論文
3-2-9

자바 에이전트를 이용한 분산컴퓨팅 환경 구현
(Implementation of Distributed Computing Environment using Java Agent)

서 건 원* 이 길 흥**
(Gun-Won Seo) (Kil-Hung Lee)

요 약

컴퓨팅 환경의 변화 때문에 최근 에이전트 기술이 주목을 받고 있다. 네트워크에 에이전트를 분산 배치시킴으로서 서비스의 실현이 용이한 위치에서 필요한 서비스를 신속하게 제공하게 함으로서 네트워크의 효율성을 증대시킬 필요성이 점점 커지고 있다. 본 논문에서는 자바로 프로그래밍한 에이전트를 실제 서비스의 실현이 용이한 위치에 있는 네트워크에 배치하고, 매니저에서는 에이전트에 접속하여 서비스코드의 URL을 에이전트에 알려주었다. 에이전트는 해당 URL에서 서비스코드를 다운로드 받아서 실행시켜 그 결과를 매니저에게 돌려주는 분산컴퓨팅 환경을 자바로 구현해본다.

ABSTRACT

Because of the change of computing environment, an agent technology is spotlighted recently. By deploying agent distributedly in network and offering the necessary service quickly in place easing realization of service, the enlargement of the effectiveness of network is necessary more and more. In this paper, agents programmed in java are distributed in place easing realization of service in network. And a manager connects to agents and informs URL of service code of the agent. This paper implements distributed computing environment in which agent downloads service code from URL of service code, executes the code, and returns the result of execution to manager.

1. 서론

이제 거의 모든 컴퓨터가 인터넷에 연결돼 있고, 많은 기업에서 전사적인 애플리케이션을 마련하기 위해 준비하고 있다. 이는 서로 다른 컴퓨터에서 실행하는 서로 다른 프로그램들이 실제로 상호 작용하는 것이며, 이를 위한 기반 구조로 제시되는 것이 DCOM과 CORBA, 자바 RMI 와 같은 분산객체기술이다. 썬 (Sun) 사의 자바는 순식간에 인터넷을 등에 업고 유력한 분산객체 기술의 기반으로 자리를 잡았다. 이렇게 된 이유는 자바 애플리케이션은 멀티 플랫폼 환경에서 거의 완벽한 이식성을 제공한다. 웹 브라우저가 자바 인터프리터 (JVM)을 지원하면서 인터넷과 웹 환경에서 가장 확실한 수행환경을 보장받은 것이 자바활성화에 큰 역할을 했다. 또한 C++

* 정회원 : 경기고등학교 교사
** 정회원 : 서울산업대학교 컴퓨터공학과 전임강사

논문접수 : 2002. 1. 30.
심사완료 : 2002. 2. 20.

와 같은 기존의 객체지향 언어에 비해 익히기 쉽고 골치 아픈 메모리 관리로부터 해방될 수 있는 쓰레기 수집 (garbage collection)이 제공되는 등 언어적인 측면에서도 장점을 가지고 있다. 자바는 RMI라는 일종의 자체 ORB (Object Request Broker)를 갖고 있다. 문제는 RMI가 자바에만 국한하여 사용할 수 있다는 점이다. 왜냐하면 RMI는 자바언어의 자체적인 특성에 많이 의존하기 때문인데 예를 들면 자바 객체 직렬화를 구현하여야만 한다. 그러다가 썬에서 JDK1.2를 발표하면서 CORBA의 IIOP (Internet Inter-ORB Protocol)를 표준 프로토콜로 지원한다고 선언했다. 따라서 개발자는 자바의 RMI 객체를 다른 언어에서 작성한 분산객체와의 통신도 가능하게 되었다. 현재 다른 언어를 지원하는 분산 객체기술이 연구되어 오고 있으나 자바를 응용한 제품들이 가장 경쟁력이 있으며 널리 받아들여지고 있다.[1]

에이젼트 기술은 독립적이고 비동기적으로 업무를 수행하는 프로그램인 에이젼트가 사용자의 요구를 대행하여 수행하는 형태의 기술이다.[2] 본 논문에서는 자바로 프로그래밍 한 에이젼트를 서비스의 실현이 용이한 위치에 배치시킨 후 어떤 서비스가 필요할 때마다 신속히 해당 서비스 코드를 자바로 개발하여, 언제 어디서나 경량의 코드 작성으로 필요한 서비스를 관리자가 받을 수 있도록 하였다. 또한 필요할 때 실행 시간에 코드를 갱신할 수 있는 자바의 클래스로더의 유연한 기능을 이용하여 다른 분산객체기술을 사용하는 것보다 더 유연한 분산컴퓨팅 환경을 자바로 구현해 보았다.

