be created. For example, a set of services that permit the management of a workstation might be organized together into a single object interface called Workstation. This interface (abstraction) can be treated programmatically as though it really were a workstation, even though physically it is not.

Object granularity is a term used to describe the relative levels of abstraction between different definitions of the same thing. A specification Y is said to be more granular (finer grained) than X when Y defines as objects the non-object characteristics of X. In CORBA terms, Y defines interfaces for parts of X that are

not interfaces, as illustrated in Figure 1. In Specification X in this illustration, B is considered an object by CORBA because it is defined using the keyword *interface*; A is only a data structure. In Specification Y, both A and B are CORBA objects.

```
struct A {
        attribute string a;
        attribute string b;
};
interface B {
        attribute A a;
        attribute string b;
};
```

(a) Specification X

```
interface A {
    attribute string a;
    attribute string b;
};
interface B {
    attribute A a;
    attribute string b;
};
```

(b) Specification Y

Figure 1 Relative levels of object granularity in a specification.

Being a CORBA object implies a certain status within a CORBA environment (first class citizenship, so to speak). Being a 'first class CORBA object' means tighter integration with other CORBA services. The CORBA object (designated using the IDL keyword interface) is the basic unit that CORBA recognizes—it is the only unit for which you can obtain object references. Common object services, such as the OMG Naming Service and Transaction Service (Object Management Group, Inc., 1996) typically are defined in terms of first class CORBA objects. Other CORBA features such as interface inheritance (with its corresponding type checking) only apply to CORBA objects.

Despite these benefits, CORBA object granularity must be considered carefully in very large environments because it directly impacts the number of CORBA objects in the system. Finer-grained approaches mean more CORBA objects, which could mean a significant increase in system resource requirements due to overhead associated with CORBA objects.

Care must also be taken to ensure the appropriate level of abstraction is being applied. For example, some of the applications referred to in this investigation use CORBA to provide a higher level of abstraction of SNMP objects. It would not be appropriate to simply create a CORBA object for each SNMP object.

2.2 Proprietary versus Interoperable Object References (IORs)

An object reference is a CORBA-defined data structure that clients use to make requests for object services. CORBA permits two types of object references. Interoperable Object References (IORs) (Object Management Group, Inc., 1995) enable one to reference objects across interoperating ORB domains (across different ORBs). IORs are required for inter-ORB (cross-domain)

communications. ORB vendors may choose to also use the IOR for intra-ORB (single domain) communications.

Alternatively, vendors may choose to use a proprietary object reference structure for communication that takes place within their ORB's domain. Proprietary references give vendors much more opportunity to optimize the implementation of the reference. Common optimizations are to reduce resource consumption and to increase performance. For example, some ORBs support object references as small as four bytes on average using proprietary protocols. Contrast this with 100+ bytes average (or more) for an IOR in the most efficient ORB implementations.

Some ORBs exclusively use the IOR. Others use the proprietary reference by default, and require the developer to request an IOR when interoperability is needed. Some applications will virtually always need IORs. Object directory services—such as the Naming Service—are a prime example. These services know nothing about the objects registered with them; they simply hand out references on request. A client in one ORB domain who queries the directory service and receives a reference to an object in another ORB domain must receive an IOR in order to do anything with the reference. In large, enterprise environments, it is reasonable to expect that ORBs from many vendors will exist and that interoperability across those ORBs is required. One must weigh the additional overhead required for an IOR against the relative frequency that interoperability is expected.

2.3 Partitioning the 'Object Space'

In order to access an object, one must have a reference to it. While there are several techniques for obtaining references, the general purpose technique is to query some sort of directory service. The OMG Naming Service is probably the most commonly used directory service today for CORBA application development. This service takes a name as input and returns an object reference. Examples of other directory services are a trading service and a factory finder. (Object Management Group, Inc., 1996) These services each return references in response to some type of query.

The common CORBA technique that was observed for making objects generally available utilizes a single-tiered lookup strategy: objects are registered with the naming service, and clients query the naming service to obtain IORs. This approach does not scale well. As the number of objects increases, the burden on the directory services becomes too great. Ways of addressing this problem typically involve some kind of partitioning of the name space or a multi-tiered lookup strategy; however, there are no generally available guidelines for how to do this effectively.

Another consideration is the fact that a generic directory service has limited potential to optimize searches. The significance of this limitation increases as the number of objects grows large.

2.4 Object Identity

Comparisons often need to be made to determine whether two objects are in fact the same object. For example, when traversing a graph of objects, one needs to determine whether a given node (object) has already been visited. Objects need to support some notion of identity in order to permit such comparisons to be made.

