정보가전용 Java 네트워크 API 설계 및 구현

유용선[0], 성영락*, 이철훈
충남대학교 컴퓨터공학과, *국민대학교 전자정보통신공학부
{ysryu[0], chlee}@ce.cnu.ac.kr, *yeong@mail.kookmin.ac.kr

Design and Implementation of Java Network API for Information Appliances

YongSun Ryu[0], Yeong Rak Seong*, and Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.
*School of Electrical Engineering, Kookmin Univ.

요 약

정보가전기기에 네트워크 기능을 탑재하게 되면 인터넷을 통해 사이버교육, 재택업무 등을 쉽고 편리하게 하여 삶의 질을 향상시키는데 기여할 수 있다. 이러한 정보가전기기에 네트워크 기능을 제공하기 위해 자바가상머신에서는 NET API를 제공한다. NET API를 구현하는데 있어 시스템에 의존적인 부분들이 존재하게 되는데, 이는 native 함수에서 구현한다. 본 논문에서는 리눅스 기반 자바 NET API를 구현하는데 있어 플랫폼 독립적인 자바 부분과 의존적인 native 부분으로 나누어 설계 및 구현하였고, 소켓을 통한 서버/클라이언트간의 데이터 전송 부분에 초점을 두었다.

1. 서 론

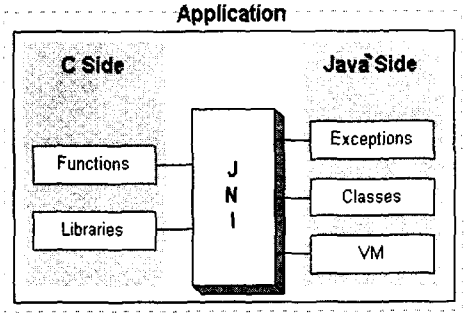
정보가전(Information Appliance)이란 가택내 구성된 홈 네트워크를 이용하여 연결 가능한 디지털 TV, 인터넷 냉장고, 컴퓨터, 통신기기등을 의미한다. 이러한 정보가전기기에는 스마트 디바이스라는 성능 좋은 마이크로프로세서가 장착되고, 이것들이 네트워크라는 하나의 매개체로 연결된다. 이렇게 연결된 정보가전기기는 가택내에서 인터넷을 통한 전자상거래, 인터넷 게임, 사이버교육, 재택업무 등을 쉽고 편리하게 하여 삶의 질을 향상시키는데 크게 기여할 수 있다. 본 논문에서는 정보가전용 제품에 네트워크 기능을 제공해 주는 리눅스 기반의 자바 NET API를 구현하는데 있어 JDK1.1.8 Specification(설계서)을 참조하였으며, 리눅스 시스템에 의존적인 부분은 c 언어로 작성하고, 자바 레벨과 통신하기 위해 JNI 함수를 이용하였다.

본 논문의 2장에서는 JNI(Java™ Native Interface)의 역할을 살펴보고, 3장에서는 JDK1.1.8 설계서를 기초로 네트워크 관련 API의 설계 및 구현에 대해 기술하고, 4장은 실험환경 및 결과에 대해 설명하며, 5장에서는 결론 및 향후 연구과제에 대해 기술한다.

2. 관련연구

2.1 JNI (Java Native Interface)

JNI란 자바와 자바 이외의 언어로 만들어진 애플리케이션이나 라이브러리가 상호 작동할 수 있도록 연결시켜 주는 인터페이스로 자바에서 지원하지 못하는 플랫폼 종속적인 기능들과 이미 다른 언어로 만들어진 애플리케이션이나 라이브러리를 사용할 수 있게 한다.



[그림1] JNI

native 언어를 이용한 자바 프로그램에서 자바로된 부분은 여전히 플랫폼 독립적이지만, native 언어로된 부분은 호스트 환경에 맞게 다시 컴파일 되어야 함을 의미한다. c/c++ 같은 native 언어들이 데이터 타입에 대한 형 안정성을 지원하지 못하기 때문에 자바 또한 형 안정성을 보장하지 못하게 된다.

그리고, JNI는 java와 native code와의 인터페이스를 제공해 주지만, 자바의 특징인 플랫폼 독립을 깨뜨릴 수도 있다. 전체 응용프로그램의 붕괴를 초래할 수 있으므로 native 함수를 사용하는데 있어서는 신중해야 한다.

3. 설계 및 구현

자바 사용자는 실제적인 네트워크의 하부구조를 알지 못하더라도 API를 사용하여 손쉽게 네트워크 애플리케이션을 구현할 수 있다. 이것은 호스트 환경의 자원들에 의존적인 부분들과 설계서에 정의되지 않는 코드들을 자바 API가 특정한 루틴을 통해 호출하기 때문이다.

본 논문에서는 리눅스 환경을 타겟으로 플랫폼 의존적인 소켓과 특정 루틴에 해당하는 프로토콜(http, file)을 구현하여 서버/클라이언트간의 데이터의 전송을 가능케 한다.

