Journal of Korea Multimedia Society Vol. 13, No. 2, February 2010(pp. 265-278)

다국어 전자 메일 주소 표준화를 위한 메일 시스템 구현

염대영[†], 한동윤[††], 김경석[†††]

요 약

현재 국내에서 서비스하고 있는 대부분의 E-mail 시스템들은 제목, 내용, 첨부 파일 등에 한글이 지원되고 있다. 그러나 E-mail 주소만은 한글이 지원되지 않고 있다. E-mail 주소의 구성 요소 중에서 '@' 마크의 오른쪽 부분인 domain-part는 2003년 IDNA에서 영문자, 숫자, '-' 이외에 다국어를 지원하도록 국제 표준을 마련하여 한글이 사용이 가능하게 되었다. 그러나 '@' 마크의 왼쪽인 local-part는 아직도 다국어를 지원하지 않아서 한글을 사용할 수 없다. 그리하여 본 논문에서는 현재 사용되는 E-mail 주소의 local-part 부분에 다국어를 지원할 수 있도록 IETF에서 진행되고 있는 EAI 표준화 연구에 따라 E-mail 시스템을 모델링하고 구축하고자 한다.

E-mail System Implementation for Standardization of E-mail Address Internationalization

Daeyoung Yeom[†], Dongyun Han[††], Kyongsok Kim[†††]

ABSTRACT

Most recent E-mail systems, that are serviced in Korea, are supporting Hangul in many fields of E-mail(For example, contents, subject, attachment, etc.). However, these systems do not support Hangul in the address field of E-mail. In 2003, IDNA built the international standard in order to support multi-lingual besides the alphabetic character, a number, and '-'. So, Hangul cab be used in the domain part where is the right of '@' mark among the compositional element of the E-mail address. But, the local-part, where is the left of '@' mark, doesn't still support multi-lingual. So Hangul cannot be used in the local-part. In order to support multi-lingual including Hangul in the local-part, this paper designs and implements the E-mail system according to the EAI standardization research that it is progressed in IETF.

Key words: E-mail(전자 메일), Local-part(로컬 파트), Internationalization(다국어화)

1. 서 론

인터넷이 보급된 이후로 현재까지 가장 많이 사용되고 있는 응용 분야는 월드 와이드 웹(World Wide Web)과 전자 메일(E-mail)이다. 두 가지 중 전자 메일은 초기에 영문으로 된 메시지를 간단히 주고받을 수 있는 정도였으나 메일에 관련된 기술이 발전함에 따라 메일에 각종 기능이 추가되고 다국어로 된 메일을 주고받을 수 있게 되었다[1]. 이제 전자 메일은 사람이 수동으로 처리해야 할 업무들을 일부 대신할

※ 교신저자(Corresponding Author) : 염대영, 주소 :부산시 금정구 장전동 산 30번지(609-735), 전화 : 051)510-2874, FAX : 051)515-2208, E-mail : dyyoum@asadal.pnu.kr
접수일 : 2009년 6월 30일, 수정일 : 2009년 9월 5일
완료일 : 2009년 11월 16일
[†] 정회원, 부산대학교 컴퓨터공학과 박사과정
[††] 정회원, 부산대학교 컴퓨터공학과 박사과정
(E-mail : dyhan@asadal.pnu.kr)
[†††] 정회원, 부산대학교 정보컴퓨터공학부 교수
(E-mail : gimgs0@asadal.pnu..kr)
※ 본 연구는 부산대학교 자유과제 학술연구비(2년)에 의하여 연구되었음

수 있는 정도까지 발전하여 현대 사회에서는 없어서는 안 될 인터넷의 킬러앱이 되었다. 그러나 메일과 관련된 기술이 발전하면서도 메일의 제목이나 내용 등은 다국어로 가능하게 되었지만 아직까지 메일의 주소 전체를 다국어로 보내는 일은 불가능한 상태이다. 현재 우리나라는 인터넷의 발전, 개인용 컴퓨터의 발전으로 인해 사회 일부 특수 계층이 아니라 남녀노소, 빈부격차를 불문하고 컴퓨터를 쉽게 접할 수 있는 상황이 되었으며, 또한 문맹률이 1%미만으로 대부분의 국민들이 한글을 읽고 쓸 수 있다. 그러나 영어는 사용률이 한글만큼 높지 못해서 영문자와 숫자, 일부 특수 문자만으로 표현할 수 있는 전자 메일 주소 때문에 인터넷 보급률과 전자 메일의 사용률은 많은 차이가 나는 편이다. 특히 일부 영어를 잘 접하지 못한 세대 같은 경우, 전자 메일을 사용하기 위해 한글 자판의 본인의 이름을 영어 입력 모드로 전환한 후 나오는 알파벳의 조합으로 만드는 등 영어 사용에 익숙하지 않은 경향이 있다. 국내 뿐만 아니라 해외 비영어권국가에서 인터넷의 킬러앱이 된 전자 메일의 주소의 다국어화는 시급한 문제로 등장하였다. 즉, 우리나라 및 다른 비영어권 국가에서의 원활한 전자 메일 사용과 여러 나라 문화의 다양성 접목을 통해 더 많은 사용자 계층이 전자 메일을 사용할 수 있도록 영문자와 숫자, 일부 특수문자로만 표현이 되는 전자 메일 주소의 다국어화는 꼭 필요한 것이다.

그리하여 IDNA (Internationalizing Domain Names in Application)에 의해 2003년 IDN (International Domain Name)이라는 도메인 이름이 만들어졌다[2]. 이것은 ASCII 코드가 아닌 다른 문자를 포함하는 도메인 이름으로 다국어 도메인 또는 자국어 도메인이라고도 부른다. 이러한 다국어 도메인 이름은 액센트 표시를 포함하는 유니코드로 된 많은 유럽의 언어나 한국어, 일본어, 중국어 등과 같이 비 라틴어계의 문자를 포함할 수 있도록 한 것이다. IDN은 1998년부터 논의가 시작되어 클라이언트에서 문자열을 기존의 표준에 맞게 변환해서 전송하는 방식을 IDNA라 명명하였고 2002년에는 IDNA를 퓨니코드라는 코드 변환방식을 이용하기로 한 표준안이 만들어졌다. 그리고 2003년에는 이와 관련된 RFC (Request For Comments) 표준문서들을 IETF (Internet Engineering Task Force)가 최종 작성하여, ICANN (The Internet Corporation for Assigned Names and Numbers)의 관리 하에 각 레지스트리들이 등록업무를 수행하고 있다.

이로 인해서 전자 메일 주소의 domain-part, 즉 '@' 마크의 오른쪽 부분을 다국어로 사용할 수 있게 되었다. 예를 들어 'dyyeom@데이터베이스.kr'과 같은 전자 메일 주소를 사용할 수 있게 된 것이다. 그러나 주로 계정이라고 불리는 전자 메일 주소의 local-part는 아직까지 다국어로 된 표준이 제정되지 않았다. 즉, '대영@데이터베이스.kr'과 같은 형태의 주소는 아직 사용할 수 없다는 것이다. 그래서 IETF 회의를 통해 세계 각국에서 local-part를 다국어화하는 작업이 진행 중에 있으며 몇 가지 표준화에 대한 Draft들이 나온 상태이다. 그리하여 본 논문은 Draft를 기반으로 하여 '@'마크의 왼쪽 부분인 local-part까지 다국어화를 지원하는 서버를 모델링하고 사용할 수 있도록 구축을 해 보았다.

2. 전자 메일 관련 연구 동향

2.1 전자 메일에서 다국어를 지원하기 위한 연구

2.1.1 UTF-8 (8bit UCS Transformation Format)

UTF-8은 유니코드를 위한 가변 길이 문자 인코딩 방식 중 하나로, 켄 톰프슨과 롭 파이프가 만들었다. 본래는 FSS-UTF (File System Safe UCS (Universal Character Set)/Unicode Transformation Format)라는 이름으로 제안되었다.

