Jini 기반의 인터페이스 공유 서비스
(An Interface Sharing Service based on Jini)

문 창 주 † 문 석 현 †† 백 두 권 †††
(Chang-Joo Moon) (Suck-Hyun Moon) (Doo-Kwon Baik)

요 약 자바 기반의 분산객체 시스템을 개발 혹은 사용할 때, 시스템 개발자는 사전에 정의된 인터페이스를 구현한 분산객체를 네트워크상에 배치(deploy)해야 하고, 시스템 사용자는 분산객체에 대한 참조(reference)를 획득하고 인터페이스 타입으로 변환한다. 이러한 작업을 위해서 인터페이스의 공유는 필수적이다. 본 논문에서는 Jini 시스템 개발 시 정보 공유의 불편함을 해결하고자 IRSJ(interface repository service based on jini)를 제안한다. IRSJ는 Jini 시스템과 인터페이스 공유 서비스를 위한 자바 클래스들로 구성된다. IRSJ는 근거리에서 작업할 때 검색(lookup) 서버나 작성자의 정보 없이도 인터페이스 파일을 검색할 수 있다. 원거리에서 개발 팀들이 공동 작업을 수행한다면 검색(lookup) 서비스가 실행되고 있는 서버의 주소만으로도 인터페이스 검색이 가능하다. 검색에 실패한 인터페이스 객체는 원격 이벤트 모델을 사용하여 더 이상의 검색 요청을 하지 않아도 인터페이스를 다운로드 받아 사용할 수 있게 하였다. 메시지 서비스는 인터페이스 검색에 실패하면 IRSJ를 사용하는 개발팀원 모두에게 자동적으로 인터페이스 요청 메시지를 전송하여 빠른 시간 안에 인터페이스 공유가 가능하도록 설계되었다.
키워드 : 인터페이스 공유, JINI, 분산객체, 메시지 서비스

Abstract When we develop and use the java-based distributed object system, the system developer must deploy distributed object, which is implemented to predefined interface, to network and the system user acquires reference about distributed object and changes reference to interface type. Sharing of interface is essential for these works. In this paper, we propose the IRSJ(interface repository service based on jini) to improve efficiency of interface sharing at the JINI system development. the IRSJ is consisted of the java classes for JINI system and interface sharing service. If development teams perform collaborative work in short distance, the IRSJ can search interface file without information of lookup server or interface writer. If development teams perform collaborative work in long distance, the IRSJ can search interface with only an address of server that executes lookup service. Although the interface object not requires additional search, the interface object that failed search interface can download interface and use interface using remote event model. When the message service fails in interface search, the message service sends interface request message to a development team member who use the IRSJ automatically, so that interface sharing may be possible in short time.
Key words : *Interface Sharing, JINI, Distributed Object, Message Service*

1. 서 론

자바 분산객체(distributed object)를 기반으로 하는 서비스 시스템을 구축할 때, 먼저 분산객체에 접근 할 수 있는 인터페이스를 정의하고 서비스 사용자들에게 공개한다. 서비스 제공자는 정의된 인터페이스를 구현(implement)한 분산객체를 네트워크상에 배치한다. 배치된 분산객체들이 자신에게 접근하는 사용자들에게 구현된 서비스를 제공한다. 서비스 사용자는 분산객체에 대한 참조를 획득하고 인터페이스 타입으로 변환한다. 그 다음 인터페이스에 선언된 메소드(method)의 형식에 맞추어 메소드를 호출함으로써 분산 객체가 제공하는

† 비 회 원 : 고려대학교 컴퓨터학과
mcj@swsys2.korea.ac.kr
†† 비 회 원 : Net Brain 핵심망연구팀 연구원
shmoon90@net-brain.co.kr
††† 종신회원 : 고려대학교 컴퓨터학과 교수
baikdk@korea.ac.kr
논문접수 2001년 5월 25일
심사완료 2003년 3월 14일

서비스를 이용한다[1,2].

서비스 제공자와 사용자 모두 구현과 형 변환을 위해서 사전에 정의된 인터페이스의 공유가 필요하다. 예를 들어 사용자측 객체를 구현할 때 획득된 참조를 원격 인터페이스 타입으로 형 변환을 실행하게 된다. 이때 자바 컴파일러는 원격 인터페이스의 정보를 읽어들이고, 이 정보를 기반으로 하여 사용자측 객체 클래스 파일을 컴파일 한다. 만약 원격 인터페이스 파일이 존재하지 않는다면 컴파일은 불가능하다. 따라서 분산 객체의 인터페이스 정보는 런타임과 컴파일타임에 꼭 필요한 요소이다.

분산 객체 시스템을 개발할 때 인터페이스를 이용하기 위한 방법으로는 프로그래머 각자가 인터페이스를 구현하여 사용하는 방법이 존재한다. 이 방법은 동일한 인터페이스를 여러 사람이 중복 개발을 하여야 하며, 오타로 인한 에러가 발생할 가능성이 크다. 이 방법 외에 인터페이스를 공유하기 위해 사용하는 방법으로는 FTP, 전자 메일, 데이타 베이스, 네트워크 드라이브와 같은 방식으로 인터페이스 정보를 공유하고 있다. 이 방법들은 인터페이스 정보가 저장되어 있는 서버 주소와 같은 정보나 인터페이스를 누가 작성했는지를 알아야만 인터페이스를 검색할 수 있다. 뿐만 아니라 이와 같은 방식의 인터페이스 공유는 인터페이스의 다운로드가 실패하는 경우, 다른 서버를 검색하거나 검색에 실패한 서버에 대하여 임의의 시간이 흐른 뒤에 인터페이스의 등록 여부를 지속적으로 검색하여야 하는 단점이 존재한다.

본 논문에서는 Jini를 기반으로 하여 분산 객체 시스템을 개발하는 개발팀원간에 인터페이스 작성자에 대한 정보 없이 인터페이스 공유를 보다 효율적이며, 자동적으로 공유할 수 있도록 IRSJ(interface repository service based on jini)를 제안한다. IRSJ는 SUN사(Sun Microsystems)에서 제공하는 분산 시스템의 인프라(Infra)인 Jini 시스템[3,4,5,6]과 인터페이스를 보다 효율적이고 쉽게 공유할 수 있도록 제안하는 자바 클래스들로 구성된다. Jini 시스템의 검색(lookup) 서비스[3,4,5,6]는 네트워크 상에 존재하면서 서비스의 프록시(proxy) 객체를 검색(lookup) 서비스에 저장 및 관리하며, 사용자가 요청한 프록시 객체를 검색 및 사본을 전송하는 매개자로서의 역할을 실행한다. IRSJ는 Jini 시스템의 이러한 특징을 인터페이스 공유에 적합하게 사용할 수 있도록 설계하였다. IRSJ 사용자는 멀티캐스팅 패킷의 전송범위 안에서 작업하는 팀원들간에 인터페이스 작성자에 관한 정보나 혹은 인터페이스가 저장되어 있는 서버에 관한 정보 없이 인터페이스를 공유할 수 있도록 한다. 인터페이스 검색이 실패한 경우 원격 이벤트 모델을 사용하여 자동적으로 인터페이스 다운로드가 가능하며, 메시지 서비스에 의하여 어떤 인터페이스를 필요로 하는지의 정보를 개발자에게 통보할 수 있다. 뿐만 아니라 인터페이스의 이름을 정확하게 기억하지 못하는 경우라고 하더라도 검색(lookup) 서비스에 등록된 모든 인터페이스 객체를 검색함으로써 인터페이스 정보를 공유할 수도 있다.

본 논문은 다음과 같다. 2 장에서는 관련 연구에 대해서 알아보고, 3 장에서는 설계 시 고려된 사항에 대하여 논한다. 4 장에서는 IRSJ의 구조와 흐름에 대하여 설명한다. 5 장에서는 IRSJ의 설계와 실행 결과에 대하여 살펴본다. 6 장에서는 IRSJ와 기존의 인터페이스 공유 방식을 비교 평가한다. 마지막으로 7 장에서는 결론과 향후과제에 대하여 논한다.

