RMI와 CORBA 환경하의 분산 액티브 객체의 설계 및 구현에 대한 비교 분석

이 도 학[†] · 김 식[††] · 현 무 용[†††]

요 약

분산 프로그래밍은 분산 커뮤니케이션에 대한 언어적 지원을 기반으로 상당히 단순화 될 수 있다. 현재, 많은 웹 브라우저들은 다양한 형태의 액티브 객체들을 제공하고 있으며, 그 수와 유형은 빠른 속도의 증가 추세에 있다. 자바 애플릿은 널리 알려진 웹 브라우저 관련 액티브 객체중의 하나이다. 이 논문은 인터넷 상에 분산되어 있으면서 서로 정보를 교환할 수 있는 분산 액티브 객체의 구현에 관하여 기술한다. 분산 액티브 객체를 구현함에 있어서, 접근방식이 다르고 상호 호환성이 결여된 주요한 두 프로그래밍 환경은 RMI와 CORBA IDL 방식이다. 분산 액티브 객체의 구현상 쟁점들을 명확하게 하기 위해서, RMI 메커니즘을 채택한 HORB와 CORBA를 채택한 OrbixWeb2.0.1 환경 하에서 하나의 어플리케이션 프로그램을 각각 구현하였다. 분산 객체 사이의 바인딩, 상속성, 다형성, 객체의 전달, 콜백은 구현상 중요한 쟁점들이었다. 실험결과는 분산 액티브 객체를 구현하는데 있어서 작은 차이가 분산 어플리케이션의 구성에 상당한 영향을 미칠 수 있음을 보여 주었다. 두 프로그래밍 환경 하에서 구현된 어플리케이션 간의 비교는 각각의 환경에서 구현된 어플리케이션 사이의 상호 변환 시스템을 구축하기 위한 기초 연구가 될 것이다.

Comparison of Design and Implementation for Distributed Active Objects based on RMI and CORBA environment

Dohak Lee[†] · Shik Kim[††] · Muyong Hyun[†††]

ABSTRACT

Distributed programming can be greatly simplified by language support for distributed communication. Many web-browsers now offer some form of active objects and the number and types of them are growing daily in interesting and innovative ways. Java applets are well known as one kind of active object related to web-browser. This paper focuses on distributed active objects which is one kind of active objects that can communicate with other active objects located on different machines across the Internet. Java RMI and CORBA IDL are two major programming environments for distributed active objects which are non compatible with each other. To make discussion concrete, we introduce a single application as implemented on two environments: the HORB, adopting RMI mechanism, and the OrbixWeb2.0.1, adopting CORBA specification, respectively. Binding, inheritance, polymorphism, object passing and callbacks across the machine boundary on distributed programming environments are issued. The results show that some differences in the implementation of distributed active

† 정 회 원: 세명대학교 대학원 전산정보학과
†† 정 회 원: 세명대학교 전산정보학부 교수
††† 정 회 원: 대원전문대학 전자계산학과 교수
논문접수: 1997년 8월 13일, 심사완료: 1997년 10월 8일

objects can have a significant impact on how distributed applications are structured. The comparison between two implementations on the programming environments will be the basis for building the translation system between HORB to OrbixWeb and vice versa.

1. Introduction

In distributed computing, people have used a "socket" so far to communicate between two or more separate machines. Socket provides programs a read/write interface of network protocols such as TCP/IP. Since such programming style is too painful, many distributed programming languages have been studied [1, 2, 3, 4, 5, 6]. When we write a widely usable network program, we have to consider two important factors, portability and interoperability. One of the most exciting recent developments in Web-browser technology is *active objects*, where the browser downloads a program, executes it, and displays the program's user interface in a Web page. Sun's HotJava browser with Java applets pioneered active objects, showing Web pages with a wide range of content, from bouncing ball to spreadsheets to simulated science experiments. Most browsers now offer some form of active objects, written in a variety of languages.

This paper focuses on how to implement *distributed active objects*[7], across the machine boundary, that is, active object that can communicate with other active objects located on different machines across the Internet. High-level support for distributed computation makes it easy to write groupware, computer-supported cooperative work(CSCW) applications, and multiplayer games as active objects.

Common Object Request Broker Architecture (CORBA) and Java, which are two of major object models, introduce a different approach to distributed computing. CORBA provides an infrastructure which enables invocations of operations on objects located anywhere on a network as they were local to the application using them. Java introduces platform-independent, low-level code, which, when integrated with World Wide Web protocols and browsers, results in applets which are known as active objects.

