순수 ATM 서비스를 제공하는 자바 API

성 종 진† · 이 근 구†† · 김 장 경†††

요 약

본 논문에서는 자바 프로그래밍에서 사용될 수 있는 ATM API를 제안한다. 제안하는 자바 ATM API는 기존 Java Core API 패키지들 중에서 통신 기능을 정의하고 있는 java.net 패키지를 확장하여 순수 ATM 서비스를 제공할 수 있도록 정의한 것이다. 순수 ATM 서비스의 표준인 ATM 포럼의 "Native ATM Services : Semantic Description, Version 1.0" 규격에 따른 표준화된 ATM 서비스 기능들을 제공할 수 있도록 고려하였다. ATM 서비스 제공을 위해 java.net에 추가적으로 정의한 자바 ATM API 용 클래스로는 ATM 어드레싱을 위한 *AtmAddress*, ATM BLLI/BHLI 정보의 이용을 위한 *AtmBLLI*와 *AtmBHLI*, 그리고 소켓 개념의 통신 프로그래밍을 위한 *AtmSocket*, *AtmServerSocket*, *AtmMulticastSocket*, *AtmSocketImpl* 등이 있으며, ATM 통신의 장점인 연결의 특성 표현을 위해서 *AtmConnAttr*을 정의하였다. 본 논문에서는 또한 이렇게 정의된 자바 ATM API를 WinSock 2 환경 상에서 구현한 내용을 기술한다.

Java API for Native ATM Services

Jong-Jin Sung† · Keun-Ku Lee†† · Jang-Kyung Kim†††

ABSTRACT

In this paper, we propose an ATM API for Java application programming. The proposed Java ATM API is an extended form of java.net package of the Java Core API. Our Java ATM API is defined based on the ATM Forum's semantic standard for native ATM services, "Native ATM Services : Semantic Description, Version 1.0." In order to provide native ATM services, we defined several new classes within the java.net package, including *AtmAddress* for ATM addressing, *AtmBLLI* and *AtmBHLI* for ATM BLLI and BHLI information, *AtmSocket*, *AtmServerSocket*, *AtmMulticastSocket*, and *AtmSocketImpl* for socket programming over native ATM communication, and *AtmConnAttr* for native ATM connection characteristics. Software structure for constructing the Java ATM API over WinSock 2 environment and its implementation method are also presented.

1. 서 론

ATM(Asynchronous Transfer Mode) 통신 환경에서 ATM 계층과 ATM 적응 계층(AAL : ATM Adaptation Layer)의 서비스를 직접 응용에게 제공하여 순수한 ATM 서비스를 이용하도록 하는 ATM API는 ATM 서비스의 기능과 성능을 최적으로 제공할 수 있는 응용 프로그래밍 인터페이스라 할 수 있다[1]. 마이크로소프트 윈도우 운영체제 환경에서는 WinSock 2 API가 ATM API의 기능을 수행할 수 있도록 확장 정의되면서[2] 업계의 표준 ATM API로 인정받고 있으며, 유닉스와 애플 운영체제 환경에서는 XTI와 X/Socket이 ATM API 기능을 수행할 수 있도록 확장 정의되면서[3] 역시 업계의 표준 ATM API로 인식되고 있다.

근래에는 마이크로소프트 윈도우나 유닉스와 같은

† 정 회 원 : 한국전자통신연구원 네트워크장비 시험센터 선임연구원
†† 정 회 원 : 한국전자통신연구원 네트워크장비 시험센터 선임연구원
††† 정 회 원 : 한국전자통신연구원 네트워크장비 시험센터 센터장/책임연구원

논문접수 : 1998년 8월 26일, 심사완료 : 1999년 5월 19일

운영체제 상에서 자바 가상 운영체제를 이용하는 응용 프로그램의 개발이 늘고 있다. 통신을 기본으로 하여 운영된다고 할 수 있는 자바 운영체제는 현재 인터넷 통신 서비스를 주로 이용하고 있다. 그러나 자바 환경에서도 ATM과 같은 우수한 통신 서비스의 이용이 곧 확산될 것이고 순수 ATM 서비스를 활용하려는 필요성이 대두될 것으로 보인다.

ATM 통신 서비스를 자바 인터페이스를 통하여 제공하는 ATM 기술과 자바 기술의 결합은 우수성과 실용성을 모두 살린 효과적인 통신 응용 개발 방법이 될 수 있을 것이다.

본 논문에서는 이러한 ATM과 자바의 결합 형태의 하나인 자바 환경에서 사용될 수 있는 ATM API를 제안하며 이를 구현한 내용을 기술한다. 제안하는 자바 ATM API는 순수 ATM 서비스를 제공함에 있어서, 기능상의 의미적 내용(semantic) 면에서는 ATM 포럼의 표준인 "Native ATM Services : Semantic Description, Version 1.0"[4]을 따르도록 하였고, API의 외형적 형태(syntax) 면에서는 기존의 Java Core API(JDK)[5]의 통신용 패키지인 java.net의 형태를 따르도록 하였다.

기존의 java.net 패키지에 포함되어 있는 클래스들은 인터넷 서비스를 위주로 작성된 것들이다. 이 패키지에 순수 ATM 서비스 기능을 위한 ATM 클래스들을 추가로 정의하여 이 패키지가 ATM API 역할을 할 수 있도록 만들었다. 이때 추가되는 ATM 클래스들은 기존의 인터넷 서비스를 위한 클래스들의 외형과 사용 방법과 최대한 유사하게 정의하였다. 이렇게 함으로써 기존 java.net 프로그래밍에 익숙한 사용자들에게 보다 사용하기 쉬운 자바 ATM API를 제공할 수 있다.

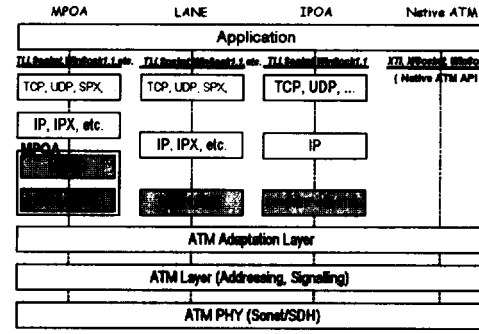
본 논문에서는 먼저 순수한 ATM 서비스에 대해 간단히 설명하고, 이를 자바 환경에서 제공하기 위한 자바 인터페이스인 자바 ATM API의 개발 내용을 기술한다. 우리가 제안하는 자바 ATM API 실현 방법인 java.net 패키지의 확장 정의 방법을 설명하고, ATM 서비스를 위해 정의한 새로운 클래스들의 내용을 하나씩 기술한다. 그리고 이렇게 정의된 자바 ATM API의 이용 및 구현 방법을 설명한다.

