로컬 의미에 의한 원격객체 설계 기법

김윤호*
*안동대학교

A technique for creating remote objects by local semantics

Yun-Ho Kim*
*Andong NationalUniversity
E-mail : unokim@andong.ac.kr

요 약

본 논문에서는 로컬 의미만으로 원격객체 생성을 할 수 있는 기법을 제안하고자 한다. 즉, 로컬 인터페이스와 같은 메소드 시크니쳐의 형태를 가진 원격 인터페이스를 생성함으로써, 애플리케이션 서버 인터페이스와 RMI runtime 사이의 레이어를 인터페이스 형태로 생성시켜준다. 본 논문에서 제안하는 RMI 지역 기법을 도입하면, RMI application의 코딩작업을 간소화할 수 있게 된다.

ABSTRACT

This paper presents a technique that allows objects written with local semantics to be made remote. It works by creating interfaces that have roughly the same method signatures as the local interfaces but are remote. These generated interfaces form a layer between the application server interfaces and the RMI runtime. This technique results in the coding of tediously repeated try/catch blocks for the RMI remote interface.

키워드
RMI, distributed object system, java, internet computing

Ⅰ. 서 론

분산 객체 시스템에서 객체들을 RMI [1]를 사용하여 배포하기 위하여는 원격 접근을 전제하여 애플리케이션을 설계하여야 한다. 이때 원격으로 접근되는 인터페이스들은 java.rmi.Remote 클래스를 확장하여 설계하여야 하고, 메소드들은 java.rmi.-RemoteException 예외를 발생시킬 수 있도록 해야 한다. 즉, 서버 측에서는 이 인터페이스가 사용되는 부분마다 RemoteException을 처리하는 try/catch 블록을 설정해 주어야 하며, 클라이언트 측에서도 이 인터페이스를 가진 객체들이 사용되는 부분마다 RemoteException을 처리하는 try/catch 블록을 설정해 주게 된다. 결과적으로 애플리케이션 소스 코드 내에 RemoteException을 처리하는 try/catch 블록들이 필요한 곳곳마다 빈번하게 반복되어 나타나게 되어, 프로그램 작성 시에 많은 시간과 노력이 필요할 뿐 아니라 이로 인하여 프로그램의 가독성이 현저히 떨어지게 된다. (비효율적인 코딩의 전형적인 한 예가 된다.)

이러한 문제는 클래스를 선언할 때 원격 메소드에 'throws RemoteException 절'을 둠으로 해서 발생하는 것이다. 즉, RemoteException은 애플리케이션 자체에서 발생될 수 있는 예외가 아니라 자바 가상머신이 원격 접근을 시도할 때에 발생될 수 있는 것이다. 그럼에도 불구하고 이러한 예외에 대한 처리를 애플리케이션의 설계에 직접적으로 반영시킴으로써 문제가 발생하는 것이다. 그러므로, 애플리케이션 자체의 설계와는 독립적으로 이러한 예외를 처리하는 방안이 요구된다.

본 논문에서는 애플리케이션의 설계 시에 로컬 의미로서 작성된 객체들이 원격의 기능을 가지도록 하는 기법을 제시하고자 한다. 즉, 원격 접근이 필요한 인터페이스를 설계할 때에 원격이 아닌 지역 인터페이스의 형태로 작성되지만 실행시에 시

그니쳐가 거의 동일한 형태로 생성되도록 하여, 결과적으로 클래스 작성 시에 RemoteException을 처리하는 것과 동일한 효과를 가지되 코드 작성 상에 try/catch 블록의 설정을 제거할 수 있도록 하고자 하는 것이다.

II. RMI RemoteException의 처리

II.1 RMI 원격 인터페이스

원격으로 접근되는 인터페이스들은 java.rmi.Remote 클래스를 확장하여 설계하여야 하고, 메소드들은 java.rmi.RemoteException 예외를 발생시킬 수 있도록 해야 한다 [2]. 서버 측에서는 이 인터페이스가 사용되는 부분마다 RemoteException을 처리하는 try/catch 블록을 설정해 주어야 하며, 클라이언트 측에서도 이 인터페이스를 가진 객체들이 사용되는 부분마다 RemoteException을 처리하는 try/catch 블록을 설정해 주게 된다. 예로서 클라이언트에서 정보를 요청하면 서버에서 요청된 정보를 보내주는 간단한 RMI 애플리케이션을 고려한다.