These two object models converge when a mapping is defined from CORBA's interface definition language, Object Management Group Interface Definition Language(OMG IDL), to Java. When combined with a run-time system which supports this language mapping, the result is a Java Object Request Broker(Java ORB)[2, 3, 8]. An alternative approach enabling the invocation of methods on remote Java objects is provided by a mechanism called Remote Method Invocation(RMI)[4, 5]. The idea is to create objects whose methods can be invoked from another Java virtual machine. This approach provides remote procedure call(RPC) mechanisms for Java objects[6].

RMI mechanisms allow the optimization of protocols for communication between Java classes. However, The main advantage of the CORBA IDL approach over RMI is that it supports multiple programming languages. As a point of active objects, Java is a more friendly language to use than COBOL, C, or C++ and a good candidate to replace those languages in commercial software development. Java applications can access those legacy applications if they are wrapped in a Java object wrapper or CORBA object, using the most appropriate language binding[1].

In this paper, firstly we examine some of the fundamental design and implementation issues in distributed active objects which are written in RMI mechanism and CORBA IDL environment respectively, from the point of their impact on distributed applications. Our goal is not to praise or criticize either of two programming systems, but to use the comparison to expose design issues in distributed active objects. Secondly, we study the basis for translation system between two major non-compatible distributed programming environments: Java extension with RMI mechanism.

Java with CORBA IDL.

To make our discussion more concrete, we introduce an example application, called *WB_DAO(White-Board based on Distributed Active Object)*, as implemented in two distributed programming environments :HORB and OrbixWeb2.0.1 respectively. WB_DAO is a distributed program that manages 'whiteboards', which are windows that several workstation users can write or draw on simultaneously, with each seeing an up-to-date image of the whiteboard's state on his or her screen. HORB[4] is an representative example of RMI mechanisms and OrbixWeb2.0.1[3] is one of CORBA IDL whose language support is limited to Java language.

The scope of the paper is as follows. In Section 2, Distributed programming environments, HORB and OrbixWeb, are reviewed. WB_DAO, one kind of realistic application, is described how to design and implement on these environments respectively. In Section 3, we examine WB_DAO from the perspective of several issues:binding, remote object creation and connection, inheritance, object passing, polymorphism and callbacks across the machine boundary. Our objective is simply to use HORB and OrbixWeb to help us examine the influence of different environments and their implementations on distributed programming. Section 4 summarizes our discussion.

2. WB_DAO as evaluation experiments

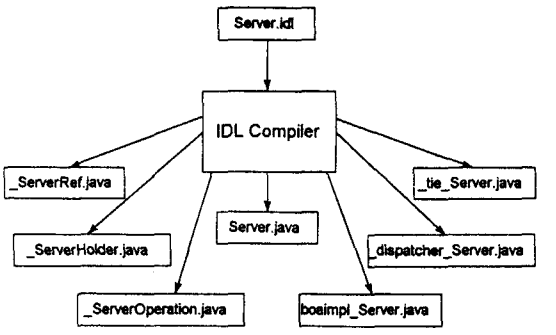
2.1 Programming environments for distributed active objects

Every distributed programming system, whether explicitly or implicitly, motivates a specific style of programming. In this section, we first review the distributed programming environments, which have become the basis for many distributed active objects.

The Object Management Group's CORBA(OMG CORBA) is a standardized specification for the creation of distributed, object-oriented software systems. This specification defines the functionality of an Object Request Broker(ORB). An ORB is a software which allows developers to define objects which can be accessed across a network through clearly defined, high-level interfaces. The Interface Definition Language(IDL) is a standard language, defined by the OMG, for defining such interfaces. It provides a universal notation for specifying APIs. IDL supports library function interfaces just as well as distributed objects across a network[9].

Top six commercial CORBA ORB products are IONA's Orbix, IBM's SOM, Digital's ObjectBroker, Sun's DOE, HP's ORB Plus, and Expersoft's XShell. OrbixWeb implements Orbix, IONA's full implementation of the CORBA specification, in the Java language. In this paper, the objective is to examine the difference of programming style between CORBA IDL and RMI mechanism for distributed active objects. OrbixWeb2.0.1 is selected for CORBA programming environments because RMI mechanism is dedicated to Java.

Figure 1 shows seven Java source files generated by IDL compiler. Each generated file contains a Java class or interface which serves a specific role in application development.