UTF-8 인코딩은 유니코드 한 문자를 나타내기 위해 1바이트에서 4바이트까지 사용한다. 예를 들어서, U+0000부터 U+007F 범위에 있는 ASCII 문자들은 UTF-8에서 1바이트만으로 표시된다. 4바이트로 표현되는 문자는 모두 BMP (Basic Multilingual Plane) 바깥의 유니코드 문자이며, 거의 사용되지 않는다. UTF-16과 UTF-8 중 어느 인코딩이 더 적은 바이트를 사용하는지는 문자열에서 사용된 코드 포인트에 따라 달라지며, 실제로 DEFLATE와 같은 일반적인 압축 알고리즘을 사용할 경우 이 차이는 무시할 수 있을 정도이다. 이러한 압축 알고리즘을 사용하기 힘들고 코드의 크기가 중요할 경우 유니코드 표준 압축 방식을 대신 사용할 수 있다.

UTF-8은 여러 표준 문서에서 다른 방법으로 정의되어 있지만, 일반적인 구조는 모두 동일하다. 유

표 1. UTF-8의 구조

코드 범위 (16진법)	UTF-16BE 표현 (이진법)	UTF-8 표현 (이진법)	설 명
000000-00007F	00000000 0xxxxxxx	0xxxxxxx	ASCII와 동일한 범위
000080-0007FF	00000xxx xxxxxxxx	110xxxxx 10xxxxxx	첫 바이트는 110 또는 1110으로 시작하고, 나머지 바이트들은 10으로 시작함
000800-00FFFF	xxxxxxxx xxxxxxxx	1110xxxx 10xxxxxx 10xxxxxx	
010000-10FFFF	110110yy yyxxxxxx 110111xx xxxxxxxx	11110zzz 10zzxxxx 10xxxxxx 10xxxxxx	UTF-16 서로 게이트 쌍 영역 (yyyy=zzzzz-1). UTF-8로 표시된 비트 패턴은 실제 코드 포인트와 동일하다.

니코드의 코드 포인트를 나타내는 비트들은 여러 부분으로 나뉘어서, UTF-8로 표현된 바이트의 하위 비트들에 들어간다. U+007F까지의 문자는 7비트 ASCII 문자와 동일한 방법으로 표시되며, 그 이후 문자는 다음과 같은 4바이트까지의 비트 패턴으로 표시된다. 7비트 ASCII 문자와 혼동되지 않게 하기 위하여 모든 바이트들의 최상위 비트는 1이다[3].

UTF-8은 다음과 같은 성질을 가지고 있다.

- 1바이트로 표시된 문자의 최상위 비트는 항상 0이다.
- 2바이트 이상으로 표시된 문자의 경우, 첫 바이트의 상위 비트들이 그 문자를 표시하는데 필요한 바이트 수를 결정한다. 예를 들어서 2바이트는 110으로 시작하고, 3바이트는 1110으로 시작한다.
- 첫 바이트가 아닌 나머지 바이트들은 상위 2비트가 항상 10이다.

2.1.2 MIME (Multipurpose Internet Mail Extensions)

MIME은 전자 메일을 위한 인터넷 표준 포맷이다. 인터넷 전자 메일 전송 프로토콜인 SMTP는 7비트 ASCII 문자만을 지원하는데 이것은 7비트 ASCII 문자로 표현할 수 없는 영어 이외의 언어로 쓰인 전자 메일은 제대로 전송될 수 없다는 것을 의미한다. 그래서 MIME은 ASCII가 아닌 문자 인코딩을 이용해 영어가 아닌 다른 언어로 된 전자 메일을 보낼 수 있는 방식을 정의한다. 또한 그림, 음악, 영화, 컴퓨터 프로그램과 같은 8비트 바이너리 파일을 전자 메일로 보낼 수 있도록 한다[4]. MIME은 또한 전자 메일과 비슷한 형식의 메시지를 사용하는 HTTP와 같은 통신 프로토콜의 기본 구성 요소이다. 메시지를 MIME 형식으로 변환하는 것은 전자 메일 프로그램이나 서버 상에서 자동으로 이루어진다.

전자 메일의 기본적인 형식은 RFC 2821에서 정의하고 있다. 이 문서는 RFC 822를 대체한다. 이 문서는 텍스트 전자 메일의 헤더와 본문의 형식을 명시하고 있으며, 그 중에는 우리에게 익숙한 'To:', 'Subject:', 'From:', 'Date:' 등의 헤더가 포함되어 있다. MIME은 메시지의 종류를 나타내는 content-type, 메시지 인코딩 방식을 나타내는 content-transfer-encoding과 같은 추가적인 전자 메일 헤더를 정의하고 있다. MIME은 또한 ASCII가 아닌 문자를 전자 메일 헤더로 사용할 수 있도록 규정하고 있다.

또한, MIME은 확장 가능하다. MIME 표준은 새로운 content-type과 또 다른 MIME 속성 값을 등록할 수 있는 방법을 정의하고 있다.

MIME의 명시적인 목표 중 하나는 기존 전자 메일 시스템과의 호환성이다. MIME을 지원하는 클라이언트에서 비 MIME이 제대로 표시될 수 있고, 반대로 MIME을 지원하지 않는 클라이언트에서 간단한 MIME 메시지가 표시될 수 있다.

MIME 헤더는 그림 1과 같이 정의할 수 있다.

2.1.3 IDN (Internationalized Domain Names)

IDN이란 도메인 이름의 영역에 영문 도메인이 아

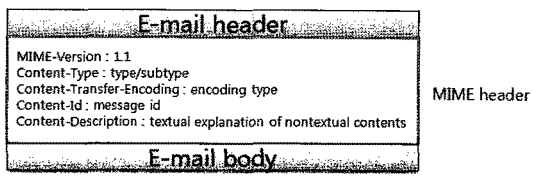


그림 1. MIME 헤더의 정의

닌 한글과 같은 자국어를 사용한 도메인을 말한다. 향후 몇 년 안에 인터넷 전체 사용자의 66% 이상이 비영어권 사용자가 될 것으로 추정되는 상황 속에서 추진된 기술이다.

추진된 현황은 1998년 APNG (Asia Pacific Networking Group)에서의 다국어 도메인 서비스 개발 프로젝트를 시작으로 많은 논의와 시행착오를 거쳐 2002년 10월 다국어 도메인의 IETF 국제 표준안이 확정되었다. 한글을 포함한 다국어를 지원할 수 있는 DNS (Domain Name Server)는 영어 뿐만 아니라, 전 세계의 모든 언어로 도메인 이름을 표현해야 하므로 인프라에 많은 변화가 필요하고 해결해야 할 많은 과제들을 가지고 있다.

따라서 표준의 방향은 DNS가 인식할 수 있도록 다국어 도메인을 영어, 숫자, 하이픈(-)으로만 구성된 아스키 문자열로 변환(ACE : ASCII Compatible Encoding)하는 과정이 클라이언트 어플리케이션 레벨에서 수행되는 방식으로 추진되었다(IDNA : Internationalized Domain Names in Application). 물론 이 방식은 단기간에 다국어 도메인을 구현할 수 있는 방안이었지만 안정적이지 못하다는 많은 엔지니어의 반대에도 불구하고, 결국 2002년 10월 IDN 표준안으로 확정되었으며 2003년 3월에 관련 문서들이 RFC 표준문서로 등록되었다.

이와 동시에, 비영어권 국가 및 몇몇 gTLD (Generic Top Level Domain) 레지스트리를 중심으로 다국어 도메인 서비스 시행 및 언어별 이슈를 반영하기 위한 움직임이 활발히 진행되어 왔는데 그 대표적인 단체는 각국 NIC (Network Information Center)와 업체의 콘소시엄 형태인 MINC (Multilingual Internet Names Consortium)와 한중일 NIC 엔지니어 모임인 JET (Joint Engineering Team)이다.

표준안으로 확정된 문서는 IDNA[RFC 3490], Nameprep[RFC 3491], Punnycode[RFC 3492]이다. 각 표준의 상세 기술 내용을 살펴보면 다음과 같다.

비영어권 국가들은 각기 EUC-KR (KSC 5601), Shift-JIS, GBK, Big5 등 자국 언어 코드를 가지고 있어서 코드 체계가 다르므로 한국에서 중국으로 다국어 도메인 이름을 질의하는 경우 답변을 하지 못하는 문제가 발생하기 때문에 아스키 문자열로 변환하기 전에 전 세계에 공통적으로 적용할 수 있는 일원화된 코드체계인 유니코드(Unicode)를 사용하는 방법을 이용하게 되었다[2].

