An Approach for
Constructing Web Enterprise Systems
on Distributed Objects

Hiroshi Kashima

Abstract
This paper describes problems and solutions in constructing Web-based enterprise systems with
distributed object technology.

Since Java was announced, distributed object-oriented technology has been arousing much interest in the
Web environment. It does not negate HTML; but rather has been accepted as a technology to strongly
support the Web environment in such areas as client-side processing and communication traffic
optimization, together with its descriptive capability. The systems that will be required for network
computing in the future must be mutually cooperative and be equipped with expandability to cope with the
scaling up of the network. When such a background is considered, the distributed object is well-suited for
the communication infrastructure for application development in a Web environment.

On the other hand, as for the distributed object-based communication traffic and architecture, there are
many things that can be determined only by actual trials, such as whether satisfactory performance can be
achieved in an Internet environment.

This paper concentrates on issues to consider when investigating the applicability of using distributed
object technology in the Internet environment.
Contents

1. Introduction .................................................................................. 3
   1-1. Background ........................................................................... 3
   1-2. Characteristics of Web System .............................................. 3
   1-3. Characteristics of Enterprise System ....................................... 4
   1-4. Distributed Objects under Web Environment ......................... 4
2. The Concept of CORBA .............................................................. 5
   2-1. CORBA Development Environment .................................... 5
   2-2. CORBA Execution Environment ....................................... 7
   2-3. Other Functions ..................................................................... 8
   2-4. CORBA Advantages in a Web Environment ......................... 9
   2-5. Issues to be Solved in Adopting CORBA .............................. 10
      2-5-1. Issue in Application Design ..................................... 10
      2-5-2. Issues of System Design and Planning ......................... 11
   2-6. Summary .............................................................................. 12
3. Approach to Application Design .................................................. 12
   3-1. Realization of Thin Client ................................................ 13
   3-3. Light Object Oriented Design .......................................... 13
   3-4. Location Transparency .................................................. 14
   3-5. Scalability ........................................................................ 14
   3-6. Solution of Design Issues .............................................. 14
      3-6-1. Implementation of Memory Management Class ............. 14
      3-6-2. Connection Model Expansion by Factory Function ........ 15
      3-6-3. Method of Object Copying ...................................... 15
      3-6-4. Implementation Inheritance by TIE ........................... 16
      3-6-5. Two-way Communication by Callback ....................... 16
      3-6-6. Implementation of Japanese Support Class ................. 17
4. An Approach to System Design .................................................... 17
   4-1. Client Platform .................................................................. 18
   4-2. Security ............................................................................ 18
      4-2-1. Internet Security .................................................. 18
      4-2-2. Access Control .................................................. 18
   4-3. Failure Management ..................................................... 19
      4-3-1. Network failures .................................................. 19
      4-3-2. Process Failures .................................................. 19
   4-4. Operation Control .......................................................... 20
      4-4-1. Failure Monitoring ............................................... 20
      4-4-2. Automation ....................................................... 21
      4-4-3. Module Distribution ............................................ 21
5. System Implementation and Evaluation ...................................... 22
6. Concluding Remarks ................................................................. 23

References ..................................................................................... 24

Author profile ............................................................................... 24
1. Introduction

The computer world is facing a time of big change. Internet technology, like the telephone and the airplane did in the past, has suddenly “made the earth smaller”. The Web environment has made it possible for people to enjoy new experiences easily and inexpensively without leaving their home or office. If viewed from the business perspective, it readily provides a platform where all people throughout the world are potential customers.

This paper discusses some of the problems and solutions for applying distributed object-oriented technology to the construction of Internet-based enterprise systems. It is based in part on our experiences with an actual project to connect Java clients on the Internet to a current legacy system.

1-1. Background

In many enterprises in the past, a large number of individual systems were developed without a strategic vision on how the enterprise-wide information systems should function, and how they might work together. What has developed in many large organizations is “information islands”, where information is used only by the owner department and segregated from others.

On the other hand, customers expect all their needs to be satisfied by the enterprise with a single call. To maintain such a service level, customer requirements have to smoothly flow across various departments or sections within the enterprise. Information must flow within a brief amount of time across different platforms. It must be remotely maneuverable and cooperatively processed by various people with different responsibilities. All of these necessities strongly motivate the adoption of distributed processing.

As for the open systems which were introduced with great expectations of reducing system costs, various problems have surfaced, such as disordered design distribution that imposes excessive load to clients. This increases the management cost and eventually makes clients very fat, or specialized as fixed purpose terminals. While its advantages for the client, such as superb user interfaces, must be fully utilized, the importance of well-controlled design to centralized processes has been recognized anew. From that viewpoint, the concept of “thin client” will be widely accepted for operational systems typical on the Intranet. By the same token, large mainframe systems, that process a huge volume of transactions, should be reevaluated as distributed servers with an immense processing power.

Such an evolution of computing style, combined with Internet considerations, has brought about a new information processing style called “network computing”.