2. 자바 에이젼트를 이용한 분산컴퓨팅 환경 구현

2.1 분산컴퓨팅 환경 고찰

하나의 기계에서 동작하는 경우에는 필요하지 않은 일들이 클라이언트와 서버로 구성된 분산환경에서는 반드시 고려되어야 할 것들이 많다. 클라이언트는 서버의 위치를 알아야 하고, 클라이언트의 호출을 이해한 서버가 처리결과를 클라이언트에게 전달하기 위해서는 클라이언트와 서버간의 약속이 필요하다.

이러한 약속을 특정 개발자가 나름대로 정의해 사용하면, 본인이 개발한 프로그램에만 적용할 수 있고, 다른 개발자가 개발하는 프로그램들과는 통신을 할 수 없게 될 것이다. 이런 것을 피하도록 탄생한 기술이 분산객체의 개념이다. 개발자에게는 객체간의 세부적인 통신 프로토콜은 가려지고 단지 객체간의 메소드 호출에만 관심을 가지면 되었다.[3]

개발자는 다른 분산객체기술 예를 들면 RPC (Remote Procedure Call), 자바의 RMI(Remote Method Invocation), CORBA (Common Object Request Broker Architecture) 와 같은 기술을 사용하여 분산컴퓨팅 환경을 구현하는 것이 프로그래밍작업도 용이하고 더 편리할 수도 있다고 생각할 수도 있으나 소프트웨어의 크기가 커지고 시스템간의 호환성 때문에 기능상의 제약이 생긴다. 이에 반하여 동일한 일을 하는 서비스코드를 개발하여 본 논문의 분산컴퓨팅 환경을 이용해 네트워크에 배치한다고 가정하면, 단지 이미 정의된 서비스 인터페이스를 구현하는 서비스코드를 개발 컴파일하여 임의의 위치에 배치하고, 곧 바로 에이젼트에게 코드저장소의 위치만 알려주는 것 이외의 다른 과정을 요구하지 않는다. 이와 같이 RMI 와는 달리 서버 측 컴퓨터에 소프트웨어를 설치하거나 또는 배포하는 등의 복잡한 작업이 필요 없게 되어 소프트웨어 컴포넌트의 개발과 배치전략이라는 측면에서 RMI와 비교할 때 훨씬 이점이 많다.

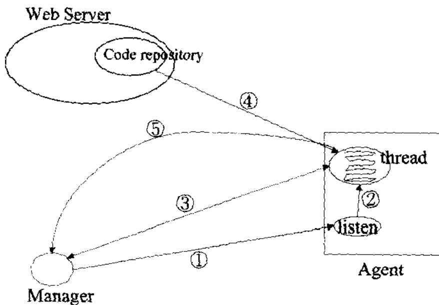
2.2 제안 모델

에이젼트를 서비스의 실현이 용이한 위치에 있는 네트워크에 배치하고 관리자에서는 에이젼트에 접속하여 서비스코드의 URL을 넘겨주면, 에이젼트는 해당 URL에서 서비스코드를 다운로드받아서 실행시킨 결과를 관리자에게 돌려주는 분산컴퓨팅 환경을 자바로 구현해본다. 관리자 측 컴퓨터에서 여러 에이젼트에 접속하여 서비스코드의 수행결과를 통보 받을 수 있다. 서비스를 받기 위하여 소프트웨어를 설치하거나 또는 배포하는 등의 작업이 필요 없이, 해당 서비스를 제공하는 서비스코드를 개발하여 임의의 위치에 배치하고, 에이젼트에게 코드 저장소의 위치를

알려 줌으로써 서비스를 받을 수 있다. 따라서 소프트웨어의 설치, 배포 등의 복잡한 절차가 필요 없이 언제라도 경량의 서비스코드를 개발하여 에이전트로부터 서비스를 받을 수 있을 뿐만 아니라 실행시간에 서비스코드의 갱신 등 좀 더 유연한 분산컴퓨팅 환경을 구현할 수 있다. 자바 어플리케이션의 경우와 자바 애플릿의 경우로 나누어서 제시한다.

2.2.1 자바 어플리케이션인 경우

다음 [그림 1]은 자바 어플리케이션인 경우의 제안 모델이다.



[그림 1] 서비스 요청 구조

[Fig. 1] Service request architecture

서비스가 요청되고 실행되는 순서는 다음과 같다.

① 관리자에서는 에이전트의 IP 주소로 접속한다.

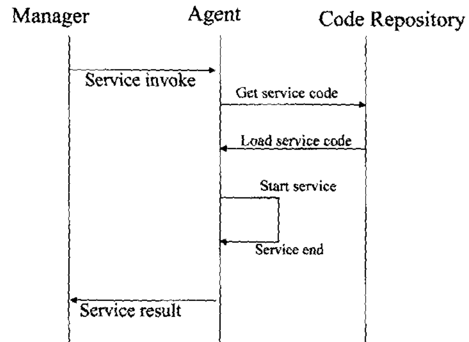
② 에이전트에서는 서버소켓을 열어 놓고(listen), 기다리고 있다가 관리자에서 접속해 오면 생성된 클라이언트 소켓을 매개변수로 하여 새로운 스레드를 발사한다.