2.2 전자 메일 프로토콜

2.2.1 ESMTP (Extended Simple Mail Transfer Protocol)

전자 메일을 보내고 받기 위한 인터넷 표준 프로토콜인 SMTP는 RFC 821에 정의되어 있으며 계속 내용이 추가되어 표준이 발전하면서 최근에는 RFC 5321에 정의된 표준을 사용하고 있다. 오늘날 널리 사용되는 ESMTP (Extended SMTP)도 SMTP의 연장선상에 있다.

1982년 SMTP가 최초로 형성된 이래, 상당한 변화와 발전을 거듭해 현재의 메일 시스템을 위한 기초를 형성하고 있다. 메일 서버간의 송수신 뿐만 아니라, 메일 클라이언트에서 메일 서버로 메일을 보낼 때에도 사용되는 경우가 많다.

SMTP는 단순 텍스트 메시지 포맷을 명령 기반의 프로토콜로 25번 PORT로 통신한다. SMTP는 클라이언트가 서버에 명령을 하면 서버는 클라이언트가 보내온 명령에 대해 처리한 후 그 명령에 대한 응답을 보내주는 동기식 통신 프로토콜이다. 그리고 SMTP 메시지는 ASCII 라인피드 문자(LF) 앞에 나오는 ASCII 캐리지 리턴 문자(CF)로 행을 분리한다. 이 행 분리자는 플랫폼에 독립적이고 메시지가 처리되는 모든 운영체제에 적용이 가능하다. 한 가지 유의할 점은 SMTP 메시지의 경로다. 유저 이름은 최대 64문자까지, 도메인 또한 최대 64문자까지이다. 그리고 명령어 라인(command line)은 CRLF를 포함하여 512문자 미만으로 제한하고 있다. 그리고 응답 라인(reply line)은 명령어 라인과 동일하게 CRLF를 포함해 512문자 미만으로 제한한다. 텍스트 라인(text line)은 CRLF를 포함하여 1000문자 미만으로 제한하고 있다. 하지만 대부분의 MTA와 MUA가 긴 행을 잘 처리하지 못하므로 관행적으로 한 행을 80문자 미만으로 작성하게 되어 있다.

그래서 나온 것이 ESMTP로 1995년 RFC 1869로 정의되었다. SMTP의 문제점을 해결하기 위해 IETF에서 만든 것으로 SMTP의 문제점 해결 뿐만 아니라 앞으로 발생할 수 있는 다양한 전송 요구를 수용할 수 있도록 프로토콜 표준 확장에 관한 프레임

워크이다[5].

ESMTP로 확장되면서 추가된 기능들로는 다음과 같은 것들이 있다..

- 8BITMIME (8bit data transmission)
- ATRN (Authenticated Turn for On-Demand Mail Relay) : 하나 이상의 도메인을 매개 변수로 사용 가능.
- SMTP-AUTH (Authenticated SMTP) : SMTP 로그인 인증
- CHUNKING (Chunking) : SMTP 호스트가 데이터의 끝부분을 계속적으로 검사하지 않도록 메시지 전체 바이트 수가 포함된 인수와 함께 보내고 수신 서버에서는 메시지 바이트를 계산하여 받은 값과 같을 경우 모든 메시지 데이터를 받았다고 가정
- DSN (Delivery status notification) : 메일 배달 상태 알림
- ETRN (Extended Turn) : 다른 서버에서 자체 전자 메일 메시지를 모두 보내도록 요청 가능
- HELP (Supply helpful information) : 도움말
- PIPELINING (Command pipelining) : 각 명령 이후 응답을 기다리지 않고 명령어를 전송할 수 있게 함
- SIZE (Message size declaration) : 메시지 크기 확인
- STRATTLS (Transport layer security) : 전송 계층의 보안

ESMTP에 위와 같은 기능들이 추가가 되었으나 메일 주소를 다국어로 보낼 수 있는 기능이 존재하지 않는다.

2.2.2 POP3 (Post Office Protocol 3)와 IMAP (Internet Messaging Access Protocol)

POP3는 전자 메일을 수신하기 위한 표준 프로토콜인 POP의 가장 최신 버전이다. POP3는 서버가 사용자를 위해 전자 메일을 수신하고 그 내용을 보관하기 위해 사용되는 클라이언트/서버 프로토콜이다. 사용자(또는 전자 메일 수신용 클라이언트 프로그램)는 주기적으로 서버에 있는 자신의 메일 수신함을 점검하고, 만약 수신된 메일이 있으면 클라이언트 쪽으로 다운로드한다[6]. POP3는 가장 유명한 전자

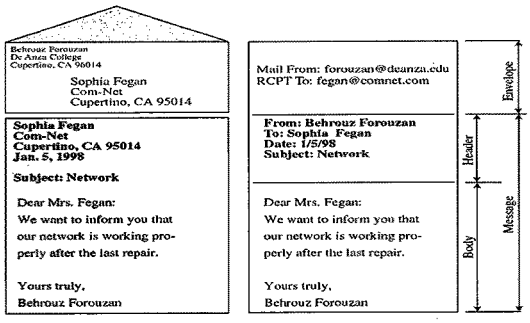


그림 2. SMTP에서 메일을 보내는 구조

메일 제품 중의 하나인 유도라에 적용되었으며, 넷스케이프와 마이크로소프트 익스플로러 브라우저에도 역시 적용되었다.

POP3의 대안으로 사용할 수 있는 프로토콜이 IMAP이다. IMAP은 사용자가 설령 자신의 클라이언트 컴퓨터에 메일이 있다고 해도 서버에 있는 메일을 본다. 자신의 컴퓨터에서 지워진 메일도 서버에는 아직 남아 있다. 메일은 서버에 보관될 수 있고 검색할 수 있다[7].

즉, POP3는 '보관하고 전달하는'서비스라고 생각할 수 있으며, IMAP은 '원격지 파일 서버' 라고 생각할 수 있다. 그러나 현재의 POP3는 많은 발전을 거듭해 IMAP과 같이 메일서버에 메일을 그대로 남길 수 있어 자신의 컴퓨터에서 메일을 지우더라도 다시 메일 서버에서 가져올 수가 있다.

POP과 IMAP은 둘 모두 전자 메일을 받는 일을 담당하므로, 인터넷을 통해 전자 메일을 전달하는 프로토콜인 SMTP와는 다른 것이며 송신자가 SMTP를 이용해서 메일을 보내면 상대방 메일 서버에 있는 MTA (Mail Transfer Agent)가 수신자를 대신해서 그것을 수신한다. 그 후 그 메일을 POP3나 IMAP을 이용하여 수신자가 읽게 되는 것이다.

3. EAI를 위한 IETF에서의 표준화 진행 과정

3.1 EAI 지원을 위한 SMTP (Simple Mail Transfer Protocol)의 UTF-8 확장

EAI (E-mail Address Internationalization) 지원을 위한 SMTP의 확장 표준은 지난 9월 RFC 5336 SMTP Extension for Internationalized E-mail Address라는 제목으로 정해졌다. 기본적인 내용은

전자 메일의 Envelope부분과 Header부분에 다국어로 된 전자 메일 주소를 사용할 수 있게 하는 SMTP 확장을 규정한 것이다[8].

SMTP의 확장은 EAI를 지원하기 위한 것이며, SMTP의 HELO의 새로운 버전으로 EHLO (Extended HELO) 키워드가 추가되었다. 둘째로 "Mail From"과 "RCPT To" 명령에 선택적 매개 변수로 ALT-ADDRESS가 추가되었다. ALT-ADDRESS는 다국어로 된 주소를 지원하지 않는 메일 서버로 메일을 보낼 때 기존 형식으로 된 메일 주소를 저장해 놓은 매개 변수이다. 셋째로 "VRFY"와 "EXPN" 명령에 UTF8REPLY라는 선택적 매개 변수를 추가하여 해당 명령을 클라이언트가 서버로 보냈을 때 서버가 UTF-8 글자를 보내면 UTF-8글자를 받아들일 수 있다는 것을 의미한다. 넷째로 8BIT MIME을 지원해야 하며, 다섯째로 메일박스 이름에 UTF-8 글자를 쓸 수 있도록 해야 한다. 마지막으로 "MAIL From"과 "RCPT To" 명령행의 최대 길이는 ALT-ADDRESS 지원을 위해 460자로 늘어났다.