1-2. Characteristics of a Web System

In constructing a Web-based system, attention must be paid to various points that are different from those in more traditional systems. Following are the characteristics of a Web-based system:

- Client environment cannot be specified
- Low speed, low quality lines are the base
- The scale is large
- Security is an issue
When many and unspecified users are the target, prerequisite conditions for the client hardware and software must be flexible. The resource requirement should be as small as possible and a thin client orientation, where the logic and the data are processed at the server side, is preferred. It is also important to make an assumption that the speed and quality of the base network is relatively low. A design that permits reconnection at the time of failure can be achieved by limiting the number of communication packets between the client and the server. In the sense that applications are made open on the large-scale network and future application scale cannot be specified, a great attention must be paid to scalability. Needless to say, security is the key to protect the system from malicious external attacks.

1-3. Characteristics of Enterprise Systems

Following are some of the characteristics to incorporate when constructing enterprise systems on the Internet:

- Transaction processing
- Cooperation with current systems
- Continuous operation and rigorous failure management
- Adaptation to technology innovation and education plan

In many cases, an enterprise system provides processes with clear objectives and executes certain types of transactions. Between the Internet terminal and the server, the transaction property is not needed in most cases, yet the server has to perform the data processing functions without any conflicts. Also, for transaction processing, cooperation with current systems is usually required. A new system has to maintain a high connectivity level with current systems.

Since there are no national borders in Web applications, it is difficult to adopt a system stop time due to the time differential, which necessitates a continuous operation in many cases. Also, because the system is open to the public, its quality and availability could affect the image of the enterprise. Countermeasures for system failures, not to mention the maintenance of application quality, are important.

The development of an enterprise system generally requires a large-scale development process as well as long term maintenance. It is necessary to keep up any technological innovations during that time, and continuous education is very important for the developers who are working on the base technology.

New distributed technologies under the Web environment are frequently announced by many companies these days. Expansion of HTML, the base structure to run stand-alone applications on a Web browser, functions to support development at server side, and so forth, are just a few of them. However, so far as product quality is concerned, many of them seem to have been released without enough testing just to meet the competition with other companies. Some of the problems are specifications that are frequently changed, programs that are full of patches, and systems too complicated to be widely accepted in the marketplace. Since stability is more important than novelty for an enterprise system, it is important to select a platform that has been developed with enough buildup time and that reflects the best features available among vendors.

1-4. Distributed Objects under Web Environment

Since the announcement of Java, distributed object-oriented technology has rapidly drawn attention in the Web environment. It does not negate HTML, but rather is accepted as a technology that strongly supports the Web environment in such areas as client-side processing and communication traffic optimization, as
well as providing good descriptive capability. To apply Java does not immediately mean to adopt
distributed object technology; yet, systems constructed hereafter must be mutually cooperative and be
equipped with the expandability to meet the requirements of rapid network expansion. With such a
background, the distributed object should be more suitable than the socket or RPC-based environment as
the communication platform for application development under the Web environment. (The reason will be
discussed later.)

Following are the typical distributed object technologies that are usable under the Web environment:

**CORBA (Common Object Request Broker Architecture)**
An industry standard made public by OMG (Object Management Group), which is a consortium
organized by computer vendors and various research and academic institutes.

**DCOM (Distributed Component Object Model)**
An expansion for distributed environment of COM, which is developed and distributed by
Microsoft as the object technology for Windows environment. When a system is built in a
Microsoft-exclusive environment, it provides some powerful functions such as, for instance, the
initiation of Excel on a remote machine. Currently, it is supported by Windows environment only,
but Microsoft is positioning it as the essential function for WIN32 environments. (It is provided
with OS for free.)

**RMI (Remote Method Invocation)**
A package included in Java since JDK1.1, which adds RPC technology to Java. It can be used for
free.

**HORB (Hirano ORB)**
A free, Java-specific product for distributed objects, which was developed by S. Hirano of MITI,
Electro Technical Laboratory. Programming of objects without distinctions of remote or local is
possible, which makes it very useful as an introductory product for the distributed object.

As the distributed objects used in the heterogeneous Web environment, CORBA is best suited because of
its complete independence from vendor/platform. When the interoperability with current systems is
considered, the technologies that are limited to a specific platform or those for Java-only as the supporting
language are not appropriate for medium-scale and larger enterprise systems. In the case of CORBA, the
connectivity with current systems is kept high because of its mainframe and COBOL support.

### 2. The Concept of CORBA

This section presents an overview of the architecture upon which CORBA is based.

The CORBA specification defines IDL (Interface Definition Language), control objects such as ORB and
BOA (which will be discussed later), mapping to languages such as C++ and COBOL, and standard
protocols (such as IIOP). In addition to these core functions, it also specifies common services such as a
naming service and an event service (the implementation of these functions vary for each developing
company).

#### 2-1. CORBA Development Environment

While RPC takes a remote function call as a type of the remote call, CORBA takes remote object calls.
Its programming is similar to that of RPC; interface definitions called IDL are compiled into stabs and skeletons, which are then used to generate codes for clients and for servers. In Figure 1, the squares with thick lines represent the portions to be newly developed. It indicates many portions are generated automatically.