③ 관리자와 스레드 사이에 커넥션을 확립하고, 커넥션을 통하여 코드 저장소(code repository)의 URL을 에이전트에 넘겨준다.

④ 에이전트는 코드저장소의 URL에서 클래스코드를 다운로드받는다.

⑤ 에이전트는 다운로드된 클래스코드를 실행시켜, 그 결과를 ③에서 확립된 커넥션을 통하여 관리자로 넘겨준다.

위와 같은 서비스 요청 구조를 메시지 교환 시퀀스로 나타내면 다음 [그림 2] 와 같다.



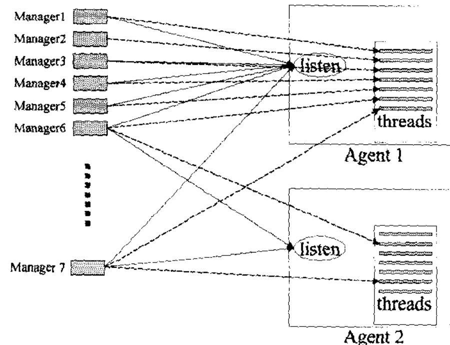
[그림 2] 메시지 교환 시퀀스

[Fig. 2] Message exchange sequence

예를들면, 관리자에서 아래와 같이 클라이언트 어플리케이션을 실행시킨다.

java ComputeClient 127.0.0.1
http://user.chollian.net/~jd7332/HelloAgainService.class

에이전트의 IP address 과 Code Repository의 URL 이 ComputeClient의 명령행 인자로 전달된다. 다음 [그림 3]에서 다수의 관리자가 여러 에이전트에 접속하여 서비스를 요청하고 있다.



[그림 3] 여러 관리자에서 다수의 에이전트로 서비스요청

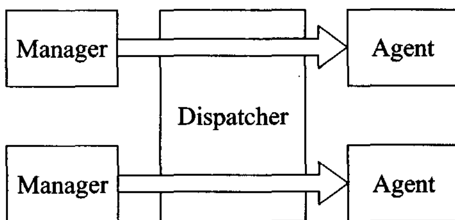
[Fig. 3] Service request from multiple manager to agent

다수의 관리자가 여러 에이전트에 접속하여 별도의 독립된 스레드로 서비스가 처리되고 있다. 에이

젼트는 서버소켓을 열어놓고 관리자가 접속해 오기를 기다리고 있다. 관리자가 접속하면, 에이전트는 별도의 클라이언트 소켓을 할당하여 독립된 스레드로 관리자의 요청을 처리한다. 예를 들면, 관리자 6와 관리자 7 은 에이전트 1 와 에이전트 2 모두 접속하여 서비스를 요청하고 있다.

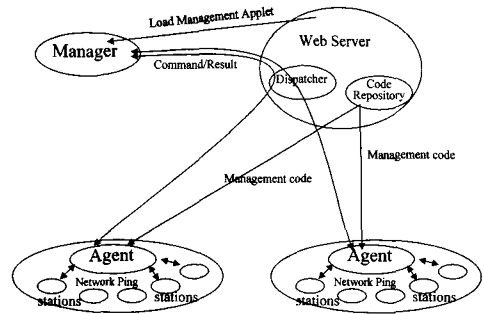
2.2.2 자바 애플릿인 경우

애플릿인 경우에는 보안관리자에서 예외를 발생시키기 때문에 애플릿코드가 있었던 웹서버이외에는 접속할 수 없다. 따라서 애플릿이 원격지에 있는 에이전트에 접속하기 위해서는 관리자와 에이전트 사이에 디스패쳐 (dispatcher)를 둔다. 관리자-디스패쳐-에이전트는 클라이언트가 서버와 직접 통신하지 않는다. 대신에, 관리자는 디스패쳐 컴포넌트에 접속하고, 디스패쳐 컴포넌트는 관리자의 요청을 에이전트로 전달해 주는 역할을 한다. [그림 4]와 같이 관리자와 에이전트 사이의 중간 계층인 디스패쳐 컴포넌트를 이용하여 위치 투명성을 확보한다.



[그림 4] 관리자-디스패쳐-에이전트 패턴
[Fig. 4] Manager-dispatcher-agent pattern

그리고 디스패쳐는 웹서버에서 실행되고 있는 일종의 게이트웨이 프로그램이다. 예를 들면, [그림 5]에서와 같이 관리자에서 dispatcher를 경유하여 여러 에이전트에 접속하여 서비스를 요청하고 있다.



