무선 인터넷 응용 서비스를 위한 WAP 게이트웨이의 설계 및 구현

Design and Implementation of the WAP Gateway for Wireless Internet Applications Services

이양선
서경대학교 컴퓨터공학과 조교수
임철수
서경대학교 컴퓨터공학과 조교수
이용학
김포대학 컴퓨터계열 인터넷 정보전공 조교수

Yang-Sun Lee
Professor, Dept. of Computer Engineering, SeoKyeong Univ.
Chul-Su Lim
Professor, Dept. of Computer Engineering, SeoKyeong Univ.
Yong-Hak Lee
Professor, Div. of Computer, Kimpo College

중심어 : WAP, WAP 게이트웨이, WSP, WTP, WDP, WML

요 약

무선 인터넷 응용 서비스를 위한 WAP(Wireless Application Protocol) 게이트웨이는 사용자 단말기(핸드폰, PDA등)와 WAP/WEB 서버 사이에 위치하여 유선 인터넷 환경에서 제공되는 각종 콘텐츠를 무선 인터넷 사용자에게 제공하기 위해 HTTP 프로토콜과 해당 WAP 프로토콜 사이의 변환 및 WML(Wireless Markup Language) 문서의 인코딩(encoding) 기능 등을 제공하는 미들웨어 시스템이다. 구현된 WAP 게이트웨이는 WAP 포럼에서 발표한 WAP 규격 1.2를 토대로 설계되었으며 Java 언어로 구현하여 플랫폼 독립적으로 운영될 수 있도록 하였다. 개발된 시스템을 Nokia Tool Kit로 테스트한 결과 Kannel 제품 보다 성능이 우수한 것으로 나타났다.

Abstract

The WAP gateway. for wireless internet applications services, is a software that is placed between the user terminals(PDA, mobile phones) and the WAP/Web server. It is a middleware system that converts the protocol of a packet network to the protocol on the WAP network, and supports the encoding of WML documents. In this paper, we designed and implemented the WAP gateway based on the WAP specification version 1.2 announced by the WAP forum. It provides cross-platform compatibility by being implemented by the Java language. Compared with a Kannel system using a Nokia tool kit version 1.2, it provides a more excellent performance and a stability.

Ⅰ. 서론

인터넷 비즈니스 시장은 물리적인 시간과 공간의 제약을 받는 유선 인터넷 환경의 시장 개념에서 시·공간의 제약을 해소하고 언제(any time), 어디서나(any where) 자유롭게 사용할 수 있는 무선 인터넷 환경의 시장 개념으로 이동 중에 있다. 그러나 아직까지도 현재의 인터넷 환경은 단위 네트워크들을 케이블로 연결한 유선망에 기반을 두고 있기 때문에 시·공간적 제한 사항들을 완전하게 해소해 주지 못하고 있는 실정이다. 이런 이유로 이동 단말기나 PDA와 같은 Post PC를 이용한 무선 인터넷에 관한 연구가 활발하게 진행되고 있는 상황에서 Phone.com, Ericsson, Nokia, Motorola 등의 기업이 WAP 포럼을 결성하여 무선 인터넷 규격인 WAP(Wireless Application Protocol)을 발표하였다. WAP의 구조는 크게 Application Layer(WAE), Session Layer(WSP), Transaction Layer(WTP), Security Layer(WTLS), Transport Layer(WDP)로 구분할 수 있는데 이는 유선 인터넷상의 각각의 프로토콜 계층을 무선 환경으로 변환시킨 것이다.

기존 HTML 기반의 유선 인터넷 환경의 콘텐츠를 이동 단말기에서 서비스 받기 위해서는 유선 환경에 비해 상대적으로 열악한 전송 속도와 디스플레이 환경을 극복해야만 하는데 WAP 모델에서는 XML 기반의 WML(Wireless Markup Language)과 Gateway를 통해 인코딩된 바이너리 형태의 전송을 통하여 해결하였다. 이 중 WAP Gateway는 WDP (Wireless Transport Layer), WSP(Wireless Session Layer)와 같은 WAP 프로토콜 스택의 구현을 통한 HTTP 프로토콜과

해당 WAP 프로토콜 사이의 변환 및 WML 문서의 인코딩 기능 등을 제공한다[1,10,11,13].

본 프로그램은 무선 인터넷 서비스 제공을 위한 WAP Gateway로 사용자 단말기와 Web/WAP 서버 사이에 위치하여 프로토콜 변환기능과 제공되는 문서의 인코딩 기능 등을 제공하는 미들웨어 시스템이다. 구현된 WAP Gateway는 플랫폼에 독립적으로 작동될 수 있도록 JAVA 언어를 이용하여 구현하였으며 WDP, WSP layer 만을 이용하여 통신하는 WSP 비연결형 통신 기능과 WDP, WTP, WSP layer를 이용하는 WSP 연결형 통신 기능이 모두 수행 가능하다. Gateway 구현에 사용한 WAP 규격은 1.2 버전이다.

II. WAP의 개요

WAP은 1997년에 에릭슨(Ericsson), 모토롤라(Motolora), 노키아(Nokia), 그리고 언와이어드 플래닛(Unwired Planet- 현재의 Phone.com)이 주축으로 WAP 포럼(http://www.wapforum.org)을 창설하고 기존 인터넷에서 사용된 개념과 기술을 무선 서비스 플랫폼에 도입하여 무선 통신망에서도 인터넷을 자유롭게 이용하기 위해 발표된 기술이다[1,13].