**Language support**

The CORBA specification specifies the language mapping for C, C++, COBOL, Smalltalk and Ada. Java support was also announced in CORBA2.2 in February, 1998. Many products will support Java and C++, but COBOL support is very important for the products that support mainframe systems, such as CBConnector. If a project is free to select a language, Java would be the best choice. Java has a close affinity with CORBA, and programming in C/C++ is comparatively difficult, for the following reasons:

- The concept of object reference in CORBA is the same as in Java, which makes the programming natural and smooth.
- While the memory management related with call semantics is fairly cumbersome in C/C++, the garbage collector does the job in the case of Java.
- CORBA architecture has high adaptability to multithread models. In Java, the language itself supports threads to make the implementation easy, while in C++, developers themselves have to implement the concurrent processing by using OS functions such as the semaphore and mutex.
- The uniqueness of the name space is important in the Web environment. IDL supports it in the module sentence. Although the module sentence is supposed to be mapped into the namespace in C++, many compilers do not support the namespace yet due to the relative newness of this function, so most CORBA implementations map it into a single class. This means that a single compilation of a huge file is needed in C++ even if the close modularity is implemented in IDL.

**Data type**

The parameter types exchangeable with remote objects are fairly extensive, including structures, arrays, object references, etc., in addition to primitive types such as character strings and numerical values. Other
unique types, such as Sequence, which allows more dynamic memory management than arrays, and Any, to which any data can be stored, can also be defined. While primitive types and structures are handled in call by value, objects are managed in call by reference so that the real value is not actually transferred. For these data types, the format of the communication protocol is defined and ORB does the translation for the local environment (called martialing/unmartialing, the same as in RPC).

**Object reference and call**

The access to distributed objects is gained by using object reference (the same as reference in C++). In IDL, the call semantics of object reference have to be explicitly specified, as in the following:

```java
interface UserClass {
    UserObject createObject (in string name);
}
```

This example shows that the method "createObject()" is a method which receives a string type as the input parameter and returns an object reference of "UserObject" type. There are two other modes in call semantics: `out` (returns a value from the output parameter = server) and `inout` (both input and output are possible). Distributed objects are always passed via reference; value passing and object copying are not supported (they will be supported in CORBA3.0). The object reference can be freely passed in this way across processes and machines.

**2-2. CORBA Execution Environment**

At execution time, a function called ORB (Object Request Broker) plays a central role as a software bus for services such as server detection and request transfer. It makes it possible for clients to access distributed objects in the same way as through local access. In the server side, there is a function called BOA (Basic Object Adapter) which does the initiation and control of distributed objects.

In addition to the static call based on stabs and skeletons as prerequisites, CORBA supports the dynamic call, which manipulates objects by making reference to the interface information called interface repository. DII (Dynamic Invocation Interface) is for the client to dynamically access servers without stabs, and DSI is for the server to accept requests from clients without skeletons.

<table>
<thead>
<tr>
<th>Client applications</th>
<th>Implementations of server objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stabs</td>
<td>Skeletons BOA</td>
</tr>
<tr>
<td>ORB</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2. Logical module structure at execution time (left: static, right: dynamic)**

**Communication protocol**

As shown in Figure 2, communication protocols are hidden by ORB. CORBA specifies the protocol as IIOP (Internet Inter-ORB Protocol) when TCP/IP is used for communication, which secures the connectivity across the products of different vendors.
Some implementations provide functions to automatically switch among communication methods, so that applications are not required to be conscious of the method selected, such as an ordinary local function call when the target object is in the same process, a pipe or shared memory for different processes within the same host, and TCP/IP for cross-host communications.

**Initiation of server process**

As BOA is in charge of the server process initiation, CORBA defines initiation semantics as follows:

- **Resident server**: a method other than BOA to pre-initiate processes. The automatic initiation at OS startup time (/etc/inittab in UNIX) and timer initiation (cron in UNIX) are included in this category.
  - **Shared server**: a server is first initiated by a bind (connection) request from a client, and is shared by multiple clients thereafter.
  - **Non-shared server**: a server is initiated by each bind from a client and exclusively occupied by the client. The server terminates at the end of the session or the disconnection of the client.
  - **Method-specific server**: a server is initiated for each method request. The server terminates when the method ends.

**Thread policy**

Java-based products are always multithread oriented because of the nature of the language. In the case of C++ based products, how to use threads is a matter of choice; yet, since the C++ language itself is not multithread oriented, the programming would be easier for a single thread if it is allowed. However, as will be discussed later, CORBA architecture goes better with multithread mode, as in the case of using callback or status inquiry during processing. Since there are several variations in the relationship between the client and the server thread, as shown in the examples below, each situation has to be closely investigated, depending on the service content:

- 1 client for 1 server process (non-shared server)
- 1 client for 1 server thread (resident, shared server)
- Fixed number of processes/thread (resident, shared server)

Although dependent on the product, the thread policy has to be decided for each object implementation process:

- **Thread pooling**: a fixed number of threads are pre-initiated and allocated for each method request. Each method uses a different thread.
- **Session-specific thread**: a thread is initiated for each session (started with bind). The thread will be fixed while the same connection is in use.

**2-3. Other Functions**

**Caffeine**

Caffeine is an outcome of the joint research by Visigen and Netscape, which provides functions to define IDL in Java. As a product, it is onboard VisiBroker. Caffeine has some unique extensions to a portion of IDL to realize object copy in the CORBA environment. Care must be taken due to the fact that communication with the C++ version would become impossible, even between VisiBrokers, if these extensions are used. Also, in most enterprises of medium-scale and larger, the IDL would be managed at
the company level as in the case of a database, where the final management of objects would often be done in IDL language even if the IDL development itself has been done in Java. It should be avoided so that excessive IDL knowledge is not required.