3.2 EAI 지원을 위한 메일 헤더 확장

EAI 지원을 위한 전자 메일의 Header부분을 확장하기 위한 표준이 9월 RFC 5335 Internationalized E-mail Headers로 정해졌다. 주요 내용으로는 첫째, MIME 규정(RFC 2045)에 따라 ASCII 아닌 내용을 보낼 수 있어야 한다. 둘째, 전자 메일 Header부분에 주소와 정보를 나타낼 수 있어야 하며, 셋째로 Envelope Address에 다국어를 나타낼 수 있어야 한다는 것이다[9].

그러나 Header field에 UTF-8로 된 문자를 쓸 수 있다는 것은 Header field의 값을 UTF-8로 된 문자를 쓸 수 있다는 것이지 Header field 이름을 UTF-8로 쓸 수 있다는 것은 아니다.

3.3 기존 메일 서버와의 호환을 위한 다운그레이드

기존의 메일 시스템은 SMTP Envelope과 Header field의 값은 ASCII만을 허용하게 되어 있으나 EAI에서는 SMTP의 UTF-8 확장과 함께 un-ASCII 값을 허용하게 되었다. 그러나 표준화가 된다 하더라도 전 세계의 모든 메일 서버가 동시에 표준을 지원하지 않을 것이기에 EAI 표준을 지원하지 않는 메일 서버로 메일을 보낼 때 거절되는 것을 회피하기 위한 방법으로 다운그레이드 방법을 고안하였다. 기본적인 내용은 다운그레이드 메커니즘을 사용하여 Envelope과 Header field를 ASCII형태로 변환하여 보내는 것이다. 다운그레이드는 새로운 프로토콜을 정의하는 것이 아닌 새로운 Header field를 정의하여 사용하는 것을 목적으로 한다[10].

다운그레이드는 크게 네 부분으로 나누어져 있다. 첫째, 새로운 Header field의 정의이다. 새로운 Header field는 아래와 같다.

- Downgraded-Mail-From
- Downgraded-Rcpt-To

즉, 다운그레이드 된 SMTP Envelope정보는 Original non-ASCII Address와 All-ASCII Address로 구성된다.

두 번째로, SMTP의 다운그레이드이다. Mail From의 <reverse-path>와 RCPT To의 <forward-path>는 <path>가 non-ASCII Address를 포함하고 ALT-ADDRESS에 ASCII로 정의된 주소가 있을 경우에만 다운그레이드가 가능하며, Header field와 body MIME Header field가 non-ASCII 문자를 포함하고 있다면 반드시 다운그레이드를 해야 한다. 또한 "RCPT To" 명령이 DSN extension을 지원한다면 ORCPT 매개 변수를 사용할 수 있는데 non-ASCII를 포함하고 있다면 UTF-8로 반드시 변환해야 한다.

세 번째는 전자 메일 Header field 다운그레이드이다. Header field가 non-ASCII 문자를 포함하고 있다면 UTF-8로 인코딩하고 메일박스는 ASCII로 수정한다.

네 번째는 MIME Header field 다운그레이드이다. 이 부분들은 non-ASCII를 포함하는 부분이 있다면 모두 다운그레이드를 실행한다.

3.4 UTF-8을 지원하기 위한 POP3의 확장

UTF-8을 지원하기 위한 POP3는 아직 표준이 확정되지는 않았지만 네 번째 드래프트까지는 나와 있으며 주요 내용은 UTF-8을 지원하는 확장된 POP3를 지원하며 ASCII가 아닌 로그인 이름과 사용자에게 UTF-8 Protocol-level error 문자를 지원을 할 수 있게 하는 것이다[11].

크게 두 가지가 중요한 내용인데 첫째는 LANG capability 추가이다. LANG을 서버 측으로 보냈을 때 서버 측 응답은 POP3를 사용할 때 사용자가 사용할 수 있는 언어를 결정할 수 있도록 가이드라인을 제공하며, LANG을 사용하여 언어를 바꾸기 전까지 사용할 I-default 언어로 응답을 하며 LANG을 사용하여 성공적으로 언어를 바꾸었을 시에는 해당 언어로 구성된 응답들을 볼 수가 있다. 해당 언어들은 기본적으로 UTF-8 인코딩이 된다. LANG이 실패 시에는 "-ERR" 응답과 함께 I-default 언어로 응답을 할 수 있다.

두 번째는 UTF8 capability 추가이다. UTF8 capability는 세션을 ASCII에서 UTF-8 모드로 바꾸어 준다. 그러나 UTF-8을 지원하지 않는 환경에서는 UTF-8로 구성된 메시지를 수신할 때 반드시 EAI 다운그레이드 메커니즘에 의해서 다운그레이드를 해야 한다.

4. EAI 지원 메일 시스템 모델링

4.1 메일 시스템에서 EAI 지원하기 위한 구현 방법

EAI를 지원하는 메일 시스템을 만들기 위해 표준 문서들을 참조하면 정확하게 어떤 방법으로 구현해야 한다고 정의되어 있지 않다. 메일의 송신과 수신 시, 출력되는 결과물이 어떻게 되어야만 하는가, 어떤 명령어 및 키워드들이 추가될 것인가만 기재되어 있다. 또한, 국내에서 아직 이 표준에 맞춰서 개발된 메일 시스템은 없으며, 해외에서는 일본 NIC(Network Information Center)와 캐나다의 afilias라는 기업에서 개발 중이다. 그러나 일본의 NIC이나 캐나다의 afilias에서도 개발 중인 소스가 공개되어 있지 않아서, 내부의 모델링이나 어떤 언어로 어떻게 구현되어 있는지는 알 수가 없다. 단지 IETF 회의 때 테스트 버전을 시연한 것만 알 수 있었다. 그래서 본 논문에서는 독자적으로 표준에 따라서 개발하기 위해 SMTP, POP3 서버들을 새로이 모델링하고, 개발하였다. 그리고 표준에서 제시되지 않은 POP3 사용을 위한 클라이언트도 모델링하고, 개발하였다. 그래서 본 논문에서는 같이 현재의 메일 시스템과의 호환성을 고려하여 다음과 같이 Case by case로 구현 방법들을 제안하고자 한다.

(1) EAI 미지원 메일 송신 서버 A1
(2) EAI 지원 메일 송신 서버 A2
(3) EAI 미지원 메일 수신 서버 B1
(4) EAI 지원 메일 수신 서버 B2

○ Case 1 : A1 → B1

- 현재 널리 사용되고 있는 메일 시스템으로, 영문자 및 숫자, 일부 특수 문자를 사용하는 전자 메일 주소만 사용할 수 있다.

○ Case 2 : A1 → B2

- 현재 사용되고 있는 메일 서버에서 EAI를 지원하는 메일 서버로 메일을 보내는 경우인데, EAI를 지원하는 서버에서는 당연히 영문자를 사용할 수 있으므로 정상적인 작동이 가능하다.

○ Case 3 : A2 → B1

- EAI를 지원하는 메일 서버에서 현재 사용되고 있는 메일 서버로 메일을 보내는 경우인데, 이러한 경우에는 문제가 발생하게 된다. 그 이유는 EAI 지원 서버에서는 메일 주소 표기를 UTF-8 형태로 인코딩을 하는데 반해, 현재 사용되고 있는 메일 서버에서는 UTF-8으로 인코딩된 메일 주소를 인식하지 못하기 때문이다. 그러므로 EAI 지원 서버에서 미지원 서버로 메일을 송신 시에는 메일 주소의 특별한 변환 방법이 요구된다.

○ Case 4 : A2 → B2

- 각 메일 송수신 서버가 EAI를 지원하므로 UTF-8으로 인코딩된 메일 주소를 사용함에 이상이 없다.

위의 내용에 따라, EAI 지원 메일 시스템을 구현하기 위해서는 기존 메일 시스템과의 호환성을 위해 EAI 미지원 메일 시스템으로 메일을 송신할 때, 메일 주소의 다운그레이드(UTF-8에서 기존 메일 주소 형태)라는 과정이 필요한 것을 알 수 있다. 또한 각 EAI를 지원하는 메일 서버들 간에 메일을 송수신할 때에도 메일 주소를 정상적으로 인코딩과 디코딩하는 과정이 필요하다.

앞선 과정에서는 EAI를 지원하는 메일 시스템을 직접적으로 구현하고 서버 내에서 변환하는 과정을 포함하는 방법을 고려해 보았다. 이와 다른 방법으로

현재의 메일 시스템을 그대로 사용하고 중간에서 필터처럼 메일 주소를 업그레이드(기존 메일 주소 형태에서 UTF-8)와 다운그레이드를 해 주는 서버를 송신 측에 따로 두는 방법을 생각해 볼 수 있다.