2. 관련연구

2.1 용어(Terminology)

1) 발견(discovery) : Jini 시스템을 기반으로 하는 서비스나 사용자들이 네트워크에 접속하여 처음에 수행하여야 하는 과정이다. 네트워크에 존재하는 검색(lookup)서비스를 찾기 위한 과정이며, MRP (Multicast Request Protocol), UDP(User Datagram Protocol), MAP (Multicast Announcement Protocol) 등의 프로토콜을 사용한다. MRP와 UDP는 서비스 제공자나 서비스 이용자로부터 시작되는 프로토콜이다[5,6,7,8].
2) 검색(lookup) 서비스 : RMI(Remote Method Invocation)의 rmiRegistry처럼 분산 객체에 대한 정보를 저장하고 있으며, 사용자로부터의 요구를 받아들여 분산 객체를 검색하고 전송하는 역할을 담당한다. 검색(lookup) 서비스는 자신이 서비스하고 있는 객체들의 유효성 검사를 위하여 임대기간(lease)를 검사한다[5,6,7,8].
3) 임대기간(lease) : 검색(lookup) 서비스에 등록된 프록시 객체들이 검색(lookup) 서비스를 받을 수 있는 계약 시간이다. 검색(lookup) 서비스는 자원의 효율적 사용을 위하여 등록된 객체에 대하여 시간 계약을 수행하고 주기적으로 계약을 갱신토록 강요하고 있다[5,6,7,8].
4) 서비스 제공자 : 분산 객체 시스템에서 임의의 서비스를 제공하는 프로그램이나 장치 혹은 사람을 서비스 제공자라고 정의한다[5,6,7,8].
5) 서비스 사용자 : 서비스 제공자가 제공하는 서비스

를 사용하는 프로그램이나 사람을 서비스 사용자라 정의한다. 서비스 제공자와 서비스 사용자는 수행하는 작업에 따라 역할을 바꿀 수 있다[5,6,7,8].

2.2 인터페이스 공유 관련연구

자바 기반의 분산 객체 시스템을 개발할 때 인터페이스의 공유는 필수적이다. Jini 시스템의 경우 각종 장치들을 네트워크에 연결하고 사용하는 과정의 자동화를 주목적으로 삼고 있다. 따라서 장치들이 제공하는 서비스의 사용방식을 표준화하기 위해서는 인터페이스를 표준화하고, 공유하여야 하며 Jini를 이용하여 시스템을 개발하는 개발자들의 모임인 Jini Community에서 인터페이스의 표준화와 공유에 대한 연구를 진행중이다[9].

인터페이스를 공유를 위해 FTP, 전자 메일, 데이타베이스, 네트워크 드라이브 등을 이용하는 방법은 통용되는 방법이기는 하지만, 학문적인 연구 대상으로써 부적합하여 논문으로는 발표되지는 않았다. CORBA(common object request broker architecture)경우는 인터페이스를 정의하기 위해 IDL(interface definition language) 이라는 별도의 언어를 이용하여 인터페이스를 정의한다[10,11]. IDL에 의해 정의된 인터페이스를 각 언어에 의존적인 컴파일러를 이용하여 컴파일 함으로써 언어에 적합한 인터페이스를 생성한다. 생성된 인터페이스는 IR(interface repository)에 저장되고 이 정보를 이용하여 각 사용자들은 분산 시스템에서의 서비스를 사용하게 된다[12,13]. IR은 CORBA가 사용하는 인터페이스 저장소로서 인터페이스의 정보를 관리하는 서비스이다. CORBA에서 인터페이스 정보는 주로 런타임의 공유를 목적으로 한다. 그러나 각 벤더들이 작성한 특정 컴파일러를 이용하여 컴파일타임에 동적으로 인터페이스 정보를 공유할 수도 있다[10,11,14]. 이 방법 역시 IR이 동작하고 있는 서버의 정보를 알아야만 인터페이스를 공유할 수 있으며, CORBA를 이용하기 위하여 ORB(object request broker)의 초기화와 같은 복잡한 과정을 거쳐야만 한다. 그러나 이 방법은 CORBA에 종속되어 있으므로 CORBA 이외의 다른 시스템을 이용하여 분산 객체 시스템을 구현하는 경우 CORBA의 IR을 사용할 수 없다는 단점이 존재한다.

3. IRSJ 설계의 고려사항

3.1 인터페이스의 흐름

Jini의 검색(lookup) 서비스는 자바 객체를 대상으로 서비스를 제공한다. 그러나 IRSJ는 인터페이스 클래스 파일을 전송하고, 이를 검색하여 서비스 사용자의 저장장치에 영구히 저장한다. 따라서 인터페이스 파일의 자료형태를 변환하지 않으면 Jini의 검색(lookup) 서비스를 바로 사용할 수는 없다. 즉 저장장치에 파일로 존재하는 인터페이스를 자바 객체로 변환하는 과정이 필요하다. 본 논문에서 Wrapper관련 클래스들이 파일 형태의 인터페이스와 자바 객체 형태의 인터페이스간의 변환 기능을 수행한다. 다음 <그림 1>은 서비스 제공자로부터 서비스 사용자에게 인터페이스 파일이 전송되어 저장되는 과정에서 인터페이스의 형태 변화를 나타낸다.

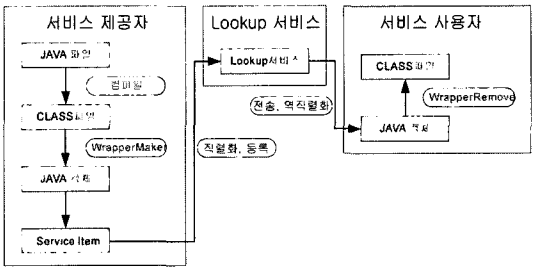

그림 1 IRSJ에서의 인터페이스 형태 변화

<그림 1>에서 서비스 제공자는 인터페이스 파일을 생성하고 컴파일 하여 인터페이스 클래스 파일을 생성한다. 검색(lookup) 서비스에 인터페이스를 등록하기 위하여 인터페이스 클래스 파일을 자바 객체로 변환해야 한다. IRSJ는 인터페이스 클래스 파일을 자바의 객체로 만들기 위하여 파일을 읽어 바이트 배열로 파일의 내용을 저장 한 후, 배열과 인터페이스 이름을 포함하는 ServiceItem객체를 생성한다. ServiceItem객체는 직렬화 되고 검색(lookup) 서비스에 등록된다.

서비스 사용자는 인터페이스의 이름을 이용하여 원하는 인터페이스 객체의 검색을 검색(lookup) 서비스에 요구한다. 검색 요구를 받은 검색(lookup) 서비스는 해당 인터페이스의 Wrapper객체를 검색하여 검색된 객체를 서비스 사용자에게 전달한다. 서비스 사용자는 전송받은 객체를 역직렬화 하고 WrapperRem 클래스를 이용하여 자바의 클래스 파일인지의 여부를 검증한 후 파일형태로 저장한다.

3.2 발견(discovery) 과정

Jini 시스템의 MRP는 멀티캐스팅 패킷을 이용하여 검색(lookup) 서비스를 검색하므로 이 목적에 적합하다. 단 문제가 되는 것은 네트워크 관리자가 라우터를 어떻게 설정하는가에 따라 패킷이 전송되는 범위가 결정된다는 것이다. IRSJ는 인터페이스의 표준화를 위하여 세

계적으로 공유하기로 결정된 인터페이스가 존재하는 경우 혹은 프로그래머가 속한 회사에 유용한 정보가 존재하는 경우 이 정보 역시 쉽게 다운로드받아 사용할 수 있어야 한다. 이 기능은 Jini의 UDP를 이용한다. UDP를 이용하는 것은 서버의 주소를 명시적으로 제공하여야 한다는 단점이 존재 하지만, 발견(discovery)[3,4,5,6] 과정을 수행하기 위한 패킷이 네트워크에 폭주하는 것을 막을 수 있다.

IRSJ에서 MRP와 UDP의 사용은 사용자가 서버의 주소를 제시하는지의 여부에 따라 결정된다. 만약 사용자가 서버의 주소를 입력하지 않았다면 MRP를 이용하여 한 세그먼트 안에서 검색(lookup)서비스를 자동 검색하도록 실행되고, 서버의 주소를 입력한 경우에는 UDP를 이용하여 입력된 주소의 특정 서버만을 검색한다. 뿐만 아니라 IRSJ가 실행된 후 활성화되는 검색(lookup) 서비스에도 IRSJ는 반응하여야 한다. 이를 위하여 IRSJ는 검색(lookup) 서비스로부터 시작되는 발견(discovery) 프로토콜인 MAP에도 반응하도록 설계하였다. MAP는 검색(lookup) 서비스가 네트워크에 접속한 후 검색(lookup) 서비스의 정보를 담은 멀티캐스팅 패킷을 네트워크에 전송하는 것으로부터 시작되는 프로토콜이다.

3.3 다중 검색(lookup) 서비스

하나의 세그먼트 안에 하나 이상의 검색(lookup) 서비스가 실행될 수 있다. <그림 2>는 IRSJ가 다중 검색(lookup) 서비스에 대하여 인터페이스를 등록하고, 다운로드 받는 방식을 표현한 그림이다. 다중 검색(lookup) 서비스가 실행되는 경우 하나의 검색(lookup) 서비스에 인터페이스를 등록하고 사용자는 모든 검색(lookup) 서비스를 검색하여 인터페이스를 다운로드 받는 것이 가능하다. 그러나 이 경우 인터페이스를 등록 받은 검색(lookup) 서버가 다운되면 인터페이스 사용자는 더 이상 인터페이스를 다운로드 받을 수 없게 된다. 뿐만 아니라 인터페이스 객체의 손상에 대하여 적절히 대처할 수 없다는 단점이 존재한다.