1. WAP의 역사

WAP는 GSM(Global System for Mobile Communications), TDMA(Time Division Multiple Access), CDMA(Code Division Multiple Access), CDPD(Cellular Digital Packet Data) 등을 포함한 모든 무선 네트워크에 연결할 수 있는 모바일 컴퓨터용 아키텍처로 에릭슨, 모토롤라, 노키아, 언와이어드 플래닛(Phone.com) 등 이동통신 업체들이 1997년 결성한 WAP 포럼에서 개발했다[27]. 이후 AT&T, 벨사우스 와이어리스 데이터, IBM을 포함해 세계 60여 개의 유력 통신, 컴퓨터 업체들이 WAP 규격을 지원하고 있다. 이와 관련, 핀란드의 노키아 등의 업체에서 최근 WAP 규격의 GSM 방식 휴대전화를, 한국의 삼성전자 등의 업체에서는 CDMA 방식의 휴대전화를 각각 선보이고 이를 통한 인터넷 접속 서비스에 적극 나서고 있다. 또한, 통신망 제공 업체들은 WAP 프로토콜을 통해 전자우편은 물론 날씨, 교통정보, 홈뱅킹(Home-Banking)과 같은 인터넷서비스를 제공하고 있다. 뿐만 아니라, 모바일 컴퓨터 업체들도 단말기에 채용되는 인터넷 검색용 브라우저(Micro Browser) 기능을 강화하기 위해 WAP을 채용하고 있는 실정이다[24].

WAP 포럼에서는 WAP에 대한 프로토콜 규격을 제정하여 발표하고 있으며 1.0, 1.1, 1.2, Technical June 2000과 2.0 draft를 2001년 10월에 발표하였다[13].

2. WAP의 기능 및 구조

WAP은 단순한 하나의 프로토콜이 아니라 무선통신에 관한 여러 가지의 프로토콜의 집합을 지칭하는 이름이다. 이것은 디지털 이동 전화기, 페이저(Pager), 휴대형 개인정보단말기(PDA : Personal Digital Assistant) 또는 그 밖의 무선 단말기들에게 인터넷 통신과 진보된 전화 서비스를 제공하는 프로토콜을 말한다. 이런 서비스를 제공하기 위해서는 기존 WWW 모델과는 다른 형태의 모델이 필요한데 그 이유는 휴대용 기기(hand held device)들이 기존의 고정된 거치형 컴퓨터와는 그 성능에 있어서 많이 부족하기 때문이다. 기존의 컴퓨터와 휴대용 기기 간의 차이점으로는 낮은 성능의 CPU, 적은 양의 메모리, 낮은 전송률, 안정적이지 못한 연결, 안정적이지 못한 전원 공급, 컴퓨터에 비하여 상대적으로 열악한 입출력 장치 등이다[11].

WAP 모델을 도식화하면 그림 1과 같다.

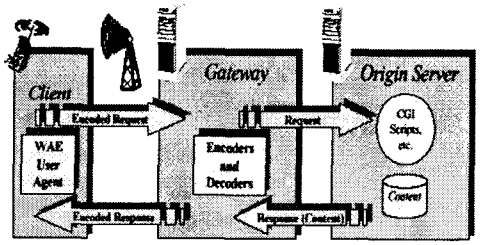

그림 1. WAP 모델

WAE 사용자 에이전트(User Agent)는 사용자에게 특별한 기능을 제공하는 클라이언트 쪽의 소프트웨어로 이것은 WWW에서의 웹 브라우저와 같은 역할을 수행한다. WAP 게이트웨이는 프로토콜 변환 작업을 수행하며 클라이언트와 웹 서버 중간에서 WML를 인코딩 하거나 디코딩 한다. 오리진 서버(Origin Server)는 웹 서버와 같은 역할을 수행한다[2].

그림 2는 WAP을 기반으로 한 네트워크의 예로 무선 인터넷 서비스를 받기 위해서는 두 가지의 통신 방법이 가능하다. 첫 번째 방법은 일단 이동 전화 단말기는 WAP 프록시에 요청(Request) 메시지를 전송하면, 웹 서버는 해당 응답(Response) 메시지를 전송한다. 이때 웹 서버가 WML을 지원하면 직접 WAP 프록시로 전송한다. 만약에 웹 서버가 WML을 지원하지 않는 경우에는 HTML 필터링을 통해 WML으로 변환 후 WAP 프록시에게 전송한다. 두 번째 방법은 무선 전

화기에서 WAP 프록시를 거치지 않고 바로 WTA 서버를 통해서 서비스를 받을 수 있다.

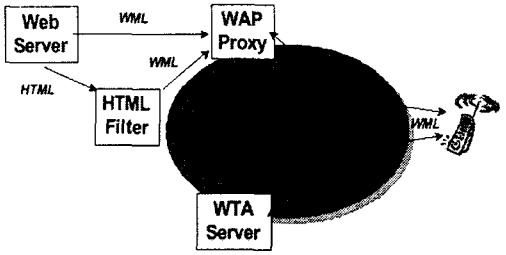

그림 2. WAP 네트워크의 예

WAP의 특징은 낮은 대역폭 등 무선환경에 최적화된 응용 프로토콜이라는 점과, 바이너리 WML과 경량 무선 프로토콜 사용으로 제한된 처리 능력을 갖는 휴대용 기기에 최적의 솔루션 제공한다. 또한, HTML/XML 기반의 인터넷 표준 응용 및 프로토콜을 수용하고 있으며 이질적인 다양한 통신망에 독립적인 프로토콜이다.

