# 분산프로그래밍 도구의 프로그래밍투명성 지원 방안

이상윤
대원과학대학 컴퓨터정보처리과
sylee@daewon.ac.kr

# A Methodology to Support Programming Transparency for Distributed Programming Tool

Sangyun Lee
Dept. of Computer Information Processing, Daewon Science College

## 요 약

분산 컴퓨팅 환경에서 동작할 프로그램을 보다 쉽게 작성하기 위한 다양한 형태의 분산프로그래밍 도구들이 개발되어왔다. 그러나, 이러한 도구들을 이용하기 위해서는 각각의 도구가 요구하는 프로그래밍 방법을 숙지하여야 하는 부담을 감수해야 한다. 이런 부담은 분산프로그래밍 도구가 만족할 만한 프로그래밍 투명성을 제공하지 못하기 때문이고, 프로그램 작성시, 문제의 본질에만 집중하는 것을 방해한다. 본 논문에서는 프로그래밍 투명성을 지원하는 분산프로그래밍 도구를 개발하기 위한 방법을 제안한다. 제안된 방법의 테스트를 위하여 독자적인 분산실행환경을 설계하였으며, 이 환경에서 동작하는 분산 프로그램을 작성하여 단위테스트를 수행하였다. 이 프로그램은 분산처리를 위하여 특별히 정의된 문법(Syntax)을 전혀 사용하지 않았으며, 후처리만을 통하여 설계된 분산실행환경에서 분산프로그램의 역할을 수행한다.

## 1. 서론

분산 컴퓨팅 환경에 적용해야할 응용 프로그램이 점점 많은 비중을 차지하게 됨에 따라, 이를 지원하기 위한 다양한 형태의 분산 프로그래밍 도구들이 제안되어 있다[1,2,3]. 그러나, 이러한 분산 프로그래밍 도구들은 프로그래머에게 각각의 도구가 지원하는 독자적인 프로그래밍 방법을 숙지하여야 하는 부담을 준다. 이러한 부담은 분산 프로그래밍 도구가 프로그래밍 투명성을 만족할 만큼 지원하지 못하기 때문이다. 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 분산프로그래밍 도구를 개발하기 위한 방법을 제안한다. 제안된 방법으로 개발된 분산프로그래밍 도구를 이용하면, 프로그래밍 투명성을 충분히 보장하는 분산 프로그램을 Java로 작성할 수 있고, 이를 분산환경에서 동작시킬 수 있다. 프로그래머는 단지 작성하고 있는 프로그램이 분산프로그램으로 변환되어 분산환경에서 실행될 것이라는 사실만 염두에 두고, 일반적인 방법으로 프로그램을 작성하면 된다. 분산프로그램의 프로그래밍 투명성을 확보하기 위하여, 일반적인 프로그래밍 방법으로 작성된 자바클래스로부터 분산환경에서 실행하기 위한 적절한 후처리를 수행하여 변환된 클래스와 대행객체를 위한 코드를 생성한다. 후처리를 통하여 수정된 자바클래스는 생성된 대행객체와 연동하여 독자적으로 설계된 분산실행환경에서 동작하며 분산처리를 수행한다.

본 논문은 프로그래밍 투명성을 충분히 보장하는 분산 프로그래밍 도구를 구현하기 위한 방법을 알아보기 위하여 다음과 같이 구성된다. 서론에 이어 2장에서는 관련연구결과를 소개하고 이들이 가지는 프로그래밍 투명성의 문제점을 논의한다. 3장에서는 일반적인 방법으로 작성된 프로그램을 분산프로그램으로 변환하는 절차와 함께 분산실행환경의 기본적인 메커니즘을 제시하고 4장에서는 분산프로그램으로 변환하기 위한 기술과 구현방법을 설명한다. 5장에서는 결론 및 향후 연구계획을 제시한다.

## 2. 관련연구

자바를 지원하는 분산처리 시스템으로 Java RMI를 비롯한 많은 형태의 객체대행자(Object Request Broker)가 개발되어 있지만, 이들 대부분은 인터페이스 정의를 통하여 원격객체를 접근하도록 고안되어 있다. 따라서, 이들을 사용하여 분산프로그램을 작성하려면 프로그래밍 외적인 작업이 상당히 필요하고, 프로그램을 작성 할 때에도 프로그래밍 투명성의 침해를 감수해야 한다[2]. Java Cass Broker와 HORB는 이러한 문제를 줄이기 위해서 개발되었지만 다음과 같은 프로그래밍 투명성의 부족문제를 남겨놓았다[3,4]. 이들을 사용하여 분산프로그램을 작성하려면 그림1과 같은 프로그램을 그림2와 같이 작성해야 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

자바언어의 확장을 통한 분산프로그래밍 도구로는 JavaParty가 있다[5]. JavaParty는 자바언어에 단지 "remote"라는 키워드를 추가해서 분산프로그래밍을 지원하도록 고안되어 있으므로 프로그래밍 투명성을 상당히 확보한 것으로 평가할 수 있다. 그러나, 분산프로그래밍을 작성할 때 원격객체의 생성을 위하여 최소한 "remote"라는 키워드를 사용하여야 하고, 별도의 자바컴파일러가 필요하다.