There is a common misconception that CORBA object references represent object identity. Statements that lead one to conclude that object references and object identity are synonymous can be found throughout the CORBA specifications themselves. However, one important thing that an object reference does not provide is the identity of the object. Multiple references can point to the same object. CORBA supports operations on the object reference that enable one to determine if two references are equivalent, but if the references are not equivalent, the objects could still be the same.

The best way to determine the identity of an object is to ask the object itself—assuming it provides an interface for doing so. However, CORBA makes no requirement that an object support such an interface. The OMG Relationship Service (Object Management Group, Inc., 1996) defines the *IdentifiableObject* interface, but there are still two problems related to object identity when operating in very large environments.

An *IdentifiableObject* has a read-only attribute of type *ObjectIdentifier* named *constant_random_id*. The value associated with *constant_random_id* must not change during the lifetime of the object. It is not guaranteed to be unique. Its typical use is as a key in a hash table. If the two identifiers are different, one can conclude that the two objects are different. When collisions between these identifiers occur, the client can use the *is_identical* operation to ask the object whether the other object that it is being compared against is the same.

The first problem is related to the potential amount of network traffic that might be generated when having to perform a large number of comparisons. Each call to obtain the value of *constant_random_id* or to perform the *is_identical* operation could result in a remote call through the ORB. This leads one to look at caching strategies to minimize the amount of ORB traffic. This, in turn, leads to further questions about whether the identifier should be obtained in advance or obtained on demand, and whether more can be done to strengthen the notion of identity through the use of universally unique values, possibly eliminating dependence on the *is_identical* operation altogether.

The second problem is more political than technical. It has to do with the importance of object identity to the OMG. By and large, the notion of identity is very weak in CORBA. There is very little use of the *IdentifiableObject* interface in other services. This runs contrary to the strong requirement for object identity that was found in this investigation. One of the most heavily used services among the applications in this study is the topology service. That service has a very strong dependency on object identity.

2.5 Bulk Operations

It is often the case that one will want to perform the same operation on many objects simultaneously. Common examples are querying many objects through a single call, or dispatching an update request to many objects at the same time. The primary motivation behind bulk operations is performance and resource consumption optimizations by taking advantage of co-location of objects. The value of using bulk operations increases as the number of objects being managed increases.

Consider, for example, an application that renders a graphical map of large numbers of objects. To display a map, a certain bit of information must be obtained from each object. The straightforward approach is to iterate through the list of objects requesting each object to supply the required information. The problem is that this approach can result in a tremendous amount of network traffic. To obtain just one attribute value from 100 objects will require 100 object requests. If there is a slow distribution link between the client and the objects, performance can be unacceptable. If, however, the 100 objects were partitioned among two 'collection' objects, where the link between each collection object and the objects that it manages is fast and the link between the client and the two collection objects is slow, performance is going to be much better if two calls can be made to the collection objects asking them to obtain the information from the individual objects.

ORB implementations typically take advantage of the co-location of objects within the same server process to optimize the size of object references. For example, some object references embed location information for the server process that 'houses' an object, plus a key that uniquely identifies the object within the server. An ORB can reduce the average per-object reference size by keeping one copy of the server location—which will be the same for all objects from that server—and sharing that copy among all related references. Unfortunately, the information needed for clients to take advantage of co-location is hidden within the opaque object reference structure.

To facilitate bulk operations, a partitioning model is needed that will increase the likelihood that co-location of objects occurs. Further, the client needs a means of identifying co-located objects so that bulk operations on the right set of objects can be requested. Finally, an interface capable of supporting bulk operations on the desired objects must be provided.

3 SOLUTIONS TO SCALEABILITY ISSUES

The previous section identified five issues facing developers who are building applications with high scaleability requirements. This section discusses ways of addressing four of those issues. No attempt is made to address the issue of

appropriate level of CORBA object granularity. Unfortunately, determining appropriate levels of abstraction is still largely an art. The appropriate level will vary from application to application.

The solutions to the issues raised in sections 2.3-2.5 are wrapped together into a model called the Entity-Warehouse Model, or Entity Model for short. In essence, the Entity Model provides structure that enables developers to design their applications in a consistent way to effectively manage the issues that arise when working with CORBA.

