프로그래밍 투명성을 지원하는 분산 프로그래밍 도구의 설계

(A Design of Distributed Programing Tool in support of Programming Transparency)

이 상 윤 † 김 승 호 ††
(Sang-Yun Lee) (Sung-Ho Kim)

요 약 분산 컴퓨팅 환경에 적용해야할 응용 소프트웨어의 비중이 증가함에 따라, 이를 지원하기 위한 다양한 형태의 분산 프로그래밍 도구들이 제안되어 있다. 그러나, 이러한 도구를 이용하여 분산처리 소프트웨어를 작성하기 위해서는 분산 프로그래밍 도구가 요구하는 부가적인 프로그래밍 지식을 숙지하여야 한다. 부가적인 지식 없이 분산처리를 수행하는 소프트웨어를 개발할 수 있다면, 분산처리에 대한 개발자의 부담을 줄여서, 개발하고자 하는 소프트웨어의 자체 기능에 더욱 집중할 수 있다. 본 논문에서는 새로운 분산 프로그래밍 도구를 제안하고, 이름을 TORB(Transparent Object Request Broker)라고 명명하였다. TORB를 이용하면, 프로그래밍 투명성의 지원을 통하여, 자바로 작성하는 분산처리 소프트웨어를 상대적으로 쉽게 개발할 수 있고, 후처리를 통하여 TORB가 제공하는 분산처리 환경에서 수행될 수 있다.

키워드 : 프로그래밍 투명성, 분산 프로그래밍, 자바 객체 대행자

Abstract According to the increasing demand of application software that must be applied to the distributed computing environment, the various tools are proposed to write distributed softwares. But, if using these tools, programmers have to know the usage of each tool requisite for writing distributed softwares. If programmers can write distributed software without additional knowledge, they can get better concentration of the functions of software itself to develop, because it reduces burden for distributed programming. In this paper, we introduce new distributed programming tool, named TORB(Transparent Object Request Broker). With TORB, thanks to programming transparency that is supported by TORB, we can write the distributed software with java more easily. After postprocessing, this software can run in the distributed processing environment that is supported by TORB.

Key words : programming transparency, distributed programming, Java ORB

1. 서 론

분산 컴퓨팅 환경에 적용해야할 응용 프로그램이 점점 많은 비중을 차지하게 됨에 따라, 이를 지원하기 위한 다양한 형태의 분산 프로그래밍 도구들이 제안되어 왔다. 객체지향 디자인기법을 적용한 분산 프로그램은, 이식성과 재사용성을 통하여 복잡한 시스템 구축에 적합하다는 장점 때문에, 널리 사용되고 있는 분산 소프트웨어 개발기법 중의 하나이다[1]. 이 기법은 소프트웨어의 완벽한 캡슐화를 보장하고, 분산된 객체간의 메시지 교환을 위한 기본 메커니즘인 원격 메쏘드 호출을 통하여 시스템에서 요구되는 통신 기능을 자연스럽게 구현할 수 있으므로 분산 시스템으로 자연스럽게 적용된다. 객체지향 분산 시스템을 지원하기 위한 방안으로써, CORBA, DCOM, DSOM, JAVA RMI, HORB등 여러 가지 도구들이 제안되어 있다[2-6].

이러한 도구를 이용하여 분산처리 소프트웨어를 작성하기 위해서는 분산 프로그래밍 도구가 요구하는 부가적인 프로그래밍 지식을 숙지하여야 한다. 부가적인 지식 없이 분산처리를 수행하는 소프트웨어를 개발할 수 있다면, 분산처리에 대한 개발자의 부담을 줄여서, 개발하는 소프트웨어의 자체 기능에 더욱 집중할 수 있다. 분산 프로그래밍 도구가 요구하는 부가적인 지식은 소

† 정 회 원 : 대원과학대학 컴퓨터정보처리과 교수
lsy@sam.daewon.ac.kr
†† 종신회원 : 경북대학교 컴퓨터공학과 교수
shkim@knu.ac.kr
논문접수 : 2003년 7월 21일
심사완료 : 2004년 1월 30일

프트웨어의 자체기능과 상관없는 부담이며, 이러한 부담은 기존의 분산 프로그래밍 도구가 프로그래밍 투명성을 만족할 만큼 지원하지 못하기 때문에 나타나는 오버헤드이다. 프로그래밍 투명성이란 프로그래머가 분산환경에서 동작하는 프로그램을 개발할 때, 로컬환경에서 동작하는 프로그램을 개발하는 것과 동일한 방법을 적용할 수 있도록 지원하는 것이다.

본 논문에서는 프로그래밍 투명성을 지원하는 새로운 분산프로그래밍 도구를 제안하고, 그 이름을 TORB (Transparent Object Request Broker)라고 명명하였다. TORB를 이용하면, 프로그래밍 투명성을 충분히 보장하는 분산 프로그램을 자바로 작성할 수 있고, 이를 분산처리 환경에서 동작시킬 수 있다. 프로그래머는 작성하고 있는 프로그램이 TORB의 지원을 받아서 분산처리 환경에서 실행될 것이라는 사실만 염두에 두고, 일반적인 프로그래밍 방법으로 프로그램을 작성하면 된다. 이렇게 작성된 프로그램은 TORB가 제공하는 도구에 의해 후처리를 거친 후, 분산처리 환경에서 실행된다. 또한, TORB가 지원하는 프로그래밍 투명성의 범위 내에서는 분산환경을 전혀 고려하지 않고 작성된 프로그램도 동일한 후처리를 통하여 분산환경에서 실행될 수 있다.

TORB에서는 분산처리 프로그램의 프로그래밍 투명성을 확보하기 위하여, 분산처리에 대한 부가지식 없이 작성되어 컴파일된 자바클래스로부터 객체 대행자 역할을 수행하는 클래스파일을 생성하고 기존의 클래스파일을 변환하는 후처리를 수행한다. 후처리를 통하여 수정된 자바클래스는 생성된 대행객체와 연동하여 TORB가 제공하는 분산처리 환경에서 동작하며 분산처리 기능을 수행한다.

TORB의 분산처리 환경과 대행객체를 위한 코드는 RMI(Remote Method Invocation) 메커니즘에 기반을 둔 방법으로 구현되어 있으며, 프로그래밍 투명성을 확보하는 분산처리 환경을 제공하기 위하여 다음과 같은 몇 가지 특징을 가진다. 인수전달 매커니즘의 투명성 확보를 위해서 JAVA Serialization 기능을 사용하고 있고, JAVA Reflection 기능을 사용한 동적 호출을 통하여 원격 메쏘드 호출을 수행한다. 또한, 컴파일된 자바클래스로부터 후처리만을 통하여 분산처리 환경에서 분산 기능을 수행하는 수정된 자바 클래스를 생성한다. 분산처리 환경에서 동작하도록 변환하는 작업이 후처리를 통해서 이루어지므로, 프로그램을 작성하는 동안, TORB의 사용법에 대한 지식이 없어도 된다.