(Fig. 1) Files generated by OrbixWeb IDL compiler

_ServerRef is a Java interface which defines the Java client view of the IDL interface. Server is a Java class which implements the methods defined in interface _ServerRef and provides functionality which

allows client method invocation to be forwarded to a server. _ServerHolder is a Java class which defines a Holder type for class Server. This is required for passing Server objects as inout or out parameter to and from IDL operations. _ServerOperations is Java interface which maps the attributes and operations of the IDL definition to Java methods. _boaimpl_Server and _tie_Server are Java classes which allow server-side developers to implement the Server interface using two techniques available in OrbixWeb: BOAImpl and TIE approach. _dispatcher_Server is a Java class internally by OrbixWeb to dispatch incoming server requests to implementation objects.

An alternative approach enabling the invocation of methods on remote Java object is provided by a mechanism called RMI. The idea is to create objects whose methods can be invoked from another Java virtual machine. Examples of RMI mechanism are HORB and JavaSoft's RMI. In both cases, stub and skeleton classes are directly generated from a Java class identified as remote[5, 10]. HORB is a Java ORB that extends Java for distributed active objects. The HORB package consists of the HORBC compiler and the ORB runtime. To ensure operability on existing Java environments, the compiler and ORB is entirely written using Java and HORB, and execution is possible on the nonmodified Java interpreter. All of Java's features can be accessed from an HORB program[4].

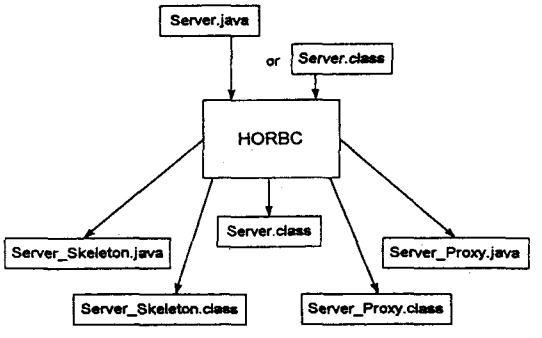
The HORBC compiler creates a proxy class and a skeleton class which includes the object code and stub of the class from the program which defines the Java class. Figure 2 shows Classes generated by the HORBC compiler.

HORB is Java ORB that extends Java for distributed object oriented computing, whereas, CORBA IDL is transparent to CORBA ORB in OrbixWeb2.0.1 even though it is also provide for distributed active objects. These different approaches have significant influences on the programming style.

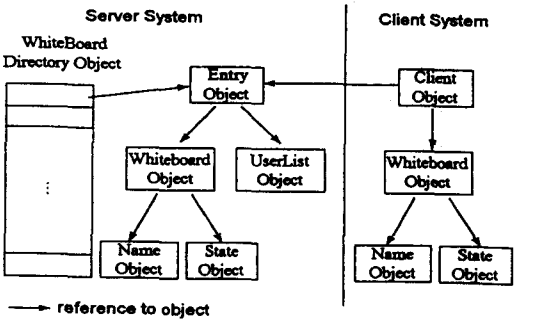
2.2 Distributed program for Whiteboard

To make our discussion more concrete, we introduce an example application as evaluation experiments based on HORB and OrbixWeb2.0.1 environments respectively. Conventional object provides three key properties: encapsulation, inheritance, polymorphism. Henry M. Levy[11] demonstrated that implementation of 'whiteboard' is a good model for examining these properties across the machine boundary.

Figure 3 shows the structure of WB_DAO. In this implementation, all of the data structures are implemented as distributed active objects. Furthermore, the system is organized in three somewhat separate parts. On the left side of figure 3 is a whiteboard directory object. This object maintains a list with entries for whiteboard that are currently in use. Each entry object, as shown in the centre of the figure, consists of references to a whiteboard object and its



(Fig. 2) Classes generated by HORBC compiler



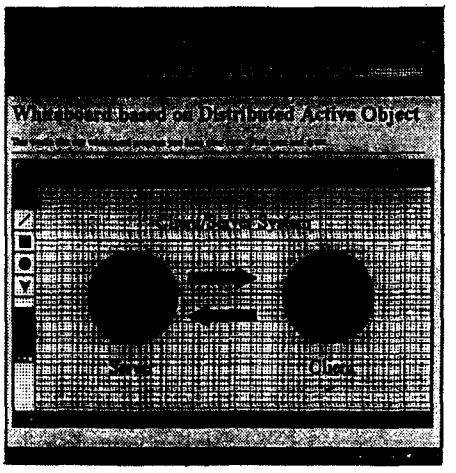
(Fig. 3) The structure of WB_DAO system

associated userlist object. Each whiteboard object has references to its name and state objects.