먼저 서버측을 보면, Remote 클래스를 확장한 InterfaceSvr, 이 인터페이스를 구현한 SvrImpl 클래스, 그리고 서버 역할을 하는 Server 클래스로 구성된다. 이들의 구현된 형태의 예는 다음과 같다.

```
public interface InterfaceSvr extends Remote {
        public String _inf()
                throws RemoteException;
}

public class SvrImpl extends
UnicastRemoteObject
                inplements InterfaceSvr {
        public String _inf ()
                throws RemoteException {
         return "Information";
        }
}

public class Server {
  public static void main(Stringg[] args)
                throws Exception {
    String hostName =

java.net.InetAddress.getLocalHost().getHostName(
);
      int port = ...;
      String regName = "//" + hostName +
        ":" + port +"/" + "Server";
      SvrImpl sv = new SvrImpl();
      Naming.rebind(regName, sv);
  }
}
```

클라이언트 측에서는 Client 클래스 하나로 처리된다. 즉, 다음의 예와 같이 Client 클래스의 main() 메소드에서 InterfaceSvr에 대한 원격 참조가 가능하다.

```
public class Client {
  private InferfaceSvr cl;
  public Client (Interface_Svr is) { cl = is; }
  public static void main(...) throws Exception
{
  String hostName =

java.net.InetAddress.getLocalHost().getHostName(
);
    int port = ...;
   String regName = "//" + hostName +
      ":" + port +"/" + "Server";
    InterfaceSvr ic =
      ( I n t e r f a c e S v r )
Naming.lookup(regName);
    Client c = new Client(ic);
  }
}
```

II.2 원격 인터페이스의 지역적 설계

본 논문에서 제시하는 방법은 애플리케이션 인터페이스와 RMI runtime [3]사이에 단순히 하나의 레이어를 설정하는 데에 기반하고 있다. 즉, InterfaceSvr와 SvrImpl을 지역적 개념으로 다시 작성하고, 각각에 상응하는 원격 인터페이스 RemoteInterfaceSvr과 그 구현인 RemoteSvrImpl을 생성하도록 하는 것이다. 이들에 대한 예는 다음과 같다.

```
public interface InterfaceSvr {
  public String _inf();
}

public class SvrImpl implements InterfaceSvr {
  public String _inf() {
    return "Information";
  }
}

public interface RemoteInterfaceSvr
                        extends Remote {
  public String remote_inf()
                throws RemoteException
}
```

앞서의 예와 비교를 해보면 InterfaceSvr와 SvrImpl에서 Remote 클래스와 RemoteException, UnicastRemoteObject 클래스와 같은 원격 접근과 예외 처리를 위한 내용이 표현되지 않으며 SvrImpl의 메소드 내에서도 RemoteException을 발생시키는 부분도 빠지게 된다. RemoteInterfaceSvr 인터페이스에 이러한 내용들이 집합되어 반영되어 있음을 알 수 있다. RemoteInterfaceSvr 인터페이스의 구현은 다음과 같다.

```
public class RemoteSvrImpl
                extends UnicastRemoteObject
        implements RemoteInterfaceSvr {
  private InterfaceSvr w;
  public RemoteSvrImpl(InterfaceSvr is)
                throws RemoteException {
    w = is;
  }
  public string remote_inf()
                throws RemoteException {
    return w._inf();
  }
}
```

여기서 RemoteInterfaceSvr의 메소드는 InterfaceSvr의 메소드와 각각 대응된다. 또한, RemoteInterfaceSvr 인터페이스는 InterfaceSvr의 처리를 지연시키는 역할을 한다.