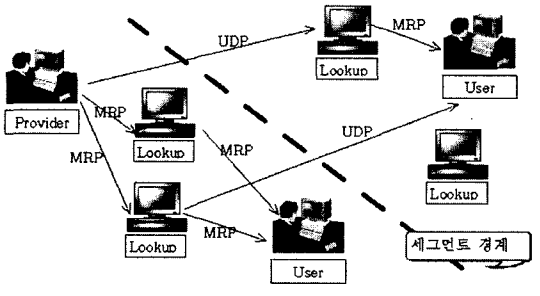

그림 2 다중 검색(lookup) 서비스

인터페이스 제공자는 모든 검색(lookup) 서비스에 인터페이스를 등록하고 사용자가 처음으로 검색되는 검색(lookup) 서비스로부터 인터페이스 정보를 다운로드 받도록 설계하는 것도 가능하다. 그러나 이 경우 역시 인터페이스 정보가 손상된 경우 해당 검색(lookup) 서비스를 제외한 다른 검색(lookup) 서비스에게 다시 인터페이스를 검색하도록 요구하여야 하는 비효율적인 상황이 발생하게 된다.

IRSJ는 MRP를 사용하여 검색되는 모든 검색(lookup) 서비스에 인터페이스를 등록하고 사용자는 모든 검색(lookup) 서비스에 인터페이스를 검색 요청하고 다운로드 받은 후 사용자에게 인터페이스의 목록을 리스트로 제시하고 사용자가 직접 리스트에서 선택하도록 설계하였다. 이렇게 설계함으로서 하나 이상의 검색(lookup) 서비스가 동작하는 환경에서 IRSJ는 안정적인 서비스를 제공할 수 있다. UDP는 특정 검색(lookup) 서비스를 지정하고 사용하기 위한 프로토콜이다. 따라서 UDP의 경우에는 사용자가 명시한 주소에서 실행되는 검색(lookup) 서비스 하나에만 인터페이스를 등록하게 된다. 인터페이스를 다운로드받는 경우에도 UDP를 사용한다면 특정 검색(lookup) 서비스만을 검색할 수 있으며, TCP(Transmission Control Protocol) 연결을 이용하므로 거리의 제약이 없다.

3.4 인터페이스 검색 방법

인터페이스 이름을 정확하게 아는 경우는 인터페이스 이름을 제시하여 정확한 인터페이스 검색이 가능하다. 그러나 경우에 따라서는 인터페이스 이름이 정확하지 않거나 모르는 경우가 발생할 수도 있다. 이 경우 IRSJ는 검색(lookup) 서비스에 등록되어 있는 모든 인터페이스 객체를 검색 할 수 있어야 한다. Jini의 검색은 자바 클래스의 타입을 기반으로 검색 과정을 수행한다. 보다 정확하고 세부적인 검색을 위하여 속성 값을 저장하는 엔트리(entry)객체를 사용한다. IRSJ를 이용하여 인터페이스 객체를 등록할 때 인터페이스 타입은 Wrapper 클래스가 구현하는 인터페이스 타입으로 등록된다. 인터페이스 이름은 Name이라는 엔트리(entry)의 서브클래스를 타입의 객체로 생성하여 등록한다. 이 성질을 이용하여 인터페이스 이름을 명확하게 제시하고 검색을 시도하는 경우 정확하게 일치되는 인터페이스 객체만을 반환하도록 하였다.

인터페이스 이름을 제시하지 않고 검색을 요구하는 경우 검색(lookup) 서비스에 Wrapper 클래스의 인터페

이스 타입만을 전달하고 검색을 요구함으로써 검색(lookup) 서비스에 등록되어 있는 모든 객체를 다운로드 받을 수 있도록 하였다. 이렇게 다운로드 받아진 객체는 IRSJ의 사용자측 서비스에 리스트 형태로 목록을 표시하고 사용자가 직접 선택하여 저장할 수 있도록 설계하였다.

3.5 보안

동적으로 프로그램이나 객체를 다운로드 받는 분산 객체 시스템의 가장 큰 문제점은 보안성이다. 악의를 가진 프로그래머가 바이러스와 같은 프로그램을 등록하는 경우가 발생 할 수 있으며, 이 프로그램이 사용자의 저장장치에 기록되어 실행된다면 치명적인 사태가 발생할 수도 있다. 이 문제를 해결하기 위하여 WrapperRem 객체를 이용하여 파일로 저장할 때 저장하고자 하는 객체가 자바의 클래스 파일인가 검사한다. 만약 다운로드 받아진 객체가 자바 클래스가 아니라면 IRSJ는 파일로 저장하는 과정을 수행하지 않고 사용자에게 경고 메시지를 출력하도록 설계하였다. 이 기능은 자바의 클래스 파일을 나타내는 매직수(Magic Number)를 이용한다. 자바 클래스 파일은 항상 16진수로 "CAFEBABE"라는 숫자 열로 시작한다. 이 코드를 검사함으로써 자바 클래스 여부를 검사하도록 하였다.

Jini를 실행하기 위하여 사용하는 보안 관리자 역시 보안성을 강화하는 역할을 수행할 수 있다. 본 논문에서는 보다 편리하게 실험을 하기 위하여 모든 포트와 저장공간을 접근할 수 있도록 보안 정책 파일을 생성하였다. 그러나 실무에 적용하기 위하여 일정 포트만 공개하거나 저장공간의 특정 디렉토리(Directory)만 IRSJ가 접근 가능하도록 보안 정책 파일을 수정할 수 있다.

4. IRSJ 전체 구조 및 흐름

IRSJ는 인터페이스를 제공하는 인터페이스 제공 서비스 와 인터페이스를 검색 및 다운로드 하는 인터페이스 사용 서비스로 구성되며, Jini의 검색(lookup) 서비스는 네트워크에서 항상 실행되며 등록된 인터페이스 객체를 저장, 검색하는 역할을 수행한다. <그림 3>은 전체적인 인터페이스 등록과 검색에 대한 흐름도 이다.

IRSJ의 구성 요소들을 살펴보면 다음과 같다. 인터페이스 제공 서비스는 공유를 원하는 인터페이스를 검색(lookup) 서비스에 등록하여 공유 가능하게 하며, 인터페이스 사용 서비스는 검색(lookup) 서비스를 통해 원하는 인터페이스를 검색하고 다운로드받아 인터페이스를 공유한다. HTTPD는 동적으로 파일을 전송하기 위한 데몬(Daemon)[7,8]으로 작용한다. 본 논문에서는

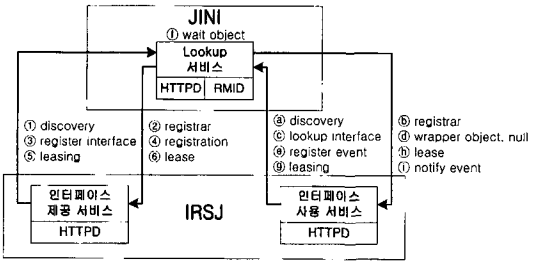

그림 3 인터페이스 등록과 검색에 대한 흐름도

Jini 시스템이 배포될 때 포함되어 있는 가장 기본적인 형태의 웹서버를 이용한다. 검색(lookup) 서비스의 RMID[7][8]는 검색(lookup) 서비스를 활성화 및 비 활성화시킴으로써 서버의 효율을 높이는 작업을 수행하며, 서버 이상으로 인하여 검색(lookup) 서비스가 다운된 경우 다운되기 이전의 상태를 기록하고 있다가 서비스를 복구하는 역할을 담당한다. 자세한 흐름을 살펴보기 위하여 숫자와 알파벳을 사용하였다. 숫자는 인터페이스 제공 서비스와 검색(lookup) 서비스간의 흐름 순서를 표현한다. 알파벳은 인터페이스 이용 서비스와 검색(lookup) 서비스간의 흐름 순서를 나타낸다. IRSJ는 검색(lookup) 서비스에 인터페이스가 등록되는 시점과 검색 요구를 받는 시점에 따라 아래와 같은 두 가지 형태의 흐름이 존재한다.

4.1 인터페이스 등록이 검색 보다 먼저 발생

인터페이스 제공자가 검색(lookup) 서비스에 인터페이스를 등록한 후 인터페이스 검색 요구가 발생하는 경우이다.

4.1.1 인터페이스 제공 서비스-검색(lookup) 서비스

<그림 3>에서 아라비아 숫자로 표시한 실행 흐름에 관하여 살펴보자.

① 인터페이스 제공 서비스는 사용자가 인터페이스 명을 입력하는 순간 네트워크에 존재하는 검색(lookup) 서비스를 찾기 위한 발견(discovery)과정을 수행한다. 사용자가 URL을 입력 여부에 따라 발견(discovery) 과정에서 사용하는 프로토콜이 결정된다.