**Push type design**
In CORBA an event service is defined, by which objects (called event channels) to collect and distribute events can be defined, even among applications not known to each other. This makes it possible, even in the Internet and Intranet environment, to establish data push type applications, where data is periodically updated from the server side (such as is done with stock quotes), without periodical polling to clients.

2-4. CORBA Advantages in a Web Environment
The CORBA architecture was discussed so far. Then, the next subject is why CORBA is useful for the Web environment. Following are the advantages of CORBA when compared with other distributed objects like socket and RPC:

- **Ease of use:**
  The capability to design and code without consciousness of the communication makes it possible not only to reduce the number of development steps, compared with socket, but also to realize distributed computing by a very simple design. Since its base is object-oriented, what has been designed can be implemented by the semantics as it is. In the case of socket and RPC, since the communication portion cannot be object-oriented by itself, the design would be split there and the wrapper code is often said to be more than 20% of the total code.

- **Object-oriented:**
  In a software design, it is essential that all the system states to be processed are covered. The number of system states increases as the system becomes more versatile, which could make software development more costly and downgrade the quality. Since all the nodes that run in parallel have their own status, the total number of states significantly increases in a distributed environment. The fact that the versatility of the distributed system can be easily managed by the encapsulation and the concept of an exception of the object-oriented technology has been proved by various research activities of CORBA, DCOM and others in this field. For the Web environment, which is a large-scale distributed system, object-oriented technology is also useful in productivity as well as in quality improvement.

- **Communication traffic optimization:**
  It is well known in local programming that passing by reference and passing by pointer are more efficient than passing by value. Although this is also true in a distributed environment, passing by reference was practically impossible in traditional technologies including RPC. In CORBA, passing by reference is the base in passing objects, by which distributed objects can be freely passed across processes and machines. The optimization of communication traffic is easier this way than actually passing the real entities of an object, so that high-speed communication can be designed even in a Web environment where line speed is assumed to be relatively low.

- **Common services:**
  Services useful in a distributed environment, such as distributed object initiation and naming service, are readily available. They not only improve development productivity, but also make it possible under the distributed environment to construct a mutually interoperable environment without using vendor-unique facilities.
Platform independent:
In the Web environment, various techniques on various platforms exchange information. The importance of CORBA, which is independent of vendor/platform and where mutual connectivity has been proved, will be further enhanced in the future.

2-5. Issues to be Solved in Adopting CORBA

CORBA is not free from problems. This section discusses the issues to be solved in applying CORBA.

2-5-1. Issue in Application Design

Following are the design issues to be solved in the actual use of CORBA:

- Connection model is restrictive.
- Objects copying is not supported.
- Inheritance of an implementation is impossible.
- Overhead of two-way communication is heavy.
- Japanese support.

Each of these issues is described in detail in the rest of this section.

Connection model

The basic style of CORBA distributed object is that multiple clients share a single remote object. In such a connection model, multiple objects have to be previously instantiated at the server side if the client has to bind multiple remote objects. This makes the design fairly complicated. Moreover, to prepare many bindable objects is not desirable from the viewpoint of access control and overhead. In the case of HORB mentioned previously, a function called generation model is supported to generate objects according to the client request. The same function should be implemented in CORBA in some way.

Object copying

CORBA generally does not support the transfer of an object copy. For a case such as when the execution itself at the Client has the meaning (for instance, to execute a method to play music at the server is meaningless), an object copy is required to be resident at the client.
**Inheritance of an implementation**

Most products do the implementation of CORBA server objects by inheriting the skeleton classes that IDL compiler produced as the output. On some platforms that do not support multiple inheritance, as is the case with Java, it is impossible to inherit implementations from other classes (although the inheritance of an interface is supported).

![Class diagram by Rose](image)

**Figure 4. Inheritance of interface and implementation (Class diagram by Rose)**

**Two-way communication**

In order for the server to make a call for a client, the client has to temporarily act as a server. However, the load to the naming service would become too heavy if all the clients enroll the objects as bindable.

**Japanese support**

CORBA adopts UNICODE as the Chinese character code. UNICODE has an issue that there are delicate differences between the standard specification and the Microsoft specification. Care must be taken when to cooperate with Windows systems. Since CORBA products generally use the OS translation function for the translation between UNICODE and the OS internal code, there could exist some unusable characters.

**2-5-2. Issues of System Design and Planning**

Following are the typical issues in system design and planning:

- Designers and developers must be familiar with object-oriented technology
- Security
- Failure management

Since the distributed object is based on object-oriented technology, its knowledge is required as a prerequisite from the external design phase up to coding. However, class design as well as object-oriented programming are not widespread yet. Organization of the project members and their education, as well as the establishment of development standards, are extremely important. Even for those skilled with object-oriented technology, a certain level of experience and some trial period may be required to start designing with a full understanding of the unique restrictions and consideration points inherent to the distributed object.

One of the issues that cannot be avoided in using CORBA under the Internet environment is security. Since most enterprises have firewalls installed, CORBA protocols usually cannot pass them as they are. Moreover, some CORBA products do not provide authentication and/or access control functions. In those
cases, user's own functions of authentication and access control are mandatory; otherwise, objects could be accessed by any clients.

Although there are some functional differences by product, CORBA does not support such a function as that, when a client goes down, the server detects it and deletes objects in the current use. This could be a crucial factor in designing measures for failure recovery.