(5) 메일 송수신 서버 중간에 업그레이드와 다운그레이드를 해주는 서버 : C

○ Case 5 : A1 → C(변환 과정 없음) → B1
○ Case 6 : A1 → C(메일 주소 업그레이드) → B2
○ Case 7 : A2 → C(메일 주소 다운그레이드) → B1
○ Case 8 : A2 → C(변환 과정 없음) → B2

이러한 방법은 EAI를 지원하는 메일 서버와 지원하지 않는 서버가 공존할 때 사용할 수 있는 과도기적 방법으로 사용할 수 있다. 그러나 이 방법은 기존에 사용되는 메일 시스템을 별다른 수정 없이 그대로 사용할 수 있다는 장점이 있으나, C에 송신측 계정 정보를 중복하여 저장해야 하는 문제점과 C에서 모든 메일을 중간에서 가로채서 메일 주소를 바꾸는 작업을 하기 때문에 보안상의 문제가 생길 가능성이 있다.

그러므로 EAI를 지원하는 메일 서버를 새로 만드는 것과 동시에 기존 메일 서버와의 호환성을 위해 자체적으로 메일 주소 다운그레이드 구조를 포함하는 것이 더 낫다고 볼 수 있다.

이전까지는 서버의 구현 방법을 이야기하였다면, 메일 클라이언트 부분에서도 수정이 필요하다는 것을 알 수 있다. 그 이유는, 물론 텔넷 방식으로 서버에 바로 접속하여 전자 메일을 사용할 수는 있으나 대부분 웹 메일이나 메일 클라이언트 등 따로 접속하여 사용하는 방식을 많이 사용하고 있기 때문에, 현재 메일 클라이언트들도 서버에 맞춰 UTF-8 메일 주소를 인식할 수 있도록 새로 개발되거나 수정할 필요가 있다는 것이다. 메일을 송신할 때 클라이언트 측에서 메일 주소를 인코딩한 다음 메일 송신 서버로 송신하고, 수신할 때는 메일 서버에서 읽어온 메일의 메일 주소를 디코딩하여 출력할 수 있도록 해야 한다.

EAI POP3 draft 문서에서는 POP3를 TUI (Text User Interface), 즉, 서버에 직접 접속하여 LANG 명령어로 사용 언어를 바꾸며, UTF8 명령어를 사용해 서버에 다국어로 로그인할 수 있도록 해야 한다고 명시되어 있다. 그러나 실질적으로 POP3를 사용하면서 직접 서버를 사용하는 일은 드문 일이며 매우 불편한 일이다. 그러므로 따로 클라이언트를 GUI 기반으로 쉽게 접근할 수 있도록 만드는 것이 좋은 방법이라 할 수 있다. 이러한 과정에서 메일 클라이언트 측에서도 다국어 메일 주소를 먼저 체크하여 잘못된 입력을 방지할 수 있다.

4.2 UTF8SMTP (UTF8 Simple Mail Transfer Protocol) 모델링

EAI 지원을 위한 메일 서버는 새로운 프로토콜로 새로운 메일 시스템 구조로 만드는 것이 아니라 기존의 메일 시스템의 구조에서 크게 변하지 않는 수준으로 모델링해야 한다. 새로운 메일 시스템 구조로 만드는 것은 전 세계에서 가동되는 메일 서버를 동시에 새로운 것으로 바꿀 수 있는 것이 아니기 때문에 단계적으로 조금씩 바꾸어서 통일을 해야 하므로 기존 메일 시스템과 호환성을 유지하는 구조로 해야 한다.

그림 3에서 보이는 기존의 전자 메일 시스템의 메일 전송 과정을 살펴보면 일반적으로 많이 쓰이는 메일 클라이언트 또는 웹 메일 시스템에서 제공하는 인터페이스에 따라 내용을 작성한 뒤 User Agent를 통해 SMTP 프로토콜로 메일 서버로 보낸다. 받은 메일 서버에서는 수신자의 메일 서버로 보내고 수신한 서버는 수신자의 메일박스로 메일을 기록하게 되고 수신자는 웹 메일 시스템이나 POP3 클라이언트를 통해 메일을 볼 수 있게 되어 있다.

이러한 메일 시스템에 EAI를 지원하게 하기 위해서는 일단 메일 클라이언트 프로그램에서 다국어로 된 메일 주소로 메일을 보낼 수 있도록 해야 하고 User Agent를 통해 메일 서버로 보내지면서 UTF-8로 인코딩하여야 한다. 그 때는 메일이 내부 메일인

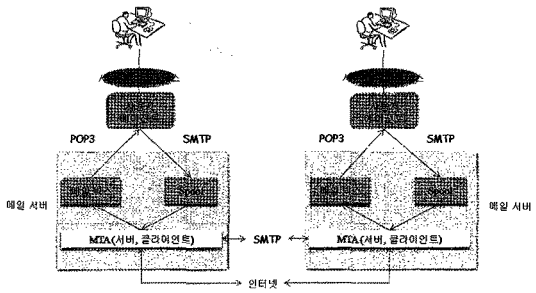


그림 3. 기존 전자 메일 시스템의 구조

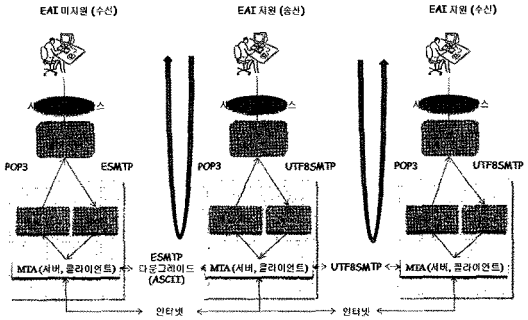


그림 4. 기존 메일 시스템을 수정한 EAI 지원 메일 시스템 구조

지 외부 메일인지를 판단한 후 내부 메일이면 바로 해당 사용자의 메일박스에 기록하고 외부로 보내질 메일이라면 목적지 메일 서버가 EAI를 지원하는지 하지 않는지를 판단한 후 지원하는 메일 서버라면 UTF8SMTP로 보내고 아니라면 ASCII 방식으로 다운그레이드를 한 후 보내게 된다.

EAI를 지원하는 수신 메일 서버는 해당 메일을 해당 수신자의 메일박스에 기록하게 되고 메일 클라이언트에서는 UTF-8로 인코딩한 메일을 디코딩하여 메일을 확인할 수 있도록 한다.

위의 그림 4에서 가운데 부분과 오른쪽 부분이 EAI를 지원하는 메일 서버이고 왼쪽 부분이 EAI를 지원하지 않는 메일 서버이다. 위에서 설명한 바와 같이 가운데 EAI를 지원하는 서버에서 작성된 메일은 UTF-8로 인코딩하여 수신 서버의 EAI 지원 여부에 따라 UTF8SMTP나 ASCII로 다운그레이드한 후 ESMTP로 보내게 된다.

크게 수정된 부분은 그림 5와 같이 진하게 표시된

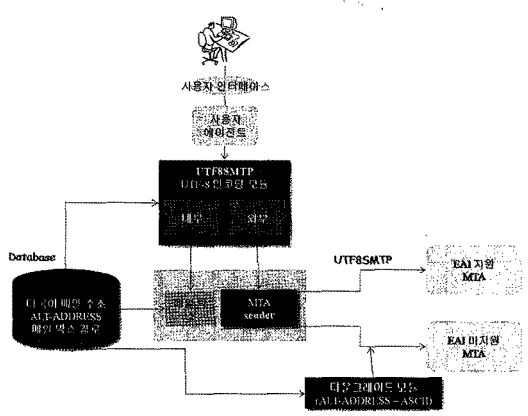


그림 5. UTF8SMTP 지원 메일 서버 구조

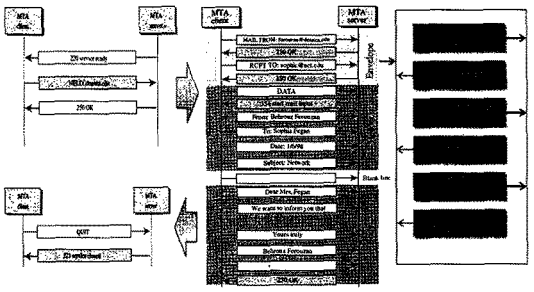


그림 6. 기존 메일 시스템에서 EAI 시스템으로 수정한 후 클라이언트와 서버 간 메시지 송수신 방법의 변화

부분으로 메일 클라이언트에서 메일 서버로 다국어 주소로 된 메일을 보낼 때 UTF8SMTP의 UTF-8 인코딩 모듈에 의해 인코딩하고 EAI 미지원 메일 서버로 보낼 땐 다운그레이드 모듈에 의해 데이터베이스의 ALT-ADDRESS로 다운그레이드해서 보낸다. 기본적으로 데이터베이스에는 다국어로 된 메일 주소와 메일박스 경로, ALT-ADDRESS가 있다.