3.1 IORs to Represent Managed Entities

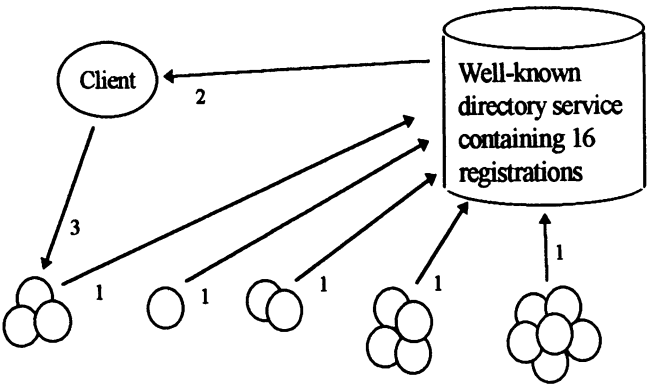
It was decided that if an entity was implemented as a CORBA object, the implementation would exclusively use IORs for those objects, rather than using proprietary references. The applications being developed will run on several vendor platforms, as well as on several different ORBs. These applications make heavy use of naming, trading, and topology services to gather collections of objects to operate on. Many instances of these services will be deployed throughout the enterprise. The instances will be federated together to present a single 'logical' view of the enterprise, rather than a view of disjoint islands of objects. A client is equally likely to obtain a reference to an object implemented over ORB A as from ORB B. Using IORs is the only reasonable way to ensure that the client will be able to use the object once it gets a reference to it.

3.2 Entities and Warehouses — A Partitioning of the Object Space

The Entity Model partitions the object space into entities and warehouses. Entities are instances of objects that generally represent the 'real' objects (physical or logical resources) that the user is interested in managing. Entities are not constrained to being CORBA objects. Warehouses are instances of objects that manage collections of entities. Warehouses may themselves be entities. Warehouses generally are CORBA objects. Warehouses may be managed by other warehouses. An installation might consist of millions of entities, managed by thousands of warehouses, residing on hundreds of management hosts throughout the enterprise.

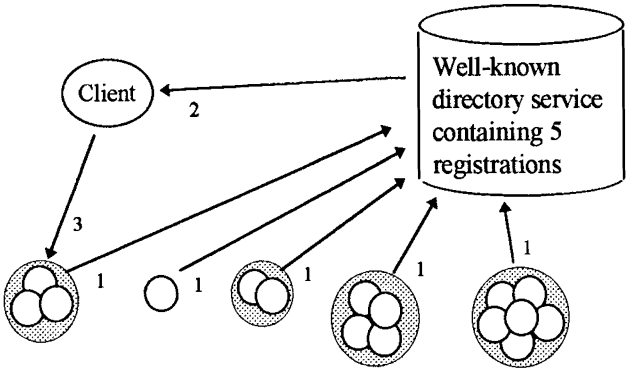
Warehouses play a role in an entity's object lifecycle. Warehouses are often entity factories, or work in cooperation with entity factories. Warehouses are involved in the relocation and destruction of entities. Warehouses may also provide the interface for operating on the entities themselves (i.e., clients manage entities indirectly using an interface provided by the warehouse).

Warehouses also provide directory services for the entities they manage. Rather than utilizing generic directory services to locate all objects, the model encourages limiting the use of the generic services to locating the relatively small number of warehouses, and using warehouses to locate the larger number of entities.



1. White circles represent entities. All entities are CORBA objects. All entities register with a well-known directory service.
2. Client obtains a reference to the desired entity from the directory service.
3. Client uses the entity's object reference to operate on the entity.

Figure 2 Using CORBA without the Entity Model.



1. White circles represent entities, gray circles represent warehouses. All warehouses are CORBA objects, and register with a well-known directory service. Entities *may* be CORBA objects. Entities *may* register with a well-known directory service.
2. Client obtains a reference to the warehouse containing the desired entity from the directory service.
3. Client uses the warehouse's reference to indirectly operate on the entity in directory, or to obtain a reference to the entity itself.

Figure 3 Using CORBA with the Entity Model.

Figures 2 and 3 illustrate CORBA with and without the Entity Model. Object partitionings that often naturally exist are formalized into warehouses. Warehouses can be located using the public object directory, but individual

entities generally cannot. IDL interfaces are provided as base interfaces for both entities (*Entity*) and warehouses (*EntityWarehouse* and *EntityFactory*), as shown in Figure 4.

```
struct PartitionedIdentifier {
    string id_context;
    string id_base;
};

interface Entity {
    readonly attribute PartitionedIdentifier entity_id;
};

struct EntityKey {
    PartitionedIdentifier id;
    Object ior;
};
typedef sequence<EntityKey> EntityKeyList;

interface EntityFactory {
    EntityKey create_entity(in string criteria);
};

interface EntityWarehouse {
    Entity get_entity(in PartitionedIdentifier entity_id);
    EntityKeyList query(in string query_string);
    void destroy_entity(in EntityKeyList key_list);
};
```

Figure 4 IDL for the Entity Model

3.3 Object Identity

The Entity interface defines the read-only attribute *entity_id*, through which each entity provides a unique identifying value. *entity_id* is a struct consisting of two parts. By convention, *id_context* represents the identity of the warehouse containing the entity. *id_base* uniquely identifies the entity within the context given by *id_context*. Individually, each part is immutable, collectively they are unique in space and time.