2-6. Summary

The advantages and the problems of CORBA so far discussed are summarized in the following table. It is important to design to cover the problems by fully utilizing the advantages.

<table>
<thead>
<tr>
<th>Issues/Advantages</th>
<th>How to improve</th>
<th>Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design to improve problems</td>
<td>Complicated memory management in</td>
<td>Implementation of memory management</td>
</tr>
<tr>
<td></td>
<td>Restrictive connection model</td>
<td>Factory</td>
</tr>
<tr>
<td></td>
<td>No object copying</td>
<td>Method to copy objects</td>
</tr>
<tr>
<td></td>
<td>Restriction of implementation</td>
<td>Implementation inheritance by TIE</td>
</tr>
<tr>
<td></td>
<td>Heavy two-way communication</td>
<td>Two-way communication by callback</td>
</tr>
<tr>
<td></td>
<td>Issues of Japanese support</td>
<td>Implementation of Japanese support class</td>
</tr>
<tr>
<td></td>
<td>Reduction of client load</td>
<td>Implement Thin client</td>
</tr>
<tr>
<td></td>
<td>Low-speed/Low quality line</td>
<td>Traffic optimization, recovery system</td>
</tr>
<tr>
<td>Firewall management</td>
<td>Internet security</td>
<td>4</td>
</tr>
<tr>
<td>Design to utilize advantage</td>
<td>Large-scale network</td>
<td>Scalability, location transparency</td>
</tr>
<tr>
<td></td>
<td>Communication traffic optimization</td>
<td>Method to optimize communication traffic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Light object oriented design</td>
</tr>
<tr>
<td></td>
<td>Platform independent</td>
<td>Client platform</td>
</tr>
</tbody>
</table>

Table 1. Summary of advantages and problems

3. Approach to Application Design

In order to fully utilize the advantages of CORBA when it is adopted, it is necessary to strictly follow the correct methods of design and use. Application design considerations will be discussed in the following order:

- Realization of thin client
- Optimization method of communication traffic
- Light object-oriented design
3-1. Realization of Thin Client

In order to achieve thin clients, it is important to pursue designs that make the server do the processing as much as possible, and that make the client concentrate on the communication traffic optimization as well as the presentation processing. A hierarchical architecture with a structure of more than three layers fits well in this respect. It also has the advantages listed below, and goes very well with the system design concept under the Web environment.

**Client workload reduction**: can remove business logic from the presentation layer, decrease the client workload and minimize hardware restrictions. This is also a design to withstand future changes.

**Efficient database access**: can optimize communication traffic and reduce transactions by allocating the business logic layer to the server.

**Database access transparency**: can absorb database locations and differences among vendors by the data abstraction layer.

3-2. Method for Communication Traffic Optimization

CORBA architecture that is based on pass-by-reference can be appreciated from the viewpoint that it does not degrade performance in manipulating large objects. On the other hand, when a large object is to be processed as a whole, it could degrade performance if the contents are accessed one by one. This could be a big design issue because a low-speed line is the base assumption in a Web environment. If it is known from the start that the whole content is to be transferred, as in the case of file transfer, a better performance can be obtained by sending it by bulk transfer from the beginning. *Sequence* type is used for bulk transfer. *Sequence<octet> is particularly fast for file transfer and others.*

The speed of bulk transfer is as fast as that of socket communication, but in the transfer of fine granularity it could be extremely slow under the environment of slow turnaround. The issue of granularity is one of the major tuning items of CORBA.

3-3. Light Object-Oriented Design

Although there are differences in implementations, CORBA distributed objects are classified into two categories: one is the persistent object which is bindable from the client, and the other is the transient object which cannot be bound directly. Since persistent objects directly accept connection requests so that they require additional work such as the registration to the naming service (often an independent server process) and security checking, the workload of object generation tends to be heavy. In the case of transient objects, the object generation workload is light in general, although they require indirectly to pass the reference by return of a distributed method or by an output parameter. It is important from the workload and the security viewpoint to do design based on transient objects as much as possible, and restrict the use of persistent objects to limited areas only.
3-4. Location Transparency

Most traditional TCP/IP applications do connections by specifying the IP address and the port number. Standard CORBA products provide the following methods for obtaining object references for a server implementation:

- Naming service: a common service defined by OMG.
- Product-unique methods: often provide recovery functions and simple query functions.
- Passing object reference as character string: send and receive a reference in the form of a character string.
- Obtaining as a parameter of other object: often used in transient objects. Not usable for binding of initial connection.

The naming service is usually used to realize location transparency, but when the multivendor connection is not available, product-unique methods tend to be superior in usability.

3-5. Scalability

Scalability is the expandability to meet changes in scale, maintaining the resource sharing as a single distributed system, regardless of how many terminals access the system from the Internet. As designing a scalable system is accompanied with difficulties, it is important to make the access to shared resources independent from the system scale. In addition to caching techniques such as data replication and server replication, the realization of the following mechanisms is required to be investigated:

- **Distributed locks**: When multiple clients can access the same object under the distributed environment, use of an exclusive control by a distributed lock has to be investigated in order to maintain the data integrity. A distributed lock for the exclusive control is defined in the OMG concurrency control service. It is an object by itself and not a control function for other objects. Whether to force using this distributed lock for an exclusive control is left for the implementation. The distributed lock function is not implemented by all products; if not, the best way is to newly create one since its implementation is easy and small in scale.