[그림 5] 모니터링 관리구조
[Fig. 5] Monitoring management architecture

위의 [그림 5]에서 관리자는 디스패쳐를 경유하여 다수의 에이전트에 접속함으로써 각 서브넷의 망에 관한 정보를 에이전트로부터 통보 받을 수 있다.

자바 어플리케이션의 경우와는 달리 에이전트의 IP address 과 Code Repository 의 URL은 HTML문서의 applet tag의 parameter 변수로 제공된다.

```
<applet codebase="." code="JComputeClientApplet.class"
   width=950 height=500>
 <param name=agent value="203.246.81.246">
 <param name=dispatcher value="203.246.81.201">
 <param name=service

 value=http://user.chollian.net/~jd7332/HelloAgainService.
 class>
</applet>
```

위와 같이 agent의 IP 주소, dispatcher의 IP 주소 그리고 서비스코드의 URL이 applet tag의 parameter 변수로 제공된다.

2.3 RMI와 본 논문의 제안모델과의 비교

비슷한 일을 하는 서비스코드를 RMI를 이용해 개발하여 실제 네트워크에 배치한다고 가정하면, 인터페이스를 설계하는 것에서부터 시작하여, 인터페이스 구현객체 소스코드 작성, 컴파일하고 그리고 RMI 컴파일러(rmic)를 이용해 스텁(stub), 스켈레톤(skeleton) 객체 생성, 해당되는 객체가 클라이언트에서 접근이 가능하도록 classpath 관리 및 배치, RMI registry의 시작, 해당 원격객체의 naming registry에

의 등록, 서버 측 어플리케이션 시작, 클라이언트 측 어플리케이션 시작 등 서버측에서 상당히 복잡한 설치 및 관리과정을 요구한다.

이에 반하여 동일한 일을 하는 서비스코드를 본 논문의 분산컴퓨팅 환경을 이용해 개발하여 실제 네트워크에 배치한다면, 단지 이미 정의된 Service 인터페이스를 구현하는 서비스코드를 개발 컴파일하여 임의의 위치에 배치하고, 곧 바로 에이전트에게 코드저장소의 위치만 알려주는 것 이외의 다른 과정을 요구하지 않는다. 이와 같이 RMI와는 달리 클라이언트 측 어플리케이션 시작으로 에이전트에 서비스코드가 적재되어 매니저측에서 서비스를 받을 수 있으므로 서버 측 컴퓨터에 원격객체를 설치 또는 관리하는 등의 복잡한 작업이 필요 없게 되어 소프트웨어 컴포넌트의 개발과 배치전략이라는 측면에서 RMI와 비교할 때 훨씬 이점이 많다.

2.4 프로그램 흐름

2.4.1 자바 어플리케이션인 경우

우선 에이전트는 서버소켓을 열어놓고 클라이언트에서 접속해 오기를 기다리고 있다.(listen). 관리자에서는 에이전트의 IP 주소로 접속하여 커넥션을 확립한다. 그러면 에이전트에서는 확립된 커넥션의 클라이언트 소켓을 매개변수로 하여 별도의 독립된 스레드로 서비스요청을 처리한다. 그 다음 관리자 측에서는 확립된 커넥션을 통하여 코드 저장소의 URL을 에이전트에게 넘겨준다. 에이전트 측에서는 코드 저장소의 URL을 읽어 들인 후 URL이 http://로 시작하는 문자열이면 서비스코드를 해당 URL에서 다운로드받고, 아닌 경우 에이전트의 캐쉬에서 서비스코드를 끌어온다. 그 다음 에이전트는 서비스코드의 start() 메소드를 호출하여 서비스코드를 실행시킨다. 서비스코드의 수행결과는 확립된 커넥션을 통하여 관리자 측에 돌려진다. 그 후 관리자 측과 에이전트 측에서 해당 소켓을 닫고 서비스를 종료한다.

2.4.2 자바 애플릿의 경우

애플릿의 경우에는 보안관리자에서 예외을 발하므로 애플릿자신이 다운로드 된 웹 서버이외의 IP 주소로는 접속할 수 없다. 따라서 관리자 측에서 직접 에이전트에 접속하는 것이 아니라 애플릿이 다운로드 된 웹 서버에서 수행되고 있는 디스패쳐에 접속하여 에이전트의 IP 주소를 넘겨준다. 그러면 다시 디스패쳐는 넘겨받은 에이전트의 IP 주소를 이용하여 에이전트로 접속한다. 이와 같이 관리자는 디스패쳐를 중계로 하여 에이전트와 커넥션(connection)을 확립한다. 관리자는 이와 같이 확립된 커넥션을 통하여 코드저장소의 위치를 에이전트에 넘겨주고, 에이전트로부터 수행 결과를 돌려 받는다.