2. 순수 ATM 서비스와 API

본 장에서는 순수 ATM 서비스와 ATM API에 대한 정의, 기술개발 및 표준화 추세에 대하여 설명하고, 본 논문에서 제안하는 자바 ATM API의 목적과 필요성에 대해 나타낸다.

2.1 순수 ATM 서비스

ATM의 통신 서비스를 이용하는 방법으로는 ATM 계층이나 ATM 적응 계층이 제공하는 ATM 서비스를 직접 응용 프로그램이 이용하도록 하는 방법과 ATM 서비스 상에서 기존의 네트웍 프로토콜들을 올려서 응용 프로그램이 이를 통하여 이용하도록 하는 방법이 있다. 전자의 방법이 순수 ATM 서비스 이용 방법이며, 후자의 방법은 IPOA(IP over ATM)[6], LANE (LAN Emulation over ATM)[7], MPOA(Multiprotocol over ATM)[8] 등과 같은 방법들이다. 근래에는 인터넷 응용들이 인기를 끌고 있고 보편화되어 있는 까닭에 후자의 방법을 이용하여 IP를 ATM 서비스 상에 올려서 인터넷 서비스를 제공하는 방법들이 많이 이용되고 있다. (그림 1)에서 이러한 방법들의 프로토콜 계층 구조를 비교하여 나타내었다.



(그림 1) MPOA, LANE, IPOA, 순수 ATM 기술의 프로토콜 계층

ATM 서비스 상에서 IPOA, LANE, MPOA 등의 기술을 적용하여 인터넷의 TCP/IP 서비스를 제공하게 되면 ATM의 고유한 특성 중 QoS(Quality of Service) 제어와 ATM의 멀티캐스팅 기능 등 중요한 장점부분을 잃어버리게 된다. 이를 극복하기 위해서 MPOA나 차세대 인터넷 기술들에서는 RSVP(Resource Reservation Protocol), MARS(Multicast Address Resolution Service), NHRP(Next Hop Resolution Protocol) 등의 새로운 기술들을 모색하고 있으나 프로토콜 구조가 복잡해져 처리속도의 지연을 유발하는 등 또 다른 단점들이 나타나게 된다.

순수 ATM 서비스를 이용할 경우는 응용이 ATM 계층이나 ATM 적응 계층의 서비스를 직접 이용하므로 IPOA 등과 같은 방법들에서 나타나는 단점들 없이 ATM 고유의 장점을 충분히 살릴 수 있다.

2.2 순수 ATM API

순수 ATM 서비스를 이용하기 위해서 응용이 접근하는 인터페이스를 순수 ATM API(이하, 줄여서 ATM API라 칭함)라고 하며, 이것을 통하여 응용이 ATM 계층이나 ATM 적응 계층에 직접 접근할 수 있게 된다.

이 ATM API 기술에 대하여 표준화된 형태와 기능을 제정하려는 노력이 수년간 ATM 포럼에 의해 이루어져 왔다. 기본적으로 ATM 포럼에서는 ATM API 자체의 표준을 정의하는 대신에 순수 ATM 서비스에 대한 의미 명세인 "Native ATM Services : Semantic Description"을 표준으로 제정하여 이를 따르는 ATM API의 개발을 권고하고 있다. 1996년 2월에 UNI(User-Network Interface) 3.0과 3.1의 서비스를 바탕으로 한 버전 1.0 표준을 발표하였으며, 현재는 UNI 4.0에서 추가된 기능을 바탕으로 하는 차기 버전에 대한 승인 표결을 추진 중에 있다.

ATM 장비 개발 업계에서는 이 의미 명세 표준을 바탕으로 하여 실제적인 ATM API를 개발하도록 되어 있다. 본 논문에서 제안하고자 하는 자바 ATM API도 이 의미 명세 표준을 바탕으로 하여 개발한 실제적인 ATM API의 하나인 셈이다. 현재까지 개발된 실제적인 ATM API의 대표적인 두 가지 예를 들면 유닉스와 애플 계열 플랫폼에서 사용될 수 있는 XTI 및 X/Socket과 마이크로소프트사의 윈도우 계열 플랫폼에서 사용될 수 있는 WinSock 2 API를 꼽을 수 있다. 이들 두 가지 ATM API들은 ATM 포럼에 의해 표준 의미 명세를 따르는 사양을 정의한 것으로 인정받았다. 현재 대부분의 ATM 장비 개발 업체들은 자사의 장비 위에 업계의 표준 통신 API로 인식되고 있는 WinSock 2나 XTI, X/Socket을 지원하기 위한 서비스 제공 드라이버를 개발하여 제공하고 있다.

이와 같이 마이크로소프트 윈도우 환경에서의 ATM API로는 WinSock 2 API 그리고 유닉스 환경에서의 ATM API로는 XTI와 X/Socket이 정의되어 있는 상황에서, 근래에는 자바 컴퓨팅 및 통신 환경이 확산되고 있어 자바 환경에서 사용하기 위한 ATM API의 필요성이 대두되고 있다. 따라서 자바 ATM API의 정의 및 개발 그리고 표준화에 대한 연구가 요구된다고 할 수 있다.