- **Shared data**: This will enhance the expandability if it is centrally provided as in the concept of a repository, because any number of applications can access it. Since applications are required to be integrated with the repository only, without any application-to-application integration, the application separation and reuse of the repository can be achieved. As for the implementation, a method to provide it as a distributed object is also conceivable, in addition to traditional methods such as shared memory.

3-6. Solution of Design Issues

3-6-1. Implementation of Memory Management Class

The reason why memory management in C++ is complicated is because there is no parent class that would be the base class for the whole objects in C++. It is useful to previously develop base classes for almost all objects:

- Garbage collection (to release unused memory)
Concurrent processing under a multithread environment
Object location information and callback function
Debugging facility such as dump function

As for memory management, there is also the choice of leaving it to education and standardization. Any way, debugging of access-related troubles at the leakage or the deleted object always presents a lot of difficulties.

### 3-6-2. Connection Model Expansion by Factory Function

In a CORBA environment where only the connection model is supported, generation models are created in general by preparing a so-called factory object which generates objects and returns them to the client. Clients are designed to be bound to the factory object, make it generate other remote objects and receive them. In this way, the instance management for each client is easily realized and the number of heavy connection processes of binding can be reduced. A release operation is always required at the server side after use since the transient objects generated by a factory are not deleted automatically.

![Factory to realize generation model](image)

**Figure 5. Factory to realize generation model**

### 3-6-3. Method of Object Copying

When the copying of objects is needed, it can be achieved by the design to generate a wrapper class as in the following:

- **Content transfer**: Contents (property values) are transferred as a structure which is to be regenerated at the client as an object. A wrapper class to do the job is required in the client.

- **Cache in local**: By using the characteristic of CORBA objects that they can be generated at any place, it is possible to generate objects for both local and remote with the same contents and reference the local object as the cache for access the second time on. Of course, a certain mechanism to reflect the update of the contents is required. However, this can be an effective design method when the same objects are accessed repeatedly without frequent update operations. In the CORBA design pattern [1], this is realized by replacing the client stabs that have been generated automatically. The author is of the opinion that, generally speaking, when the issue of maintenance is considered, the self-generated codes should not be modified in ordinary projects. It could easily generate the "Never change in interface " syndrome which would directly clog the artery of the system. Therefore, a wrapper class is required in this case as well.
3-6-4. Implementation Inheritance by TIE

The CORBA specification provides a method called TIE mechanism for realizing the server implementation by a delegation model, not by a skeleton inheritance. It is realized by having the skeleton contain (has-a) an implementation object and mapping all the methods to the methods of the implementation class. When Java is used as the server language in a CORBA environment, the products that support TIE function should be selected. An individual measure will be required for the environment where TIE is not supported.

TIE mechanism has certain unique characteristics, such as that parameters cannot be passed to the contractor (object generation method). By creating a complementary class as in the following, such characteristics can be completely hidden from the outside.

![Class diagram by Rose](image)

Figure 6. Wrapping technique of TIE mechanism (Class diagram by Rose)

3-6-5. Two-way Communication by Callback

The cases where two-way communications are required (which often happens for such reasons as broadcasting from the server to clients) can be managed by temporarily reversing the role of the client and the server. Since CORBA objects have location transparency, if the server calls a method after an object reference generated in a client is passed to the server, the method will be executed at the client.

(Multithread is prerequisite for the client). If clients generate persistent objects, the workload to the naming service and others would become heavy; therefore, a transient object reference should be passed as a method parameter. By the same token, the callback to asynchronously call clients from the server can be realized:
3-6-6. Implementation of Japanese Support Class

The issue of UNICODE will not be resolved until all the makers agree upon to the same implementation. For the time being, tentative solutions as shown below would be required:

Avoid using characters in question: When questionable characters are uses, you must prepare a process which notices an error for the character of this category in such areas as the character input for a display and the input of a database update batch. Sometimes it is difficult to get agreement for this from the customer since some popular characters such as "~" cannot be used. The translation process will use functions of the client OS, Java and the server OS.

Newly develop a translation process: A process has to be developed to translate everything without any conflicts. For that, it must be explicitly specified which standard the code belongs to at the time of input and output.

In an actual project that the author participated in, where Java was used on Windows for clients and C++ on AIX for the server, problems occurred from both sides of Java and CORBA. Since the support of the characters used in customer names and addresses in the current database was the specification of the customer requirement, the perfect support of Kanji characters defined in the IBM Kanji specification was mandatory. To solve the problem, a Japanese class was newly developed which internally holds standard UNICODE, MS-UNICODE and shift-JIS and does the translation on demand at execution time whenever required.

4. An Approach to System Design

This section discusses the following system design considerations for using CORBA under the Web environment:

Client platform
Security
Failure management
Operation control
4-1. Client Platform

In order to fully utilize the advantage that CORBA is platform independent, the client environment to be supported must be carefully investigated. Although it is important to support both Netscape Navigator and Internet Explorer when their current usage level in the market is considered, there are slight differences in JVM (the core of Java) on these browsers, so the continuous support of both browsers would make the development workload very heavy. A JDK-compatible plug-in called Java Plug-in, which supports both of them, is provided from Sun; the best way, therefore, to guarantee the operation is with this plug-in. A dynamic installation via the network is possible for the clients to which the Java Plug-in has not been introduced yet.