프로그래밍 투명성을 충분히 지원하는 분산실행환경으로는 Addistant가 있다[6]. Addistant는 분산환경을 전혀 고려하지 않고 작성된 자바프로그램을 분산환경에서 동작하도록 지원하기 위하여 정책파일을 정의하고, 이것을 기준으로 각각의 자바객체가 생성될 호스트를 명시하며, 분산환경에서의 동작을 위하여 실행시간에 자바 바이트코드의 변환이 이루어진다. Addistant의 주된 목적이 분산프로그래밍을 지원하는 도구로서의 역할이 아니라 이미 작성된 자바 프로그램을 분산환경에서 실행되도록 지원하기 위하여 개발된 것으로 평가할 수 있지만, Addistant를 염두에 두고 프로그래밍을 한다면 분산프로그래밍 지원도구로서의 역할을 담당할 수 있다. 그러나, Addistant에서

는 하나의 이름으로 정의된 자바객체의 인스턴스를 여러 호스트에서 생성하도록 명시하는 것이 불가능하다. 이와 같은 제약은 여러 개의 분산호스트에서 동일한 역할을 수행하는 자바객체의 인스턴스를 동적으로 생성할 수 없는 결과를 초래하게 된다.

본 논문에서는 자바의 interface 기능으로 이 문제를 해결하였다. 즉, 프로그래머는 미리 정의된 "SayURL"이라는 interface 객체의 구현객체로서 원격객체를 정의하고 구현 메쏘드인 "void _SayURL(...)"을 사용하여 원격객체가 생성되어 동작해야 하는 호스트를 지정할 수 있다. 이때, 구현 메쏘드 "void _SayURL(...)"은 내용이 없이 형태만 존재하는 메쏘드로 삽입하면 충분하다. 필자는 이러한 절차가 자바의 일반적인 프로그래밍 요소를 활용하는 것이고, 이 기능을 사용하지 않을 때는 프로그램 작성시 적용하지 않으면 되므로 프로그래밍 투명성을 해치는 것이 아니라고 주장하고 싶다.

## 3. 분산프로그램으로의 변환

분산환경을 전혀 고려하지 않고 작성된 프로그램도 본 논문에서 제안하는 방법을 사용하면 분산환경에서 실행되도록 변환할 수 있다. 그러나, 분산환경을 고려하면서 프로그램을 작성한다면, 목적과 더 잘 부합되는 프로그램을 작성할 수 있다는 것은 자명한 사실이다. 본 논문에서는 위의 두 가지 중 어떠한 경우라도 프로그래밍 투명성을 충분히 보장하는 분산 프로그램을 작성할 수 있도록 지원하는 분산프로그래밍 도구를 제안한다. 3장에서는 작성된 자바프로그램을 분산실행환경에 적용할 수 있도록 변환하는 절차와 이것이 실행되는 과정을 논의한다.

### 3.1. 후처리를 통한 프로그래밍 투명성 확보

일반적인 방법으로 작성된 자바프로그램을 분산실행환경에서 동작하도록 변환하기 위해서는 후처리를 위한 독자적인 변환도구가 사용된다. 변환도구를 사용하기 위해서는 그림3과 같은 설정파일이 필요한데, 이는 작성된 자바프로그램의 원격객체들이 생성될 호스트와 원격객체와 관련된 객체의 목록을 명시하기 위한 것이다.

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

그림 3. 설정파일의 예

따라서, 일반적인 프로그래밍 방법으로 작성된 자바프로그램은 간단한 형태의 설정파일과 몇 가지 옵션을 지정하는 후처리만을 통하여 분산실행환경에 적합한 형태의 분산프로그램으로 변환된 후, 표준 자바가상기계에서 분산처리를 수행하는 프로그램으로서의 역할을 수행한다.

### 3.2. 분산실행환경

변환도구에 의해 변환된 분산프로그램은 본 논문에서 설계된 분산실행환경에서 분산처리를 수행하며 동작한다. 분산실행환경은 표준 자바가상기계에서 동작하는 서버프로그램과 변환된 분산프로그램의 연계작용에 의해 동작하며 그림4와 같은 구조로 동작한다.