3. WAP의 구성요소

WAP 프로토콜 스택은 6개의 레이어 계층으로 구성되며 각 계층은 Application Layer(WAE), Session Layer(WSP), Transaction Layer(WTP), Security Layer(WTLS), Transport Layer(WDP) 그리고 베어러(Bearer)들로 이루어진다. WAP의 아키텍처는 그림 3과 같다[1].

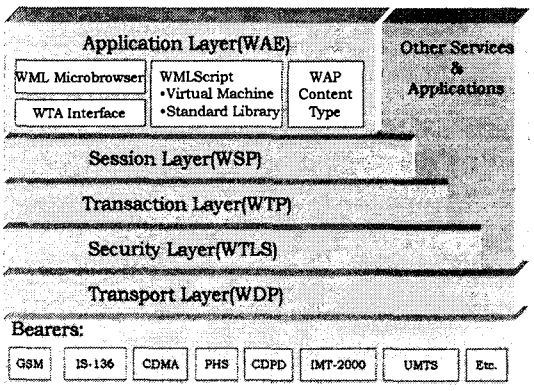

그림 3. WAP 아키텍처

WAP 프로토콜 스택이 TCP/IP 모델에 기반한 HTTP(Hyper Text Transfer Protocol) 아키텍처의 각 레이어를 무선환경에 적합한 해당 WAP 레이어로 변환시킨 관계를 그림으로 나타내면 그림 4와 같다.

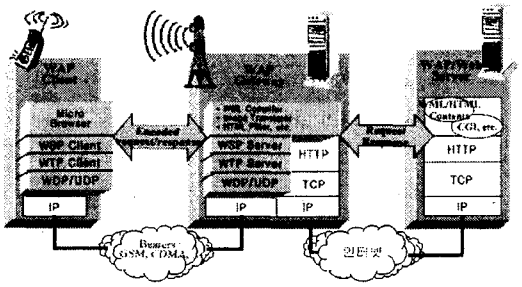

그림 4. 프로토콜 및 레이어 간의 통신

III. WAP 게이트웨이의 설계 및 구현

WAP 게이트웨이는 WAP 클라이언트의 request가 들어오면 이를 해석한 후에 해당 요청에 대한 결과를 Web/WAP 서버로부터 받아들여 이를 무선 전송에 알맞은 데이터 형식으로 인코딩 하여 다시 WAP 클라이언트에게로 reply하여야 한다[1,11].

구현된 WAP 게이트웨이는 플랫폼에 독립적으로 작동될 수 있도록 자바 언어를 이용하여 구현하였으며 WDP, WSP 레이어 만을 이용하여 통신하는 WSP 비연결형 통신 기능과 WDP, WTP, WSP 레이어를 이용하는 WSP 연결형 통신 기능이 모두 수행 가능하다. 제한 사항으로는 WML 컴파일러가 WML 문서만 처리가능하며 Push 기능과 선택 사항인 WTLS 레이어의 구현은 제외하였다. 게이트웨이 구현에 사용한 WAP 규격은 Technical June 2000이다[13].

1. 시스템 구성도

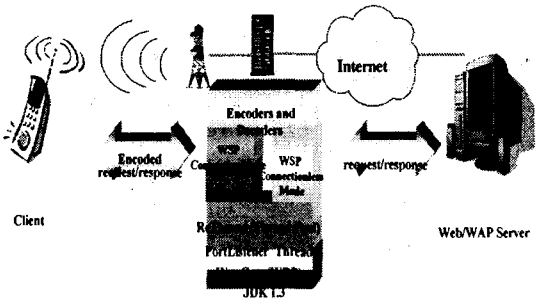

그림 5. WAP 게이트웨이 전체 구성도

게이트웨이는 크게 다음과 같은 다섯 부분으로 구성된다.

- 자바 패키지 - 전체 게이트웨이에서 사용하는 기본 라이브러리로 JDK(Java Development Kit) 1.3을 사용하며, 실제로 사용되는 패키지는 java.net(DatagramSocket, DatagramPacket, InetAddress), java.io(InputStream, BufferedInputStream), java.util(Timer,Vector), java.lang (Thread, String, Exception)이다.
- WDP 레이어 및 관련 스레드 - WAP 클라이언트로부터 UDP 패킷을 송수신하는 부분으로 Runnable 인터페이스를 구현하여 스레드로 실행되는 메인 클래스인 WapGate와 동시에 여러 포트의 UDP 패킷을 감시하는 PortListener 스레드 그리고 Thread Pool인 ReThread로 구성된다.
- WTP 레이어 - WSP 연결형 통신에 함께 사용되며 트랜잭션 관리 기능을 담당한다.
- WSP 레이어 - WSP 레이어의 기능은 크게 연결형과 비연결형으로 구별되는데 전자는 WTP 레이어 함께 사용되어 WSP의 세션 관리 기능을 담당하고 후자는 WSP의 메소드 호출과 응답 그리고 웹 서버와의 통신을 담당한다.
- 인코더 & 디코더 - WML 문서를 무선 전송에 적합한 바이너리 형태의 WML로 상호 변환하는 기능을 수행하며 별도의 WMLComplier 패키지로 구성된다.

2. WDP 레이어의 설계 및 구현

그림 6은 WDP 레이어의 클래스 다이어그램이다. WDP 레이어는 3개의 클래스와 1개의 인터페이스로 구성되는데 GatewayConfigTable 인터페이스는 게이트웨이가 수행되는 포트의 상수 정의용으로 사용되며 나머지 클래스들의 기능은 다음과 같다.