The right-hand side of figure 3 shows the structure of a distributed program on the client node. Each client runs a copy of the client object, which provides the client interface. The client object has a reference to the entry in the whiteboard directory's list that contains the whiteboard it is using, and a reference to a whiteboard object that is the client's local copy.

A user opens whiteboard by specifying the whiteboard's name. The client object sends this name, together with a self reference, to the whiteboard directory. The directory examines its list for an entry with the named whiteboard. If there is an entry with a whiteboard of that name in the list, it will reply with a reference to that entry. Otherwise, it will create a new entry for a whiteboard of that name and return a reference to that entry. In either case, the directory adds a reference for the client to the whiteboard's userlist, and returns a whiteboard reference to the client for the client's use.

Figure 4 shows web based WB_DAO system running on the client side. Client object, implemented by distributed active objects, communicates with other active objects located on the server across the Internet. Whenever a client draws some image to a whiteboard, it modifies the local whiteboard's state, along with the image on the user's display. The client then modifies the whiteboard object whose reference it received from the directory on the server. It does this by invoking that whiteboard object on the server. That whiteboard object then notifies the other clients of the changes by using references on the userlist to contact them.



(Fig. 4) Web based WB_DAO system

3. Comparison of distributed active objects in programming environments

There are two interface implementation approaches in OrbixWeb: BOA(Basic Object Adapter) and TIE approach. The TIE approach can be very useful when using a pre-existing Java class as a implementation class for an IDL interface. This is why WB_DAO was implemented by TIE approach. We learn some different looks of WB_DAO on HORB and OrbixWeb programming environments. These differences result from embedded attributes of RMI mechanisms and CORBA specification. This section presents some of issues in the design and implementation of distributed active objects for WB_DAO in two programming environments. Binding, remote object creation and connection, inheritance, object passing, polymorphism, callbacks across the machine boundary are issued from embedded attributes of RMI and CORBA specification which are based on object-oriented computing. Java codes in design issues are compared with each other and some possiblities to translate them between two programming environments are described.

3.1 Binding

Binding is the process of establishing communication between two parties. Through binding, the parties determine the addresses of their partners and the protocols to be used for the communication. In a network, there are various choices for how and when binding is made, and these choices affect both the performance

and flexibility of the application. Both programming environments have chosen dynamic binding, which permits flexibility. Here is an example of binding client object to remote server object in WB_DAO. In HORB, a remote server class is written as follows:

```
class Server
{
    Entry_Proxy openWhiteboard(String name, String userid)
    {
        Entry_Proxy entry;
        ...
    }
}
```

The above class is exactly the same as Java class, even though Entry_Proxy class is generated by HORBC compiler. There is no need to add anything to the Java code. Programs when binding to a Server object remotely on an HORB process which operates on the machine of galaxy.semyung.ac.kr are written like this:

```
class Client
{
    public static void main(String arglist[])
    {
        HorbURL url=new HorbURL
                    ("horb://galaxy.semyung.ac.kr/");
        Server_Proxy server=new Server_Proxy(url);
        server.Server();// remote object creation
    }
}
```

The proxy class of Server is Server_Proxy. The instance server is a remote object reference. In the above example, TCP/IP connection is established when the Server and remote object are created.

In OrbixWeb, the first step in writing an distributed program is to define the interfaces to the application objects using IDL. A part of interface to our WB-DAO can be defined as follows.

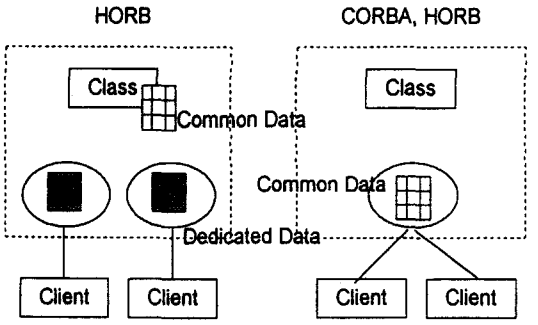
```
// WB-DAO.idl
interface Server {
    ...
    Entry openWhiteboard(in String name, in String userid);
    ...
}
interface Entry {
    ...
}
```