3. 자바 에이전트 모듈 구현

3.1 에이전트에서 수행되는 서비스 코드가 구현하는 인터페이스

에이전트에서 수행되기를 원하는 서비스가 반드시 구현해야 할 기본 인터페이스부터 정의하자. Service 인터페이스를 구현한 서비스 코드가 코드 저장소로부터 에이전트로 다운로드 되어 실행될 때 외부에 노출되는 인터페이스는 오직 Service에서 정의한 인터페이스만 알려진다. 에이전트 쪽에서는 오직 Service 인터페이스를 구현한 서비스코드만이 수행될 수 있는 환경을 구성해 놓으면 서비스코드 개발자는 인터페이스 설계에 신경을 쓰지 않고 오직 코드개발에만 전념할 수 있으며, 또한 서비스코드의 관리와 배포전략에도 많은 이점이 있다. 특히 클라이언트에서 서버의 메소드를 호출하기도 쉽고 코드를 읽고 이해하기도 훨씬 쉬워진다.[4]

```
public interface Service extends java.io.Serializable
{
    public void start( ) throws Exception;
    public void stop()  throws Exception;
    public void suspend() throws Exception;
    public void resume() throws Exception;
    public String getStatus();
    public String getID();
}
```

위와 같이 Service 인터페이스는 서비스의 시작과 멈춤을 위한 start( ) 와 stop( ), 중지와 재기동을 위

한 suspend( ) 와 resume( ) 서비스의 상태를 얻기 위한 getStatus( ), 서비스의 ID를 얻기 위한 getID( ) 메소드를 갖는다.

3.2 서비스 랩퍼 클래스가 구현해야하는 인터페이스

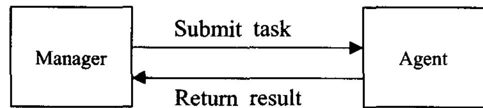
에이젼트에는 하나의 서비스코드만 수행되는 것이 아니다. 여러 관리자에서 다양한 서비스요청들이 들어올 수 있으므로 에이젼트에 적재되어 수행되고 있는 여러 서비스코드들이 원활하게 수행될 수 있도록 인터페이스 Service를 둘러싼 랩퍼 클래스인 Server를 둔다. 예를 들면, 서비스코드가 에이젼트에 적재되어 수행될 때 서비스코드의 초기화 작업, 서비스 메소드 호출이 무한루프에 빠지거나 또는 어떤 이유에서든 블록 되었을 때 에이젼트시스템 전체가 블록되는 것을 미연에 방지하는 등의 작업이 필요할 수 있다. 반드시 이 Server를 통해 Service 인터페이스에 접근하도록 함으로써 갑작스러운 서비스 오류로부터 전체 시스템을 보호할 수 있다. 이와 같은 패턴을 프락시 (proxy) 패턴이라고 하며, 이와 같이 다른 객체에 대한 게이트웨이로서 다른 객체 인스턴스를 정의하는 것을 말한다. 이와 같은 프락시 패턴을 이용해서, 에이젼트의 응답성과 강인성을 증진시킬 수 있다.

```
public interface IServer extends java.io.Serializable
{
   public boolean start();
   public boolean stop();
   public boolean suspend();
   public boolean resume();
   public void terminate();
   public String getStatus();
   public String getID();
}
```

위와 같이 모든 메소드가 리턴 타입을 제외하고는 Service 인터페이스와 동일하다. 단 하나 다른 것은 terminate( ) 메소드가 하나 더 있다. 이것은 에이젼트에 적재된 서비스가 일정한 시간, 예를 들면 15초 동안에 초기화과정을 마치고 (독립된 스레드로 수행되면서) 리턴 해야 되는 데, 주어진 시간 안에 초기화 과정이 실패하면 해당 서비스를 삭제한다.

3.3 서비스코드를 실행시키는 에이젼트를 구현하는 모듈

ServerManager는 서버 안에서 수행되고 있는 원격객체로, 클라이언트로부터 업무를 받아서 (클래스코드의 형태로 코드저장소에서 다운로드받는다), 해당 서비스코드를 실행시켜서 결과를 다시 클라이언트에게 돌려주는 에이젼트이다. ServerManager 에이젼트는 에이젼트에 적재되어 수행되고 있는 서비스코드에 대하여 사전에 아무런 정보도 가지고 있지 않다. 즉 에이젼트는 서버 측 컴퓨터에서 돌고 있는 일종의 컴퓨팅 엔진이다. 다음 [그림 6]과 같이 에이젼트에 임의의 서비스코드를 던져 넣으면 ,어떤 서비스코드인지 관심도 없지만, 알지도 못하고 무조건 실행시켜서, 결과를 관리자에게 돌려주는 일종의 컴퓨팅 엔진이다.