3. 자바 ATM API

3.1 자바 프로그래밍을 위한 ATM API 제공

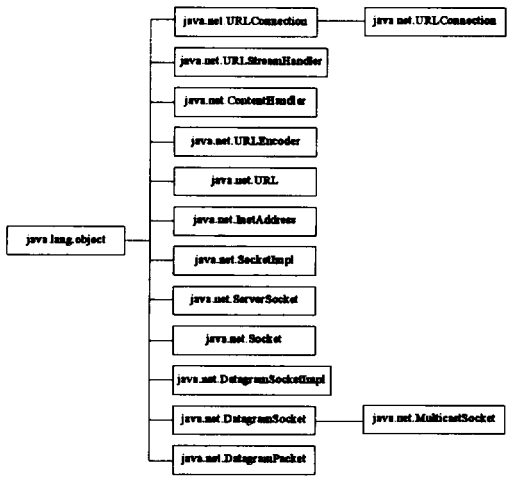
자바 환경에서의 통신 응용 제작시 ATM 통신 서비스를 이용할 수 있도록 하기 위한 자바 API는 여러 형태로 정의될 수 있다. 그러나 하나의 표준화된 형태의 API를 정의하여 제공하는 것이 바람직하므로 ATM 포럼에서는 이 자바 ATM API를 위한 표준을 제정하려 하고 있다.

이 논문에서는 Java Core API 패키지들 중의 하나인 java.net 패키지를 확장 정의하여 표준화된 자바 ATM API의 형태로 사용하는 방안을 제안한다. 기존의 TCP/UDP 통신 서비스에 대한 API를 정의하고 있는 java.net 패키지에 ATM 통신 서비스를 위한 클래스를 추가하는 것이다. 이때 이 추가된 클래스들은 순수 ATM 서비스의 표준인 ATM 포럼의 "Native ATM Services : Semantic Description, Version 1.0" 규격에 따른 표준화된 ATM 서비스 기능들을 제공할 수 있도록 고려되었다.

기존 java.net을 바탕으로 자바 ATM API를 정의하는 방법을 선택한 이유는 대부분의 네트웍 프로그래머들이 기존의 java.net을 이용하는 것에 익숙해져 있기 때문에 기존 API의 모습과 사용방법을 최대한 수용하기 위해서이다. ATM 서비스를 위해 추가된 클래스들의 형태와 사용방법은 TCP/UDP 서비스를 위한 클래스들과 유사하게 정의하였다. java.net에 포함되어 있는 소켓(socket) 관련 클래스들은 BSD 소켓을 수용하여 정의된 것이므로 기존의 소켓 프로그래밍에 익숙해져 있는 네트웍 응용 프로그램 제작자들에게는 더욱 쉽게 받아들여질 수 있을 것이다.

3.2 Java.net의 확장 정의

일반적으로 자바 API라고 하면 JDK(Java Development Kit) API를 의미하며, 이 논문에서 참조한 버전으로 보면 "Java Platform 1.1.6 Core API Specification"[5]을 그 내용으로 한다. 이 Java Core API는 여러 종류의 API 패키지들로 구성되어 있다. 이 패키지들 가운데 통신 서비스와 관련된 것으로는 java.net 패키지가 존재한다. Java.net 패키지는 URL, TCP 소켓, UDP 소켓, IP 주소, binary-to-text 변환 등과 관련된 기능들을 제공한다. (그림 2)에 java.net에서 정의하고 있는 클래스들을 나타내었다.



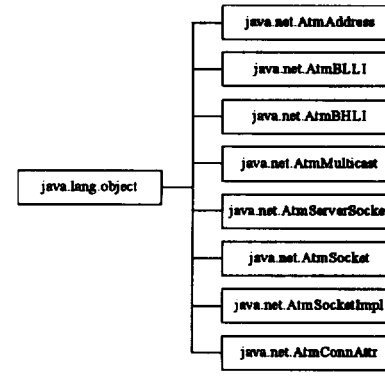
(그림 2) java.net 패키지 내부 클래스들의 계층구조

이러한 기존의 java.net 패키지에 다음과 같은 새로운 클래스와 인터페이스를 추가하여 순수 ATM 서비스를 지원할 수 있도록 한다.

- Class java.net.AtmAddress
- Class java.net.AtmBLLI
- Class java.net.AtmBHLI
- Class java.net.AtmMulticastSocket
- Class java.net.AtmServerSocket
- Class java.net.AtmSocket
- Class java.net.AtmSocketImpl
- Class java.net.AtmConnAttr
- Interface java.net.AtmSocketImplFactory

순수 ATM 서비스의 지원을 위해서는 먼저 ATM 어드레싱에 필요한 ATM 주소와 BLLI(Broadband Low Layer Information) 정보, BHLI(Broadband High Layer Information) 정보를 표현할 수 있어야 한다. 이를 위해서 AtmAddress, AtmBLLI, AtmBHLI 클래스들을 정의하였다. java.net이 기본적으로 소켓 개념을 바탕으로 한 통신 API로 정의되어 있으므로 ATM 통신을 위한 확장된 java.net에서도 역시 소켓 개념의 통신에서 필요한 서버/클라이언트 소켓 클래스들이 정의되어야 한다. ATM 소켓 클래스들로는 AtmServerSocket, AtmSocket, AtmSocketImpl을 정의하였다. 그리고 ATM 통신의 장점인 연결에 대한 특성을 표현하기 위한 클래스로 AtmConnAttr를 정의하였다. 인터페이스 AtmSocketImplFactory는 AtmSocket과 AtmServerSocket의 실제 동작 기능을 구현하는 AtmSocketImpl을 생성하고자 할 때 필요한 인터페이스로 정의하였다.

확장된 java.net 패키지의 새로운 클래스들은 (그림 3)과 같은 계층구조를 갖는다.



(그림 3) 순수 ATM 서비스를 위해 확장된 java.net 패키지 클래스 계층구조

3.3 ATM 서비스를 위한 java.net 클래스 정의

이 절에서는 앞에서 설명한 추가된 java.net 클래스들의 정의 내용을 기술한다. 클래스의 상세 정의는 부록에 기술하였다.

3.3.1 Class java.net.AtmAddress

AtmAddress는 ATM 주소체계를 이용하기 위한 클래스이다. 이 클래스는 IP 어드레스 제공을 위한 기존의 InetAddress 클래스와 유사하게 정의한 것으로 InetAddress에서 사용되는 메소드들과 유사한 기능을 수행하는 메소드들을 정의하였다.

AtmAddress 클래스에는 아래와 같은 3개의 변수와 8개의 메소드를 정의한다.