분산산실행환경의 동작 메커니즘을 그림4를 통하여 알아보자. 프로그래머는 "ObjA"와 "ObjB"를 사용하는 프로그램을 일반적인 프로그래밍 방법으로 작성하였고, 이는 변환과정을 거쳐 "ObjA_Proxy"와 "ObjB_Proxy"를 사용하는 분산프로그램으로 변환된다. 이들은 미리 정의된 "Agent"객체의 지원을 받아 원격 컴퓨터의 "Server"에게 적절한 요청을 한다. 원격 컴퓨터의 "Server"는 객체서비스를 위하여 미리 정의된 객체인 "ServerObject"를 통하여 "ObjA"와 "ObjB"에 대한 서비스를 수행하여 "Agent"의 요청에 응답한다. 한편, 원격의 "ObjB"객체는 동일한 방법으로 로컬 컴퓨터의 "ObjC"와 "ObjD"를 접근할 수 있다.

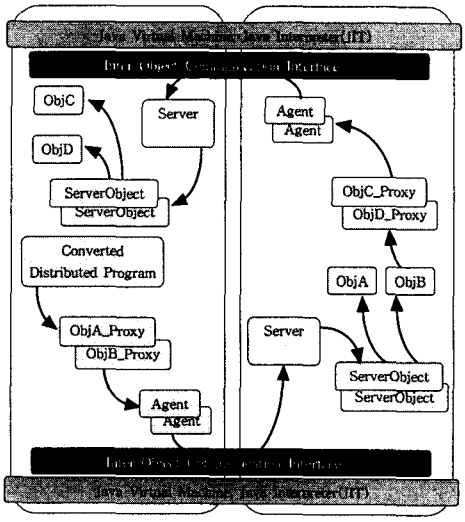


그림 4. 분산실행환경

한편, "Server"는 원격 컴퓨터에서 동작해야하는 객체에 대한 서비스를 지원해야 하므로 프로그램이 실행을 시작하기 전에 동작 중이어야 한다. 즉, 분산처리에 참여하는 모든 컴퓨터에 "Server"를 기동하는 조치가 필요하다. "Server"를 기동하는 방법은 두 가지가 제공된다. 하나는 사용자가 명령라인에서 직접 서버를 기동하는 방법이고, 다른 하나는 프로그래머가 작성한 프로그램을 기동할 때 자동으로 기동하는 방법이다. 두 번째 방법은 변환도구를 통하여 분산프로그램으로 변환될 때, "Server"를 기동하는 코드를 자동으로 삽입함으로써 구현된다.

## 4. 분산실행환경 및 변환도구의 구현

충분한 프로그래밍 투명성을 보장하는 분산 프로그래밍 지원도구가 되기 위해서, 변환도구에 의한 후처리를 통하여 이미 작성된 프로그램이 분산실행환경에서 동작되도록 변환한다.

### 4.1 Server

Server는 임의의 객체 ObjA를 원격에서 접근하기 위하여 대행객체 역할을 수행하는 ObjA_Proxy로부터 네트워크를 통하여 전달되어오는 요청을 분석하여 ObjA에 존재하는 메쏘드를 수행한 후, 그 결과를 원격에 존재하는 ObjA_Proxy에게 네트워크를 통하여 응답한다. 이때, 하나의 _Proxy 객체에 대해서 하나의 서비스객체 ServerObject가 대응되도록 구현함으로써 여러 개의 원격객체를 동시에 처리할 수 있도록 하였다.

### 4.2 대행객체

변환도구에 의해 변환된 분산프로그램이 적절한 분산처리를 수행하기 위해서는 원격시스템에서 동작할 객체 ObjA로부터 생성된 객체 ObjA_Proxy가 ObjA의 대행객체 역할을 완벽히 수행해야 한다.

#### 4.2.1 대행객체의 생성

임의의 자바객체에 대하여, 그 객체에 정의된 모든 속성(메쏘드, 생성자 등)은 java.lang.Class를 사용하여 얻을 수 있다.

변환도구는 이렇게 얻어진 속성정보를 대행객체(_Proxy)를 생성하기 위하여 활용한다.

4.2.2 대행객체의 내용

일반적인 자바프로그램에서 임의의 객체에 대하여 적용할 수 있는 조작은 다음과 같다.

첫째, 자바의 키워드 "new"를 사용한 해당 객체의 생성이다.
둘째, 해당 객체의 인스턴스에 대하여 연산자 "."을 사용하여 해당객체에 정의되어 있는 메쏘드를 정적으로 호출하거나 Java Reflection 기능을 활용하여 동적으로 호출하는 것이다.
셋째, 임의의 객체에 대하여 임의의 생성자나 메쏘드를 호출할 때, 인수로서 전달하는 것이다. 자바의 모든 객체는 인수로서 전달될 때 참조에 의한 인수전달 방법이 적용된다. 여기에는 자바의 키워드 "this"에 의해 전달되는 것도 포함된다.
넷째, 해당 객체의 인스턴스에 대하여 연산자 "."와 "="을 사용하여 데이터멤버에 값을 할당하거나 데이터멤버의 값을 다른 변수에 할당하는 데이터멤버의 접근이다.