When Server is created, it is assumed that its name is assigned to 'WbSrv' by programmer. We find that _EntryRef and _ServerRef generated by IDL compiler is straightforward to Entry_Proxy and Server_Proxy generated by HORBC compiler in their functionality at source code level. As the method for binding, _bind() showed following class, is mapped to the code for "Server_Proxy("horb://galaxy.semyung.ac.kr/")" showed above class. Binding is established by _bind() method call in OrbixWeb, whereas, that is provided by remote object instantiation in HORB. The binding by Naming Service in OrbixWeb is beyond the scope of our study because HORB does not supports this property.

```
class ServerImpl implements _ServerOperations
{
    _EntryRef openWhiteboard(String name, String userid)
    {
        _EntryRef entry;
        ...
    }
}
class Client
{
    _ServerRef server;
    ...
    server=Server._bind(":WbSrv", "galaxy.semyung.ac.kr");
    _EntryRef entry=server.openWhiteboard(...);
```

3.2 Remote Object creation and connection

Figure 5 illustrates the difference of remote object creation(left) and remote object connection(right). In case of the remote object connection model, all clients share one remote object. While the clients can share data in the instance variables of the remote object, there is no room to keep client's dedicated data such like user names. Such data must be transferred in each remote method call or be stored in a special data storage such like hashtables. In case of the remote object creation model, a remote object is created for each client. Thus, a remote object can have client's specific data. Class variable of the remote object is used to share between clients. HORB supports both models. OrbixWeb supports only the remote object connection model. We find that it is possible to happen side effect, for example, object initialization when codes with HORB map to those of OrbixWeb.



(Fig. 5) Remote object creation and Remote object connection.

3.3 Inheritance

Inheritance is one of key properties in object-oriented programming. Because basic notion of HORB is the extension of Java for distributed active objects, HORB fully supports inheritance across machine boundary. OrbixWeb supports only inheritance specified in IDL for their distributed active objects. Implementation classes corresponding to interfaces specified in IDL should also maintain their inheritance. Following codes are the example of inheritance in OrbixWeb.

```
// idl file
interface graphicObject {
    ...
};
interface graphicLineObject extends graphicObject {
    ...
};

// implementation class for graphicLineObject
class graphicLineObjectImpl extends graphicObjectImpl
                    implements _graphicLineObjectOperations
{
    ...
}
```

WB_DAO shows that the classes specified with remote method call in HORB should be denoted in IDL for inheritance across the machine boundary. The other parts of implementation for inheritance are mapped to each other because both environments support inheritance in OOP except for multiple inheritance.

3.4 Polymorphism

Polymorphism is another key property in OOP. Polymorphism is a high-brow way of saying that the same method can do different things, depending on the class that implements it. WB_DAO would be expected to invoke the draw operation on just about any graphic object-a square, or circle, or line, or whatever-and have it draw itself on a screen. However, the client invoking the same operation on a set of objects actually results in different things happening, since each object has its own methods, for example, a square needs drawing position, height and width, compared as a circle needs drawing position and radius, in drawing operation.

HORB fully supports polymorphism between the client and server site. The object type is correctly transferred even if it is casted. However, OrbixWeb does not provide any polymorphic property since CORBA 2.0 specification is not still defined this property by OMG[2]. The OrbixWeb version of WB_DAO shows the programming overhead to implement this property using conventional control statements such as ***if*** or ***switch***. Following codes show the example of how to implement it in OrbixWeb.

```
public void appendNode(_graphicObject grobj)
{
    switch (grobj.getid())
    {
        case Line:
            line = graphicObjectLine._narrow();
            break;
        case Rectangle:
            rect = graphicObjectRectangle._narrow();
            break;
        ...
    }
}
```

3.5 Object passing

In HORB, an object can be passed by value or reference for arguments or return value of the remote method. If the object for transferring is a local object to a remote object on another site, then passing by value is used. If the object for transferring is a remote object on another site to a remote object on the other site, then passing by reference is used. However, OrbixWeb only provides passing by reference between the objects located on different sites. In WB_DAO, the client object first modifies the local copy of the whiteboard's state and the image on the user's display when a user writes to a whiteboard. This is mainly a performance issue: it is most important that the drawing user have real-time feedback on his changes. In such a situation, the OrbixWeb version of WB_DAO shows additional programming needs that the local copy of referenced remote objects should be maintained explicitly. Following class in OrbixWeb show the example of local copy by programmer.

```
class ServerImpl implements _ServerOperations
{
    ...
    void changeStatus(_graphicLineObjectRef g)
    {
        // creation of local copy for referenced remote object
        _graphicLineObjectRef localObj =
                    new _tie_graphicLineObject
                    (new graphicLineObjectImpl());
        localObj.set_currentColor(g.get_currentColor());
        ...
    }
}
```

3.6 Callbacks

A callback is a mechanism that enables a server to call methods of client and necessary to design an application as Client/Server model. WB_DAO shows that the client sends the changes to the server, which updates the global data structures and propagates the changes to other users of that whiteboard.