- int addressType
- int numOfDigits
- byte addr[20]
- isMulticastAddress()
- getHostName()
- getAddress()
- getHostAddress()
- equals()
- toString()

- getByName()
- getAllByName()
- getLocalHost()

이 클래스에서는 생성자(constructor)를 정의하지 않았다. 새로운 AtmAddress 객체를 생성하려면 생성자를 사용하는 대신에 getLocalHost(), getByName(), 또는 getAllByName() 메소드를 이용하도록 한다. ATM 주소 유형을 표현하는 addresType 변수에는 E.164 방식을 나타내는 "ATM_E164" 값이나 NSAP 형태의 AESA(ATM End System Addressing) 방식을 나타내는 "ATM_AESA" 값을 적용할 수 있다. AESA 방식을 addressType에 적용할 경우 numOfDigits 변수는 항상 20 바이트로 나타내어 진다.

3.3.2 Class java.net.AtmBLLI

ATM 시스템은 AtmAddress 정보로 식별이 가능하다. 그러나 그 시스템 내부에 존재하는 여러 응용 프로그램들 중에서 특정한 것을 식별해내기 위해서는 BLLI 정보와 BHLI 정보가 필요하다. 이 정보를 나타내기 위하여 Java ATM API에서는 AtmBLLI 클래스와 AtmBHLI 클래스를 정의하였다.

AtmBLLI 클래스는 다음과 같은 변수와 그 변수에 관계된 상수들의 정의로 구성하였다.

- Int layer2Protocol
- Int layer2UserSpecifiedProtocol
- Int layer3Protocol
- Int layer3UserSpecifiedProtocol
- Int layer3IPI
- byte snapID[5]

BLLI 정보는 하위 계층인 2 계층과 3 계층의 정보를 나타내기 위한 것이다. 따라서 AtmBLLI 클래스에 정의된 변수들은 2, 3 계층 프로토콜 정보들을 표현하기 위한 것들로 정의하였다. 이 클래스 정의에서는 이러한 변수들 뿐만 아니라 이들 변수에 적용될 수 있는 표준화된 2, 3 계층 프로토콜들을 표현한 다수의 상수 값들도 정의해 두었다.(부록 참조)

3.3.3 Class java.net.AtmBHLI

AtmBHLI 클래스에서는 다음과 같은 변수를 정의하였다.

- Int highLayerInfoType
- Int highLayerInfoLength
- byte highLayerInfo[8]

이 클래스는 이러한 변수들을 통하여 BHLI 정보인 상위계층의 정보를 표현한다. AtmBHLI 클래스 정의에서는 이상의 클래스내 변수 정의 외에도 highLayerInfoType 변수에 적용될 수 있는 상수 정의도 포함하도록 하였다.(부록 참조)

3.3.4 Class java.net.AtmMulticastSocket

AtmMulticastSocket 클래스의 내부 상세 정의는 아직 ATM 포럼에서 작업이 진행 중에 있는 "Native ATM Service : Semantic Description, Version 2.0"이 완성되면 작성할 예정이며, UNI 4.0에 포함된 멀티캐스팅 기능을 표현하도록 정의할 것이다.

3.3.5 Class java.net.AtmServerSocket

이 클래스는 순수 ATM 서비스를 위한 서버 소켓을 표현한다. 이 서버 소켓은 ATM 네트웍에서의 어떠한 요구자로부터의 특정 요구를 받아들여 그 요구에 따른 동작을 수행한 후, 그 결과를 그 요구자에게 돌려준다. AtmServerSocket 클래스에서는 다음과 같은 메소드를 제공하도록 하였다.

- getAtmAddress()
- getConnectionAttribute()
- accept()
- implAccept()
- close()
- setSoTimeout()
- getSoTimeout()
- toString()
- setAtmSocketFactory()

정의된 메소드들 중에서 accept()와 close()는 서버 소켓의 기본적인 기능을 위한 메소드들이며, getAtmAddress(), getAtmBlli(), getAtmBhli(), getConnectionAttribute() 등은 ATM 주소와 연결의 특성을 확인하기 위한 부가적 기능들을 제공하는 메소드들이다.

implAccept()는 기본적인 acccpt()와는 다른 연결 요청 수신 기능을 이용하고자 할 때 사용된다. setSoTimeout()와 getSoTimeout()은 accept()에서의 블록킹 타임아웃 시간을 설정하기 위해 정의하였다.

ATM 서버 소켓의 실제 동작은 AtmSocketImpl 객체에 의해 수행된다. 따라서 이 클래스에서 정의한 메소드들의 실제 동작은 AtmSocketImpl에 정의해 두었다.

3.3.6 Class java.net.AtmSocket

이 클래스는 순수 ATM 서비스를 위한 클라이언트 소켓을 표현하도록 하였다. 소켓은 두 시스템간의 통신을 수행할 때 하나의 끝점에 해당한다. ATM 소켓을 생성하도록 하는 시작점은 클라이언트 측에서는 AtmSocket의 생성자이며, 서버측에서는 AtmServerSocket 클래스들의 생성자이다. AtmSocket의 생성자는 연결이 되어 있지 않은 소켓의 생성만 수행하는 단순한 것으로부터, 소켓 생성 후 특정 상대방과의 연결 설정과 자신의 ATM 주소와의 바인딩까지의 기능들을 일괄적으로 수행하는 복잡한 것까지 여섯 가지를 정의하였다.

AtmSocket 클래스에서는 다음과 같은 메소드들을 포함하여 ATM 소켓이 필요로 하는 기능들을 제공하도록 하였다.

- getAtmAddress()
- getAtmBlli()
- getAtmBhli()
- getConnectionAttribute()
- getLocalAddress()
- getInputStream()
- getOutputStream()
- setSoLinger()
- getSoLinger()
- setSoTimeout()
- getSoTimeout()
- close()
- toString()
- setAtmSocketImplFactory()

AtmSocket 클래스의 주요 기능인 데이터 송수신 기능을 위한 메소드로는 getInputStream()과 getOutputStream()을 정의하였다. 이들은 각각 데이터 수신과 송신 기능을 수행하게 된다. getAtmAddress(), getAtmBlli(), getAtmBhli() 등은 연결 상대방의 주소와 BLLI/BHLI 정보를 알아내기 위한 것이며, getLocalAddress()는 자신의 주소를 알아보기 위한 것이다. getConnectionAttribute()를 통해서는 연결의 특성에 대한 정보를 파악할 수 있도록 한다.