기존의 메일 시스템은 대부분이 ESMTP를 지원하고 있기에 HELO로 확인한 후 MAIL FROM과 RCPT TO를 보낸다. 그러나 EAI 지원 메일 시스템은 그림 6의 가장 오른쪽 부분처럼 바뀌게 된다. 시작할 때 EHLO를 통하여 상대방 서버가 UTF8SMTP를 지원하는지 ESMTP만을 지원하는지를 확인한 후 그에 따라 다국어로 된 메일 주소나 ASCII로 된 주소를 선택해서 보내게 된다.

4.3 EAI 지원 POP3 (Post Office Protocol version 3) 및 메일 클라이언트 모델링

UTF8SMTP로 전송된 메일을 사용자가 보기 위해서는 직접 서버에 접속을 하거나 POP3나 IMAP 프로토콜 서버를 따로 구성해서 메일 클라이언트 프로그램으로 받아서 볼 수 있도록 해야 한다. 그러나 메일 주소가 UTF-8로 된 것은 기존의 POP3나 IMAP 서버에서 수용하는 것이 불가능하므로, 앞의 장에서 설명한 draft를 통해 만들어야 한다.

이 부분에서 핵심적인 부분은 메일박스에 들어 있는 메일들을 UTF-8로 인코딩한 상태이기 때문에 POP3를 이용해서 받을 때 디코딩을 해서 화면에 보여주어야 하므로 기존의 POP3 측에 디코딩 모듈이 추가되어야 한다.

4.1에서 모델링된 서버에서는 메일이 UTF-8형식

의 파일로 저장이 되어 있기 때문에 POP3의 "retr" 명령으로 메일을 읽을 때 파일을 읽어서 화면에 뿌려 준다. 그러나 메일 클라이언트 같은 경우에는 원격지에서 접속을 하므로 서버에서 UTF-8로 인코딩 된 메일을 읽어서 보내주기 때문에 디코딩 과정을 거쳐야 한다. 그렇지 않은 경우에는 메일 클라이언트에서 다국어로 된 메일 주소를 표현할 수 없게 된다. 그래서 메일 클라이언트로 접속해서 사용하기 위해서는 메일박스에 있는 메일을 가져와서 Header의 각 부분별로 파싱을 해서 각각 디코딩을 하도록 한다.

메일 클라이언트에서는 그림 7과 같이 디코딩 모듈에 의해 디코딩한 다국어 메일 주소를 표현할 수 있도록 하였다. 이와 같은 과정은 앞에서 서술한 바와 같이 꼭 필요한 과정이다.

또한, 기존에 사용되던 메일 클라이언트들 중 많이 쓰이는 MS 오피스의 아웃룩2007은 메일을 수신할 때 제대로 한글로 된 메일 주소를 표시하지 못했으며, MS 윈도 라이브 메일 2008은 한글로 된 메일 주소를 제대로 표시는 하나 한글로 된 주소로 메일을 보내지 못했고, 모질라의 썬더버드 2.0은 한글로 된 메일 주소를 제대로 표시를 하고 또한 한글 주소로 된 메일을 보냈을 때 메일이 보내어 지지만 해당 메일을 수신할 때 한글로 된 메일 주소가 제대로 표시되지 않는 문제가 발생했다.

이러한 문제가 생기는 이유는 2007에 발표된 아웃룩2007이 메일을 수신할 때 다국어에 대한 디코딩이 전혀 이루어지지 않고 있기 때문이다. 또한, 윈도 라이브 메일2008과 썬더버드 2.0은 2008년에 출시된 버전인데 메일을 수신할 때는 UTF-8 디코딩을 지원하여 다국어 메일 주소를 제대로 표시가 되는 반면에 아직 메일을 송신하지는 않는다. 즉, 최신 버전의 메일 클라이언트에서는 다국어 메일 주소가 포함된 메일을 수신할 수 있도록 개발되고 있다. 메일 주소의 다국어화 프로젝트가 몇 년 전부터 인터넷 드래프트 문서로 계속 업데이트됨으로 인해 각 개발사들이 조금씩 반영하여 개발하고 있는 상태지만 메일을 송수신할 때 모두 다국어 메일 주소를 제대로 표현할 수 없기 때문에 메일 클라이언트도 바꿀 필요가 있다.

그래서 본 논문에서는 EAI 지원 POP3 클라이언트 모델링과 함께 다국어 메일 주소로 메일을 송신할 때도 메일이 이상없이 보내질 수 있도록 클라이언트 측에도 UTF-8 인코딩 모듈을 추가하였다.

그림 8은 4.2의 EAI 지원 서버 모델에서 User Agent에서 메일 서버로 메시지를 보낼 때 UTF-8 인코딩을 한다는 것을 보여 준다. 즉, 서버 측에서 메일을 받으면서 인코딩을 한다는 것인데 그와 같은 경우는 메일 서버에 직접 접속해서 메일을 사용하는 경우이거나 해당 메일 서버에 웹 메일 서버까지 포함이 되었을 때 서버에서 인코딩하는 것으로 처리가 가능하다는 것이다. 그래서 메일을 원격지에서 클라이언트 프로그램을 통해 보내게 될 경우 클라이언트 측에서 보낼 때는 자체적으로 인코딩을 해서 보내야 서버 측에서 다국어 메일 주소를 인식하게 된다.

5. EAI 지원 메일 시스템 구현

5.1 구현 환경

본 논문에서 서버와 클라이언트 2대의 컴퓨터를

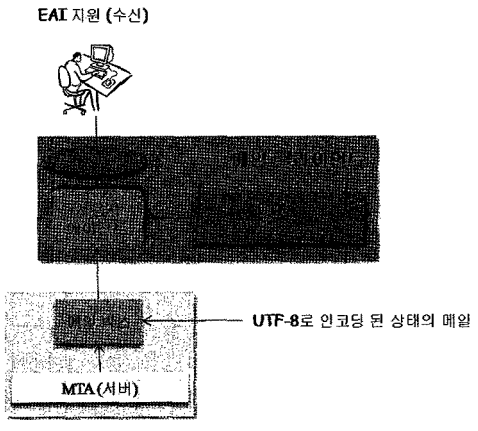


그림 7. POP3로 메일 수신시 디코딩 모듈로 디코딩 한 후 메일 확인

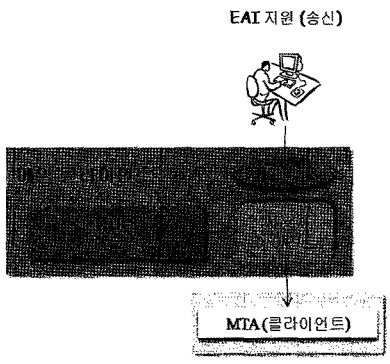


그림 8. 메일 클라이언트에 UTF-8로 인코딩하는 모듈을 포함

표 2. EAI 지원 메일 서버 및 메일 클라이언트 구현 환경

구 분	Server	Client
CPU	Pentium4 2.4GHz	PentiumD 3.0GHz
RAM	512MB	2.5GB
HDD	80GB	200GB
OS	CentOS 5.1	Fedora Core 9
Development	Language : Python 2.4.2	Language : Python 2.5.1
	DBMS : sqlite 2.8.17	Tool : Eclipse

사용하여 구현 환경을 구축하였다.

서버와 클라이언트 모두 Redhat Linux 계열의 Linux이며 커널은 2.6이상이며 개발 언어로는 Python이 사용되었다. Python은 문자열 관련 파싱 기능이 강력하며 문자 인코딩과 관련된 라이브러리들을 많이 제공하고 있다.

5.2 구현 내용

일단 서버 부분과 클라이언트 부분으로 나뉘게 된다.