② 발견(discovery) 패킷을 수신한 검색(lookup) 서비스는 패킷에 포함되어 있는 그룹 이름을 참조하여 자신이 서비스를 제공할 수 있는지의 여부를 파악한다. 만약 검색(lookup) 서비스가 서비스를 제공할 수 있는 그룹인 경우 자신의 프록시 객체인 ServiceRegistrar를 서비스 제공자에게 전달한다. ServiceRegistrar는 RMI의 스텁(stub) 객체이다.

③ ServiceRegistrar를 전송 받은 인터페이스 제공자는 검색(lookup) 서비스에 특정 객체를 등록하도록 요구할 수 있다. IRSJ에서는 인터페이스 객체를 등록하기 위해 WrapperMaker객체를 이용하여 인터페이스를 읽어들이고 바이트 배열에 저장한 후 Wrapper 객체를 생성한다. Wrapper객체는 보다 정밀한 검색을 가능하도록 하기 위하여 인터페이스 이름을 속성으로 생성하고 Wrapper객체와 속성을 나타내는 엔트리(entry)객체를 이용하여 ServiceItem객체를 생성한다. 이 객체가 검색(lookup) 서비스에 등록되는 객체이다.

④ 인터페이스 객체의 등록 요청을 받은 검색(lookup) 서비스는 남아있는 자원의 양과 서버의 상태 서비스 요구시간 등을 살펴보고 인터페이스 객체를 서비스 할 수 있는 서비스 보장 시간을 결정한다. 검색(lookup) 서비스가 결정한 서비스 보장 시간은 객체를 등록 요청할 때 같이 요청된 시간과 동일하거나 더 짧은 시간일 수 있다. 검색(lookup) 서비스는 등록 요청의 결과로서 서비스 보장 시간과 ServiceID를 포함하는 ServiceRegistration을 전송한다. 이 객체는 등록된 객체를 관리하기 위하여 인터페이스 제공 서비스에서 사용한다.

⑤ ServiceRegistration 객체를 전송 받은 인터페이스 제공 서비스는 임대기간(lease) 객체를 추출하여 임대기간(lease) 갱신을 위하여 저장한다. 임대기간(lease) 갱신에 실패한 인터페이스 객체는 검색(lookup) 서비스에서 제거되기 때문에 임대기간(lease)에 명시된 시간이 지나기 전에 인터페이스 제공 서비스는 임대기간(lease)를 갱신하는 작업을 수행한다.

⑥ 임대기간(lease) 갱신 요청을 받은 검색(lookup) 서비스는 다시 서버의 상태에 따라 최적의 서비스 지속 시간을 산출하고 이 값을 임대기간(lease) 객체로 생성하여 인터페이스 제공 서비스에 전달한다.

4.1.2 인터페이스 사용 서비스-검색(lookup) 서비스

ⓐⓑ 인터페이스 사용 서비스가 검색(lookup) 서비스를 이용하기 위하여 검색하고 프록시 객체를 얻는 발견(discovery) 과정이며 인터페이스 제공 서비스와 동일한 작업이다.

ⓒ 인터페이스 제공 서비스와는 달리 인터페이스 사용 서비스는 인터페이스를 검색하기 위하여 검색 요청용 ServiceTemplate 객체를 생성한다. 이 객체는 ServiceItem과 비슷한 구조를 지니고 있다. 사용자가 인터페이스 이름을 지정한 경우 인터페이스를 Wrapping 하고 있는 클래스의 인터페이스 타입을 주 검색 요소로 설정하고 인터페이스 이름을 상세 검색을 위한 속성 값으로 사용한다. 인터페이스 이름은 엔트리(entry) 객체로 생성되어 ServiceTemplate를 생성할 때 사용된다. Service Template 객체를 생성한 인터페이스 사용 서비스는 이 객체를 검색(lookup) 서비스에 전달하면서 검색을 요청한다.

ⓓ 인터페이스 검색 요청을 받은 검색(lookup) 서비스는 등록된 인터페이스 객체를 검색한다. 검색하는 과정의 효율성을 위하여 MashalledObject의 equals 메소드를 이용한다. MashalledObject는 직렬화 된 객체의 최상위 클래스로서 역할을 수행한다. 이 클래스의 equals 메소드는 직렬화 되어 있는 객체를 역직렬화 하지 않고 동치성의 여부를 검사할 수 있다. 따라서 직렬화 되어 있는 객체를 역직렬화 하기 위해 소요되는 시간을 절약한다. 검색된 인터페이스 객체는 Wrapper 클래스의 객체 형태로 사용자 서비스에 전달된다. 인터페이스의 Wrapper 클래스 객체를 전송 받은 인터페이스 사용자 서비스는 WrapperRem 객체를 이용하여 보안 검사를 수행하고 파일 형태로 저장한다.

인터페이스 등록이 검색 요구보다 먼저 발생하는 경우 반드시 검색에서 성공함으로 ⓓ 까지만 진행되고 종료된다.

4.2 인터페이스 등록보다 검색이 먼저 발생

인터페이스의 등록보다 검색 요구가 먼저 발생하면 검색(lookup) 서비스는 인터페이스를 검색 후 Null 값을 반환하게 된다. 인터페이스 제공 서비스와 검색(lookup) 서비스간의 흐름은 4.1.1절과 동일하다. 인터페이스 사용 서비스와 검색(lookup) 서비스간의 흐름은 4.1.2.의 ⓒ까지 동일하다. 이 절에서 다루는 내용은 서비스가 등록되기 전에 검색됨으로 검색에 실패하여 Null이 반환되는 경우를 처리해야 함으로 ⓓ부터 언급한다.

ⓓ 검색 요청을 받은 검색(lookup) 서비스는 자신이 저장하고 있는 객체들을 검색하여 일치하는 객체를 찾는다. 일치하는 객체가 찾아진 경우에는 검색된 객체를 전송함으로써 모든 절차가 종료된다. 그러나 검색을 요청 받은 객체와 일치하는 객체가 없다면 검색(lookup) 서비스는 이 사실을 통보하기 위하여 Null을 반환하게 된다.

ⓔ Null 값을 반환 받은 인터페이스 사용 서비스는 검색을 위하여 생성한 ServiceTemplate 객체와

관심 있는 이벤트의 종류, 이벤트를 처리할 클래스 등을 지정하고 이벤트 등록 객체를 생성한다. 또한 이벤트를 수신하고 처리하기 위한 이벤트 처리 객체도 생성하여야 한다. 이벤트 처리는 RMI를 이용하므로 스텁을 생성하여야 한다. 이벤트 등록 객체를 생성한 후 검색(lookup) 서비스의 프록시 객체의 notify 메소드를 호출하여 검색(lookup) 서비스에 생성한 이벤트 등록 객체를 전달하는 것으로 이벤트 등록 과정을 실행한다.

ⓕ 이벤트 등록 요청을 받은 검색(lookup) 서비스는 이벤트 객체를 저장하고 요청 받은 이벤트 상황이 발생하는지의 여부를 살피게 된다. 이벤트의 종류는 <표 1>과 같다.

표 1 Jini 이벤트 종류

종 류	발 생 상 황
NoMatch Match	이벤트가 등록되는 시점까지는 일치하는 객체가 없었는데 이벤트 등록 후 일치하는 객체가 등록되는 경우의 이벤트
Match Match	검색에 일치하는 객체가 존재하고 있었지만 객체가 갱신되는 경우의 이벤트
Match NoMatch	검색에 일치하는 객체가 검색(lookup) 서비스에서 제거되는 경우에 대한 이벤트

본 논문에서는 검색 실패의 경우에 NoMatch-Match 이벤트 형식을 사용하였다. 따라서 검색(lookup) 서비스는 인터페이스 객체가 등록될 때마다 이벤트로 등록된 객체에서 요구하는 객체와 일치하는가의 여부를 검증하는 작업을 수행하게 된다. 이벤트 등록 요청을 받은 검색(lookup) 서비스는 이벤트 관리를 위한 정보를 포함하고 있는 EventRegistration 객체를 반환한다.

ⓖ EventRegistration 객체를 전송 받은 인터페이스 사용 서비스는 이벤트 등록 객체의 관리를 위하여 임대기간(lease)정보를 추출하고 저장, 관리하기 위하여 LeaseManager 객체에 임대기간(lease) 객체를 전달한다. LeaseManager객체는 자신이 저장하고 있는 임대기간(lease) 객체들의 갱신이 실패하지 않도록 지속적으로 관리하고 검색(lookup) 서비스에 임대기간(lease)갱신 요구를 한다.

ⓗ 임대기간(lease) 갱신 요구를 받은 검색(lookup) 서비스는 이벤트 객체의 임대기간(lease)를 갱신하고 새롭게 설정된 서비스 지속시간을 반환한다.