ATM 소켓의 실제 동작은 AtmSocketImpl 객체에 의해 수행된다.

3.3.7 Class java.net.AtmSocketImpl

이 클래스는 ATM 클라이언트와 서버 소켓들의 동작들을 실제로 구현하게 되는 곳이다. AtmSocket과 AtmServerSocket에 정의된 기능들의 기본적인 형태들이 이 클래스에 의해 제공되도록 하였다. 따라서 아래에 정의된 바와 같이 AtmSocket과 AtmServerSocket에서 정의되었던 메소드들이 이 클래스에 망라되어 있는 것처럼 보인다.

- create()
- connect()
- bind()
- listen()
- accept()
- getInputStream()
- getOutputStream()
- available()
- close()
- getFileDescriptor()
- getAtmAddress()
- getAtmBlli()
- getAtmBhli()
- getConnectionAttribute()
- toString()

이 클래스 자체는 추상클래스(abstract class)로써 실제 동작 구현 내용을 담게될 클래스들의 공통 수퍼클래스(super class)가 된다. 이 클래스에서 정의된 내용 그대로를 구현하게 되는 클래스를 "PlainAtmSocketImpl" 이라는 이름으로 만들어 이용하도록 하였다. 이 클래스에 구현되어 있는 메소드들은 AtmServerSocket 클래스와 AtmSocket 클래스에 의해 사용된다. 이러한 방식의 사용 예를 (그림 5)에서 찾아볼 수 있다.

3.3.8 Class java.net.AtmConnAttr

이 클래스는 ATM 연결의 특성을 표현하도록 하여, ATM 연결의 생성시 그 연결에 대한 특성을 설정하고자할 때 사용되도록 하였다. 이 클래스에서는 AAL 정보, ATM 트래픽 설명, broadband bearer capabilities, BLLI 정보, BHLI 정보, 어드레싱 정보, QoS 정보 등의 내용을 변수로 정의하였다. 이 내용들은 "Native ATM Services : Semantic Description, Version 1.0"의 "ANNEX A : Connection Attributes" 부분을 충실히 반영한 것이다. 이들 변수에 의해 표현되는 연결의 특성을 확인하고자 할 때는 주로 AtmServerSocket과 AtmSocket 클래스에 정의되어 있는 getConnectionAttribute() 메소드를 이용하도록 한다.

3.3.9 Interface java.net.AtmSocketImplFactory

이 인터페이스는 AtmServerSocket과 AtmSocket에 의해 사용된다. 이 인터페이스에서 정의한 단 하나의 메소드인 createAtmSocketImpl()은 AtmServerSocket과 AtmSocket 클래스의 기능들을 실제로 구현해둔 AtmSocketImpl 객체를 생성할 때 이용된다.

4. ATM 포럼의 ATM 서비스 의미적 표준과의 비교

이상과 같이 java.net 패키지의 클래스 형태로 정의된 자바 ATM API는 기본적으로 ATM 포럼의 순수 ATM 서비스 의미적 표준 규격인 "Native ATM Services : Semantic Description, Version 1.0"을 따르도록 하였다. <표 1>에서 이 의미적 표준 규격의 서비스 프리미티브(primitive)와 우리가 정의한 자바 ATM API의 프리미티브를 비교하여 나타내었다.

<표 1>에서 ATM 포럼의 의미적 표준 프리미티브들과 비교된 자바 ATM API 프리미티브들은 실제로 의미적 표준 프리미티브들의 동작과 정확히 일치되는 것들인 경우도 있으나 그 동작을 포함한 좀더 포괄적인 기능을 수행하는 프리미티브들인 경우도 있다. ATM_set_connection_attributes와 같은 경우가 그 대표적인 예로써, AtmSocket() 생성자에 의해서 ATM 클라이언트 소켓의 생성(create())과 연결의 설정(connect())이 일괄적으로 처리될 때 부분적인 하나의 절차로 수행되는 기능이다. 즉, AtmSocket() 생성자에 의

〈표 1〉 ATM 포럼의 ATM 서비스 의미적 표준 API와 자바 ATM API의 프리미티브 매핑

ATM Forum Primitives	Java ATM API
ATM_abort_connection(req)	
ATM_accept_incoming_call(res)	Accept()
ATM_add_party(req)	
ATM_add_party_reject(con)	
ATM_add_party_success(con)	
ATM_arrival_of_incoming_call(ind)	
ATM_associate_endpoint(req)	create() by AtmServerSocket() and AtmSocket()
ATM_call_release(req)	close()
ATM_call_release(ind)	close()
ATM_connect_outgoing_call(req)	connect() by AtmSocket()
ATM_drop_party(req)	
ATM_drop_party(ind)	
ATM_get_local_port_info(req)	getAtmBlli(), getAtmBhli()
ATM_P2MP_call_active(con)	
ATM_P2P_call_active(con)	connect() by AtmSocket(), Accept()
ATM_prepare_incoming_call(req)	bind() and listen() by AtmServerSocket()
ATM_prepare_outgoing_call(req)	bind() by AtmSocket()
ATM_query_connection_attributes(req)	getConnectionAttribute()
ATM_reject_incoming_call(res)	
ATM_set_connection_attributes(req)	connect() by AtmSocket()
ATM_wait_on_incoming_call(req)	listen() by AtmServerSocket()
ATM_send_data(req)	getOutputStream()
ATM_recv_data(req)	getInputStream()

해 수행되는 AtmSocketImpl의 내부 메소드 connect()의 수행시 ATM_set_connection_attributes에 해당하는 기능인 연결의 특성 설정을 수행하게 된다.

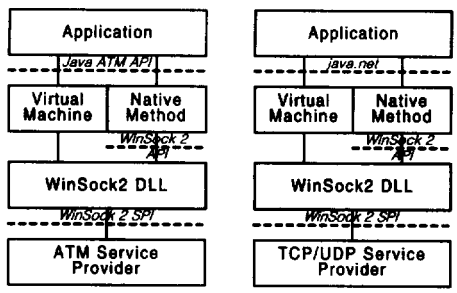
<표 1>에서 자바 ATM API 프리미티브들로 나타내어지는 create(), connect(), bind(), listen(), accept(), close() 등은 AtmSocketImpl 클래스 내부에 정의된 메소드들이다. 이들은 AtmServerSocket과 AtmSocket 클래스들의 생성자인 AtmServerSocket()과 AtmSocket()에 의해 내부적으로 불리워져 수행되는 부분들이다.