본 논문은 프로그래밍 투명성을 충분히 보장하는 분산 프로그래밍 도구를 제안하기 위하여 다음과 같이 구성된다. 서론에 이어 2장에서는 자바와 관련된 분산처리 지원환경의 연구결과를 소개하고 이들이 가지는 프로그래밍 투명성의 문제점을 논의한다. 3장에서는 TORB를 이용하여 작성하는 분산처리 프로그램의 프로그래밍의 투명성과 분산처리 기능의 수행을 위하여 후처리를 적용하는 절차를 소개한다. 4장에서는 TORB가 지원하는 분산처리 수행환경의 구조와 동작 메커니즘을 설명하고, 5장에서는 TORB를 구현하기 위한 핵심적인 사항을 제시한다. 6장에서는 TORB를 이용한 분산처리 프로그램의 설계 및 구현과정을 보이고, 7장에서는 결론 및 향후 연구계획을 제시한다.

2. 자바와 관련된 분산처리 지원 환경

자바로 작성된 프로그램을 분산환경에서 실행하도록 하기 위하여 확장된 자바 가상기계를 적용한 cJVM, Java/DSM, JESSICA와 같은 시스템이 개발되어 있다[7-9]. 이러한 시스템들은 분산처리 환경을 통하여 동시에 동작하는 여러 개의 자바 쓰레드가 공유 메모리를 가지는 하나의 병렬컴퓨터에서 실행하는 효과를 제공한다. 또한, 일반적인 방법으로 작성된 자바프로그램을 전혀 수정하지 않고, 이들 가상기계에서 실행할 수 있다. 그러나, 이들의 목적은 분산처리 환경을 이용한 병렬처리이므로, 특정기능을 담당하는 자바객체를 특정 컴퓨터에서 수행하도록 하는, 기능별 분산처리를 수행할 수 없다. 또한, 특별히 고안된 형태의 자바 가상기계에서만 동작한다는 제약을 가진다.

자바를 위한 분산 프로그래밍 지원 시스템으로, CORBA의 규약에 따라 IDL을 정의하고 이에 따라 원격 객체를 접근하도록 고안되어 있는 VisiBroker, OrbixWeb, Java IDL 등이 있으며, 원격 객체의 인터페이스를 정의하고 분산처리 프로그램을 작성하도록 지원하는 Java RMI, Voyager 등의 객체 대행자(Object Request Broker)가 개발되어 있다[5,10]. 이들을 사용하여 분산 프로그램을 작성하기 위해서는 각각의 시스템에 요구하는 부가적인 프로그래밍 지식을 적용하여야 한다. 부가지식을 적용해야하는 이러한 요구사항은 프로그래머로 하여금 개발하는 분산 프로그램의 자체기능에 집중하는 것을 방해한다.

Java Class Broker와 HORB는 이러한 문제를 줄이기 위해서 원격객체에 대한 IDL이나 인터페이스를 정의하지 않고 분산 프로그램을 작성할 수 있도록 개발되어, 상대적으로 쉬운 프로그래밍 방법을 제공한다[6,11]. 그러나, 이들도 일반적인 프로그래밍 방법과는 상당히 다른 방법으로 분산 프로그램을 작성하여야 하고, 분산 프로그래밍을 위하여, 다음과 같이 각각의 도구가 요구하는 방법이 적용되어야 한다.

```
ObjA oA = new ObjA("Hello");
ObjB oB = new ObjB(oA);
oA.DoJob();
```

그림 1 일반적인 자바 프로그램

```
Object[] params1 = {"Hello"};
ObjA oA = (ObjA) objectBroker.create("ObjA", params1);
Object[] params2 = {oA};
ObjB oB = (ObjB) objectBroker.create("ObjB", params2);
oA.DoJob();
```

그림 2 Java Class Broker에 적용할 자바 프로그램

```
ObjA_Proxy oA =
        new ObjA_Proxy("HORB://remote1", "Hello");
ObjB_Proxy oB =
        new ObjB_Proxy("HORB://remote2", oA);
oA.DoJob();
```

그림 3 HORB에 적용할 자바 프로그램

그림 1은 원격객체인 ObjA를 생성하여, 또 다른 원격객체인 ObjB를 생성하기 위한 인수로 전달한 후, 원격객체 ObjA의 DoJob()이라는 메쏘드를 실행하는 프로그램이다. 그림 2는 이러한 역할을 수행하는 프로그램을 Java Class Broker를 이용하여 개발하고자 할 때, 프로그래머가 작성하여야 하는 코드이다. HORB를 이용하여 동일한 역할을 수행하는 프로그램을 개발하기 위해서는 그림 3과 같이 작성하여야 하고, ObjB의 클래스 정의에서는 ObjA_Proxy라는 ObjA의 대행객체를 전달받도록 작성되어야 한다.

자바 언어의 확장을 통한 분산처리 지원 환경으로 JavaParty가 있다[12]. JavaParty는 "remote"라는 키워드를 사용할 수 있도록 자바 언어를 확장하여 분산 프로그래밍을 지원한다. 그러나, 원격객체의 생성 및 접근을 위하여 최소한 "remote"라는 키워드를 사용하여야 하고, 확장된 키워드의 처리를 위하여 별도의 자바 컴파일러가 필요하며, 특별히 고안된 자바 가상기계에서만 실행된다.

레거시 자바 프로그램을 분산환경에서 실행할 수 있도록 변환하고 이를 위한 실행환경을 제공하는 시스템으로 addistant가 있다[13]. Addistant는 분산환경을 고려하지 않고 이미 작성된 자바프로그램을 분산환경에서 동작하도록 지원한다. 이를 위하여 정책파일을 정의하고, 이것을 기준으로 원격 객체가 생성될 호스트를 명시하며, addistant가 지원하는 분산환경에서의 동작을 위하여 자바 클래스의 적재 시간에 분산처리를 위한 바이트 코드의 변환이 이루어진다. 또한, addistant가 지원하는 분산환경을 고려하며 프로그램을 작성하는 경우에는 일정 범위의 프로그래밍 투명성을 지원 받으며 분산처리 프로그램을 작성하는 효과를 나타낸다. 그렇지만, addistant는 분산 프로그래밍을 지원하는 도구가 아니고, 이미 작성된 레거시 자바 프로그램을 분산환경에서 실행하기 위하여 개발된 시스템이므로, 분산처리를 목적으로 새로 작성하는 프로그램에 대한 지원기능은 부족하다[13]. 예를 들면, 원격객체가 생성되어 동작할 호스트가 실행시간에 결정되는 프로그램은 addistant에 적용할 수 없다. 왜냐하면, addistant에서는 정책파일에 의한 정적인 방법으로 원격객체가 동작할 호스트를 명시하고 클래스파일의 적재시간에 바이트 코드의 변환이 수행되기 때문이다.

TORB는 프로그래밍 투명성을 지원하는 분산 프로그래밍 도구이므로 새로 작성하는 분산처리 프로그램에 대한 충분한 기능을 포함한다. 즉, 기존의 분산처리 프로그래밍 도구의 기능을 포함하면서 투명한 방법으로 분산처리 프로그램을 작성할 수 있도록 지원한다. 또한, TORB가 지원하는 프로그래밍 투명성의 범위 내에서는 이미 작성된 레거시 자바 프로그램도 후처리를 통하여 분산환경에서 실행할 수 있다.