대행객체는 임의의 객체에 대한 위의 4가지 조작에 대한 완전한 대행자 역할을 수행해야 한다. 따라서 다음과 같은 내용들과 기능이 포함된다.

첫째, 원(Orignal) 객체에 정의된 모든 생성자와 동일한 원형(Prototype)을 가지는 모든 생성자
둘째, 원 객체에 정의된 모든 메쏘드와 동일한 원형을 가지는 모든 메쏘드
셋째, 원 객체가 인수로서 전달되는 경우에 대행객체가 대신해서 인수로서 전달되어도 동일한 효과를 발휘할 수 있도록 필요한 기능
넷째, 데이터멤버를 접근하는 조작에 대응하는 적절한 기능
다섯째, 원 객체가 생성되어 실행될 호스트의 주소를 지정할 수 있는 기능(2장의 마지막 부분에 이 기능의 필요성을 설명하였다.)
여섯째, 원격시스템과 통신기능을 담당하기 위하여 추가되는 메쏘드와 데이터멤버

4.3. 클래스파일 변환

변환도구에 의해 변환되기 이전의 자바 프로그램(클래스파일)에는, 원격객체를 접근하기 위하여 대행객체를 사용하는 코드가 아니고, 로컬에 존재하는 실제객체를 접근하는 코드만으로 작성되어 있다. 변환도구는 변환전의 프로그램에 포함된 코드 중 원격에서 동작해야하는 객체에 접근하는 모든 코드를 해당객체에 대한 대행객체를 사용하는 코드로 변환해야 한다. 이러한 기능을 구현하기 위해서 BCEL의 jasmin과 JasminVisitor를 보조도구로서 활용한다. jasmin은 자바 어셈블리 코드를 클래스파일로 생성해 주는 기능을 가지는 자바 어셈블러이고, JasminVisitor는 모든 자바객체로부터 자바 어셈블리 코드를 생성하는 자바 역어셈블러이다[7,8]. 변환작업은 보조도구로 활용되는 JasminVistor에 의해 생성된 자바어셈블리 코드를 수정하는 것으로 충분하다. 왜냐하면, 수정된 자바어셈블리 코드는 jasmin에 의해 분산실행환경에 적합한 클래스파일로 변환될 수 있기 때문이다.

4.4 투명성 확보를 위한 핵심기법

분산 프로그래밍 도구의 투명성 확보를 위하여 본 논문에서 제시하는 핵심기법은 다음과 같다.

첫째, Java Serialization 기능을 적용할 수 있는 대행객체를 생성하고 이를 네트워크를 통하여 전송할 수 있도록 해서 참조에 의한 인수전달의 투명성을 확보한 것
둘째, 자바 역어셈블리 코드의 생성 및 적절한 처리에 의해 변환된 자바 클래스파일을 생성하는 방법을 적용하여 프로그래밍 자체의 투명성에 기여한 것

## 5. 결론 및 향후연구

프로그래밍 투명성은, 분산프로그램을 작성할 때, 프로그래머로 하여금 문제의 본질만 고려할 수 있도록 하여 상대적으로 고품질의 소프트웨어를 개발할 수 있도록 지원한다. 본 논문에서는 분산환경에 적용하기 위하여 작성하는 분산프로그램의 프로그래밍 투명성을 확보하기 위하여 적용한 기초적인 연구결과를 제시하였다. 따라서, 제시한 기법이 완전한 프로그래밍 투명성을 보장하는지에 대한 검증은 빠져있다. 그러나, 여러 가지 단위프로그램에 대한 테스트를 통하여, 상당한 수준의 프로그래밍 투명성을 지원하는 것을 확인하였으며, 실제의 분산프로그래밍에 적용하여도 수용 가능한 것으로 평가하였다. 향후, 보다 완전한 프로그래밍 투명성을 지원하기 위한 지속적인 연구와 내부코드의 개선이 필요하다.

## 참고문헌

[1] J. Siegel, CORBA Fundamentals and Programming, John Wiley & Sons, 1996
[2] 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
[3] Satoshi Hirano, "HORB: Distributed Execution of Java Programs", In Lecture Notes in Computer Science 1274, Springer, pp.29-42, 1997
[4] 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, Academic Press, pp.1223-1237, 2000
[5] Michael Philippsen and Matthias Zenger, "JavaParty - Transparent Remote Objects in Java", In Concurrency: Practice & Experience, Vol. 9, No. 11, Wiley, pp.1225-1242, 1999
[6] 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.
[7] J. Meyer and T. Downing. Java Virtual Machine. O Reilly, 1997.
[8] M. Dahm. "Byte code engineering with the BCEL API". Technical Report B-17-98, Freie Universitat Berlin, Institut fur Informatik, April 2001.