Because several key applications depend so heavily on object identity, a special structure called the entity key was defined. The entity key contains the entity identifier and (optionally) the IOR for the entity. Entity keys can be given to applications that depend on object identity so that those applications will not have to query the object separately to obtain the value. The entity key effectively

provides a caching mechanism for the object identity value. This caching alternative was chosen over the on-demand model because of the difficulty of ensuring high cache hit rates for some of the applications. The topology service, for example, will contain very large graphs that can be queried in a variety of ways. Until usage patterns can be studied, it is difficult to ascertain if adequate hit rates can be sustained.

3.4 Supporting Bulk Operations

The Entity Model partitions the object space in such a way as to increase the likelihood that bulk operations can be performed with sufficient regularity to be cost effective. An examination of the *id_context* field enables one to determine what groupings exist.

A second necessary ingredient is an interface that supports bulk versions of the desired operation. It is expected that the bulk versions will be very similar to the corresponding single instance versions. The definition of these will be specific to the type of object being managed. The example in Figure 5 illustrates a single instance interface and its corresponding bulk interface.

```
interface IPHost : Host {
    attribute IPAddr ip_address;
};
typedef sequence<IPHost> IPHostList;
typedef sequence<IPAddr> IPAddrList;

interface IPHostWarehouse : HostWarehouse {
    IPAddrList get_ip_address(in IPHostList ip_host_list);
};
```

Figure 5 Single instance interface and corresponding bulk interface

3.5 More on the Entity Key

There are several other aspects of the Entity Model that are not covered in this paper. One semantic of the entity key worth mentioning is the fact that the *ior* value is optional. This opens the door to a variety of implementation alternatives, such as managing non-CORBA objects. For example, applications such as the topology service accumulate entity keys, but do not examine their contents. It is not necessary for an entity to be a CORBA object for it to be manageable by such services. These entities simply must be managed by some warehouse that is a CORBA object. The entity ID can be used to identify the warehouse and the entity within the warehouse. Interfaces on the warehouse can permit operating on entities as second class objects.

4 CONCLUSION

With appropriate usage models, CORBA has been found to be adequate to support applications that must scale to manage very large numbers of objects. The Entity-Warehouse Model was developed to provide a usage model for dealing consistently with several challenges encountered. The Entity-Warehouse Model is an approach to using CORBA that enables developers to effectively manage complexity and the load on system resources. It requires no extensions to CORBA and is not specific to any particular ORB. Developers are not constrained to using this model, however, it is expected that in using the model, the scaleability potential of their applications will be greatly increased.

CORBA has evolved considerably since this investigation was begun. ORB implementations are maturing such that some issues originally raised in the investigation are no longer of major concern. The biggest example of this is evident in the improvement in the size of the IOR. In the first IOR implementations, the optimized size was very large (well over 100 bytes). This was clearly an issue to applications that accumulate and cache large numbers of IORs (such as the topology service). Recent modifications to the specification of the IOR, however, have enabled ORB vendors to significantly reduce that size, with expectations that sizes of 32-64 bytes are obtainable.

Other recent CORBA developments have yet to be investigated to determine the impact on scaleability. Work needs to be done to determine how to best align the Entity-Warehouse Model with specifications such as the Collections service, which provides a model for partitioning objects into collections, and the Query service, which provides a model for querying for objects. Warehouses are a collection of entities, and provide a method for locating entities via a generalized query. Additionally, the Collections and Query services must be examined to determine best practices that will ensure scaleability goals can be achieved.

5 REFERENCES

Object Management Group, Inc. (1996) Naming Service Specification, in *CORBAservices: Common Object Services Specification*, 3-1 - 3-18.

Object Management Group, Inc. (1996) Transaction Service Specification, in *CORBAservices: Common Object Services Specification*, 10-1 - 10-86.

Object Management Group, Inc. (1995) The Common Object Request Broker: Architecture and Specification, 7-5 - 7-6.

Object Management Group, Inc. (1996) Trading Service Specification, OMG Document *orbos/96-05-06*.

Object Management Group, Inc. (1996) Life Cycle Services Specification, in *CORBAservices: Common Object Services Specification*, 6-7 - 6-14.

Object Management Group, Inc. (1996) Relationship Service Specification, in *CORBAservices: Common Object Services Specification*, 9-19 - 9-20.

6 BIOGRAPHY

RICHARD B. (RICK) WHITNER is an architect for the HP OpenView program at Hewlett-Packard. Since joining HP in 1988, he has been active in the design of platforms for network and systems management, and in the investigation and application of object-oriented technologies. During the past two years he has represented HP in the Object Management Group's (OMG) Telecommunications Domain Task Force. Prior to joining HP he was a senior programmer/analyst for the Virginia Cooperative Extension Service. He holds a B.S. in Mathematics from Clinch Valley College of the University of Virginia and an M.S. in Computer Science from Virginia Tech.