3. 분산처리 프로그래밍

자바를 이용하여 분산처리를 수행하는 프로그램을 작성하기 위하여, 프로그래머는 원격시스템에서 생성되고 수행해야할 객체(원격객체)를 정의하고, 원격객체를 접근하는 코드를 작성해야한다. 기존의 분산처리 프로그래밍 도구들은 원격객체를 접근하는 코드를 작성할 수 있도록 지원하기 위하여 복잡한 형태의 코딩 규칙을 요구한다. 더욱이, 하나의 원격객체가 또 다른 원격객체를 접근하거나, 원격 메쏘드를 실행하기 위하여 인수로서 전달되어야 하는 경우에는 더욱 복잡한 코드가 요구된다[2,5,6]. 3장에서는 TORB가 지원하는 프로그래밍 투명성 의해 분산처리 프로그램이 어떻게 작성되어야 하는가 하는 점과 작성된 프로그램을 분산처리 환경에서 실행되도록 변환하는 절차를 설명한다.

3.1 후처리에 의한 프로그래밍 투명성

기존의 분산 프로그래밍 도구와 달리, TORB가 지원하는 분산처리 환경에서 동작할 분산 프로그램을 작성하기 위하여 별도로 요구되는 코드의 규칙(code sequence)은 없다. 즉, TORB를 이용하는 분산처리 프로그램을 작성하기 위해서, 프로그래머 자신이 분산처리 프로그램을 작성하고 있다는 사실과 TORB가 제공하는 프로그래밍 투명성의 범위의 인식만이 추가로 필요할 뿐이다. 왜냐하면, 분산처리를 위한 모든 변환작업이 후처리에 의해 이루어지기 때문이다. 이에 대한 부수적인 효과로, TORB가 지원하는 프로그래밍 투명성의 범위 내에서는, 이미 작성된 레거시 자바 프로그램도 분산처

리를 수행하도록 변환할 수 있다.

3.2 후처리

TORB가 지원하는 분산처리 환경을 고려하면서 작성된 프로그램은 분산처리가 목적이지만 실제로 분산처리 동작을 수행하지 않는다. 이는 아직 분산처리를 위한 아무런 조치가 이루어지지 않았기 때문이다. TORB는 작성된 프로그램을 분산처리 환경에서 수행할 수 있도록 변환하기 위하여 두 가지의 후처리를 수행한다. 하나는 원격 호스트에서 동작할 객체의 대행객체를 생성하는 것이고, 다른 하나는 원격 호스트에서 동작할 객체를 접근(access)하는 프로그램의 모든 코드를 해당 객체의 대행객체를 통해서 접근하도록 변환하는 것이다.

3.2.1 대행객체의 생성

대행객체는 원격 호스트에서 생성되어 동작할 자바객체를 대신해서 로컬 호스트에서 생성되고 동작할 객체이다. 자바의 API중 java.lang.Class를 이용하면, 임의의 자바객체 인스턴스로부터 그 객체에 정의된 모든 속성(메쏘드, 생성자, 멤버데이타 등)을 얻을 수 있다. TORB의 후처리 도구는 원격 호스트에서 동작할 객체의 인스턴스로부터 대행객체를 위한 소스코드를 생성하고 표준 자바 컴파일러를 통해 바이트 코드로 번역된다. 대행객체는 원시객체에 "_Proxy"가 추가된 이름으로 생성되며, 원시객체의 인스턴스로부터 생성하기 때문에, 소스 프로그램을 구할 수 없는 자바객체(third party supported java object)에 대한 대행객체도 생성할 수 있다.

3.2.2 클래스파일 변환

TORB의 후처리 도구에 의해 변환되기 이전의 자바 프로그램(자바 클래스 파일)에는, 원격 객체를 접근하기 위하여 대행객체를 사용하는 바이트 코드가 아니고, 로컬에 존재하는 객체를 접근하는 바이트 코드로 컴파일되어 있다. 변환도구는 변환전의 프로그램에 포함된 바이트 코드 중 원격 호스트에서 동작해야하는 객체를 접근하는 모든 바이트 코드를 해당객체에 대한 대행객체를 사용하는 바이트 코드로 변환한다. 이러한 기능을 위하여 BCEL의 jasmin과 JasminVisitor를 보조도구로 활용한다. Jasmin은 자바 어셈블리 코드를 클래스파일로 생성해 주는 기능을 가지는 자바 어셈블러이고, JasminVisitor는 모든 자바객체로부터 자바 어셈블리 코드를 생성하는 자바 역어셈블러이다[14,15]. 변환작업은 보조도구로 활용되는 JasminVistor에 의해 생성된 자바어셈블리의 코드를 수정하는 것으로 충분하다. 왜냐하면, 수정된 자바어셈블리 코드는 jasmin에 의해 분산실행환경에 적합한 클래스파일(바이트 코드)로 변환될 수 있기 때문이다.

3.2.3 설정파일

TORB의 후처리 도구를 사용하기 위해서는 그림4와 같은 설정파일이 필요한데, 이는 자바로 작성된 분산처리 프로그램의 원격객체들이 생성될 호스트와 원격객체와 관련된 객체의 목록을 명시하기 위한 것이다.

```
#Remote Classes
Third=TORB://remoteA.ac.kr:9000
Server.java=TORB://remoteA.ac.kr:9000
Client.java=TORB://remoteB.ac.kr:9001
Go.java=TORB://self
```

그림 4 설정파일의 예

설정파일의 파일형식은 자바의 java.util.Properties에 적용하는 형식이고, 그림 4와 같이 원격객체의 이름과 URL의 순서쌍으로 구성된다. 객체이름에 ".java"가 부가되면 변환작업을 수행할 때 원시객체에 대한 컴파일 작업이 선행되고, URL을 "TORB://self"로 지정하면 로컬 호스트에서 생성되는 객체이며 원격객체를 접근하는 코드가 포함되어 있다는 의미이다.

4. 분산처리 수행환경

TORB의 후처리 도구에 의해 변환된 프로그램이 분산처리를 수행하며 동작할 환경이 필요하다.

TORB는 그림 5와 같이 정의된 분산처리 수행환경을 제공하며, 표준 자바 가상기계에서 동작하는 서버, 변환된 분산처리 프로그램, 생성된 대행객체 사이의 연계된 작용에 의해 동작한다. 다음은 그림 5에 나타난 세부적

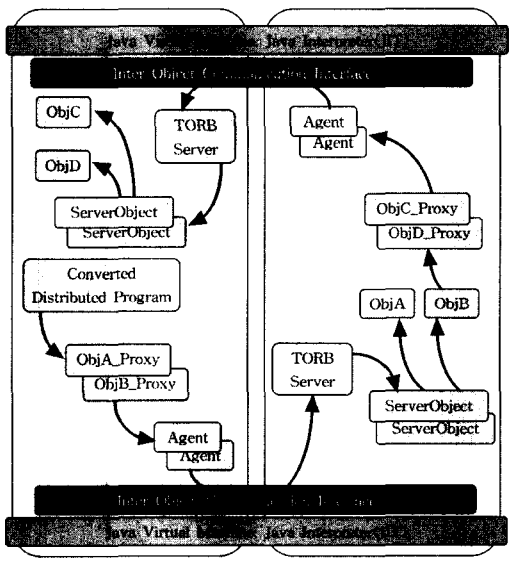

그림 5 TORB의 분산처리 수행환경

인 동작 메커니즘이다.