ⓘ 검색(lookup) 서비스에 등록한 이벤트가 발생한 경우이다. 인터페이스 제공 서비스와 인터페이스 사용 서비스는 비동기 적으로 동작한다. 따라서 ⓘ의 과정은 검색(lookup) 서비스에 이벤트를 등록한 이후 언제든지 발생할 수 있는 과정이다. <그림 5>에서는 단지 편의를 위하여 마지막 과정으로 표시하였다.

이벤트 통보를 요청 받은 검색(lookup) 서비스는 지속적으로 등록 받은 이벤트에 해당하는 인터페이스가 자신에게 등록되는지의 여부를 감시한다. 만약 일치되는 인터페이스가 등록되면 검색(lookup) 서비스는 이벤트 등록을 요청한 객체에 이벤트가 발생하였다는 사실과 함께 인터페이스 객체의 복사본을 전달한다. 이벤트 발생을 통보 받은 이벤트 관리자의 행동은 해당 이벤트 객체에 대한 임대기간(lease) 관리를 중단함으로써 더 이상 검색(lookup) 서비스에 이벤트 객체가 자원을 점유하지 못하도록 제거하는 방법과 지속적으로 임대기간(lease)를 관리하여 이벤트 객체를 검색(lookup) 서비스에 지속적으로 남겨두는 방법이 존재한다. IRSJ의 경우 NoMatch-match 에 대한 이벤트를 이용하므로 이벤트가 발생된 이후에 지속적으로 검색(lookup) 서비스가 이벤트 객체를 관리할 필요가 없다. 따라서 이벤트 관리 객체로 하여금 임대기간(lease) 관리를 중단하도록 설계하였다. 만약 Match-Match 이벤트를 사용하는 경우라면 지속적으로 이벤트 등록 객체에 대한 임대기간(lease)를 관리하는 것이 더 효율적이다.

4.3 인터페이스 요청 메시지 서비스

인터페이스 검색에 실패하는 경우에 4.2절에서 설명한 내용 외에도 한가지 추가적으로 처리하여야 하는 작업이 있다. 인터페이스 작성자는 어떤 인터페이스의 검색 요청이 있었는지, 인터페이스 검색 요청의 성공여부 등을 전혀 모르고 있다. 따라서 인터페이스 검색 요청이 실패한 경우 요청자는 무한히 기다려야 하는 상황이 발생할 수도 있다. 이러한 문제점을 해결하기 위하여 인터페이스 검색이 실패한 경우 어떤 인터페이스를 필요로 하는지 개발자들에게 전달하고, 인터페이스 작성자로 하여금 인터페이스를 검색(lookup) 서비스에 등록하도록 요청하기 위한 수단이 필요하다. 이 작업을 개발자가 직접 통보하는 것은 기존의 인터페이스 공유 방식처럼 인터페이스 작성자에 대한 정보를 필요로 한다는 점에서 비효율적이다.

IRSJ는 검색 요청 후, 검색(lookup) 서비스가 Null 값을 반환하면 검색에 실패하였다고 판단하고 검색(lookup) 서비스에 이벤트를 등록한다. 이와 동시에 검색 요청에 사용하였던 인터페이스 이름을 이용하여 어떤 인터페이스를 필요로 하는지를 설명하는 메시지를 생성하고 검색(lookup) 서비스에 메시지를 등록한다. <그림 4>는 인터페이스 요청 메시지 서비스에 대하여

간략히 표현한 흐름도이다.

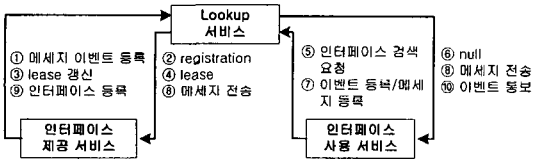

그림 4 메시지 서비스

<그림 4>에서는 편리상 인터페이스 제공 서비스와 인터페이스 사용 서비스를 분리하여 표시하였다. 메시지 서비스는 IRSJ가 실행됨과 동시에 실행되는 개별 서비스이다. 이렇게 설계하는 이유는 경우에 따라 인터페이스 사용자가 인터페이스 제공자가 될 수도 있기 때문이다. 메시지 서비스의 특징과 흐름에 대하여 간략히 살펴보면 다음과 같다.

① IRSJ는 검색(lookup) 서비스를 검색하기 위한 발견(discovery) 과정과 검색(lookup) 서비스의 프록시 객체인 ServiceRegistrar를 전송 받은 직후 검색(lookup) 서비스에 메시지 수신 이벤트를 등록하여야 한다. 이 과정에서 사용하는 이벤트 모델은 Match-Match구조를 사용한다.

② 메시지 이벤트 등록 요청에 대하여 검색(lookup) 서비스는 이벤트 등록 확인 객체인 EventRegistration을 전송한다.

③ 메시지 이벤트 등록 객체는 IRSJ가 종료될 때까지 지속적인 서비스를 받아야 한다. 따라서 임대기간(lease)에 의한 지속적인 관리가 필요하며 IRSJ는 주기적으로 임대기간(lease)를 갱신 요청한다.

④ 임대기간(lease) 갱신 요청에 대하여 검색(lookup) 서비스는 임대기간(lease)를 갱신하고 새로운 서비스 지속시간을 전송한다.

⑤ 인터페이스 사용 객체가 검색(lookup) 서비스에 인터페이스 검색을 요청한다.

⑥ 검색을 요청 받은 검색(lookup) 서비스는 자신이 저장하고 있는 객체들을 검사한다. 만약 인터페이스가 발견되면 인터페이스를 전송한다. 그러나 인터페이스가 존재하지 않는다면 Null 값을 검색 요청 객체에 전송하게 된다. 메시지는 검색(lookup) 서비스가 Null 값을 반환하는 경우에 전달된다.

⑦ Null을 반환 받은 인터페이스 사용 서비스는 4.2절에서 설명한 것과 같이 인터페이스 등록 감시를 위한 이벤트를 등록한다. 이때 자동적으로 인터페이스 요청 메시지를 작성하여 메시지도 검색(lookup) 서비스에 등록한다. 메시지 객체는 지속적으로 사용될 필요는 없다. 활성화되어 있는 IRSJ 서비스들만 메시지를 수신하고 일정 시간이 지난 뒤에는 메시지 객체는 파기되어야 한다. 따라서 메시지 등록 객체는 임대기간(lease)를 관리하지 않는다.

⑧ 검색(lookup) 서비스는 인터페이스 이벤트 등록 요청에 대하여 해당 인터페이스가 등록되는지를 감시하기 시작한다. 이와 동시에 현재 자신에게 메시지 이벤트를 등록한 모든 IRSJ 서비스에 인터페이스 요청 메시지를 객체를 전달한다.

⑨ IRSJ가 수신한 메시지는 사람에게 출력된다. 메시지를 확인한 개발자는 해당 인터페이스를 검색(lookup) 서비스에 등록한다. 이때 IRSJ의 인터페이스 제공 서비스를 사용한다.

⑩ 인터페이스 등록을 감지한 검색(lookup) 서비스는 인터페이스 요청 이벤트를 등록한 객체에 이벤트를 통보하고 인터페이스를 전송한다.

5. IRSJ 설계 및 실행 결과

본 장에서는 IRSJ의 세부 구조로서 서비스를 구성하는 클래스들의 구성과 클래스간의 흐름을 살펴보고 IRSJ를 실행하였을 때의 결과를 살펴보도록 하자.

5.1 IRSJ 설계

Jini의 검색(lookup) 서비스는 SUN사에서 제공하는 서비스이므로 검색(lookup) 서비스 자체에 대한 언급은 생략하고, IRSJ를 구성하는 클래스 중 본 논문을 작성중에 설계하였던 클래스들만 설명하도록 한다. IRSJ 는 하나의 사용자 인터페이스(GUI)를 가지고 있다. GUI와 IRSJ를 구성하는 각 서비스들간의 연결은 맵퍼 클래스에 의해 통제되고, 관리되도록 설계하였다. <그림 5>는 GUI와 각 서비스들을 연결하는 방식을 표현하는 구조도 이다. IRSJ 클래스는 단순히 그래픽 인터페이스 생성과 이벤트 처리만을 담당하는 클래스이다. IRSJ를 구성하는 각 서비스의 생성과 메시지 서비스의 시작을 관리하는 클래스는 Mapper 클래스이다. Mapper 클래스를 따로 둔 이유는 사용자의 입력의 유효성을 검증하고 검색(lookup) 서비스로부터 전달받은 인터페이스 객체들

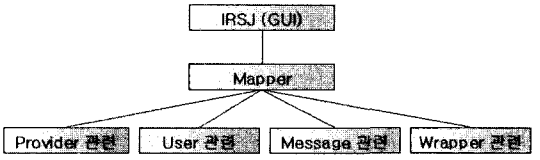

그림 5 IRSJ 서비스 객체 생성관계

의 임시로 저장하며, 인터페이스 제공 서비스 및 사용 서비스의 생명 주기를 관리하기 위함이다. <표 2>는 <그림 5>에 표시되어 있는 각 클래스들의 기능을 간략히 정리한 것이다. IRSJ 클래스와 Mapper 클래스는 단일 클래스이다. Message 관련 클래스들은 인터페이스 검색에 실패한 경우 사용하는 이벤트 모델과 매우 유사한 구조를 사용하며, 4.3절에서 흐름을 자세히 설명하였다. 따라서 이 클래스들을 제외하고 IRSJ를 구성하는 각 구성 요소별로 세부 설계를 살펴보도록 하자.