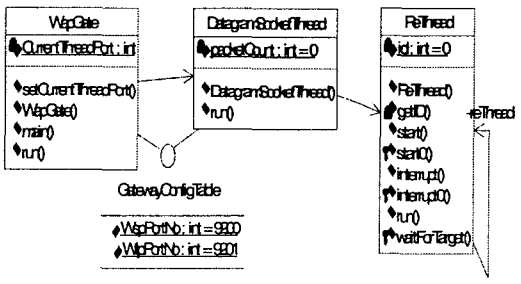

그림 6. WDP 레이어 클래스 다이어그램

2.1 WapGate

WapGate 클래스는 전체 게이트웨이를 실행시키는 main() 메소드를 포함하는 클래스로 실행되면 GatewayConfigTable 인터페이스에서 정의한 WspPortNo(UDP 9200번)와 WtpPortNo(UDP 9201번)의 데이터 입력을 감시하는 Listener 스레드를 Datagram SocketThread로부터 각각 생성시킨다. 또한, WapGate 클래스도 자바의 Runnable 인터페이스를 구현하였기 때문에 Listener 스레드가 UDP 패킷의 입력을 감지하면 WapGate의 run() 메소드를 호출하여 WapGate 클래스 자체가 하나의 스레드로 생성되어 기능을 수행한다. 스레드로 수행되는 WapGate 클래스는 자신의 private 필드인 CurrentThreadPort의 값을 확인하여 그 값이 WspPortNo(UDP 9200번)이면 WspInStream 클래스의 인스턴스를 생성시키고 WtpPortNo(UDP 9201번)이면 WtpInStream 클래스의 인스턴스를 생성시킨다. 또한, 입력된 UDP 패킷을 Source (WAP 클라이언트)의 IP 주소와 포트 번호, Destination(게이트웨이)의 IP 주소와 포트 번호, 그리고 데이터(payload)로 분리하여 상위 레이어로 전달한다. 이로서 입력 패킷의 포트에 따라 WSP 비연결형 요청과 WSP 연결형 요청 모두의 처리가 가능하다.

2.2 DatagramSocketThread

WSP 비연결형 요청과 WSP 연결형 요청을 모두 처리하기 위해서는 동시에 두개 이상 포트의 입력을 감시해야 하는데 이를 스레드로 작성하였다. 이 클래스는 포트 번호를 매개변수로 전달받아 그 포트의 UDP 패킷 입력을 감시하는 기능을 수행하는데 만약 해당 포트로부터의 입력이 감지되면 새로운 WapGate 스레드를 생성하고 생성된 스레드의 CurrentThreadPort 필드를 setCurrentThreadPort() 메소드를 통해 자신의 포트로 설정한다. 또한 스레드의 관리와 재사용을 가능하게 해주는 Thread Pool인 ReThread를 생성시키고 새로 생성된 WapGate 스레드를 Thread Pool에 포함시킨다 [16][17][20].

2.3 ReThread

Port Listener인 DatagramSocketThread 스레드가 UDP 포트의 패킷 입력을 감지하면 새로운 WapGate 스레드를 생성하게 된다. 즉, 클리이언트의 요청이 있을 때마다 새로운 WapGate 스레드를 생성하여 처리하는 것이다. 하지만 자바에서 스레드를 생성하는 것은 매운 느린 작업이며 메모리 낭비가 심하기 때문에 새로 스레드를 생성하는 것 보다 이전에

생성했던 스레드 객체를 재사용하는 것이 바람직하다.

이를 위해 클래스 내부에 Thread 객체의 리스트를 가지고 있으면서 스레드와 유사한 동작을 하는 클래스로서 기능 수행을 완료한 스레드를 소멸시키지 않고 Thread 객체 리스트에 보관하여 스레드의 재사용을 가능하게 해주는 ReThread 클래스를 구현하였다[20].

3. WTP 레이어의 설계 및 구현

WTP 레이어는 WapGate 스레드를 통해 입력된 데이터그램 패킷을 받아 트랜잭션 처리를 하는 WtpInStream 클래스와 WtpInStream 클래스의 처리결과에 따라 전송되는 각 PDU (Protocol Data Unit)들을 규정한 WtpPDU 클래스들로 구성된다. 그림 7은 WTP 레이어의 클래스 다이어그램이다.

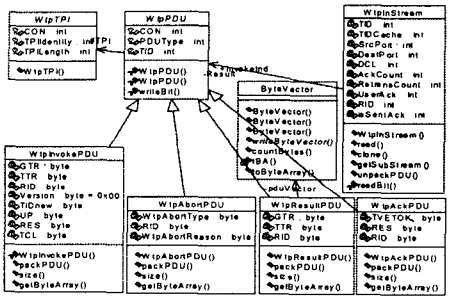

그림 7. WTP 레이어 클래스 다이어그램

3.1 WtpInStream

WtpInStream 클래스는 WapGate 스레드로부터 받아들인 메시지 스트림 형태의 패킷을 분석하여 메시지의 종류를 판단하고 그에 따른 처리를 수행하는 클래스이다. 또한, 패킷 내부의 TID(Transaction Identifier)와 TCL(Transaction Class)을 통해 메시지의 트랜잭션 처리를 담당한다.