[그림 6] 관리자에서 에이젼트에 서비스요청
[Fig. 6] service request from manager to agent

Client에서 에이젼트에 작업을 의뢰하기 위해서는 에이젼트가 이해하고 있는 방식으로 서비스코드를 구현해야한다. 따라서 개발자는 서비스코드를 개발할 때, 반드시 정해진 인터페이스 즉 Service 인터페이스를 구현하도록 서비스코드를 개발해야한다. 그리고 에이젼트는 Service 인터페이스를 구현하는 서비스코드만을 실행시킬 수 있도록 환경을 구성한다. 이와 같은 식으로 서비스코드가 구현해야 할 인터페이스를 정해 줌으로써, 개발자는 인터페이스 개발에 신경을 쓰지 않고 오직 코드개발에만 전념할 수 있으며, 또한 서비스코드의 관리와 배포전략에도 많은 이점이 있다.[4]

이와 같은 형태로 다수의 클라이언트가 에이젼트에 서비스를 요청하고 결과를 돌려 받을 수 있도록 하기 위하여 에이젼트는 다수의 서비스코드를 적재하고, 실행시키고, 또는 중지시키고 재가동 시키고,

그리고 제거하는 등 서비스코드를 관리할 수 있는 환경을 갖추어 놓아야 한다. 이와 같이 서비스코드를 관리하기 위하여 에이전트 ServerManager가 반드시 갖추어야하는 메소드 들, 즉 인터페이스 설계와 그 인터페이스를 구현하는 클래스 설계가 필요하다. ServerManager 가 반드시 구현해야 하는 인터페이스가 다음의 IServerManager 이다.

```
public interface IServerManager
{
    public void shutdown();
    public IServer loadService(Service svc);
    public IServer addService(String svcName);
    public void removeService(String instanceID);
    public void terminateService(String instanceID);
    public String[] getServices();
    public IServer getService(String instanceID);
    public void log(String msg);
    public void error(String msg);
}
```

shutdown( ) 메소드는 에이전트 자기 자신을 위한 메소드로 에이전트가 작동을 멈추기 전에 이미 서비스하던 모든 서비스들을 정리해 주는 역할을 한다. 즉 Dictionary 타입의 변수인 m_servers에서 현재 서비스중인 각각의 IServer 인스턴스를 뽑아내고, 각각에 대하여 removeService( ) 메소드를 호출한다. loadService( )는 Service 인스턴스를 ServerManager의 해시 테이블에 추가한다.

addService( )는 인자로 넘어온 Service 인스턴스를 랩퍼 클래스인 IServer 인스턴스로 한번 감싸서, 그 IServer 인스턴스를 ServerManager의 해시 테이블에 추가하고 Service 인스턴스의 start( ) 메소드를 호출하여 해당 서비스코드를 실행시킨다. 만약 어떤 이유로 서비스 실행이 실패하면 removeService를 호출하여 서비스를 에이전트 ServerManager에서 삭제한다.

getServices( )는 해시 테이블에 등록되어 있는 서비스들의 이름을 얻는다. getService( )는 해당 ID를 갖는 서비스를 해시테이블에서 추출한다. removeService( )는 해당 ID를 갖는 메소드를 현재 실행상태이면 stop( ) 메소드를 호출해 줌으로써 IServer 인스턴스에게 서비스를 정상적으로 멈출 수 있는 기회를 제공한다. 그리고 해시테이블에서 해당 서비스를 제거한다. terminateService( )는 해당 ID를 갖는 메소드를 해시 테이블에서 제거한다. log( )와 error( )는 로그와 에러를 표준출력, 표준에러로 각각 내보낸다.

3.4 분산컴퓨팅 환경의 관리자 측 어플리케이션

분산컴퓨팅 환경의 관리자 측 어플리케이션이다. 명령행인수로 에이전트의 IP 주소와 Code Repository의 URL을 받아들인다. 에이전트로의 커넥션을 확립하며, 확립된 커넥션을 통하여 코드 저장소의 URL을 출력스트림에 쓰며, 입력스트림으로부터 에이전트의 처리 결과를 얻는다.

3.5 네트워크 클래스 로더

네트워크 클래스 로더는 자바의 ClassLoader를 확장한 클래스로 네트워크를 통하여 클래스를 로드하기 위하여 반드시 ClassLoader 의 loadClass 메소드를 재 정의하여야만 한다. 일단 클래스를 구성하는 바이트들을 내려 받은 후에 defineClass 메소드를 사용하여 Class 인스턴스를 생성한다.

3.6 웹 서버에서 수행되는 디스패쳐

애플릿인 경우에 애플릿코드가 있는 웹서버이외에는 보안관리자에서 예외를 발하므로 접속할 수 없다. 따라서 관리자와 에이전트 사이에 gateway를 두어 중계 역할을 한다. gateway는 웹서버에서 실행되고 있는 디스패쳐 역할을 하는 프로그램이다. 우선 관리자 PC의 JVM 안으로 다운로드 된 애플릿은 웹서버의 정해진 포트에서 수행되고 있는 gateway에 접속한다.