5.1.1 Provider, User 관련 클래스

<그림 6>은 Provider, User 관련 클래스들을 간략하게 표현한 클래스도 이다. <표 3>은 <그림 6>에 있는 클래스의 기능을 서술한 테이블이다.

5.1.2 Wrapper 관련 클래스

Wrapper 관련 클래스는 파일 형태의 인터페이스를 자바 객체로 변환하고 다시 파일로 변환하는 과정을 담당한다. <그림 7>은 Wrapper 관련 클래스 구성도이다. <그림 7>에서 Mapper와 ClassProvider는 객체 생성의 책임을 표시하기 위하여 삽입하였다. <표 4>는 Wrapper 관련 클래스들의 기능을 설명한 테이블이다.

표 2 클래스들의 기능

클 래 스	기 능
IRSJ	GUI 제공
Mapper	사용자 입력 검증 메시지 서비스 시작 및 관리 인터페이스 파일의 관리 인터페이스 제공자 서비스 생성 및 관리 인터페이스 사용자 서비스 생성 및 관리
Provider 관련	검색(lookup) 서비스와 연동 인터페이스 객체 등록 및 관리
User 관련	검색(lookup) 서비스와 연동 인터페이스 객체 검색 이벤트 객체 생성 및 관리
Message 관련	메시지 이벤트 등록 및 메시지 수신
Wrapper 관련	클래스 파일의 자바 객체화 인터페이스 객체의 보안 검사 인터페이스 객체를 파일로 저장

5.2 IRSJ의 실행

검색(lookup) 서비스가 임의의 위치에 실행되고 있다고 가정을 하고 IRSJ의 실행순서에 맞추어 결과화면을 설명하도록 하겠다. <그림 8>은 IRSJ의 등록할 인터페이스 이름을 입력받는 과정이다. 등록할 인터페이스 이

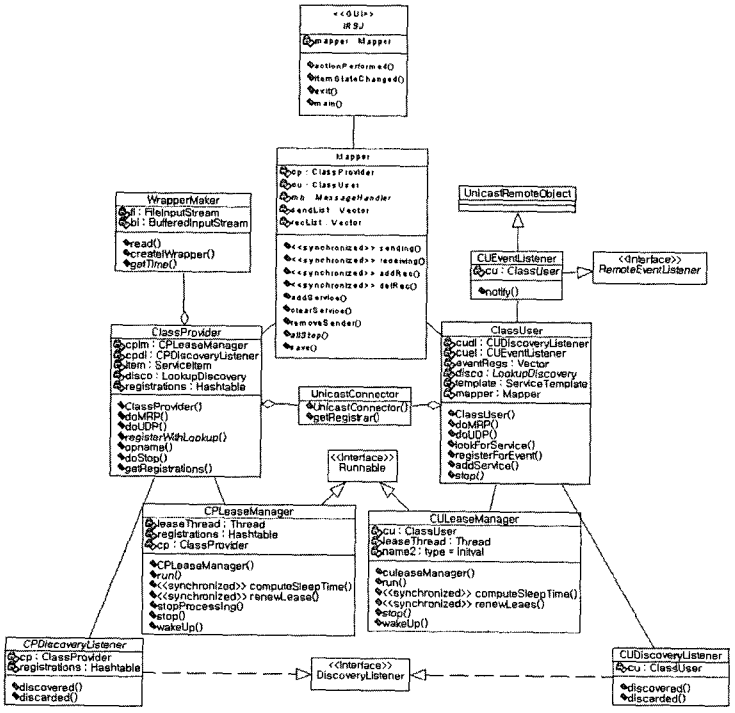

그림 6 ClassProvider, ClassUser 관련 클래스도

표 3 Provider, User 관련 클래스 설명

클래스	기 능
ClassProvider	- 서비스를 구현하기 위해 필요한 모든 객체들을 생성, 관리 - MRP와 UDP에 의한 발견(discovery) 실행 - IRSJ가 서비스하는 하나의 인터페이스 객체별로 ClassProvider 객체가 생성됨
CPDiscovery Listener	- MRP와 MAP에 대해서 비동기적으로 응답하고 관리하는 객체 - UDP는 검색(lookup) 서버와 동기적으로 실행되기 때문에 DiscoveryListener가 필요 없다.
CPLease Manager	- 검색(lookup) 서비스에 등록된 인터페이스 객체가 지속적으로 검색(lookup) 서비스를 받을 수 있도록 시간 계약의 갱신을 관리
UnicastConnector	- 사용자가 특정 검색(lookup) 서버의 주소를 명시한 경우에 생성됨 - UDP는 검색(lookup) 서버와 동기적으로 동작하기 때문에 주소 값을 잘못 입력한 경우나 혹은 네트워크의 이상으로 접속이 종료된 경우에는 무한히 기다려야 하는 단점이 있다. - 단점을 제거하기 위해서 일정 시간이 지나면 에러 메시지 출력과 함께 UDP 과정을 종료하도록 하였다. - ClassProvider와 ClassUser가 공용으로 사용
Class User	- 인터페이스 객체를 다운로드 받기 위한 과정을 담당 - MRP와 UDP를 사용하여 인터페이스 검색 - 검색에 성공한 경우 Mapper 클래스에 검색된 서비스 객체를 저장하고 실행을 종료한다. - 인터페이스 검색이 실패되고 null을 반환 받은 경우에는 이벤트와 메시지를 전송하는 역할을 담당 - 검색(lookup) 서비스에 등록된 이벤트는 30분 뒤 자동 제거함으로써 검색(lookup) 서비스의 자원을 최대한 보호한다.
CUEvent Listener	- 검색(lookup) 서비스가 통보하는 이벤트를 수신 - 인터페이스 검색에 실패한 경우에 생성 - Jini의 이벤트 모델이 RMI의 분산 이벤트 모델을 사용하기 때문에 UnicastRemote Obejct를 상속받고 RemoteEventListener 인터페이스를 구현한 형태 가지고 있다. - RMI 통신을 위한 스텁도 생성 한다.
CUDiscovery Listener	- ClassUser 클래스가 발견(discovery) 과정을 수행할 때 검색(lookup) 서비스로부터 프록시 객체를 수신 - CPDiscoveryListner와 유사한 기능과 구조 가진다
CULease Manager	- 이벤트 객체의 임대기간(lease)를 관리 - CPLeaseManager와 유사한 구조와 기능을 수행

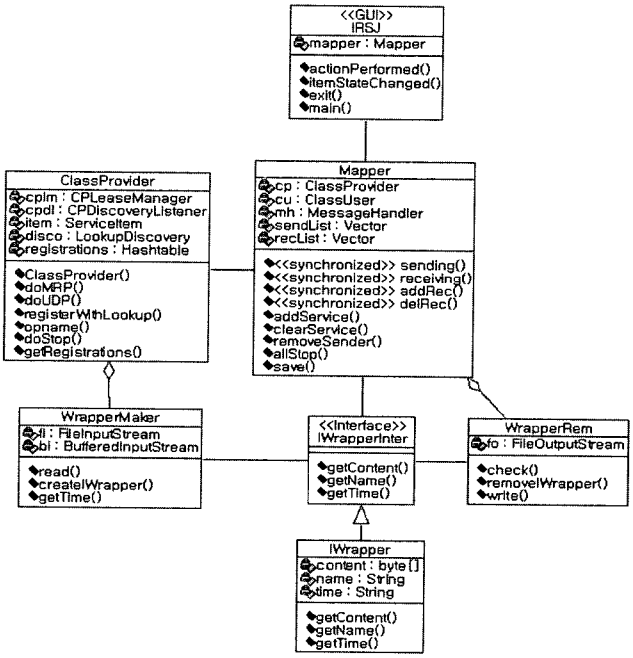

그림 7 Wrapper 관련 클래스도

표 4 Wrapper 관련 클래스 설명

클래스	기 능
IWrapperInter	- 실제로 생성되는 IWrapper의 인터페이스 - 인터페이스 제공 서비스와 사용 서비스간에 Wrapper 객체를 접근하고 사용하는 표준화된 방법 제공
IWrapper	- 파일로 저장되어 있는 인터페이스 파일을 읽어들이고 이를 저장 - 실제 인터페이스 내용을 저장하고 있는 배열과 인터페이스의 이름 관리
WrapperMaker	- IWrapper객체 생성 - IWrapper 객체를 생성할 때 ClassProvider가 제공한 인터페이스 이름과 IWrapper 객체가 생성되는 시간을 추가하여 인터페이스 사용 서비스에서 사용할 수 있도록 하였다.
WrapperRem	- 실제 인터페이스 객체들이 저장되는 공간은 Mapper클래스이기 때문에 사용자가 저장할 인터페이스를 선택하면 Mapper에서 WrapperRem의 객체를 생성하고 파일로 저장하도록 메소드를 호출한다. - WrapperRem는 WrapperMaker와는 달리 저장할 객체의 내용이 자바 클래스 파일인지의 여부를 검증하는 코드를 가지고 있다.