현재 상태로는 이 논문에서 제안하는 자바 ATM API에서는 멀티캐스팅 기능에 대한 프리미티브들이 미정의된 상태이다. ATM 멀티캐스팅 기능에 대해서는 UNI 4.0을 바탕으로 하는 "Native ATM Services : Semantic Description, Version 2.0"이 완성되면 이 규격에 따른 ATM 멀티캐스팅 소켓을 정의하여 추가할 계획이다.

이 논문에서 제안하는 자바 ATM API에서는 서비스 프리미티브에 대한 부분이외에도, ATM 주소 및 ATM BLLI와 BHLI 그리고 연결의 특성에 대한 여러 변수들에 대해서도 "Native ATM Services : Semantic Description, Version 1.0"에 정의된 내용을 충실히 반영하여 자바 ATM API로 정의된 java.net 패키지의 클래스 내부 변수로 표현될 수 있도록 고려하였다.

5. 자바 ATM API의 구현

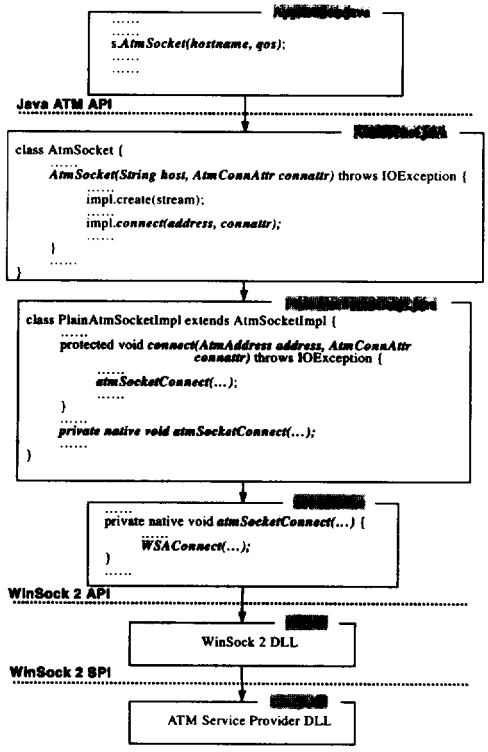
java.net을 확장하여 정의한 자바 ATM API는 (그림 4)의 (a)와 같은 구조를 통하여 제공될 수 있다. 이 구조는 마이크로소프트 윈도우 운영체제 상에서 자바 가상운영체제 환경을 구축할 경우에 해당되는 구조이다. 유닉스나 기타 운영체제 상에서 자바운영체제 환경을 구축했을 경우는 그 구조가 다를 수 있다. 그림에서는 기존의 TCP/UDP 지원을 위한 java.net의 이용 (그림 4)의 (b)와 ATM 서비스 지원을 위해 확장된 자바 ATM API의 이용 (그림 4)의 (a)시의 소프트웨어 구조를 비교하여 나타내었다. 우리가 정의한 자바 ATM API는 java.net을 확장한 것이므로 TCP/UDP 지원을 위한 기존의 java.net의 이용 구조와 유사하다.



(a) java.net for native ATM (b) java.net for TCP/UDP

(그림 4) WinSock 상에서의 java.net 이용 구조

java.net은 TCP/UDP 지원을 위한 대부분의 기능을 native 메소드를 통해 제공하고 있다. java.net 클래스들이 정의하고 있는 소켓 기능들은 결국은 C/C++ 언어로 구현되어 있으며, 이 C/C++로 구현된 함수들을 java.net 내부 클래스들이 native 메소드 방식으로 불러 쓰도록 되어 있다. 이 논문에서 제안하는 확장된 java.net의 ATM 지원 클래스 내부의 소켓 기능들도 native 메소드 방식으로 구현하도록 하였다. (그림 4)의 (a)에서 보면 WinSock 상에서 제공되는 확장된 java.net 형태인 자바 ATM API가 native 메소드 방식으로 WinSock 2 API를 이용하도록 되어 있다.



(그림 5) WinSock 환경에서의 자바 ATM API 구현 소스 코드[연결 설정 기능 부분]

(그림 5)에서는 응용 프로그램에서 Java ATM API를 이용할 때 실제로 어떠한 방식으로 ATM 서비스를 이용하는지를 구현된 프로그램 소스 차원에서 상세하게 나타내었다. (그림 5)의 윗쪽에서부터 보면, 응용 프로그램인 application.java에서 AtmSocket.java 파일에 정의되어 있는 AtmSocket() 생성자를 이용하여 ATM 소켓을 생성하고 연결을 설정하는 부분을 나타내었다. 여기에서 사용된 AtmSocket() 생성자는 ATM 소켓 생성(create()) 후 상대방 시스템 'hostname'과 연결의 특성 'connattr'를 가지고 연결을 설정(connect())하는 것까지를 일괄적으로 수행한다.

수행되는 과정 중에서 연결 설정 과정 부분만을 계속해서 설명하면 다음과 같다. AtmSocket() 생성자는 연결을 설정하기 위해 AtmSocketImpl 클래스의 connect() 메소드를 이용하게 된다. 이 메소드는 PlainAtmSocketImpl.java 파일에 구현되어 있으며, 그 메소드 내부에서는 다시 C/C++ 함수인 atmSocketConnect()를 native 메소드로 불러 사용하고 있다. C/C++ 함수 atmSocketConnect()는 결국 WinSock 2 API인 WSAConnect() 등을 사용해서 ATM 연결을 생성하게 된다.

이상과 같이 (그림 5)에서는 연결 설정 부분만을 예로 들어 도식적으로 설명하였으나, 앞에서 정의한 자바 ATM API의 나머지 부분들도 이러한 방법으로 ATM 서비스를 제공하도록 구현하였다.