4. 실험 및 검토

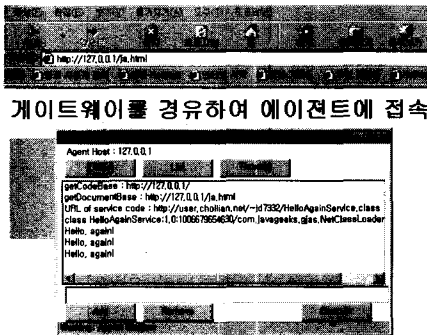
4.1 자바 애플릿인 경우

본 실험에서 웹서버는 마이크로소프트의 PWS (Personal Web Server)를 사용하였다. 에이전트는 네트워크의 어떤 IP 주소에서 기동되어 있다. 애플릿은

에이전트에 직접 접속할 수 없으므로, 우선 웹서버가 돌고 있는 컴퓨터에서 명령 java gateway를 내려 디스패처 역할을 하게 한다. 웹브라우저에서 웹서버의 ja.html 라는 이름의 HTML문서를 요청한다고 하자. html 문서의 내용은 다음과 같다.

```
<HTML>
<HEAD>
  <h1> 게이트웨이를 경유하여 에이전트에 접속 </h1>
</HEAD>
<APPLET CODE=JComputeClientApp.class >
   <PARAM name=agent  value="127.0.0.1">
   <PARAM name=dispatcher  value="127.0.0.1">
   <PARAM name=service value="http://user.chollian.net/~jd7332/HelloAgainSer
   </APPLET>
</HTML>
```

위의 HTML 코드에서 웹서버에서 다운로드 되는 자바애플릿은 JComputeClientApp.class 이다. 그리고 PARAM 매개변수로 agent, dispatcher 의 IP 주소가 지정되어 있다. PARAM service 에는 코드저장소의 URL이 지정되어있다. 애플릿은 디스패처를 중계로 하여 에이전트에 접속하게 된다. 웹브라우저의 URL 창에서 http://127.0.0.1/ja.html과 같이 입력하면 HTML 코드의 Applet 태그에서 지정된 자바 애플릿이 실행되어 [그림 7]과 같이 UI (user interface) 창이 뜬다.



[그림 7] 애플릿이 실행되어 HelloAgainService를 요청한 화면

[Fig. 7] HelloAgainService request view from applet

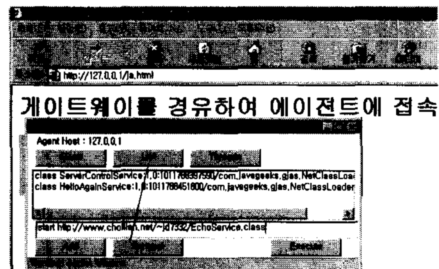
[그림 7]에서 getCodeBase는 애플릿의 위치를 나타내고, getDocumentBase는 HTML 파일이 있는 위치를 나타낸다. 또, URL of service code는 서비스코드의 URL을 나타낸다. 위의 화면에서는 에이전트의 HelloAgainService 서비스에서 보내오는 Hello, again! 라는 인사말이 디스패처를 중계로 하여 자바 애플릿의 UI 화면에 나타나고 있다.

4.2 웹브라우저를 에이전트의 서비스들을 제어하기 위한 콘솔로 사용

웹브라우저를 에이전트의 서비스들을 제어하기 위한 콘솔로 사용하기 위해서는 아래 코드에서와 같이 html 파일에서 PARAM매개변수의 service의 값으로써 에이전트의 서비스들을 제어하는 서비스인 ServerControlService를 지정해야 한다.

```
<HTML>
<HEAD>
  <h1> 게이트웨이를 경유하여 에이전트에 접속 </h1>
</HEAD>
<APPLET CODE=JComputeClientApp.class >
   <PARAM name=agent  value="127.0.0.1">
   <PARAM name=dispatcher  value="127.0.0.1">
   <PARAM name=service value=  value="http://192.168.0.91/ServerControlS
   </APPLET>
</HTML>
```

[그림 8]은 웹브라우저에서 HTML문서 (http://127.0.0.1/ja.html)를 요청한 실행화면이다.



[그림 8] 애플릿 제어화면에서 EchoService 요청 화면

[Fig. 8] EchoService request view from applet control service

[그림 8]의 애플릿 제어화면에서는 List 버튼을 클릭하여 에이전트에서 실행되고 있는 서비스들의 리스트가 출력되어 있다. 또한 아래의 입력 란에 에이전트에서 실행시키길 원하는 서비스 (start http://user.chollian.net/~jd7332/EchoService)를 입력하여 실행시킬 수 있다.