이 클래스의 생성자는 Source(WAP 클라이언트)의 IP 주소와 포트 번호, Destination(게이트웨이)의 IP 주소와 포트 번호, 그리고 UDP 패킷의 Payload 부분을 WapGate 스레드로부터 매개변수로서 전달받는다. 그 후에 전달받은 스트림을 해석하기 위해 unpack() 메소드를 실행시킨다.

unpack() 메소드는 먼저 스트림으로부터 TID를 추출하여 이를 Source 주소와 포트 번호, Destination 주소와 포트 번호 쌍과 결합하여 Handle을 만든 후 저장되어 있는 cachedTID와 비교하는 TID 확인 작업을 수행하게 된다. 이 확인 과정을 통해 전송 장치에 의한 패킷의 중복 등을 검사할 수 있다. 올바른 TID로 확인이 되면 PDU의 종류를 확인하여 그에 따라 WTP Initiator와 WTP Responder의 역할을 수행하게 된다. 이 과정 중에서 Result PDU와 같이 상위 레이어의 데이터가 필요한 경우가 발생하면 WspSessionInStream 클래스와 통신을 하게 된다.

3.2 WtpPDU

WTP 레이어의 기능 수행을 위해 필요한 필수적인 PDU들은 Invoke PDU, Result PDU, Acknowledgment PDU, Abort PDU 등 네 가지이다[9]. WtpPDU는 이들 PDU들을 표현하기 위한 클래스로서 슈퍼 클래스인 WtpPDU 클래스와 이를 상속받은 서브 클래스인 WtpInvokePDU, WtpResultPDU, WtpAckPDU, WtpAbortPUD 클래스 등과 각종 통신에 관한 정보를 수록한 TPI(Transport Information Items)에 해당하는 WtpTPI 클래스로 구성된다. 이 클래스들은 packPDU() 메소드를 통해 해당 PDU의 규격에 맞는 octet들로 조합되며 getByteArray() 메소드를 통해 게이트웨이 내부의 통신에 적합한 바이트 배열 형태로 변환이 가능하다.

4. WSP 레이어의 설계 및 구현

WSP 레이어는 크게 비연결형과 연결형의 두 가지 연결 형태로 구분할 수 있는데 이의 처리를 위해서 WspInStream 클래스와 WspSessionInStream 클래스를 각각 구현하였다. WspInStream 클래스는 메소드 관련 PDU를 구현한 GET_PDU, PostPDU, Reply_PDU 클래스의 처리만을 담당하며 WspSessionInStream 클래스는 WspInStream 클래스를 상속받은 서브 클래스로서 메소드 관련 PDU의 처리뿐만 아니라 세션 처리 관련 PDU를 구현한 ConnectPDU, ConnectReplyPDU, RedirectPUD, DisconnectPDU 클래스의 처리도 가능하게 하였다. 또한, Http 프로토콜의 해석을 위한 HttpHeader 클래스와 WSP PDU의 작성을 돕는 유틸리티 클래스인 WspInteger 클래스와 ByteVector 클래스를 추가하였다. 그림 8은 WSP 레이어의 클래스 다이어그램이다.

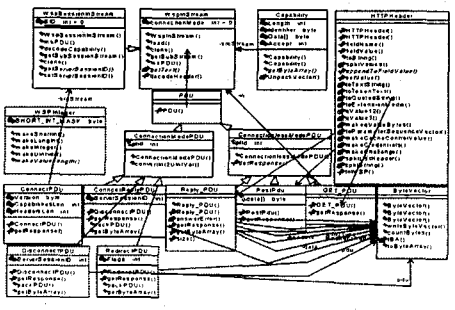

그림 8. WSP 레이어 클래스 다이어그램

4.1 WspInStream

WspInStream 클래스는 WSP 비연결형 통신을 구현하였기 때문에 WapGate 스레드로부터 직접 호출 및 입력 스트림 제공을 받는다. WSP 비연결형 통신의 입력 스트림과 WSP 연결형 통신의 입력 스트림 간에는 약간의 차이가 나타나는데 그 이유는 그림 9.와 같은 WSP PDU의 구조에 기인한다 [9].

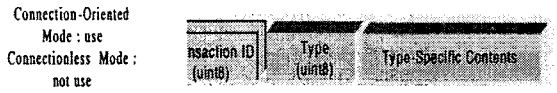

그림 9. WSP PDU의 구조

TID(Transaction ID)는 WSP 비연결형 통신에서만 사용되기 때문에 WSP 비연결형 스트림의 첫 8bit는 TID를 나타내지만 연결형 스트림의 첫 8bit는 PDU 종류를 나타내게 된다. 그 이유는 연결형의 경우 하위 레이어인 WTP 레이어에서 이미 TID 확인 작업을 수행하기 때문에 별도의 TID가 필요하지 않기 때문이다.

이 클래스는 WapGate 스레드로부터 입력받은 스트림을 mkPDU() 메소드를 통해서 해석한 후에 종류가 GET PUD이거나 Post PDU이면 해당 클래스의 인스턴스를 생성시킨 후에 HttpHeader 클래스의 도움을 받아 해석한 후에 getResponse() 메소드를 호출하여 Web/WAP 서버에 요청을 한다. Web/WAP 서버로부터 응답이 오면 그 내용을 WMLCompiler를 통하여 인코딩한 후에 ReplyPUD를 통하여 하위 레이어로 내려보낸다.

4.2 WspSessionInStream

WspSessionInStream 클래스는 WSP 연결형을 구현하였기 때문에 WtpInStream 클래스를 통해 호출 및 스트림 입력을 받는다. WSP 연결형의 경우에는 WSP PDU에 TID가 생략되므로 mkPDU() 메소드는 입력 스트림의 첫 8bit를 해석하여 종류가 Connect PDU나 Disconnect PDU와 같은 세션 관련 PDU이면 해당 처리를 하고, GET PUD나 Post PDU이면 WspInStream 클래스와 같은 처리를 한다.