6. 결 론

이 논문에서는 자바 프로그래밍에서 사용될 수 있는 ATM API를 제안하였다. 제안된 자바 ATM API는 기존의 자바 프로그래밍 환경에서 제공되는 Java Core API(또는 JDK API) 패키지들 중에서 인터넷 통신 기능을 정의하고 있는 java.net 패키지의 확장된 형태로 정의되었다. 동시에 순수 ATM 서비스의 표준인 ATM 포럼의 "Native ATM Services : Semantic Description, Version 1.0" 규격에 따른 표준화된 ATM 서비스 기능들을 제공할 수 있도록 정의되었다.

표준화된 ATM 서비스 제공을 위해 java.net에 추가적으로 정의된 자바 ATM API 용 클래스로는, ATM 어드레싱을 위한 *AtmAddress*, BLLI/BHLI 정보의 이용을 위한 *AtmBLLI*와 *AtmBHLI*, 소켓 개념의 통신 프로그래밍을 위한 *AtmSocket*, *AtmServerSocket*, *AtmMulticastSocket*, *AtmSocketImpl* 그리고 ATM 통신의 장점인 연결의 특성 표현을 위한 *AtmConnAttr* 등이 제안되었다.

이 논문에서는 이렇게 정의된 자바 ATM API를 WinSock 2 환경 상에서 구축할 경우 요구되는 소프트웨어의 구조와 구현 방법을 또한 제시하였다.

부록 : Java ATM API 클래스별 상세 정의

1. Class java.net.AtmAddress

```
public final class AtmAddress extends Object
implements Serializable {
  // Variables
  int addressType;
  Int numOfDigits;
  byte addr[20];
  // Constants used for AddressType
  Private static final int ATM_E164;
  Private static final int ATM_AESA;

  // Methods
  public boolean isMulticastAddress();
  public String getHostName();
  public byte[] getAddress();
  public String getHostAddress();
  public boolean equals(Object obj);
  public String toString();
  public static AtmAddress getByName(String host)
        throws UnknownHostException;
  public static AtmAddress[] getAllByName(String
        host) throws UnknownHostException;
  public static AtmAddress getLocalHost() throws
        UnknownHostException;
}
```

2. Class java.net.AtmBLLI

```
public final class AtmBLLI extends Object
implements Serializable {
  // Variables
  Int layer2Protocol;
  Int layer2UserSpecifiedProtocol;
  Int layer3Protocol;
```

```
  Int layer3UserSpecifiedProtocol;
  Int layer3IPI;
  byte snapID[5];

  // Constants used for Layer2Protocol
  Private static final int BLLI_L2_ISO_1745;
  Private static final int BLLI_L2_Q921;
  Private static final int BLLI_L2_X25L;
  Private static final int BLLI_L2_X25M;
  Private static final int BLLI_L2_ELAPB;
  Private static final int BLLI_L2_HDLC_NRM;
  Private static final int BLLI_L2_HDLC_ABM;
  Private static final int BLLI_L2_HDLC_ARM;
  Private static final int BLLI_L2_LLC;
  Private static final int BLLI_L2_X75;
  Private static final int BLLI_L2_Q922;
  Private static final int BLLI_L2_USER_SPECIFIED;
  Private static final int BLLI_L2_ISO_7776;

  // Constants used for Layer3Protocol
  Private static final int BLLI_L3_X25;
  Private static final int BLLI_L3_ISO_8208;
  Private static final int BLLI_L3_X223;
  Private static final int BLLI_L3_SIO_8473;
  Private static final int BLLI_L3_T70;
  Private static final int BLLI_L3_ISO_TR9577;
  Private static final int BLLI_L3_USER_SPECIFIED;

  // Constants used for Layer3IPI
  Private static final int BLLI_L3_IPI_SNAP;
  Private static final int BLLI_L3_IPI_IP;

  // Constructor
  public AtmBLLI();
}
```

3. Class java.net.AtmBHLI

```
public final class AtmBHLI extends Object
implements Serializable {
  // Variables
  Int highLayerInfoType;
  Int highLayerInfoLength;
  byte highLayerInfo[8];
  // Constants used for highLayerInfoType
  Private static final int BHLI_ISO;
  Private static final int BHLI_UserSpecific;
  Private static final int BHLI_HighLayerProfile;
  Private static final int BHLI_VendorSpecificAppId;

  // Constructor
  public AtmBHLI();
}
```

4. Class java.net.AtmMulticastSocket

```
public class AtmMulticastSocket extends Object {
  // To be defined later;
}
```

5. Class java.net.AtmServerSocket

```
public class AtmServerSocket extends Object {
  // Constructors
  public AtmServerSocket() throws IOException;
  public AtmServerSocket(int backlog) throws
        IOException;
  public AtmServerSocket(int backlog, AtmAddress
        bindAddr) throws IOException;

  // Methods
  public AtmAddress getAtmAddress();
  public AtmConnAttr getConnectionAttribute();
  public AtmSocket accept() throws IOException;
  protected final void implAccept(AtmSocket s)
        throws IOException;
  public void close() throws IOException;
  public synchronized void setSoTimeout(int timeout)
  throws SocketException;
  public synchronized int getSoTimeout() throws
        IOException;
  public String toString();
  public static synchronized void
        setAtmSocketFactory(AtmSocketImplFactory
        fac) throws IOException;
}
```

6. Class java.net.AtmSocket

```
public class AtmSocket extends Object {
```

```
    // Constructors
    protected AtmSocket();
    protected AtmSocket(AtmSocketImpl impl) throws
            SocketException;
    public AtmSocket(String host, AtmBLLI blli,
            AtmBHLI bhli, AtmConnAttr connattr) throws
            UnknownHostException, IOException;
    public AtmSocket(AtmAddress address, AtmBLLI
            blli, AtmBHLI bhli, AtmConnAttr connattr)
            throws IOException;
    public AtmSocket(String host, AtmBLLI blli,
            AtmBHLI bhli, AtmAddress localAddr,
            AtmConnAttr connattr) throws IOException;
    public AtmSocket(AtmAddress address, AtmBLLI
            blli, AtmBHLI bhli, AtmAddress localAddr,
            AtmConnAttr connattr) throws IOException;

    // Methods
    public AtmAddress getAtmAddress();
    public AtmBLLI getAtmBlli();
    public AtmBHLI getAtmBhli();
    public AtmConnAttr getConnectionAttribute();
    public AtmAddress getLocalAddress();
    public InputStream getInputStream() throws
            IOException;
    public OutputStream getOutputStream() throws
            IOException;
    public void setSoLinger(boolean on, int val) throws
            SocketException;
    public int getSoLinger() throws SocketException;
    public synchronized void setSoTimeout(int timeout)
    throws SocketException;
    public synchronized int getSoTimeout() throws
            SocketException;
    public synchronized void close() throws
            IOException;
    public String toString();
    public static synchronized void
            setAtmSocketImplFactory
            (AtmSocketImplFactory fac) throws
            IOException;
}
```