프로그래머는 원격객체 "ObjA"와 "ObjB"를 사용하는 프로그램을 일반적인 프로그래밍 방법으로 작성하였고, 이는 후처리 과정을 거쳐 "ObjA_Proxy"와 "ObjB_Proxy"를 사용하는 분산프로그램으로 변환된다. 이들은 TORB에 정의된 "Agent"객체의 지원을 받아 원격 컴퓨터의 "TORB Server"에게 적절한 요청을 한다. 원격 컴퓨터의 "TORB Server"는 객체 서비스를 위하여 TORB에 정의된 객체인 "ServerObject"를 통하여 "ObjA"와 "ObjB"에 대한 서비스를 수행하고 "Agent"의 요청에 응답한다. 한편, 원격의 "ObjB"객체는 동일한 방법으로 로컬 컴퓨터의 "ObjC"와 "ObjD"를 접근할 수 있다.

"TORB Server"는 원격 호스트에서 동작해야 하는 객체에 대한 서비스를 지원하는 기능을 수행하므로 작성된 프로그램이 실행을 시작하기 전에 동작 중이어야 한다. 따라서, 분산처리에 참여하는 모든 호스트에 "TORB Server"를 기동하는 조치가 필요하다. TORB에서는 "TORB Server"를 기동하기 위한 두 가지 방법을 제공한다. 하나는 사용자가 명령라인에서 직접 서버를 기동하는 방법이고, 다른 하나는, 변환된 분산처리 프로그램에 "TORB Server"를 중복되지 않게 기동하는 코드를 삽입하여, 작성한 프로그램이 실행을 시작할 때 자동으로 기동하도록 하는 방법이다.

5. TORB의 설계 및 구현방안

5.1 TORB 서버

TORB 서버는 임의의 객체(ObjA)를 원격에서 접근하기 위하여 대행객체(ObjA_Proxy)의 동작에 의해 네트워크를 통하여 전달되어오는 요청을 분석하여 객체(ObjA)에 존재하는 메쏘드(request)를 수행한 후, 그 결과를 원격에 존재하는 대행객체(ObjA_Proxy)에게 네트워크를 통하여 응답한다. 이때, 하나의 대행객체(_Proxy)에 대해서 하나의 서비스객체(ServerObject)가 대응되도록 구현함으로써 여러 개의 원격객체를 동시에 처리할 수 있도록 할 수 있다. Java Reflection 기능을 이용한 메쏘드의 동적호출(Dynamic Invocation)방법을 적용하면, TORB 서버가 임의의 객체(ObjA)에 포함된 메쏘드를 수행하는 공통적인 프로그램 코드를 작성할 수 있어 모든 원격객체에 대한 서비스를 수행하는 하나의 서비스객체(ServerObject)를 정의할 수 있다. 그 외에 소켓의 연결설정, 쓰레드 관리 등 임의의 객체가 원격 호스트에서 생성되어 동작하고 소멸될 때까지 TORB 서버가 적절한 기능을 수행할 수 있도록 필요한 부속 프로그램이 구현되어야 한다.

5.2 대행객체

TORB의 후처리에 의해 변환된 분산프로그램이 적절한 분산처리를 수행하기 위해서는 원격 호스트에서 동작할 객체(ObjA)에 대한 대행객체(ObjA_Proxy)가 마련되어 있어야 한다.

5.2.1 대행객체의 생성

원격 호스트에서 생성되어 동작할 자바 객체는 TORB 서버를 통해 직접 접근되며, 해당 객체에 대한 대행객체를 통하여 로컬 호스트에서 네트워크를 통하여 간접적으로 접근할 수 있다. 대행객체에는 원격 호스트에 존재하는 자바 객체를 접근하는 모든 방안이 마련되어 있어야 하며, 그 객체에 정의된 속성(메쏘드, 생성자, 멤버데이타 등)을 기준으로 생성할 수 있다. 임의의 자바 객체에 정의된 속성은 자바의 API중 java.lang.Class를 이용하여 그 객체의 인스턴스로부터 모두 구할 수 있다.

5.2.2 대행객체의 내용

프로그래밍 투명성의 지원을 받으며 작성된 분산처리 프로그램은 원격 호스트에서 동작하는 객체를 접근하는 프로그램 코드를 일반적인 자바 객체를 접근하는 방법으로 작성하고 컴파일 할 수 있다. 분산처리를 목적으로 이러한 방법으로 작성된 프로그램은 후처리를 통하여 변환된 후에야 분산처리 수행환경에서 분산처리 역할을 수행할 수 있다. 후처리는 원격 호스트에서 동작할 객체를 접근하는 프로그램의 모든 코드를 대행객체를 통해서 접근하는 코드로 변환하고 원격 객체에 대한 대행객체를 생성하는 것이다. 일반적인 자바 객체를 접근하는 방법으로 작성된 프로그램 코드가 대행객체를 통하여 원격객체를 접근하는 코드로 변환되어 분산처리 역할을 수행하는 프로그램이 생성되므로 임의의 객체에 대한 모든 접근동작을 대행객체를 통하여 수행할 수 있어야 한다. 즉, 대행객체를 통한 원격객체의 접근동작이 그 객체를 로컬객체로 취급하여 접근하는 동작을 모두 포함한다면 프로그래밍 투명성을 보장하기 위한 대행객체의 필요한 기능이 모두 갖추어졌다고 말할 수 있다.

자바 프로그램에서 임의의 객체에 대하여 적용할 수 있는 접근조작은 다음과 같다.

- 객체 인스턴스의 생성. 이것은 자바의 키워드 "new"를 사용하여 객체에 정의되어 있는 생성자를 호출하는 방법으로 이루어진다.
- 정의된 메쏘드의 실행. 이것은 객체의 인스턴스에 대하여 연산자 "."을 사용하여 해당객체에 정의되어 있는 메쏘드를 정적으로 호출하거나 Java Reflection 기능을 활용하여 동적으로 호출하는 방법으로 이루어진다.
- 메쏘드 호출시 인수(parameter)로 취급하여 전달. 이는 임의의 객체에 대한 임의의 메쏘드나 생성자를 호

출할 때 참조에 의한 인수전달 방법으로 이루어진다. 자바의 모든 객체는 인수로서 전달될 때 참조에 의한 인수전달 방법이 적용된다. 여기에는 자바의 키워드 "this"에 의해 전달되는 것도 포함된다.

- 멤버데이타의 접근. 이것은 해당 객체의 인스턴스에 대하여 연산자 "."와 "="을 사용하여 멤버데이타에 값을 할당하거나 멤버데이타의 값을 다른 변수에 할당할 때 이루어진다. 이는 객체지향 프로그래밍 기법에서 권장하지 않는 프로그래밍 방법이지만 일반적인 자바 프로그램에서 흔히 적용하는 기능이다.

위에서 언급한 4가지 이외에 임의의 로컬객체의 인스턴스에 대한 추가적인 접근조작이 없다는 참조문서는 없지만, 상기의 4가지 접근조작에 대한 완전한 대행자 역할을 수행하는 대행객체를 구현하고 단위 테스트를 수행한 결과, 기존의 분산처리 프로그래밍 지원도구의 하나인 HORB의 기능을 모두 포함하면서 프로그래밍 투명성을 지원하는 것을 확인하였다[6].