4-2. Security

4-2-1. Internet Security

One of the issues that has to be solved, by products or by the environment, is Internet security. The approaches to make it possible for CORBA protocols to pass firewalls are different among products. Here, the examples of IONA’s OrbixWeb and Visigenic’s VisiBroker will be examined.

HTTP tunneling

VisiBroker provides a method to pass through firewalls by wrapping IIOP with HTTP protocols. At the client, a run time class will format IIOP with a method following the HTTP protocol. On the server side a process to pick up IIOP from HTTP exits, called Gatekeeper, which acts as a gateway between the object implementation and the client.

Extension of firewall function

OrbixWeb provides a product called Wonderwall which extends the firewall function to make IIOP pass through firewalls. In this product, an agent object called IIOP proxy, similar to HTTP proxy and FTPproxy, works on Wonderwall.

While the extension of the firewall function can be said to be superior to HTTP tunneling because of less performance overhead, a richer product line is required to replace the firewalls being currently used at various enterprises. Standardization in this area has not been completed yet, and the discussion is now in progress.

4-2-2. Access Control

The easiest way to meet this requirement is to adopt a product that has an access control function; but, it is also true that a simple function for it can be developed without much workload. When user registration and maintenance tasks are considered, this is probably an area where new development is rather preferred.

As shown in Figure 7, a certificate is issued as the result of a log in and the client attaches it for all the method calls for persistent objects. Bindable server objects confirm the validity of the certificate by performing a session checking on the session control object that issued the certificate. Similarly, a function such as to restrict the operation according to the user authorization can also be added.
4-3. Failure Management

There are a lot of ways to recover from errors and failures; described below are the most commonly adopted methods.

4-3-1. Network failures

The common measures are either to provide multiple lines or to enable reconnection after recovery from a failure. Line multiplexing does not require any measures in the application since line switching is done by a lower layer. If the multiplexing of the line is impossible, reconnection is done after recovery from a failure. The reconnection will be realized by setting the \textit{rebind} on in the option at the binding, and by holding the object reference to make it possible to reconnect to the same server process after recovery. If an object can be implemented as a stateless server, it allows another choice - to receive the service from usable objects on a different host by rebinding, which will further improve reliability.

4-3-2. Process Failures

Server multiplexing or backup is the most common method for managing process failures. In server multiplexing, multiple servers are initiated and do services concurrently. Although the design is fairly difficult to hide the switching of the process from the user at a failure under the server multiplexing environment, it could be realized by just a setting of \textit{rewind} option, for instance, if the object is implemented as a stateless server.

The backup is realized by preparing a standby machine and by switching (called takeover) to it at the failure of the system in use by using HACMP or others. Since there is no product available that has a cooperative function to hide the switching at the time of a machine takeover from the user, the method would become a design issue. A takeover usually takes several minutes; therefore, to let users be aware of the session disconnection could be one of the choices.
Countermeasure for failures by stateless server
Different from the cases of most other distributed models, CORBA does not specify any restriction that servers must be stateless. However, the mechanism to notice the status is not provided either, even when the object is not in use any more at the termination of the client process, for instance. Therefore, making the objects to provide services stateless would generally increase the applicability to a CORBA model even if the separation of the state information is required.

![Figure 10. Separation of service and state](image)

Under the Web environment, the reliability would become higher if the operation can be continued even when the connection between the client and the server is temporarily cut off. For that purpose, the data of the interaction state needs to be preserved. Generally, the client takes the responsibility to preserve the state under the environment where only the stateless servers are supported.

4-4. Operation Control

4-4-1. Failure Monitoring

There are generally two types of failure monitoring: one is a passive monitoring which monitors the received alerts, and the other is an active monitoring to periodically monitor the predefined subjects. In practice, both are used in combination. In the project the author was involved in, both an alert transfer function which can be called from an application and a monitoring console which receives alerts and displays them on a display panel were developed. The monitoring console was implemented in Java, which supported any number of consoles at any location. It also provided an external monitor interface to send alerts to TME10 consoles. In order to minimize the workload to develop these functions, a base class to consolidate the monitoring console and the class interface of the external monitoring was designed. The number of support terminals could be changed dynamically by holding the callback list at the monitoring server side. In order to avoid blocking the sender by alert sending failures, the oneway property is given to the alert transfer method. As for the active monitoring, the implementation level would differ depending on what level of failures are to be detected. For an existence checking of an object including the process, the result of the actual ping to the object should be returned. (There are implementations that support this function.) To further confirm if the object is not hung and is still functioning as the application, a confirmation method should be actually executed.

Client failure monitoring

There are certain differences in functions for each product in detecting, from the server side, whether or not the client is still alive. However, any product has to eventually issue polling in order to detect the
status of the client in any case. (This is a kind of the function that should be covered by products.) The polling itself can be easily realized by the callback pattern as described previously, but it has to be generated as multithread for either case of the client or the server. (In the case of the server, a method to provide a unique monitoring server is conceivable.)

```
CallbackInterface
  alertNotice()
  getCBPartition()

MonitorConsolePanel
  alertNotice()
  panelDisplay()
  panelDelete()
  constructor()

CallbackList
  enroll()
  delete()

ExternalMonitor
  alertNotice()

MonitorConsolesControl
  enroll()
  delete()
  sendAlert()
  pastLogDisplay()
  pastLogDelete()
```

Figure 11. Monitoring console (passive) design

In our project, a base class was designed, which had a child-parent relationship among objects and also had a garbage collection function to delete all the child objects when the parent object should fail. By constructing a server object tree which has a callback object in the client as its parent object, all the server objects being used exclusively by the client can be deleted at the time of a client failure.