RemoteInterfaceSvr를 사용한 Server 프로그램은 앞서 InterfaceSvr를 사용한 것과 다음과 같이 약간의 차이가 있으며, RMI runtime과 작용하게 된다.

```
public class Server {
  public static void main(...) throws Exception
{
    String hostName =

java.net.InetAddress.getLocalHost().getHostName(
);
    int port = ...;
      String regName = "//" + hostName +
       ":" + port +"/" + "Server";
    ImplSvr s = new ImplSvr();
    RemoteInterfaceSvr rs = new
RemoteImplSvr();
    Naming.rebind(regName, rs);
  }
}
```

클라이언트 측에서 InterfaceSvr를 구현하는 프록시 객체를 사용은 하지만 메소드 호출은 RemoteInterfaceSvr에서 하도록 다음과 같이 한다.

```
public class ProxySvr implements InterfaceSvr
{
  private RemoteInterfaceSvr remoteSvr;
  public ProxySvr(RemoteInterfaceSvr ris) {
    remoteSvr = ris;
  }
  pubic String _inf() {
    try{
      return remoteSvr.remote_inf();
    } catch (RemoteException e) {
      e.PrintStackTrace();
      System.exit(0);
    }
  }
}
```

ProxySvr를 설정할 경우 Client 프로그램은 다음과 같이 된다.

```
public class Client {
  private InterfaceSvr a;
  public Client(InterfaceSvr ia) { a = ia; }
  public static void main(...) throws Exception
{
    String hostName =

java.net.InetAddress.getLocalHost().getHostName(
);

    int port = ...;
      String regName = "//" + hostName +
       ":" + port +"/" + "Server";
   RemoteInterfaceSvr rema =
    (RemoteInterfaceSvr)
Naming.lookup(regName);
    InterfaceSvr px = new ProxySvr(rema);
    Client c = new Client(px);
    ....
  }
}
```

이상에서 보는 바와 같이 지역적인 관점과 원격적인 관점을 분리하여 코딩함으로써 애플리케이션 클래스들을 지역적인 관점에서 프로그램할 수 있게 된다. 즉, InterfaceSvr, ImplSvr, Client는 지역적인 관점에서 코딩이 가능하여 효율적인 코딩을 할 수 있다.

전형적인 원격 애플리케이션에서는 많은 원격 인터페이스가 존재하게 된다. 주 서버 인터페이스에 대한 참조는 lookup 메카니즘에 의해 이루어지며, 원격 참조에 의해 다른 원격 인터페이스에 대한 참조가 이루어진다. 예를 들어 아래와 같이 InterfaceSvr가 다른 인터페이스 InterfaceLocal에

대한 참조를 반환하는 메소드를 가진다 하자.

```
public InterfaceLocal _infl();
```

그러면 RemoteInterfaceLocal과 그의 구현인 ImplLocal은 앞서와 같이 지역적 관점에 따라 설계하고, InterfaceLocal을 구현하고 RemoteInterfaceLocal에서 생성되는 ProxyLocal 클래스를 설계하면 된다. _infl()의 구현을 할 때 ProxySvr가 ProxyLocal을 반환하도록 하려면 다음과 같이 하면 된다. 즉, RemoteInterfaceSvr의 remote_infl() 메소드에서 InterfaceLocal이 아니라 RemoteInterfaceLocal을 반환하도록 하는 것이다. 그리고 ProxySvr에서 _infl()메소드를 다음과 같이 작성한다.

```
public InterfaceLocal _infl() {
  RemoteInterfaceLocal ril = null;
  try {
    ril = new ProxyLocal(rema.remote_infl());
  } catch (RemoteException e) {
    ......
  }
  return ril;
}
```

III. 결 론

본 논문에서는 애플리케이션의 설계 시에 로컬의미로서 작성된 객체들이 원격의 기능을 가지도록 하는 기법을 제시하였다. 즉, 인터페이스에서 원격 접근이 요구되는 부분과 그렇지 않은 부분을 분리하여, 원격 접근과 관련된 부분을 별도로 설계함으로써 나머지 부분은 지역적인 관점에서 설계를 하면 되도록 하였다. 이렇게 함으로써 원격 인터페이스에서 Remote 클래스와 RemoteException을 처리하기 위하여 지루하게 반복되는 throw RemoteException 절과 try/catch 블록에 의한 RemoteException 처리를 위한 코드 부분을 코드에 반복되어 나타나지 않게 하였다. 따라서, 코드 작업상의 효율성의 증대와 프로그램 가독성이 높아지게 하는 효과가 기대된다.

참고 문헌

[1] Campione, et. al, The Java Tutorial Continued: The Rest of the JDK, Addison-Wesley, 1998.
[2] Horstmann and Cornell, Core Java 2, Volumn II: Advanced, Parentice-Hall, 2002.
[3] Seshadri, "Fundamentals of RMI," jGuru Short Courses, jGuru, 2000.