3.1 JNI을 사용한 TCP 소켓

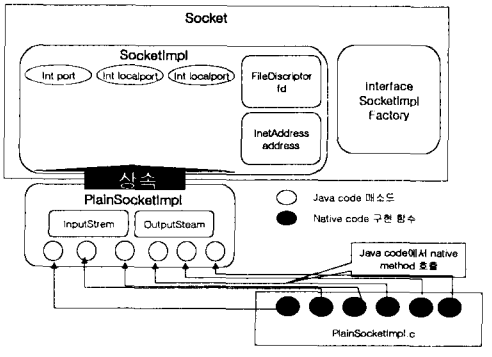
하나의 소켓은 두 호스트 사이의 연결을 말하며, [표1]과 같은 7가지 기능을 수행한다.

[표1]

① 원격호스트에 연결하기
② 데이터 보내기
③ 데이터 받기
④ 연결 닫기
⑤ 포트에 바인드 하기
⑥ 들어오는 데이터를 기다리기
⑦ 바인드된 포트로 원격 호스트의 연결 받아들이기

자바 API에서 소켓은 클라이언트 부분의 Socket클래스와 서버부분의 ServerSocket클래스로 설계되며 소켓의 기능 중 ①~④에 해당하는 메소드는 서버/클라이언트 양쪽에서 모두 필요하며, ⑤~⑦의 기능에 해당하는 메소드는 클라이언트의 연결을 기다리는 것이므로 서버쪽에서 구현한다.

Socket 클래스는 [그림3]과 같은 구조로, TCP에 관련된 메소드와 SocketImpl, SocketImplFactory 객체를 멤버변수로 갖으며, SocketImpl 객체를 상속받는 PlainSocketImpl 클래스가 TCP 기능을 구현한 native 코드를 호출함으로써 소켓을 생성한다.



[그림3] TCP 소켓 구조

[그림3]에서 PlainSocketImpl.c는 native 코드 부분으로 리눅스 상에서 동작하는 TCP 스택을 JNI를 사용하여 c와 java를 연동시킨다.

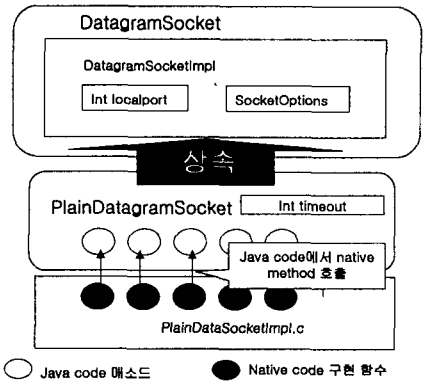
3.2 JNI을 사용한 UDP 소켓

JDK1.1.8 설계서에서 UDP 소켓 객체를 구현하기 위해서는 DatagramPacket과 DatagramSocket 클래스를 구현해야 한다. 전자는 데이터를 데이터그램이라는 패킷으로 만들고, 전달받은 데이터그램을 풀어내는 역할을 하며, 후자는 UDP 데이터그램을 송·수신한다. 따라서, UDP 프로토콜에서는 소켓 자체는 간단히 어느 포트에서 기다리고 있는지, 아니면 어떤 포트로 전송되어야 하는지만 알면 된다.

[표2]

① 원격호스트에 연결하기
② 데이터 보내기
③ 데이터 받기
④ 연결 닫기
⑤ 포트에 바인드 하기

[표2]는 UDP 소켓의 기능을 나타내며, 프로토콜 특성상 신뢰성의 보장없이 데이터를 보내기만 하므로 TCP 소켓과는 차이점이 있다. 한편, [그림4]는 UDP 소켓의 생성 과정이다.



[그림4] UDP 소켓 구조

DatagramSocket클래스는 UDP 소켓 생성에 관련된 메소드들과 DatagramSocketImpl 객체를 멤버변수로 하며, [그림4] 와 같이 PlainDatagramSocket 객체의 메소드가 JNI을 통해 native 코드인 PlainDataSocketImpl.c를 호출함으로써 소켓을 생성한다.

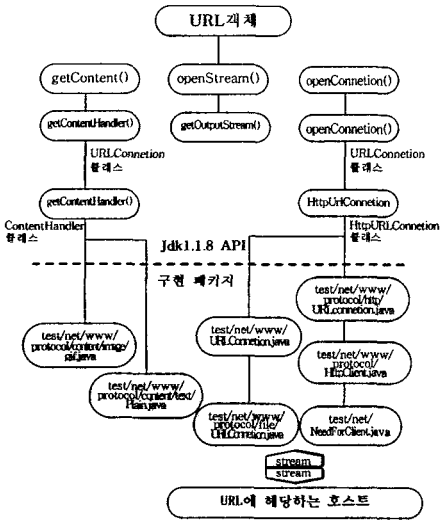
[그림4]의 PlainDataSocketImpl.c는 리눅스 상에 동작하는 UDP의 내부 동작을 JNI를 사용하여 c와 java를 연동시킨다.