4-4-2. Automation

As a prerequisite for realizing automation, appropriate methods have to be implemented to enable the manipulation of the target objects whenever needed, such as for the centralized control of distributed objects. Since it is convenient to have the following implementations especially as resident objects, it is recommended to implement them as base objects:

- Function for forced operation discontinuance and for normal process termination
- Function to return state information
- Function to switch logging
- Signal processing function

4-4-3. Module Distribution

Unless the client is a terminal to use HTML only, a function to distribute modules is required. Although Java applets can be dynamically loaded, it takes a fair amount of time even to download the CORBA modules alone. It is recommended to investigate another way to distribute modules. In our project, a cooperative function for MS Excel, Adobe Acrobat and others was required in the client. The distribution of DLLs that use cooperation functions for Java and C (JNI) was mandatory so that multiple distribution methods were provided and used in parallel as in the following:
Distribute installer-printed CD-ROMs.
Paste the installer link on a Web page.
Distribute needed modules dynamically based on the result of version checking at execution time.

5. System Implementation and Evaluation

System overview
The author was engaged in a financial system development project of a scale for 150 man-months, and there developed a lightweight framework to effectively use the distributed object technology and obtain high quality outcome from it. All the discussions in this paper are based on the experience gained to investigate, develop or apply during this project. The physical configuration of this system is shown in Figure 12 below.

AIX4.2 was used for the AP server to process the business logic, and the transaction control was processed by the MQ functions used for the UDB and the queuing control. TP Monitor was not used. All the communication was consolidated into CORBA, including ten NT servers to process Excel macros on the server side. Clients were all Java applets that ran on either Netscape or Microsoft browser, and would access servers via Gatekeeper as described earlier. The performance overhead to pass through Gatekeeper was insignificant. There was no difference either, when compared in the file transfer performance. The network was the bottleneck, as was expected.

As was described previously, this system had a module distribution function, which was implemented as a signed applet since Java applets would write to local disks. The application design was pursued along with what was described in this paper, based mainly on the access control and the factory.
Performance

As for performance, since the IDL design was carried out step-by-step, based on previous performance measurements each time, there was no deviation from presumptions without any troubles. In the case of bulk transfer, in particular, performance almost equivalent with the line speed could be obtained. The Japanese translation class resided in the server that would perform the server access with a fine granularity for each translation. There was some room left for further tuning in areas such as the listing of Japanese and display speed to the spreadsheet image.

For reference, we conducted a comparison between the transfer time required for object reference and the time required when the contents are transferred as an octet string. In this comparison, the same data contents are transferred in three different modes as follows:

- Reference transfer of an object that has four byte strings of 1 KB (class)
- Transfer of the sequence of the same class above (class string)
- Transfer the same data in an octet sequence (byte string)

The measurement was performed on Ethernet. In the case of byte string transfer, almost the same speed is obtained as the FTP. When a remote line is used, the speed is almost the same as the line speed. C++ modules were used in the measurement for both the client and the server, and the result was about the same when Java applets were used for the client.

This comparison does not imply which method is the best; yet, it indicates several interesting results:

- The transfer by object reference takes about 10 times more time than the byte string transfer in this case, because it transfers all the contents.
- The transfer by class string does not improve the transfer time because it simply sends the sequence of the object reference so that the number of requests for the byte string is not changed after all.
- The turnaround of the request to the server is the factor to affect the performance; which means, the performance would be degraded as the number of requests increases.
- It is possible to forecast the number of requests from the IDL and the logic; therefore, the performance can perhaps be preestimated with a significant accuracy by thorough investigation.

6. Concluding Remarks

This paper has discussed the adaptability of CORBA architecture for the Web environment together with an example of its application to an enterprise system.

While there still remain some items that were not clarified with the experience in the last project, the issues in applying IIOP protocols to remote lines, as well as the problems unique to distributed object, were clarified and it was proved that they can be solved. Following are some of the items to be challenged hereafter, and applied to real systems:

- Applying to Java server (C++ server was used this time)
- Reuse and framework
Development of design patterns

Improvement of development efficiency and quality

The issues discussed in this paper were not necessarily initiated and developed by the author alone; rather, most of them are the results of the close cooperation with project members, customers, and other IBM support staffs. An acknowledgment goes to them for their support.

References


The names of products and companies used in this paper are registered trademarks of each of those companies.

Author profile

Hiroshi Kashima

IBM Japan Systems Engineering Co., Ltd.

Mr. Hiroshi Kashima is an I/T specialist - OLTP Products. After he joined IBM Japan in 1985, he spent most of his SE career in the technical support organization (Systems Center) as a Network and AIX expert. Since the beginning of 1996, he has worked with a banking customer who has successfully implemented a large system utilizing Java, C++ and CORBA technologies. Since the beginning of 1999, he has been a group leader of the AIX/6000 and Linux technical support group. He received his Bachelor of Science from Waseda University in 1985.