WB_DAO is implemented successfully on both environments. In HORB, the client object is registered itself to the HORB server when it is initiated and is need to be called back. Then, it is connected to a server object and sends a signal to the server for invitation. At the server side, the server object accepts the invitation when the preparation for callback in the client is finished. After preparation for callback between the client and server, the sever invokes a method on the client object, so called callback, in the identical to the conventional method calls. Following class shows how to implement it on both sides in HORB.

```
class Client
{
```

```
    public static void main()
    {
        Server_Proxy server;
        ...
        // register client object to the HORB server
        HORBServer.registerObject(...);

        // send a signal to invite the server
        server._invite(...);
    }
    public void AppendNode(...) { ... }
    ...
}
class Server
{
    public void Append(...)
    {
        Client_Proxy client;
        ...
        Server_Skeleton skl = (Server_Skeleton)
                              HORBServer.getSkeleton();
        // accept the signal of preparation
        client = (Client_Proxy)skl. accept(0);
        ...
        // call back client's method
        client.AppendNode(...);
    }
    ...
}
```

In OrbixWeb, the client object creates additional thread which is dedicated to handling callback events. The server object receives an object reference from client. When this object reference enters the server address space, a proxy for the client object is created. It is this proxy which the server will use to call back to the client. Following class demonstrates how to implement it in OrbixWeb.

In HORB, there is a facility, provided by programming environment, to support callback. Whereas, user programmer should create additional thread in the client object and pass the client object reference to the server, which takes responsible for handling callback in OrbixWeb. We find that the mapping of callback between two environments is too complex and further study is required to build the translation system.

```
class ClientImpl implements _ClientRef
{
    public static void main()
    {
        _ServerRef server;
        ...
        Invoking of thread for handling callback event;
        ...
        // create reference of client object
        _ClientRef client = new _tie_Client
                            (new ClientImpl());
        Binding to Server object;
        // pass client object reference to server
        server.Append(client);
    }
    public void AppendNode(...) { ... }
    ...
}
class ServerImpl implememts _ServerOperations
{
    public void Append(_ClientRef client, ...)
    {
        ...
        // callback client's method
        client.AppendNode(...);
    }
    ...
}
```

4. Conclusion

The primary objective of this paper is to examine design and implementation issues in the construction of distributed active objects. Secondary of it is to study the basis for the translation system between two

major distributed programming environments: Java extension with RMI mechanism, Java with CORBA specification. To make our discussion concrete, we examine a single application, called WB_DAO, as implemented in two environments: HORB, the representative of RMI mechanism[4, 12, 13], OrbixWeb2.0.1, commercial version of CORBA IDL for Java[1, 3].

We present some of issues in the design and implementation of distributed active objects for WB_DAO in two environments. Binding, remote object creation and connection, inheritance, object passing, polymorphism, callbacks across the machine boundary are issued from embedded attributes of RMI and CORBA.

There are three essential properties in distributed active objects: inheritance, polymorphism, encapsulation. In both environments, inheritance across the network is fully supported. However, additional programming overhead is needed since OrbixWeb does not provide the polymorphism compared as HORB does. As the encapsulation across machine boundary, programming environment should support binding, object creation/connection and object passing mechanism to implement distributed active objects. OrbixWeb provides Naming Service for binding, whereas HORB provides object passing by copy between remote objects.
DAO demonstrates these properties can have a significant impact on how to design and implement distributed active objects.

RMI is an interesting approach for small and medium-sized applications completely implemented in Java. However, applications that require the integration of legacy components or the use of a particular programming language need a CORBA solution. CORBA IDL's separation of interface definitions from implementations and mappings to many programming languages, in combination with CORBA services, provide better support for large-scale applications. Even though CORBA IDL provides the integration of legacy components, there is no room to integrate RMI and CORBA IDL[2, 8].