4.3 본 분산컴퓨팅 환경의 응용분야

지역적으로 분산된 환경기초시설이 급격히 증가함에 따라 이러한 기초시설들로부터 발생하는 데이터의 효율적이고 효과적인 운영 및 관리에 대한 관심은 크게 증가하고 있다. 특히, 지리적으로 분산된 하·폐수 처리장에서 발생되는 다양한 종류의 실시간 데이터에 대한 관리와 이러한 데이터를 처리, 분석하여 필요한 정보를 즉시 각 처리장에 제공함으로서 처리장 사이의 정보 공유는 물론 감독기관의 각 처리장에 대한 원격 제어를 손쉽게 할 수 있는 실시간 원격 제어 시스템 대한 필요성이 요청되고 있다. 이러한 상황에서 각 처리장에서는 실시간으로 발생되는 데이터를 DB에 저장한다고 가정하면, 각 처리장의 DB에서 데이터를 추출, 처리하여 결과를 관리자에게 송신하는 서비스코드를 개발하여 각 처리장의 에이전트에 적재하여 놓으면 관리자는 인터넷에 연결된 임의의 장소에서 각 처리장의 정보를 파악할 수 있을 뿐만 아니라 각 처리장의 환경설정 변수 값의 수치를 원격으로 제어할 수도 있을 것이다.

5. 결론 및 향후 연구과제

본 논문을 이용하여 다음과 같은 이점이 있는 좀 더 유연하고 시스템 독립적인 분산 컴퓨팅 환경을 구현할 수 있다. 첫째, 자바로 프로그래밍 하였으므로 플랫폼 독립적인 분산 컴퓨팅 환경을 구현할 수 있다. 둘째, 서비스를 받을 필요가 있을 때, 작고 경량의 서비스코드를 제작하여 네트워크의 임의의 위치에 배치시키고, 에이전트에 서비스코드의 URL을 알려주는 것만으로 관리자는 에이전트로부터 서비스를 받을 수 있다. 이와 같이 RMI와는 달리 클라이언트 측 어플리케이션 시작으로 에이전트에 서비스코드가 적재되어 매니져측에서 서비스를 받을 수 있으므로 서버 측 컴퓨터에 원격객체를 설치 또는 관리하는 등의 복잡한 작업이 필요 없게 되어 소프트웨어 컴포넌트의 개발과 배치전략이라는 측면에서 RMI와 비교할 때 훨씬 이점이 많다. 세째, 코드가 갱신되었을 때 갱신된 서비스코드를 네트워크의 임의의 위치에 배치시키고 URL을 에이전트에 알려주는 것만으로 갱신된 서비스코드를 에이전트에서 수행시킬 수 있다. 이와 같이 서비스코드를 갱신시키기 위하여 서버를 정지시키거나 하는 등의 복잡한 과정이 필요 없다.

본 논문은 자바를 이용한 분산 컴퓨팅 환경의 구현에만 초점이 맞추어 졌기 때문에 클래스 코드가 에이전트로 다운될 때 아무런 보안이나 인증 절차 없이 에이전트로 로드되어 실행되도록 하였다. 따라서 악의적인 클래스코드가 수행되는 것을 방지할 아무런 장치가 없다. 이 점은 자바의 보안관리자를 이용하여 앞으로 보완해야 할 과제이다.

※ 참고문헌

[1] Danny B. Lange and mitsuru Oshima, Programming and Deploying Java Mobile Agent with Aglets, Addison Wesley, 1998.

[2] Communication of the ACM Journal, "Intelligent Agents," Vol37, No 7. pp 18-21, 1994

[3] 마이크로소프트웨어 2000. 11월호 분산객체 기술의 개념, pp 227-230 2000.

[4] Ted Neward, Server-based Java Programming, Manning Publications, 1998.

[5] Danny B. Lange, Mitsuru Oshima, Programming and deploying JAVA Mobile Agents with Aglets, Addison-Wesley, 1998.

[6] William R. Cockayne, Michael Zyda, Mobile Agents, Manning 1998.

[7] http://java.sun.com/docs/books/tutorial/rmi/overview.html

[8] http://topaz.kaist.ac.kr/manual/jdk1.3/docs/guide/rmi/

서 건 원

2000년3월-2002년3월 현재
서울산업대학교 산업대학원
컴퓨터공학과(공학석사)
1998년3월-2002년2월
온수고등학교 교사
2002년3월-현재
경기고등학교 교사

이 길 홍

1989년2월 연세대학교
전자공학과 졸업(공학사)
1991년2월 연세대학교 대학원
전자공학과 졸업(공학석사)
1991년-1995년 LG정보통신
연구소네트워크그룹
1999년8월 연세대학교 대학원
전기컴퓨터공학과 졸업
(공학박사)
2000년5월-현재 서울산업대학교
컴퓨터공학과 전임강사