름은 "<<" 라벨이 붙은 버튼을 클릭 하여 대화 창으로부터 입력할 수도 있으며, Name 필드에 인터페이스의 절대 경로 명을 입력함으로써 지정할 수도 있다. URL 필드는 UDP를 사용하기 위하여 특정 검색(lookup) 서버의 주소를 입력하는 필드이다. 주소는 Jini 가 사용하는 프로토콜까지 같이 입력하여야 하며 "Jini://서버주소:포트번호"와 같은 형식으로 입력한다. 포트번호의 입력은 선택사항이며, 일반적으로는 입력하지 않는다. MRP 프로토콜을 사용하는 경우에는 URL 필드를 공백으로 남겨두면 자동적으로 MRP 프로토콜을 사용한다.

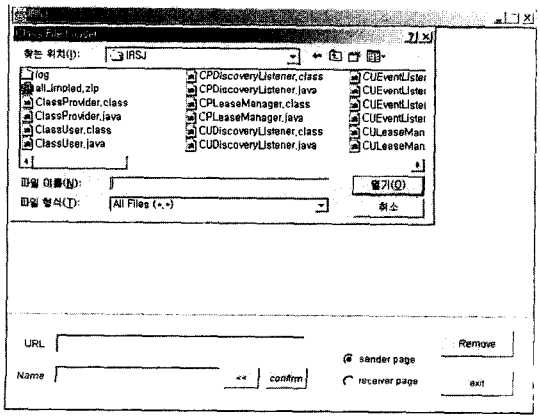

그림 8 인터페이스 이름 입력

검색(lookup) 서버의 주소 및 인터페이스 이름을 입력하고 "Confirm" 버튼을 누르면 실제로 ClassProvider 객체를 생성하면서 인터페이스를 검색(lookup) 서비스에 등록하게 된다. <그림 9>는 인터페이스가 검색(lookup) 서비스에 등록된 결과화면이다.

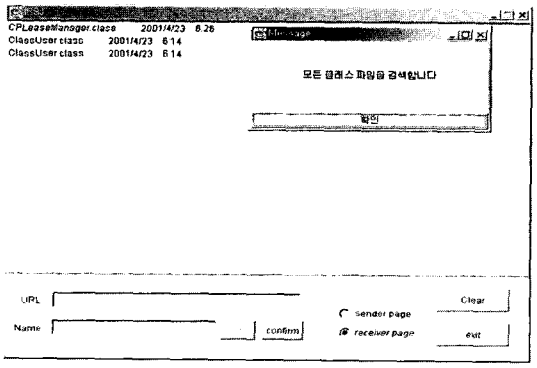

그림 9 인터페이스 등록 결과

이미 등록되어 있는 인터페이스를 제거하고 싶은 경우에는 제거하고자 하는 인터페이스 이름을 선택한 후 "Remove" 버튼을 클릭 하면 선택한 인터페이스가 IRSJ와 검색(lookup) 서비스로부터 제거된다. ClassUser의 경우처럼 다중 검색(lookup) 서비스에 등록된 경우에는 자동적으로 모든 검색(lookup) 서비스로부터 제거된다. 검색(lookup) 서비스로부터 등록된 클래스 파일을 제거하는 방법은 임대기간(lease) 시간을 1초로 설정하고 갱신 요청함으로써 이루어진다.

IRSJ 서비스의 종료는 "exit" 버튼을 클릭 하면 IRSJ는 서비스 자체가 종료된다. 이때 검색(lookup) 서비스에 등록한 모든 인터페이스 객체들도 같이 제거되도록 하였다. 이렇게 함으로써 수많은 서비스들을 상대로 작동하는 검색(lookup) 서비스의 자원을 보호할 수 있다.

다음은 인터페이스 사용 서비스에 대하여 간략하게 살펴보도록 하자. <그림 10>은 검색할 인터페이스의 이름을 지정하지 않고 검색(lookup) 서비스에 검색을 요청한 결과이다. 모든 인터페이스의 검색은 인터페이스 입력 필드를 공백으로 비워두고 "confirm" 버튼을 클릭함으로써 실행된다.

인터페이스를 등록하는 과정에서 검색(lookup) 서비스를 2개 실행하였으며 한번은 MRP를 이용하고 CPLeaseManager를 등록하는 과정에서는 UDP를 이용했었다. <그림 10>의 결과 화면에서 MRP를 이용하여 등록한 ClassUser의 경우는 2곳의 검색(lookup) 서비스로부

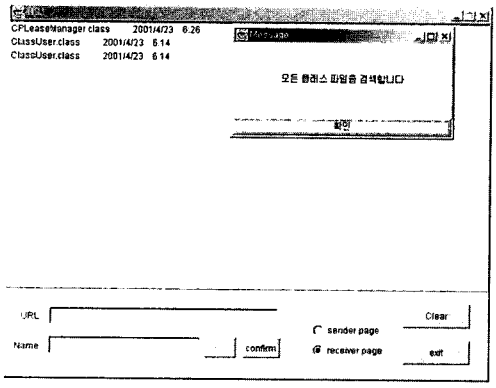

그림 10 모든 인터페이스 검색 결과

터 동일 인터페이스를 다운로드 받았기 때문에 중복해서 화면에 출력되지만 CPLeaseManager 클래스는 UDP를 이용하였기 때문에 하나의 검색(lookup) 서비스로부터 다운로드 된다. 따라서 결과화면에 한번만 출력된다.

인터페이스 이름 뒤에는 해당 인터페이스가 Wrapper 객체로 생성된 시기, 즉 검색(lookup) 서비스에 등록된 시간을 출력함으로써 인터페이스가 중복 출력되는 경우 시간 정보를 기준으로 하여 새로운 인터페이스를 선택할 수 있도록 하였다. 인터페이스의 저장은 저장하고자 하는 인터페이스 이름을 더블 클릭 하면 저장된다.

6. 비교 평가

기존의 인터페이스 공유 방식 중에서 전자 메일이나 플로피 디스크를 이용하여 직접 복사하는 경우가 있다. <그림 11>은 이 방식에 대한 구조도이다.

<그림 11>에서 알 수 있듯이 인터페이스 사용자들은 인터페이스 공유를 위해 인터페이스를 작성한 사람에 대한 정보를 알아야 한다. 누가 인터페이스를 작성하였는지를 모른다면 각각의 인터페이스 사용자들은 정보를 얻기 위하여 다른 사람들에게 정보를 묻거나 인터페이스를 가지고 있는지 질문하여 얻는 번거로운 작업을 거치게 된다. 플로피 디스크를 이용하여 복사를 하는 경우 또 하나의 문제가 발생한다. 인터페이스 작성자가 원거리에 존재하는 경우 인터페이스를 공유 할 수 없다는 것이다. 이 경우에 대한 해결책은 전화와 같은 통신 수단을 이용하여 인터페이스에 선언된 내용을 질문하고 인터페이스 사용자가 직접 인터페이스를 구현하여 사용하여야 한다. 따라서 전자 메일이나 플로피를 이용하여 복사하는 경우는 매우 비효율적이라고 할 수 있다.

IRSJ는 근거리에서는 MRP에 의하여 검색(lookup) 서버의 정보 없이 인터페이스를 공유 할 수 있으며, 원

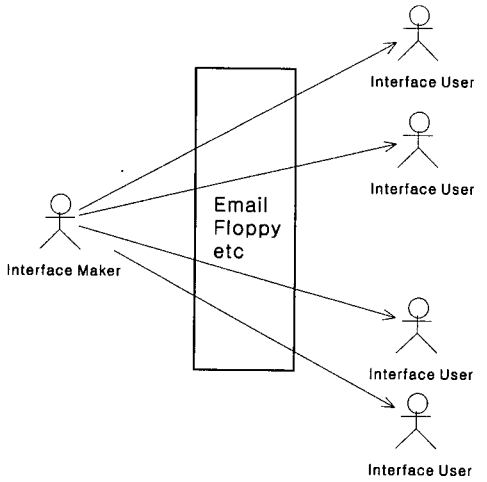

그림 11 전자 메일, 플로피 등을 이용한 인터페이스 공유 방식

거리에 검색(lookup) 서비스가 존재하는 경우는 서버의 주소를 입력하여 접속하므로 모든 경우에 인터페이스를 공유 할 수 있다. 뿐만 아니라 한번 검색에 실패한 인터페이스는 이벤트를 등록하고 자동적으로 다운로드받도록 설계되어 있다. 따라서 <그림 11>의 경우와 같이 전자 메일, 플로피 디스크 등을 이용하는 경우보다 IRSJ를 사용하는 것이 더 효율적이고 간편하다고 할 수 있다. <표 5>는 전자 우편, 플로피 디스크, IRSJ를 각각 사용하여 인터페이스 공유 시 비교 평가이다.