4.3 PDU

WSP 레이어의 기능이 연결형과 비연결형로 나누어지는 것과 같이 WSP PDU 또한 세션 관리(Session Management) PDU와 메소드 호출(Method Invocation) PDU로 구분된다. 때문에 이들 PDU의 구현 또한 ConnectionPDU 클래스와 ConnectionlessPDU 클래스를 작성한 후에 세션 관리 PDU들은 ConnectionPDU 클래스를 상속받고 메소드 호출 PDU들은 ConnectionlessPDU 클래스를 상속받도록 작성하였다. 그리고 ConnectionPDU 클래스와 ConnectionlessPDU 클래스의 슈퍼 클래스로 PDU 클래스를 두었는데 이를 통해 얻을 수 있는 장점은 각각의 PDU들을 그 종류에 상관없이 매개변수로 전달할 수 있다는 점이다. 세션 관리 PDU들은 Capability 클래스를 이용하여 클라이언트와 서버 사이의 통신 수준 협상을 수행한다.

5. WML 컴파일러의 설계 및 구현

WML 컴파일러는 여러 바이트의 WML 태그들을 짧은 한 바이트의 숫자들로 바꾸어 준다. 이것은 긴 Http 헤더들을 짧은 바이트 형태로 인코딩하는 것과 유사하다[30]. 이 컴파일러는 wmlc라는 별도의 패키지로 작성되었는데 그 구성은 메인 클래스인 WMLCompiler 클래스와 각종 코드 값을 저장한 Table 클래스, 그리고 예외 처리를 위한 Exeception 클래스들로 이루어져 있다. 또한, 내부적으로 XML Parsing을 위해 자바의 JAXP 패키지를 사용한다. 그림 10은 WML 컴파일러의 클래스 다이어그램이다.

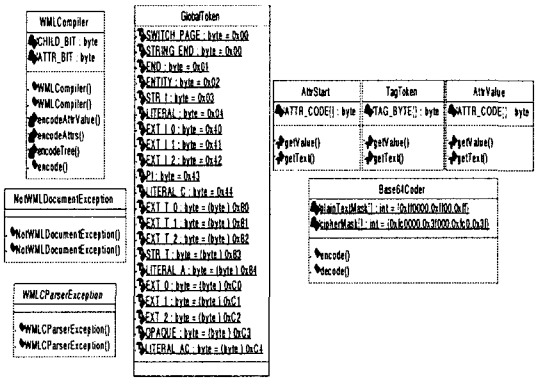

그림 10. WML 컴파일러의 클래스 다이어그램

5.1 WML 컴파일러

WMLCompiler 클래스는 WML을 바이너리 WML로 변환하는 기능을 수행한다. WMLCompiler 클래스 내부의 encode() 메소드는 하나의 WML 문서를 포함한 스트림과 두 개의 문자 집합을 입력으로 받아들인다. 첫 번째 문자 집합은 이 입력 스트림이 어떻게 인코딩되는지를 규정하고 있으며 나머지는 이 문서의 바이너리 표현을 규정하고 있다[13].

IV. 실험 결과 및 분석

WAP 게이트웨이의 기능 실험은 WAP 사용자 에이전트로서의 휴대용 단말기와 WAP 게이트웨이, 그리고 Web/WAP 서버가 상호 통신하는 것이다. 하지만 휴대용 단말기를 WAP 사용자 에이전트로 사용하는 것은 무선 통신망 라우터인 CSD(Circuit Switch Datagram) 라우터를 직접 사용해야하는 어려운 점이 있다. 이런 이유로 본 실험에서는 WAP 시뮬레이터인 노키아 WAP 툴킷 2.0을 WAP 사용자 에이전트로 사용하여 실험하였다.

다음은 Wap 게이트웨이의 테스트 환경 및 실험 모델을 나타낸 것이다.

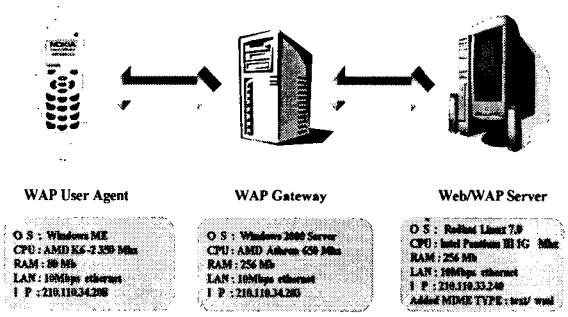

그림 11. 기능 수행 실험 환경

그림 12는 http://210.110.33.240/ex1.wml 요청을 수행한 노키아 클라이언트 화면이며, 그림 13은 위의 요청을 처리한 WAP 게이트웨이의 수행 결과이다.

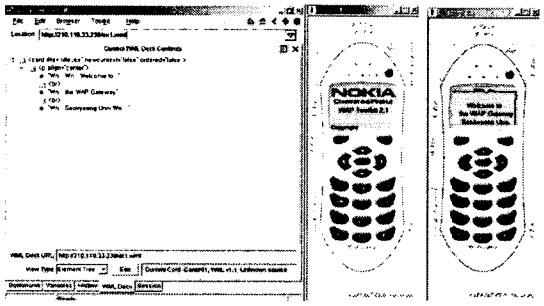

그림 12 실행결과-클라이언트 초기화면 결과화면

WAP 사용자 에이전트와 WAP 프로토콜 스택이 사이에 전송된 패킷의 내용을 분석해 보면 다음과 같다.