서버는 ESMTP에서 UTF-8 인코딩 모듈을 추가하여 다국어 메일 주소를 사용할 수 있도록 UTF8SMTP로 확장하고 수신 서버로 보내는 MTA에서는 수신 서버가 EAI를 지원하는지에 대한 판단 기능이 추가되며, 그 판단에 따라 EAI를 지원하지 않는다면 다운그레이드할 수 있는 다운그레이드 모듈이 추가 되었다. 데이터베이스에는 다국어로 된 메일 주소, 다운그레이드 할 ASCII형태의 메일 주소, 메일박스 path가 들어있다.

메일 클라이언트에서 넘어오는 메일을 서버에서 받으면서 인코딩을 하게 되고 내부로 전송되는 메일일 경우 메일박스에서 바로 읽어올 수 있으므로 읽어올 때는 디코딩을 해서 넘겨주어야 한다.

그림 9와 같은 코드로 다국어로 된 메일 주소들이 전송되어 올 때 UTF-8로 인코딩하게 된다. 또한 메일을 전송할 때 기본적인 컨텐츠 인코딩 방법으로 base64MIME 형식으로도 인코딩하게 된다.

그림 10은 인코딩되어서 들어오는 문자열들을 메일을 읽을 때 디코딩하는 부분이다. 모든 메일들을 UTF-8로 인코딩하여 저장하기 때문에 읽기 위해선 꼭 디코딩을 해야 한다.

또한, EAI를 지원하지 않는 외부 서버로 나가는 메일일 경우 아래의 그림 11의 코드를 거쳐 ALT-ADDRESS로 변환되어 나가게 된다. 변환된 후에 re.match라는 메소드로 ASCII형태인지 확인을 하게 된다.

메일 클라이언트는 UTF8SMTP를 통해 메일을 보내고 POP3를 통해 메일을 받아볼 수 있으므로 그

```
##############################################################################
def encodeCHARSET(str, incharset, outcharset):
    incharset = convertCHARSET(incharset)
    outcharset = convertCHARSET(outcharset)
    try:
        if incharset != outcharset: str = unicode(str, incharset, 'ignore').encode(outcharset, 'ignore')
        retStr = '=?%s?B?%s?=' % (outcharset, base64MIME.encode(str , True, 76, ''))
    except:
        printError("common  | Can't encodeCHARSET(%s, %s, %s)" % (incharset, outcharset, str), 'common')
        return str
    else: return retStr
```

그림 9.문자를 UTF-8로 인코딩하는 코드

```
##############################################################################
def decodeCHARSET(str, incharset, outcharset, szPROCESS, key19):
    incharset = convertCHARSET(incharset)
    outcharset = convertCHARSET(outcharset)
    retStr = ''
    try:
        tkns = re.split('(=\?.*\?.*\?.*\?=)', str)
        for strtkn in tkns:
            if strtkn == '': continue
            tkn = header.decode_header(strtkn)[0]
            if tkn[1]: incharset = tkn[1]
            incharset = convertCHARSET(incharset)
            if incharset == outcharset: retStr += tkn[0]
            else: retStr += unicode(tkn[0], incharset, 'ignore').encode(outcharset, 'ignore')
    except:
        printError("%-7s | Can't decodeCHARSET(%s, %s, %s)" % (szPROCESS, incharset, outcharset, key19), szPROCESS)
        try:
            if incharset == outcharset: retStr = str
            else: retStr = unicode(str, incharset, 'ignore').encode(outcharset, 'ignore')
        except:
            printError("%-7s | Can't decodeCHARSET.all(%s, %s, %s)" % (szPROCESS, incharset, outcharset, key19), szPROCESS)
            retStr = str
    return retStr
```

그림 10. 인코딩 된 문자를 읽어오기 위해 디코딩해주는 코드

```
#-------------------------------------------------------------
if ((lbracket == 0) & (rbracket == 0)):
    RCPT_TO = lineStr[8:].strip().strip('<>').lower()
    RCPT_TO_ALT = ''
    print "No angle brackets"
#-------------------------------------------------------------
elif ((lbracket == 1) & (rbracket == 1)):
    RCPT_TO_STRIP = lineStr[8:].strip().strip('<').upper()

    # The lowest index in the string where "ALT-ADDRESS=" is found
    ALTADDRESS = RCPT_TO_STRIP.decode('utf-8').find("ALT-ADDRESS=")

    #-------------------------------------------------------------
    # If there is ALT-ADDRESS
    #-------------------------------------------------------------
    if ( ALTADDRESS != -1 ):
        RCPT_TO = str(RCPT_TO_STRIP.partition('>')[0].strip().lower())

        RCPT_TO_ALT = str(RCPT_TO_STRIP.decode('utf-8')[ALTADDRESS:].strip().lower())
        RCPT_TO_ALT = str(RCPT_TO_ALT.decode('utf-8').partition('=')[2].strip().lower())

        print "RCPT TO: " + RCPT_TO
        print "RCPT TO ALT: " + RCPT_TO_ALT

        # EAI SMTP Extension 2.4
        # ALT-ADDRESS value MUST be an all-ASCII email address
        if re.match("^[a-zA-Z0-9._%-]+@[a-zA-Z0-9._%-]+\.[a-zA-Z]{2,6}$", RCPT_TO_ALT) == None:
            print "ALT-ADDRESS Syntax Error"
            conn.sendall('550 5.5.4 %s... Syntax Error\r\n' % (lineStr))
            common.printLOG('%s | 550 %s... Syntax Error' % (key19, lineStr), 'esmtpd')
    else:
        RCPT_TO = str(RCPT_TO_STRIP.partition('>')[0].strip())
        RCPT_TO_ALT = ''
```

그림 11. EAI를 지원하지 않는 서버로 메일을 보낼 때 메일 주소를 ALT-ADDRESS로 변환해주는 코드

```
Received: from 164.125.36.31
        by utf8smtp.cslab.kr with NIDAMail programmed by eai-devel@googlegroups.com
        for <dyyeom@mail.cslab.kr>; Fri, 14 Nov 2008 12:25:45 +0900
Received: from 127.0.0.1
        by utf8smtp.cslab.kr with NIDAMail programmed by eai-devel@googlegroups.com
        for <dyyeom@mail.cslab.kr>; Fri, 14 Nov 2008 12:25:43 +0900
Date: Fri, 14 Nov 2008 12:25:43 +0900
Message-ID: <20081114122543248ec@utf8smtp.cslab.kr>
From: "=?utf-8?B?7Je864yA7JiB?=" <신단수@테스트.kr <dyyeom@mail.cslab.kr>>
To: 신단수@테스트.kr
Subject: =?utf-8?B?7J286r0x67KI7Ke4?=
X-Priority: 3
MIME-Version: 1.0
Content-Type: text/html; charset=utf-8
Content-Transfer-Encoding: base64

PGltZyBzcmM9J2h0dHA6Ly9tYWlsLnhuLS05dDRINTF5aTYhLatyL3WwZGF0ZV9yZWFkb3Jub3Qu
aHRtbD9pZD1keXllb2Oma2Y5MTk3MjAwODExNTQxMjI1NDI4OWM3MTAwMDA4MDEnIHdpZHRoPTAg
aGVpZ2h0PTA+PEhUTUw+PEhFQUQ+DQo8LOhFQUQ+DQo8Qk9EWSBzdHlsZT1cIQtUQlaRTog
OXBDOy8GTD5ULUZBTUJM7TogZ3Vsa7lcij7snbzqs7Hrsojsp7gg7Y7M7lqk7Yq47J6F64ul64uk
LjwvQk9EWT48LOhUTUw+
```

그림 12. POP3 서버에서 보내주는 메일 정보

```
# 메일을 POP3로 읽어올 때 Header field의 각 부분들을 잘라서 디코딩 한 후 저장을 한다.
if '@' in msg['from']:
    if '"' in msg['from']:
        fromHeader = '"' + decodeHeader(re.findall(r'"(.*)"', msg['from'])[0]) + '"' + ' ' + decodeHeader(re.findall(r'<.*>', msg['from'])[0])
    else:
        fromHeader = decodeHeader(msg['from'])
else:
    fromHeader = 'I don't Know'

app.lbox1.insert(END, msg['Subject'])

selMail.append(decodeHeader(msg['Subject']))
selMail.append(fromHeader)
selMail.append(decodeHeader(msg['To']))
selMail.append(decodeHeader(msg['Date']))

for part in msg.walk():
    selMail.append(decodeHeader(part.get_payload(decode = 1)))
```

그림 13. POP3를 통해 받은 메일의 각 Header field를 디코딩해서 저장하는 코드

에 따라 코드를 작성해야 한다. 일단 POP3서버에서 메일을 바로 읽을 때는 그림 12와 같은 정보들이 넘어오게 된다.