상기의 4가지 접근조작에 대한 완전한 대행자 역할을 수행하는 대행객체를 생성하기 위하여 다음과 같은 내용과 기능이 포함된다.

- 원시객체에 정의된 모든 생성자와 동일한 원형(Prototype)을 가지는 모든 생성자. 이는 원시객체를 생성하는 프로그램 코드가 대행객체를 생성하는 코드로 변환되기 때문이다.
- 원시객체에 정의된 모든 메쏘드와 동일한 원형을 가지는 모든 메쏘드. 이는 원시객체에 정의된 메쏘드를 호출하는 프로그램 코드가 대행객체의 메쏘드를 호출하는 코드로 변환되기 때문이다.
- 원시객체가 원격객체가 되어 인수로서 전달되는 경우에 대행객체가 대신해서 인수로서 전달되어도 동일한 효과를 발휘할 수 있도록 필요한 기능
- 원시객체의 멤버데이타를 접근하는 조작이 대행객체를 통해서 수행될 수 있도록 지원하는 기능
- 원격 호스트와 통신기능을 담당하기 위하여 추가되는 메쏘드와 멤버데이타
- 원시객체가 생성되어 실행될 호스트의 URL을 지정할 수 있는 기능. 이는 원시객체가 생성되어 동작할 호스트가 실행시간에 결정되는 경우에만 필요한 기능이다.

5.2.3 대행객체의 생성 방안

TORB의 후처리 도구는 위에서 언급한 여섯 가지의 내용과 기능을 가지는 대행객체를 자동으로 생성하는 도구이고, 이를 구현하기 위하여 다음과 같은 방법이 적용된다.

- 원시객체에 포함된 모든 속성정보를 java.lang.Class에 의해 얻을 수 있으므로 원시객체의 생성자와 동일한 원형을 가지는 생성자를 포함하도록 구현할 수 있다.
- 원시객체에 포함된 모든 속성정보를 java.lang.Class에 의해 얻을 수 있으므로 원시객체에 포함된 메쏘드와 동일한 원형을 가지은 메쏘드를 포함하도록 구현할 수 있다. 특히, 원시객체가 다른 객체를 상속하고 있는 경우에 모든 부모객체의 메쏘드와 멤버데이타의 정보까지도 얻을 수 있으므로 자바객체의 상속성 문제도 자연히 해결된다.
- 대행객체에는 원시객체를 원격 호스트에서 접근하기 위한 모든 방안이 마련되어 있으므로, 원시객체 대신에 대행객체가 인수로 전달되어도 동일한 효과를 발휘한다. 이를 위하여 먼저 해결해야 하는 문제는 대행객체를 네트워크를 통하여 다른 시스템으로 전송할 수 있도록 구현하여야 한다는 것이다. 임의의 자바객체를 네트워크를 통하여 전송하기 위해서는 반드시 Java Serialization 기능에 적용할 수 있어야 한다. 대행객체를 Java Serialization 기능에 적용할 수 있는 형태로 생성하기 위해서는 대행객체 내에 기계 종속적인 멤버데이타(원격시스템과의 통신기능을 수행하는 소켓 등)가 존재하지 않아야 한다. 대행객체에 기계 종속적인 멤버데이타를 포함시키지 않고 Java Serialization 기능을 적용할 수 있도록 하기 위하여 원격 시스템과의 통신기능에 관련된 모든 요소를 멤버데이타를 사용하는 방법 대신에 메쏘드 호출에 의해 수행하도록 구현하면 된다. 이로 인하여 매번 원격객체를 접근하는 동작을 수행할 때마다 연결설정이 이루어져야 하는 필연적인 오버헤드를 가진다. 그러나, 한번 설정된 연결이 유지되는 시스템에서는 네트워크의 단절이 곧 시스템오류가 되어 해당 객체를 더 이상 접근할 수 없게되는 결과를 초래하지만, TORB에서는 원격 호스트와의 연결이 지속적으로 유지되는 형태가 아니므로, 네트워크가 회복되면 해당객체에 대한 접근을 정상적으로 재시도 할 수 있으므로 오버헤드에 대한 보상을 어느 정도 기대할 수 있다.
- 멤버데이타에 대한 접근은 원시객체와 그의 부모객체 내에 존재하는 모든 멤버데이타에 대하여, _get_x()와 _put_x(int value)와 같은 형태로, 두 개의 메쏘드를 대행객체에 추가로 삽입하여 해결할 수 있다. 특히, 원시객체에 대한 상속기능을 처리하기 위해, 원시객체의 부모객체가 보유하고 있는 멤버데이타에 대해서도 같은 규칙을 적용한다. 또한, 원시객체 차제도 멤버데이타를 접근하는 메쏘드가 추가되어 있어야 하는데, 이는 원시객체의 클래스파일을 변환하는 후처리를 수행할 때 이루어진다.
- 원격 호스트와의 통신을 담당하는 요소는 모든 객체에 대해 동일한 동작을 수행하므로, 미리 정의한 몇 개의 메쏘드를 추가하면 된다.

- 원시객체가 생성되어 실행될 호스트의 주소를 실행시간에 동적으로 지정하는 기능은 대행객체의 메쏘드 정의에 URL을 지정하는 메쏘드를 추가해 두고, 프로그램 코드 내에서 이 메쏘드를 호출할 수 있는 기능을 마련해 놓으면 된다. 이를 위하여 TORB에서는 SayURL이라는 인터페이스 객체를 정의해 두고 있다.

5.3 클래스파일(자바 바이트 코드) 변환

TORB의 후처리 도구에 의해 변환되기 이전의 자바 프로그램은, 원격 호스트에서 생성되어 동작하는 객체를 접근하기 위하여 작성되었지만, 원격객체를 접근하기 위하여 대행객체를 접근하는 자바 바이트 코드가 아니고 로컬에 존재하는 원시객체를 접근하는 자바 바이트 코드로 컴파일되어 있다. TORB의 후처리 중 클래스파일 변환단계 에서는 원시객체를 접근하는 바이트 코드를 대행객체를 접근하는 바이트 코드로 변환하는 기능이 있어야 한다. TORB는 이러한 기능을 구현하기 위해서 BCEL의 jasmin과 JasminVisitor를 보조도구로 활용한다. Jasmin은 자바 어셈블리 코드를 클래스파일로 생성해 주는 기능을 가지는 자바 어셈블러이고, Jasmin-Visitor는 자바 객체의 인스턴스로부터 자바 어셈블리 코드를 생성하는 기능을 가지는 자바 역어셈블러이다[13,14]. TORB의 후처리 도구가 처리하는 변환작업은 JasminVistor에 의해 역 어셈블 되어 생성된 자바어셈블리 코드를 수정하고, jasmin에 의해 변환된 클래스파일을 생성하는 절차로 구성된다. JasminVisitor에 의해 역 어셈블 되어 생성된 자바어셈블리 코드에 대한 변환조작의 주된 내용은 다음과 같다.