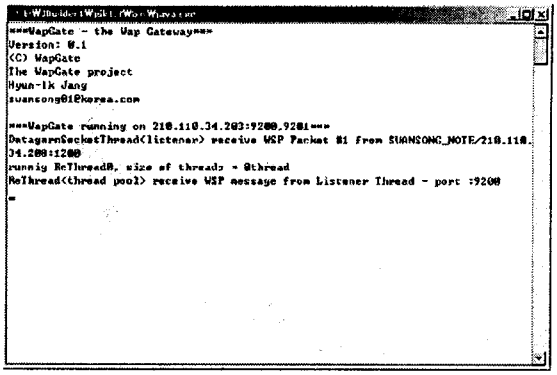

그림 13. 실행결과 - WAP 게이트웨이 화면

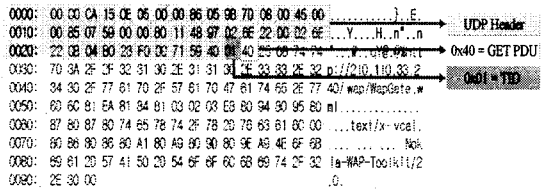

그림 14. 비연결형 패킷분석

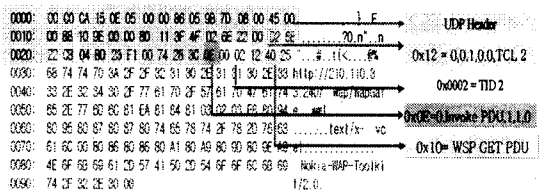

그림 15. 연결형 패킷분석

일반적으로 서버 측 프로그램들은 적은 수의 요청에 대해서는 안정적으로 작동을 하지만 동시에 많은 요청이 발생할 경우에는 그 처리과정에서의 오류로 인해 프로그램의 기능이 중단되거나 혹은 프로그램 자체가 종료되는 경우가 많이 발생한다. 하지만 서버 측 프로그램들은 자신의 처리 한도 이상의 요청이 들어와서 그 요청을 일시적으로 처리하지 못하는 경우가 발생하는 것은 수용이 가능하나 그로 인해 서비스가 중단되어서는 안된다[10,11,13].

구현된 게이트웨이 역시 서버 측 프로그램이므로 게이트웨이 프로그램이 중단되지 않고 클라이언트의 동시 요청을 안정되게 서비스하는지를 알아보기 위해 사용자로부터 게이트웨이의 주소와 UDP 포트 번호, 그리고 동시에 생성할 스레드의 수를 입력받아 일단 그 수만큼 요청 패킷을 가진 스레드를 먼저 생성한 후에 스레드의 생성이 완료되면 요청 패킷을 게이트웨이로 일제히 전송하여 테스트하였다.

다음은 WAP 게이트웨이의 성능 수행 실험 환경과 결과이다.

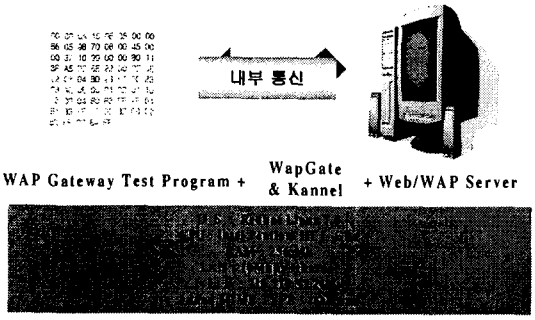

그림 16. 성능 수행 실험 환경

구현한 WAP 프로토콜 스택과 Kannel 시스템과의 성능 비교 실험 결과는 다음과 같다. 스레드를 100개 발생하였을 때의 응답시간과 300개 발생하였을 때의 응답시간, 그리고 100개의 스레드를 20회 발생시켜 평균 응답 시간을 측정하였다. 첫 번째 그림에서 보는바와 같이 스레드의 생성 개수가 많아질수록 Kannel 제품의 응답시간이 점점 증가한 반면 구현된 시스템은 안정된 응답시간을 보였으며, 두 번째 그림에서 보는 보와 같이 스레드의 개수가 100개를 넘어서면서 Kannel 시스템의 경우 일부 스레드의 요청에 대해서만 응답한 후에 시스템이 다운되는 현상이 발생하였지만 구현된 시스템은 안정되게 작동하였다. 세 번째 그림은 구현된 시스템이 Kannel 제품보다 평균 응답 시간에서 좋은 결과를 나타내고 있음을 보여준다.