From의 보내는 사람인 메일 주소는 제대로 표시가 되며 받는 사람 또한 제대로 표시가 된다. 그러나 위의 기본 세션의 모드가 UTF-8일 경우이므로 제대로 표시가 되는 것이기에 세션모드가 UTF-8이 아니거나 다른 메일 클라이언트에서 UTF-8로 인코딩한 문자를 표현할 수 없다면 UTF-8로 인코딩한 문자를 제대로 표시할 수 있도록 수정을 한다.

그래서 본 논문에서는 Python을 이용해 간단히 메일을 읽을 수 있는 클라이언트를 따로 만들었다. POP3 서버에서 넘겨주는 정보가 그림 12와 같으므로 메일 클라이언트가 UTF-8을 표현할 수 있도록 디코딩해서 화면에 보여줄 수 있도록 했다.

또한 메일 클라이언트에서 메일을 보낼 때에도 메

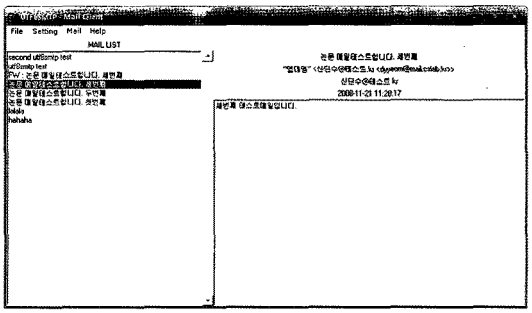
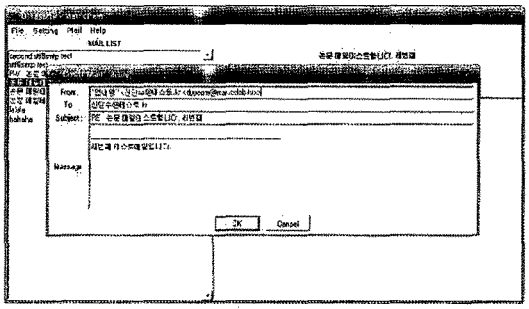


그림 14. 메일 주소가 UTF-8로 인코딩되어서 온 메일을 메일 클라이언트에서 디코딩 후 보여주는 화면



```
# 메일을 작성하여 보낼 때 각 부분을 UTF-8로 인코딩해서 보낸다.
self.From_ = self.e1.get(1.0, END).encode('utf-8').strip()
self.To = self.e2.get(1.0, END).encode('utf-8').strip()
self.Subject = self.e3.get(1.0, END).encode('utf-8').strip()
self.message = self.text.get(1.0, END).encode('utf-8').strip()

host = 'mlf8smtp.pusan.ac.kr'
msg = MIMEText(self.message, _charset = 'utf-8')
msg['Subject'] = self.Subject
msg['From'] = self.From_
msg['To'] = self.To
msg['Date'] = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(time.time()))
```

그림 15. 메일 클라이언트에서 메일 작성 시 입력받는 Header field의 값들을 UTF-8로 인코딩하는 코드

일의 Header field들에 대한 값을 UTF-8로 인코딩해서 입력을 받아서 보내야 한다.

그림 15와 같은 코드들로 메일을 보내고 받을 때 UTF-8로 인코딩과 디코딩 작업들이 필요로 한다.

6. 결 론

본 논문은 전자 메일을 보내는 데 있어서 지금까지 영문으로 된 주소만을 사용할 수 있었으나 다국어 즉, 자국어로 된 메일 주소를 사용할 수 있도록 전자 메일 시스템을 개선하는 방법을 제시하였다. 그래서 영문자 사용으로 인해 웹 메일에 로그인조차 힘들었던 초보자들도 자국어로 메일 주소를 사용할 수 있게 메일 시스템을 구축해 보았다.

그 방법으로 메일을 보내는 측에서 메일 주소를 다국어로 작성을 한 뒤 UTF-8로 인코딩해서 전송을 하고, 메일을 받는 측에서는 UTF-8로 인코딩한 메일을 읽기 위해서 해당 메일 주소를 디코딩한 후 사용자에게 보여줄 수 있도록 했다. 또한 현재 서비스 되고 있는 메일 시스템과의 호환성을 위해 UTF8SMTP를 지원하지 않는 서버로 메일을 보낼 때는 메일 주소를 기존의 메일 주소 형태로만 이루어진 ALT-ADDRESS를 두어서 사용할 수 있도록 하였다.

그러나 IETF에서 EAI의 표준화 과정이 진행되고 있지만, 정확한 구현 모델이 제시되어 있지 않고, 메일 주소의 다운그레이드, 몇 가지 명령어 추가, 키워드 추가만을 언급하고 있기 때문에, 메일 시스템 전체를 새로 디자인해야 했다. 또한 POP3 부분에서는 서버에 직접적인 접속만을 생각하는 부분만 기록되어 있어 클라이언트는 새로이 모델링해야 했었다.

앞으로 남은 것은 아직 표준화되지 않은 POP3, IMAP 등의 문서들이 빠른 시일 내에 표준으로 제정되어 해당 부분까지 표준에 맞춰 개발을 하도록 하고, 또한 다른 전자 메일 서비스 업체나 개발사들이 EAI 표준에 맞는 많은 메일 서버와 클라이언트가 만들어 세계 곳곳에서 자국어로 된 메일 주소를 사용하여 편하게 주고받을 수 있도록 해야 할 것이다.

참 고 문 헌

[1] Kim Kyongsok, "Standardization Report on E-mail Address Internationalization," 한국인터넷진흥원, 서울, 2007.

[2] H. Alvestrand and C. Karp, "An IDNA problem in right in right-to-left scripts," IETF, USA, 2006.

[3] F. Yergeau, "UTF-8, A transformation format of ISO 10646," IETF, USA, 2008.

[4] N. Freed and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies," IETF, USA, 1996.

[5] J. Klensin, N. Freed, M. Rose, E. Stefferud and D. Crocker, "SMTP Service Extensions," IETF, USA, 1995.

[6] J. Myers and M. Rose, "Post Office Protocol - Version 3," IETF, USA, 1996.

[7] M. Crispin, "INTERNET MESSAGE ACCESS

PROTOCOL - VERSION 4rev1," IETF, USA, 1996.

[8] Yao. J and W. Mao, "SMTP Extension for Internationalized E-mail Addresses," IETF, USA, 2008.

[9] Abel. Y, "Internationalized E-mail Headers," IETF, USA, 2008.

[10] Yoneya. Y and K. Fujiwara, "Downgrading mechanism for E-mail Address Internationalization," IETF, USA, 2008.

[11] Newman. C, "POP3 Support for UTF-8, IETF," USA, 2008.

염 대 영

2007년 동서대학교 컴퓨터공학과 공학사

2009년 부산대학교 컴퓨터공학과 공학석사

2009년 3월~현재 부산대학교 컴퓨터공학과 박사 과정

관심분야 : 플래시 메모리, 파일시스템, 데이터베이스, 인터넷 컴퓨팅

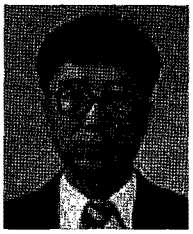
한 동 윤

2004년 부산대학교 정보컴퓨터공학부 학사

2006년 부산대학교 대학원 컴퓨터공학과 공학석사

2006년 3월~현재 부산대학교 대학원 컴퓨터공학과 박사과정

관심분야 : 플래시 메모리, 이메일 국제 표준



김 경 석

1977년 서울대학교 무역학과 경제학사

1979년 서울대학교 전자계산학과 이학석사

1988년 일리노이 주립대(어바나-샴페인) 전자계산학 박사

1988년~1992년 미국 노스다코타 주립대학교 전자계학학과 조교수

1992년~현재 부산대학교 전자전기정보컴퓨터공학부 교수

관심분야 : 데이터베이스, 멀티미디어, 한글/한말 정보처리, 인터넷 컴퓨팅 등