7. Class java.net.AtmSocketImpl

```
public abstract class AtmSocketImpl extends Object {

  // Variables
  protected FileDescriptor fd;
  protected AtmAddress address;
  protected AtmBLLI blli;
  protected AtmBHLI bhli;

  // Constructors
  public AtmSocketImpl();

  // Methods
  protected abstract void create() throws IOException;
  protected abstract void connect(String host,
          AtmBLLI blli, AtmBHLI bhli, AtmConnAttr
          connattr) throws IOException;
  protected abstract void connect(AtmAddress
          address, AtmBLLI blli, AtmBHLI bhli,
          AtmConnAttr connattr) throws IOException;
  protected abstract void bind(AtmAddress host,
          AtmBLLI blli, AtmBHLI bhli) throws
          IOException;
  protected abstract void listen(int backlog) throws
          IOException;
  protected abstract void accept(AtmSocketImpl s)
          throws IOException;
  protected abstract InputStream getInputStream()
          throws IOException;
  protected abstract OutputStream getOutputStream()
          throws IOException;
  protected abstract int available() throws IOException;
  protected abstract void close() throws IOException;
  protected FileDescriptor getFileDescriptor();
  protected AtmAddress getAtmAddress();
  protected AtmBLLI getAtmBlli();
  protected AtmBHLI getAtmBhli();
  protected AtmConnAttr getConnectionAttribute();
  public String toString();
}
```

8. Class java.net.AtmConnAttr

```
public final class AtmConnAttr extends Object {
  // Variables
```

```
int aalType;
int aal5FwdMaxSdu;
int aal5BakMaxSdu;
int aal5SscsType;
int userDefinedAalInfo;
int fwdPcrClp0;
int fwdPcrClp1;
int bakPcrClp0;
int bakPcrClp1;
int fwdScrClp0;
int fwdScrClp1;
int bakScrClp0;
int bakScrClp1;
int fwdMbsClp0;
int fwdMbsClp1;
int bakMbsClp0;
int bakMbsClp1;
int bestEffort;
int fwdTagging;
int bakTagging;
int bearerClass;
int trafficType;
int timeReq;
int clippingInd;
int connectConfig;
int applIdType;
int applId;
int layer2Id;
int layer2Mode;
int layer2WindowSize;
int layer2UserId;
int layer3Id;
int layer3Mode;
int layer3PacketSize;
int layer3WindowSize;
int layer3UserId;
int layer3IpiId;
int layer3OuiId;
int layer3PidId;
int calledAddrFormat;
int calledAddr;
int calledSubaddrType;
int calledSubaddr;
int callingAddrFormat;
int callingAddr;
int presentationInd;
int ScreeningInd;
int callingSubaddrType;
int callingSubaddr;
int causeCoding;
int causeLocation;
int causeValue;
int causeDiagnostics;
int fwdQosClass;
int bakQosClass;
int networkId;
}
```

9. Interface java.net.AtmSocketImplFactory

```
public interface AtmSocketImplFactory {
  public abstract AtmSocketImpl
        createAtmSocketImpl();
}
```

참 고 문 헌

[1] Ross, T., "ATM APIs : The Missing Links," Data Communications, pp.119-128, Sep., 1995.

[2] ATM Forum, '***WinSock 2 ATM Annex***,' ATM_Forum/96-0190, Feb., 1996.

[3] ATM Forum, '***XNET ATM API Specifications, Appendix X and Y***,' ATM_Forum/96-1169, Oct., 1996.

[4] ATM Forum, '***Native ATM Services : Semantic Description, Version 1.0***,' af-saa-0048.000, Feb., 1996.

[5] Sun Microsystems, Inc., '***Java Platform 1.1.6 Core API Specification***,' http://java.sun.com/products/jdk/1.1/docs/index.html

[6] Lauback, M., "Classical IP and ARP over ATM," IETF RFC 1577, Jan., 1994.

[7] ATM Forum, '***LAN Emulation over ATM, Version 1.0***,' af-lane-0021.000, Jan., 1995.

[8] ATM Forum, '***Multiprotocol over ATM, Version 1.0***,' Just Passed in the Final Ballot, Jul., 1997.

[9] ATM Forum, 'ATM User-Network Interface Specification, Version 3.1,' af-uni-0010.002.

성 종 진

e-mail : jsung@pec.etri.re.kr

1990년 경북대학교 전자공학과(공학사)

1992년 경북대학교 대학원 전자공학과(공학석사)

1992년~현재 한국전자통신연구원 네트워크장비 시험센터 선임연구원

관심분야 : 고속통신 응용 및 서비스, 통신 표준 시험 기술 등

이 근 구

e-mail : kklee@pec.etri.re.kr

1982년 연세대학교 전자공학과(공학사)

1985년 연세대학교 대학원 전자공학과(공학석사)

1994년~1995년 미국 NIST 객원연구원 근무

1994년~1997년 미국 해외사무소(워싱턴) 파견 근무

1984년~현재 한국전자통신연구원 네트워크장비 시험센터 선임연구원

관심분야 : 고속통신망 프로토콜 표준, 신호 프로토콜, 정보통신 상호 운용성 시험

김 장 경

e-mail : jkkim@pec.etri.re.kr

1980년 연세대학교 전자공학과(공학사)

1989년 Iowa State Univ. Computer Engineering(M.S.)

1992년 Iowa State Univ. Computer Engineering(Ph.D.)

1980년~1986년 국방과학연구소 연구원

1994년~1995년 미국 Univ. of Maryland 파견 국제 공동연구 수행

1992년~현재 한국전자통신연구원 네트워크장비 시험센터 센터장/책임연구원

관심분야 : 통신망 프로토콜 상호 연동 표준, 정보통신 표준 시험 기술, High Performance Architecture, 프로토콜 상호 운용성 시험