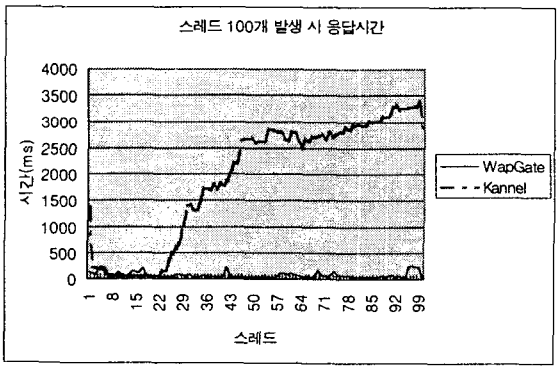

그림 17. 성능 수행 실험 결과 1

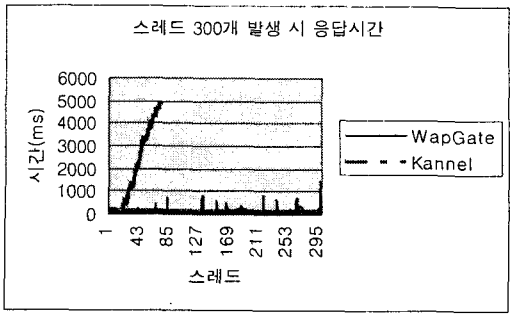

그림 18. 성능 수행 실험 결과 2

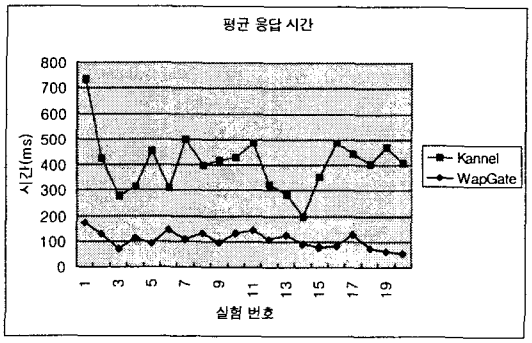

그림 19. 성능 수행 실험 결과 3

V. 결 론

WAP 프로토콜 스택은 최근 그 수요가 폭발적으로 증가하고 있는 무선 인터넷 관련 기술 중에서 WAP 포럼 중심의 WAP 기술과 WAP 기반 환경을 구축하는데 있어 필수적인 프로그램으로 WAP 규격 1.2를 기반으로 플랫폼 독립적인 Java 언어로 설계하고 구현하였다.

WAP 게이트웨이는 WAP 사용자 에이전트의 요청을 각각의 스레드를 생성하여 처리하였는데 매번 새로운 스레드를 생성할 경우에 발생하는 오버헤드와 메모리 낭비 문제를 재사용 가능한 Thread Pool인 ReThread 클래스를 이용하여 해결하였다. 또한, WSP 연결형 통신과 WSP 비연결형 통신 모두를 구현하기 위해서는 버클리 소켓의 select() 함수와 같은 다중 포트 감시 기능이 필요로 한데 자바 언어에서는 이 기능이 제공하지 않기 때문에 이를 포트 Listener인 DatagramSocketThread 클래스를 통해 구현하였다. 또한, 자바 언어로 WAP 게이트웨이를 구현하였기 때문에 별다른 소스 코드의 수정없이 자바 가상 머신이 구현되어 있는 Windows, Unix, Linux 등 여러 플랫폼에서 실행할 수 있다.

또한, 클래스를 통한 상속과 같은 객체 지향 언어의 장점을 이용하여 코드의 재사용성을 높였다.

본 논문에서 구현한 게이트웨이가 보다 효율적으로 작동하기 위해서는 WAP 규격 1.2에서 추가된 Push 기술에 관한 지속적인 연구와 이에 대한 기능 구현이 요구된다.

참 고 문 헌

[1] WAP Forum, "Wireless Application Protocol Architecture Specification," Apr., 1998.

[2] WAP Forum, "Wireless Application Environment Specification," Mar., 2000.

[3] WAP Forum, "Wireless Markup Language Specification," Feb., 2000.

[4] WAP Forum, "WMLScript Language Specification," March, 2000.

[5] WAP Forum, "Wireless Session Protocol Specification," June, 2000.

[6] WAP Forum, "Wireless Transaction Protocol Specification," Feb., 2000.

[7] WAP Forum, "Wireless Datagram Protocol Specification," Feb., 2000.

[8] W. Richard Stevens, UNIX Network Programming Volume 1,2, 2ed, Prentice Hall, 1998.

[9] W. Richard Stevens, TCP/IP Illustrated Volume 1,2 Addison-Wesley, 1995.

[10] Elliotte Rusty Harold, Java Network Programming, 2E, O'Reilly, 2000.

[11] Charles Arehart et al. Professional WAP, WROX Press, 2000.

[12] The Kannel project, http://www.kannel.org.

[13] WAP Forum, http://www.wapforum.org.

이양선(Yang-Sun Lee) 정회원

1985년 동국대학교 전자계산학과 (공학사)
1987년 동국대학교 대학원 컴퓨터공학과 (공학석사)
1993년 동국대학교 대학원 컴퓨터공학과 (공학박사)
1994년 3월 ~ 현재 서경대학교 컴퓨터공학과 교수
2000년 2월 ~ 현재 멀티미디어학회 이사
<관심분야> : 프로그래밍언어, 모바일 컴퓨팅, 분산객체기술

임철수(Chul-Su Lim) 정회원

1985년 서울대학교 계산통계학과 (학사)
1988년 Indiana University(미) 전산과학과 (석사)
1994년 서강대학교 전자계산학과 (박사)
1985년 ~ 1996년 (주)데이콤, (주)신세기통신 근무
1997년 3월 ~ 현재 서경대학교 컴퓨터공학과 교수
<관심분야> : 멀티미디어통신, 무선 인터넷, 차세대인터넷 응용기술

이용학(Yong-Hak Lee) 정회원

1990년 동국대학교 전자계산학과 (공학사)
1992년 동국대학교 대학원 컴퓨터공학과 (공학석사)
1998년 동국대학교 대학원 컴퓨터공학과 (공학박사)
1998년 3월 ~ 현재 김포대학 컴퓨터계열 교수
<관심분야> : 인터넷응용기술, 프로그래밍언어