This paper describes some possibilities to translate Java codes implemented in the RMI environment to those implemented in the CORBA IDL. As a point of view of active objects, Java is a more friendly language to use than COBOL, C, or C++ and a good candidate to replace those languages in commercial software development. Java applications can access those legacy applications if they are wrapped in a Java object wrapper or CORBA object, using the most appropriate language binding. We find the mapping of the inheritance and binding without Naming Service in both environments is straightforward to each other. In mapping of polymorphism and object passing by copy, additional programming efforts are required.

WB_DAO shows the callback mechanism is necessary to design an application as a Client/Sever model. In HORB, there is a facility, provided by programming environment, to support callback. Whereas, programmer should create additional thread in the client object and pass the client object reference to the server, which takes responsible for handling callback in OrbixWeb. We find that the mapping of callback between two environments is too complex and further study for callback is needed for the translation system.

The comparison between two implementations on the programming environments will be the basis for building the translation system from HORB to OrbixWeb and vice versa. Indeed, the authors are currently working on developing a translation system each other.

References

[1] Andreas Vogel and Keith Duddy, 'Java Programming with CORBA', John Wiley & Sons, 1997.

[2] John Siegel, 'CORBA Fundamentals and Programming', John Wiley & Sons, 1996.

[3] IONA Technologies, 'OrbixWeb programming guide', IONA Technologies Ltd., 1996.

[4] Hirano Satoshi, HORB User's guide 〈URL: http:

//www.etl.go.jp/openlab/horb〉,March, 1996.

[5] Sun Microsystems, JDK 1.1 Documentation 〈URL: http://java.sun.com/〉

[6] Shik Kim, Muyong Hyun and Sangjo Lee, "Transparent Treatments of Remote Pointers Using IPC Primitive in RPC Systems", *Proceedings of HPCA-3*, February, 1997.

[7] Marc H. Brown and Marc A. Najork, "Distributed Active Objects", *Computer Networks and ISDN Systems*, Vol. 28, pp. 1037-1052, 1996.

[8] Robert Orfali and Dan Harkey, "Client/Server Programming with Java and CORBA", John Wiley & Sons, 1997.

[9] M. Steinder, A. Uszok and K. Zielinski, "A Framework for Inter-ORB Request Level Bridge Construction", *IFIP/IEEE International Conference on Distributed Platforms*, pp. 86-99, 1996.

[10] Eric Evans and Daniel Rogers, "Using Java Applets and CORBA for Multiuser Distributed Applications", *IEEE Internet Computing*, Vol. 1 (3), May, 1997.

[11] Henry M. Levy and Ewan D. Tempero, "Modules, Objects and Distributed Programming: Issues in RPC and Remote Object Invocation", *Software-Practice and Experience*, Vol. 21(1), pp. 77-90, January, 1991.

[12] Ann Wollrath, Jim Waldo, and Roger Riggs, "Java-Centric Distributed Computing", *IEEE Micro*, Vol. 17(2), May, 1997.

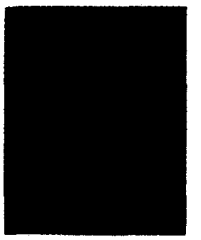
[13] Yamanaka A., Nakajima S., and Tomono M., "A HORB-Based Network Management System", ICODP'97, 1997.

이 도 학

1996년 세명대학교 전자계산학과 졸업(이학사)

1997년~현재 세명대학교 대학원 전산정보학과 석사과정

관심분야: 네트웍컴퓨팅, 분산컴퓨팅, 분산언어



김 식

1979년 경북대학교 컴퓨터공학과 졸업(공학사)

1979년~1988년 국방과학연구소 선임연구원

1990년 Texas A&M 컴퓨터공학과 졸업(석사)

1991년 금성정밀 항공전자 연구실장

1995년 경북대학교 대학원 컴퓨터공학과 박사 수료

1993년~현재 세명대학교 전산정보학부 교수

관심분야: 네트웍컴퓨팅, 분산컴퓨팅, 분산언어

현 무 용

1993년 경북대학교 컴퓨터공학과 졸업(공학사)

1995년 경북대학교 대학원 컴퓨터공학과(석사)

1995년~1996년 대원전문대학 전자계산학과 전임강사

1997년~현재 대원전문대학 전자계산학과 조교수

관심분야: 네트웍컴퓨팅, 분산컴퓨팅, 분산언어