- 변환 전의 자바 어셈블리 코드에서 원격 호스트에서 동작해야 할 객체의 이름이 나타날 때마다, "_Proxy"를 추가한 이름으로 대치한다. 이는 원격 호스트에서 동작해야 하는 객체에 접근하는 모든 바이트 코드를 해당객체에 대한 대행객체를 사용하는 코드로 변환하기 위한 것이다. 이 처리는, 원격 객체가 인수로서 전달되는 경우에도 동일하게 적용된다.
- 원격 호스트에서 동작해야 하는 객체에 포함된 모든 멤버데이타에 대하여 _get_x()와 _put_x(int value)와 같은 형태의 이름을 가지는 메쏘드를 강제로 추가하기 위하여 이에 대한 자바 어셈블리 코드를 추가해야 한다. 이는, 멤버데이타에 대한 직접접근을 추가된 메쏘드를 통하여 처리하기 위하여, 원시 객체의 클래스 정의에 멤버데이타를 접근하는 메쏘드를 마련해 두는 역할을 한다.
- 변환 전의 자바 어셈블리 코드에 포함된 어셈블리 연산자 중 "putfield"와 "getfield"가 원격객체의 멤버데이타를 대상으로 하는 연산이면, 대행객체에 강제로 포함시켜 둔 _get_x()와 _put_x(int value)와 같은 메쏘드를 호출하여, 그 결과를 저장하는 형태의 어셈블리 코드로 대치한다. 이는 프로그래밍 투명성을 해치지 않고 멤버데이타에 대한 직접접근을 지원하기 위한 것이다. 참고로, 이때의 "putfield"와 "getfield"연산이 원 객체 자신에게 포함된 멤버데이타에 대한 연산인 경우에는 이러한 변환과정을 적용하지 않아야 한다. 왜냐하면, 이는 원시객체에 강제로 추가해둔 _get_x()와 _put_x(int value)와 같은 메쏘드를 재귀호출 형태로 변환하는 결과가 되기 때문이다.
- 원격객체를 접근하는 프로그램 코드가 포함된 객체의 모든 생성자에 TORB 서버를 중복되지 않게 기동하는 어셈블리 코드를 삽입한다. 이렇게 기동되는 TORB 서버는 로컬 호스트에서 동작하는 것이고, 이 객체 자신이 직접 혹은 간접으로 다른 원격 호스트에 인수(parameter)로서 전달되었을 때, 그 호스트에 대한 원격객체 서비스를 수행하기 위한 것이다.
- 원격객체를 접근하는 프로그램 코드가 포함된 객체에 자바의 키워드 "this"를 대신하기 위한 멤버데이타를 삽입하는 어셈블리 코드를 삽입한다. 이것은 이 객체 자신이 자바의 키워드 "this"에 의해서 다른 원격 호스트에 인수로서 전달될 때 대치하기 위한 것이다. 이런 경우, 자바의 키워드 "this"를 대신해서 삽입된 멤버데이타가 동일한 역할을 수행하도록 어셈블리 코드를 적절히 변환해야 한다.
- 이상 제시한 변환조작 외에도 TORB가 지원하는 후처리 도구에 의해 변환된 자바 프로그램이 최초 목적한 분산처리를 수행하도록 적절히 변환하는 단편적인 조작이 필요하다. 이러한 변환조작은 후처리 도구를 구현하기 위한 단순한 코딩 기술이므로 설명을 생략한다.

6. TORB 기능 검증

본 논문에서 설계한 TORB가 충분한 분산처리와 프로그래밍 투명성을 지원하는지를 확인하기 위하여 몇 가지 단위 프로그램에 대한 기능실험을 실시하였다. 테스트를 위한 단위 프로그램들은 원격 메쏘드 호출, 원격객체의 멤버데이타 접근, 객체참조(object reference)의 전달에 의한 콜백 등 기존의 분산처리 프로그래밍 도구가 지원하는 기능을 테스트하기 위한 것이며, 정상적인 분산처리 동작을 수행하는 것을 확인하였다. 상속성, 다형성, 캡슐화의 지원은 자바를 포함한 객체지향 프로그램의 중요한 세 가지 속성이고, 지역적으로 떨어져 있는 여러 호스트가 협동작업을 통하여 분산처리를 수행하는 환경에서 이러한 속성을 수용하는 것은 객체지향 프로그램의 분산처리를 위한 필수 기능을 포함한다고 평가할 수 있다. 또한, 화이트 보드는 이러한 속성을 테스트

할 수 있는 좋은 예로 평가된다[16]. 6장에서는 자바 애플릿을 이용하여 웹브라우저 환경에서 동작하는 화이트보드의 구현과정을 소개하여, TORB를 이용한 분산처리 프로그램의 구체적인 작성 예를 제시하고, 분산처리 프로그래밍 도구로서의 기능을 검증하고자 한다.

6.1 프로그램 설계

TORB를 이용한 분산처리 프로그램의 예를 보이기 위하여 작성하는 "화이트보드"는 웹브라우저를 통하여 실행되는 자바애플릿으로 작성한다. 기능은 일반적인 화이트보드의 필수기능을 포함하고 있으며, 채팅 기능이 부가된 형태로 작성한다. 프로그램의 요소로서 원격 호스트에서 화이트보드의 서버역할을 담당하는 객체와 웹브라우저를 통하여 사용자 인터페이스를 담당하는 클라이언트 객체와 더불어 몇 가지 부속 객체의 정의가 필요하다. 분산처리를 위하여 클라이언트 객체는 원격 호스트의 서버 객체를 생성하고, 서버 객체의 원격 메쏘드 호출을 통하여 자신의 참조정보를 참조에 의한 인수전달(call by reference) 방법으로 등록한다. 여러 개의 클라이언트가 동일한 방법으로 서버 객체에게 자신의 참조 정보를 등록한 후, 하나의 클라이언트가 매번 화이트보드를 조작할 때마다, 서버객체의 원격 메쏘드에 대한 호출이 이루어지고, 서버 객체는 자신에게 등록되어 있는 모든 클라이언트 객체에 대하여 적절히 대응되는 원격 메쏘드를 호출(callback)하여 모든 클라이언트가 동일한 화이트보드 내용을 유지하도록 한다.

6.2 프로그래밍 투명성

TORB가 지원하는 프로그래밍 투명성을 적용하여 6.1절의 화이트보드를 작성한다면, 프로그램이 완성될 때까지 TORB의 지원은 전혀 필요 없다. 필요한 것은 프로그래머 자신이 분산처리 프로그램을 작성하고 있다는 사실과 TORB가 제공하는 프로그래밍의 투명성의 범위에 대한 인식이다. 여기서는 원격에서 생성되고 동작할 서버객체와 네트워크를 통하여 이 객체를 접근할 클라이언트 객체로 나누어서 정의해야 한다는 사실을 인식하고, 이들 객체가 수행해야할 기능을 프로그램으로 구현하면 그만이다. 이를 위하여 필요한 도구는 자바 개발킷드(JDK) 뿐이다. 즉, 일반적인 자바 프로그램을 작성하듯이 코딩, 컴파일, 테스트런을 통한 디버깅을 하며 프로그램을 구현하면 된다. 분산처리를 위한 모든 조작은 프로그램이 완성된 이후에 TORB가 제공하는 후처리 도구와 분산처리 수행환경에 의해 이루어지기 때문이다.