CORBA의 경우는 인터페이스를 IDL로 작성한다. IDL은 CORBA가 이 기종 언어를 지원할 수 있도록 새롭게 정의된 인터페이스 정의 언어이다. 따라서 CORBA를 사용하기 위해서는 IDL을 공부해야 하는 단점이 존재하지만 CORBA와 JAVA의 통합 추세에 힘입어 JAVA로 작성된 인터페이스를 IDL로 자동 변환하여 주는 프로그램도 존재한다.

표 5 전자우편, 플로피 디스크, FTP, IRSJ 비교

항 목	전자 우편	Floppy	FTP	IRSJ
근거리 서비스	가 능	가능	가능	가능
원거리 서비스	가 능	불가능	가능	가능
검색 실패 시	기능 없음	기능 없음	재검색	자동 검색 요청 메시지
인터페이스의 지속성	영구적	영구적	영구적	영구적
작성자/ 서버 정보	근거리:작성자 원거리:작성자	근거리:작성자 원거리:작성자	근거리: 서버 원거리: 서버	근거리: 없음 원거리: 서버
보안성 검증	없 음	없 음	없 음	class 확인

IDL로 작성된 인터페이스는 CORBA의 IR에 저장된다. CORBA의 IR은 인터페이스 정보를 메소드와 속성 단위로 나누어 저장한다. IR을 이용하면 런타임과 컴파일 타임에 인터페이스를 공유할 수 있다. <표 6>은 CORBA와 IRSJ를 비교한 것이다.

표 6 CORBA와 IRSJ 비교

항목	CORBA IR	IRSJ
통신 프로토콜	IIOP	TCP / UDP
지원 언어	다중언어	java
공유 방식	InterfaceDef 객체	사용자측 저장소에 파일로 저장
특정 인터페이스 검색	인터페이스 이름	인터페이스 이름
모든 인터페이스 검색	- 특정 IR의 ID - get_Interface	공문자열 (empty string)
인터페이스 검색 실패시	다시 검색	자동 다운로드
사용자측에서 인터페이스의 지속성	일시적	영구적
메시지 서비스	지원안함	지원

CORBA는 IIOP라는 특수한 네트워크 프로토콜을 사용한다. IIOP는 ORB(Object Request Broker)와 함께 사용되며, Java의 RMI처럼 원격 객체의 사용을 지원한다. IRSJ의 경우 통신 프로토콜은 제약을 받지 않는다. 기존의 TCP, UDP를 사용할 수 있으며 WAP 사용할 수도 있다. CORBA의 IR을 사용하는 경우 컴파일타임에 인터페이스 공유는 프로그램 작성 시 타입을 지정하지 않고 컴파일 한 후 런타임에 get_interface 메소드를 호출하고 IR로부터 InterfaceDef 객체를 반환 받아서 인터페이스를 사용한다. 이 방식은 Java 객체의 Meta 정보를 이용하여 객체 내부를 파악하는 인트로스펙션(introspection) 기능과 유사하다.

IRSJ는 검색 요청 방식의 종류, 지원 가능한 프로그래밍 언어, 인트로스펙션에 의한 인터페이스 검색과 같은 측면에서는 CORBA의 IR보다 기능이 떨어진다. 그러나 인터페이스 검색 요청이 실패된 후 자동적으로 다운로드 받는 기능은 CORBA의 IR에는 존재하지 않는 것이며, 인터페이스 공유 방식에 있어서 CORBA의 IR과는 달리 인터페이스 사용자의 저장소에 파일 형태로 기록하기 때문에 더 많은 보안상의 문제점을 발생시킬 수 있지만 인터페이스 정보의 지속성 측면에서는 CORBA보다 좋다. 검색 실패 시 어떤 인터페이스를 필요로 하는지 개발자에게 알리는 메시지 기능은 CORBA의 IR에는 존재하지 않는 기능이다.

7. 결론 및 향후과제

본 논문에서는 복잡해진 개발 환경 개선을 위하여 자바 기반의 인터페이스 파일을 공유하기 위한 IRSJ 서비스를 설계하고 기본적인 기능을 구현하였다. 본 논문에서 제시한 IRSJ는 근거리에서 작업할 때 검색(lookup) 서버나 작성자의 정보 없이도 인터페이스 파일을 검색할 수 있다. 원거리에서 개발 팀들이 공동 작업을 수행한다면 검색(lookup) 서비스가 실행되고 있는 서버의 주소만으로도 인터페이스 검색이 가능하다. 검색에 실패한 인터페이스 객체는 원격 이벤트 모델을 사용하여 검색(lookup) 서비스에 이벤트를 등록하고 검색(lookup) 서비스는 지속적으로 인터페이스가 등록되는 지의 여부를 감시하며, 검색(lookup) 서비스에 등록됨과 동시에 이벤트와 인터페이스 객체를 전송한다. 따라서 인터페이스 검색을 검색(lookup) 서비스에 요청한 사용자는 더 이상의 검색 요청을 하지 않아도 인터페이스를 다운로드 받아 사용할 수 있다.

메시지 서비스는 인터페이스 검색에 실패하면 IRSJ를 사용하는 개발팀원 모두에게 자동적으로 인터페이스 요청 메시지를 전송하여 빠른 시간 안에 인터페이스 공유가 가능하도록 설계되었다. 분산 객체 시스템들이 공통적으로 가지는 보안상의 문제도 다운로드 받은 객체가 자바 객체인지 검사함으로써 예상치 못한 프로그램의 다운로드에 의해 발생할 수 있는 보안상의 문제점을 해결하였다. 비록 IRSJ는 자바의 인터페이스를 대상으로 서비스가 이루어지고 있지만 제약 조건을 완화한다면 일반 파일들을 위한 서비스도 가능하다.

향후 연구과제로는 IRSJ와 기존의 자바 컴파일러와의 통합과 리플렉션(reflection)에 의한 인터페이스의 설명을 자동 생성하는 기능 등이 추가로 연구되어야 한다.

참 고 문 헌

[1] 분산 객체 기술 "http://www.inprise.co.kr/jbuilder/cases/korea/DW-GOMS_sigdb98_Final/tsld016.htm"
[2] RMI White Paper "http://java.sun.com/marketing/collateral/javarmi.html"
[3] The Network Revolution. available at "http://java.sun.com/features/1999/01/ jini_scenario.html"
[4] Alto. Why Jini Technology now? Sun microsystems. 1999.
[5] Jini Architecture Specification, Sun microsystems. 2000.
[6] Ken Arnold. The Jini Specifications. Sun microsystems. 1999.

[7] W. Keith.Edwards. Core Jini. Sun microsystems. 1999.
[8] Scott Oaks & Henry Wong. Jini In A Nutshell. O'REILLY. 2000.
[9] Jini Interface Repository for standardised service interfaces (empty at present) "http://www.artima.com/jini/interrepo/"
[10] Robert Orfali and Dan Harkey, Client/Server Programming with Java and CORBA, Jhon Wiley & Sons, Inc., 1998.
[11] IFR available at "http://www.iona.com/docs/ manuals/orbix2000/11/html/ orbix2000_progcxx/Output/ifr.htm"
[12] OMG, CORBA 3.0.2 specification, ch10. The Interface Repository, 2002.
[13] Alan Pope, The CORBA Reference Guide, ch 9 Repositories, Addison-Wesley, 1999.
[14] Doug Pedrick, Jonathan Weedon, etc. Programming with VisiBroker (A Developer's Guides to VisiBroker for java), ch3 IDL-to-java Mapping, Jhon Wiley & Sons, Inc.,1998.

문 창 주

정보과학회논문지 : 컴퓨팅의 실제
제 9 권 제 2 호 참조

문 석 현

1999 고려대학교 수학과, 컴퓨터학과 학사. 2001 고려대학교 컴퓨터학과 석사 2001~현재 Net Brain 핵심망 연구팀 주임연구원. 관심분야는 분산객체, JINI, 통신망

백 두 권

정보과학회눈문지 : 컴퓨팅의 실제
제 9 권 제 2 호 참조