3.3 URL 객체를 통한 서버/클라이언트 연결 메커니즘

주소와 포트를 갖는 URL 객체가 호스트들간에 데이터를 주고 받기 위해서는 ContentHandler 및 프로토콜에 해당하는 URLConnetion 객체를 생성해야 하며, 본 논문에서는 URLConnetion 객체를 상속받는 특정 프로토콜(http, file)과 해당하는 ContentHandler를 구현한다. [그림5]는 URL 객체와 관련된 클래스들간의 호출 메커니즘이다.

ContentHandler는 URLConnetion에서 데이터 타입을 받아 적합한 객체를 생성해 주며, [그림5]의 서브클래스인 구현패키지에서 image/gif 컨텐트 핸들러는 URLImageSource 객체를 리턴하며, text/plain 컨텐트 핸들러는 String을 리턴한다.

URLConnetion 클래스는 프로토콜 연결 및 애플리케이션과 URL 통신링크 작업을 수행하는 추상클래스로 하나의 메소드를 제외한 모든 메소드는 이미 구현되어 있다. 추가로 구현되어야 하는 메소드는 connect()메소드 인데, 이것은 구현하려는 서비스(http, file, ftp등)의 종류에 맞게 서버에 대한 연결을 만든다.

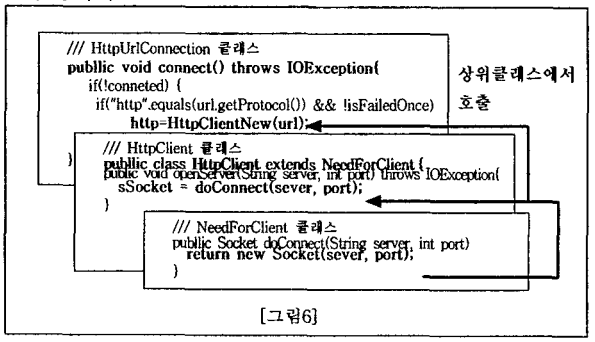


[그림5] URL의 접속 메커니즘

Test.net.www.protocol.file.**FileURLConneton**의 connect()메소드는 URL을 적절한 디렉토리에 있는 파일이름으로 변환하고 파일에 대한 MIME 정보를 생성하고 나서 그 파일에 대한 버퍼화된 FileInputStream을 연다.

또,Test.net.www.protocol.http.**HttpURLConnetion**의 connect() 메소드는 서버에 대한 연결을 책임지는 HttpClient 객체를 생성하여 TCP 소켓을 사용하여 로컬 호스트와 원격호스트가 데이터를 주고 받을 수 있도록 소켓연결을 만들어 서버/클라이언트간의 데이터를 Input/output Stream으로 전송한다.

[그림6]은 http 프로토콜을 구현한 connect()함수의 일부분이다.

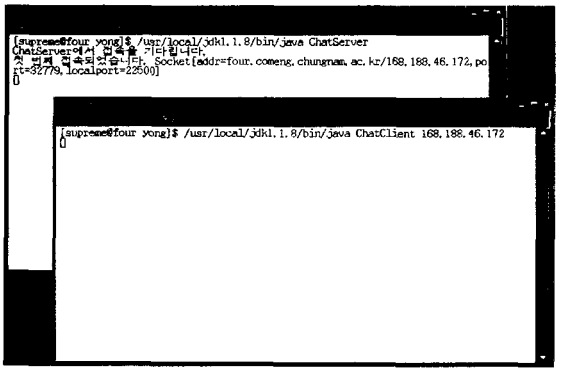


[그림6]

4. 실험환경 및 결과

실험환경은 Red Hat Linux V7.2 운영체제에 JDK1.1.8용 javac, gcc 컴파일러를 사용하였고, JDK1.1.8 클래스 라이브러리에 자체 개발한 자바 NET API를 대체하였다.

본 논문에서 결과를 검증하기 위해 서버/클라이언트간의 간단한 메시지 전송 프로그램을 수행하였다. 그 결과 [그림7]처럼 수행됨을 확인할 수 있다.



[그림7] 실험결과

5. 결론 및 향후 과제

본 논문에서는 JDK 1.1.8 설계서를 참조하여 리눅스 기반 자바 NET API를 구현하였으며, 호스트 환경에 의존적인 부분과 자바 설계서에 존재하지 않는 부분들을 native 언어(c언어)와 java 언어로 작성하였다. 호스트 환경에 의존적인 부분은 JNI를 사용하여 기존의 legacy C 코드를 재활용하였다.

향후 연구 과제로써 pdf, png, jpeg와 같은 컨텐트 핸들러나, 여러 종류의 프로토콜이 지원 가능하도록 수정 및 보완 할 것이다.

6. 참고문헌

[1] Sheng Liang, *The JavaTM Native Interface*, June 1999.

[2] Elliotte Rusty Harold, Java Network Programming, April 2000.

[3] Sun Miicrosystems, Inc., *JavaTM Native Interface Sepecification*, 1997.

[4] W.Richard stevens, *Unix Network Programming*, 1998

[5] Michael J.Donahoo, et. al, The Pocket Guide to TCP/IP Sockets, 2001.