6.3 화이트보드 서버 객체

구현된 화이트보드는 채팅기능과 연계되어 동작한다. 즉, 동일한 대화방에 입장한 사용자들이 채팅과 더불어 하나의 화이트보드를 공유하며 사용하는 구조로 작성되어 있다. 이러한 기능을 수행하기 위하여 서버 객체에는 다음과 같은 메쏘드가 정의되어 있다.

- UpdateChatRoom : 대화방이 생성되거나 없어질 때마다 수행된다. 여기에는 대화방 목록이 변동된 사실을 갱신하기 위하여 등록된 모든 클라이언트의 메쏘드를 호출(callback)하는 코드가 포함되어 있다.
- ChatRoomIn : 특정 사용자가 특정 대화방에 입장한 경우에 수행된다. 클라이언트가 원격 메쏘드 호출을 통하여 수행하며, 대화방에 입장되어 있는 다른 사용자들에게 입장한 사실을 알리는 동작을 등록된 모든 클라이언트의 메쏘드 호출을 통하여 수행한다.
- ChatRoomOut : 특정 사용자가 특정 대화방에서 퇴장한 경우에 수행된다. 클라이언트가 원격 메쏘드 호출을 통하여 수행하며, 대화방에 입장해 있는 다른 사용자들에게 퇴장한 사실을 알리는 동작을 등록된 모든 클라이언트의 메쏘드 호출을 통하여 수행한다. 입장되어 있는 사용자가 하나도 없는 경우 UpdataChatRoom을 호출하는 기능이 포함된다.
- regist : 임의의 사용자가 프로그램을 시작하여 클라이언트를 서버에 등록하기 위하여 수행된다. 등록은 클라이언트에서 서버로의 원격 메쏘드 호출 시에 이루어지며 참조에 의한 인수전달에 의해 등록된다.
- unregist : 임의의 사용자가 프로그램을 종료하여 서버와의 접속을 끊고자 할 때 수행한다. 이에 따라 원격 시스템이 쓰레기 수집 작업을 수행할 수 있다.
- say : 임의의 사용자가 채팅 메시지를 입력할 때마다 수행된다. 하나의 클라이언트가 원격에서 say를 호출하면, 동일한 대화방에 입장해 있는 다른 모든 클라이언트에게 원격 메쏘들 호출에 의해 채팅 메시지를 전달한다.
- add_gObj : 하나의 클라이언트가 공유하고 있는 화이트보드에 그림요소를 추가할 때마다 수행된다. 화이트보드에 그려진 그림요소는 서버에서 보관되며 동일한 대화방에 입장해 있는 모든 클라이언트에게 전송된다.
- update_gObjs : 하나의 클라이언트가 공유하고 있는 화이트보드에 포함된 그림요소의 속성을 변경할 때마다 수행된다. 또한, 동일한 대화방에 입장해 있는 모든 클라이언트에게 그림요소의 속성의 변동 정보가 전송된다.

6.4 화이트보드 클라이언트 객체

클라이언트 객체는 네트워크를 통하여 자신과 대응하는 서버 객체를 생성하고 원격 메쏘드 호출을 통하여 채팅과 화이트보드 기능을 수행한다. 다음은 클라이언트 객체에 정의되어 있는 메쏘드이다.

- init : 프로그램이 기동될 때 수행되며, 자바의 AWT (Abstract Window Toolkit)기능을 이용한 사용자 인

터페이스를 초기화와 서버와의 접속기능을 수행한다.

- DisplayMessage : 임의의 클라이언트가 원격 메쏘드 호출을 통하여 서버에게 전송한 채팅메시지를 서버의 콜백(서버 객체가 클라이언트 객체의 메쏘드를 원격에서 호출)에 의해 클라이언트의 채팅창에 표시되도록 하는 역할이다. 분산처리 프로그래밍에서는 클라이언트 자신의 채팅창에 대한 표시 작업도 서버를 통한 콜백에 의해 처리되도록 한다.
- UpdateChatRoom : 대화방이 생성되거나 없어질 때마다 서버에 의해 콜백되는 메쏘드이다.
- ChatRoomIn, ChatRoomOut : 사용자가 대화방에 입장 혹은 퇴장하는 조작을 수행할 때마다 수행한다. 이 메쏘드에 의해 서버의 메쏘드 ChatRoomIn, ChatRoomOut이 기동되어 다른 클라이언트에게 변동사항의 갱신을 통보할 수 있다.
- say : 임의의 사용자가 채팅 메시지를 입력할 때마다 수행한다. 이 메쏘드에 의해 서버에 존재하는 메쏘드 say가 기동되어 다른 클라이언트에게 동일한 채팅 메시지를 전달한다.
- destroy : 이것은 임의의 자바객체가 쓰레기 수집의 대상이 되어 메모리에서 사라지기 직전에 자동으로 호출되는 메쏘드로서, 대응되는 원격 객체에 대한 쓰레기 수집을 수행할 수 있도록 처리하는 기능을 수행한다. 즉, 로컬 호스트에서 특정 객체에 대한 쓰레기 수집을 수행할 때, 이 객체와 관련된 원격 객체도 쓰레기수집의 대상이 되도록 처리하기 위한 기능이다.
- 이 외에도 클라이언트 객체가 채팅 및 화이트보드 기능을 수행하기 위하여 필요한 메쏘드가 추가로 정의되어 있으며, 클라이언트 객체가 이용하는 몇 가지 부속 객체가 정의되어 있다.

이상의 화이트보드 프로그램은 하나의 클라이언트에 대하여 서버 객체의 인스턴스가 독립적으로 생성되어 동작하지만 서버 객체들은 하나의 TORB 서버에 대한 자식 프로세스로 동작하고 정적 변수를 정의하고 사용하는 것으로 상호간에 공유되는 메모리를 접근할 수 있다. 또, 자바에는 프로세스 동기화와 관련된 언어요소(키워드 synchronized)가 있으므로 동기화 문제도 쉽게 해결할 수 있다.

6.5 후처리

화이트보드 프로그램의 후처리 작업은 서버 객체(WBServer)와 클라이언트 객체(WBClient)를 대상으로 이루어진다. 원격에서 생성되어 동작할 객체의 기능을 대신해서 수행하는 각각의 대행객체를 생성하고, 서버 객체와 클라이언트 객체간의 상호 접근하는 자바 바이트 코드를 대행객체를 통하여 접근하는 자바 바이트 코드로 변환한다. 후처리 도구가 이러한 작업을 수행하도록 그림 6과 같은 설정파일이 필요하고 그 의미는 다음과 같다.

- 코멘트를 제외하고 설정 파일에 나타난 두 줄의 내용은 원격 객체를 접근하는 프로그램 코드가 포함된 객체는 WBServer와 WBClient의 두 개라는 의미이다. 따라서, 지정된 두 개의 객체에 대한 후처리 작업만 수행된다.
- 표시된 URL은 각 객체가 생성될 정적인 주소이다. TORB가 제공하는 프로그램 요소(인터페이스 객체)를 활용하여 프로그램을 작성하면, 객체의 생성 호스트를 실행시간에 결정하여 처리하도록 구현할 수 있다.
- URL의 호스트 사용된 self는 해당 객체가 프로그램이 수행되는 호스트에서 생성된다는 의미이다. TORB에서는 콜백을 지원하기 위하여 분산처리에 참여하는 모든 객체는 TORB 서버를 통하는 원격 객체로 취급한다.
- URL에 나타난 "9000"은 소켓의 포트 번호이며, 이 기능의 지원에 따라 한 개의 호스트에 여러 개의 TORB 서버를 기동할 수 있다.

```
#Remote Classes
WBServer.java=TORB://remote.ac.kr:9000
WBClient.java=TORB://self
```

그림 6 화이트보드 프로그램을 위한 설정파일

- 이 외에, 후처리의 결과로 WBServer와 WBClient로부터 WBServer_Proxy와 WBClient_Proxy가 각각 생성되며 그 내용은 5장에서 설명한 바와 같다.

6.6 실행화면

그림 7은 구현된 화이트보드의 수행화면이다. 자바 애플릿을 사용하여 구현되어 있으므로 웹브라우저를 통하여 실행된다.

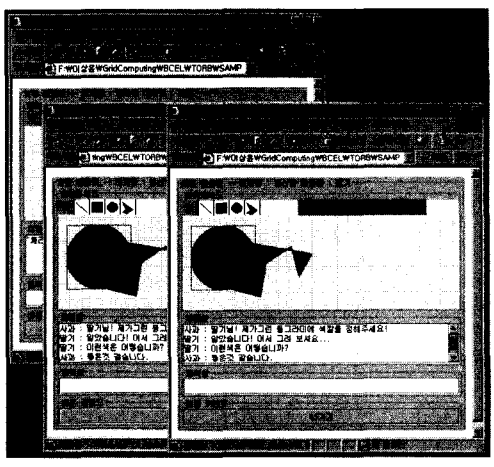

그림 7 화이트보드 실행화면

수행화면에는 대화명이 "딸기"인 사용자와 "사과"인 사용자가 화이트보드를 활용하며 채팅을 수행하는 모습이다. 실제는 두 사용자가 각각 별도의 호스트에서 네트워크를 통하여 상호작용을 수행하는 내용이지만 실행화면에 대한 신뢰를 위하여 한 개의 호스트에서 동작하는 모습을 캡처하여 표현하였다.

7. 결론 및 향후연구

부가적인 지식 없이 분산처리를 수행하는 소프트웨어를 개발할 수 있다면, 분산처리에 대한 개발자의 부담을 줄여서, 개발하는 소프트웨어의 자체 기능에 더욱 집중할 수 있다. 본 논문에서는 분산환경에 적용하기 위하여 프로그램을 작성할 때, 프로그래밍 투명성을 지원하여 상대적으로 쉽게 분산처리 프로그램을 작성할 수 있도록 해주는 분산 프로그래밍 도구를 제안하고, 이름을 TORB라고 명명하였다. TORB가 지원하는 프로그래밍 투명성의 범위가 어느 정도이고, 충분한 프로그래밍 투명성을 지원하는가에 대한 정형화된 검증은 빠져있지만 여러 가지 단위 프로그램에 대한 테스트를 통하여, 상당한 수준의 프로그래밍 투명성을 지원하는 것을 확인하였으며, 실제의 분산처리 프로그래밍에 적용하여도 수용가능한 것으로 평가하였다. 향후, 프로그래밍 투명성에 대한 정형화된 연구가 필요하고, TORB가 완전한 프로그래밍 투명성을 지원하도록 하기 위하여 지속적인 연구와 기능 개선이 필요하다.

참 고 문 헌

[1] J. R. Nicol, C. T. Wilkes and F. A. Manola, "Object Orientation in Heterogeneous Distributed Computing Systems," IEEE Computer, Vol. 26, No. 6, June, 1993.

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

[3] T. L. Thai, A. Oram, "Learing DCOM," O'reilly, April, 1999.

[4] IBM, "SOMobjets : A pratical Introduction to SOM and DSOM," International Technical Support Organization, 1994.

[5] Sun Microsystems, Inc., "Java Remote Method Invocation," Online publishing, URL http://java. sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html, 2003.

[6] Satoshi Hirano, "HORB : Distributed Execution of Java Programs," In Lecture Notes in Computer Science 1274, Springer, pp.29-42, 1997.

[7] Yariv Aridor, Michael Factor and Avi Teperman, "cJVM: a Single System Image of a JVM on a Cluster," In Proceedings of ICPP 99, IEEE, 1999.

[8] Weimin Yu and Alan Cox, "Java/DSM: A Platform for Heterogeneous Computing," In Concurrency: Practice & Experience, Vol. 9, No. 11, pp. 1213-1224, 1997.

[9] Matchy J. M. Ma, Cho-Li Wang and Francis C. M. Lau, "JESSICA: Java-Enabled Single-System-Image Computing Architecture," In Journal of Parallel and Distributed Computing, Vol. 60, No. 11, pp.1194-1222, 2000.

[10] Michael (Thuan-Duc) Ta, "Test bed for Distributed Object Technologies using Java," The University of Auckland, Online publishing, URL http://www.cs.auckland.ac.nz/~cthombor/Perf/Horb/Report2011.pdf, 1998.

[11] Zvi Har'El and Zvi Rosberg, "Java Class Broker-A Seamless Bridge from Local to Distributed Programming,"In Journal of Parallel and Distributed Computing, Vol. 60, No. 11, pp.1223-1237, 2000.

[12] Michael Philippsen and Matthias Zenger, "Java-Party - Transparent Remote Objects in Java," In Concurrency: Practice & Experience, Vol. 9, No. 11, pp.1225-1242, 1999.

[13] Michiaki Tatsubori, Toshiyuki Sasaki, Shigeru Chiba, and Kozo Itano, "A Bytecode Translator for Distributed Execution of 'Legacy' Java Software," European Conference on Object-Oriented Programming (ECOOP), Budapest, Hungary, June 2001.

[14] J. Meyer and T. Downing. "Java Virtual Machine," O Reilly, 1997.

[15] M. Dahm. "Byte code engineering with the BCEL API," Technical Report B-17-98, Freie Universitat Berlin, Institut fur Informatik, April 2001.

[16] H. M. Levy and E. D. Tempero, "Modules, objects and distributed programming: issues in RPC and remote object invocation," Software---Practice and Experience, 21, pp. 77-90, 1991.

이 상 윤

1993년 경북대학교 컴퓨터공학과 졸업(공학사). 1995년 경북대학교 대학원 컴퓨터공학과 졸업(공학석사). 2000년 8월 경북대학교 대학원 컴퓨터공학과(박사수료). 1997년~현재 대원과학대학 컴퓨터정보처리과 조교수. 관심분야는 분산 시스템, 그리드 컴퓨팅, 멀티미디어, 자바 응용기술, 임베디드 시스템 등

김 승 호

1981년 경북대학교 전자공학과 졸업(공학사). 1983년 한국과학기술원 전산학과 졸업(공학석사). 1994년 한국과학기술원 전산학과 졸업(공학박사). 1985년~현재 경북대학교 컴퓨터공학과 교수