무선 인터넷 서비스를 위한 WAP 게이트웨이용 WML 컴파일러의 설계 및 구현

(A Design and Implementation of WML Compiler for WAP Gateway for Wireless Internet Services)

최 은 정 † 한 동 원 †† 임 경 식 †††

(Eunjeong Choi) (Dong-Won Han) (Kyungshik Lim)

요 약 무선 마크업 언어(Wireless Markup Language) 컴파일러는 텍스트로 구성된 문서를 바이너리 문서로 변환, 압축함으로써, 낮은 대역폭을 갖는 무선 선로에서 트래픽을 감소시키며, 낮은 성능을 갖고 있는 이동 단말기에서 브라우징 처리를 간단하게 하는 역할을 한다. 또한 이러한 변환 과정에서 확장 마크업 언어(eXtensible Markup Language)의 well-formedness와 validation 과정을 동시에 처리함으로써, 이동 단말기에서 문서처리 부담을 대폭 경감하는 효과를 가져온다. 본 논문에서 구현한 무선 마크업 언어 컴파일러는 어휘분석기 모듈과 파서 모듈로 구성되어 있는데, 파서 모듈은 파서 생성기를 사용하여 구현하였다. 이는 향후 응용 수준에서 보안 기능을 제공하기 위하여 태그를 확장하거나 무선 마크업 언어의 버전이 업그레이드 될 때에도 변경된 부분에 해당하는 문법만 다시 설계함으로써 유연하게 대처할 수 있는 장점을 가지고 있다. 사용된 문법은 LALR(1) context-free 문법으로서, 확장 마크업 언어 1.0과 무선 마크업 언어 1.2의 문서 형태 정의(Document Type Definition)를 기반으로 무선 응용 프로토콜 바이너리 확장 마크업 언어(Wireless Application Protocol Binary XML) 문법을 고려하여 설계되었다. 구현된 컴파일러의 기능을 실험하고 데모하기 위하여 세 가지 방법(수작업, WML 디컴파일러, 노키아 WAP 툴킷)을 사용하였으며, 다양한 태그 조합을 갖는 임의의 130여 개 문서에 대해 실험한 결과, 최대 85%의 압축효과를 얻을 수 있었다. 그러나, 태그나 속성에 비해 일반 문자열 데이타가 많아지면 상대적으로 압축효과가 감소되므로, Hyper Text Markup Language 문서로부터 무선 마크업 언어 문서로 자동 변환된 텍스트를 인코딩하는 경우와 같이 특정한 응용 분야에서는 일반 문자열에 대한 확장 인코딩 기법을 적용할 필요가 있을 수 있다.

Abstract In this paper, we describe a design and implementation of the Wireless Markup Language(WML) compiler to deploy wireless Internet services effectively. The WML compiler translates textual WML decks into binary ones in order to reduce the traffic on wireless links that have relatively low bandwidth to wireline links and mitigate the processing overhead of WML decks on wireless terminals that have relatively low processing power to fixed workstations. In addition, it takes over the overhead of eXtensible Markup Language(XML) well-formedness and validation processes. The WML compiler consists of the lexical analyzer and parser modules. The grammar for the WML parser module is LALR(1) context-free grammar that is designed based on XML 1.0 and WML 1.2 DTD(Document Type Definition) with the consideration of the Wireless Application Protocol Binary XML grammar. The grammar description is converted into a C program to parse that grammar by using parser generator. Even though the tags in WML will be extended or WML DTD will be upgraded, this approach has the advantage of flexibility because the program is generated by modifying just the changed parts. We have verified the functionality of the WML compiler by using a WML decompiler in the public domain and by using the Nokia WAP Toolkit as a WAP client. To measurethe compressibility gain of the WML compiler, we have tested a large number of textual WML decks and obtained a maximum 85 %. As the effect of compression is reduced when the portion of general textual strings increases relative to one of the tags and attributes in a WML deck, an extended encoding method might be needed for specific applications such as compiling of the WML decks to which the Hyper Text Markup Language document is translated dynamically.

† 비 회 원 : 한국전자통신연구원 연구원
ejchoi@etri.re.kr
†† 비 회 원 : 한국전자통신연구원 휴대클라이언트연구팀 연구원
dwhan@etri.re.kr
††† 정 회 원 : 경북대학교 컴퓨터과학과 교수
kslim@knu.ac.kr
논문접수 : 2000년 5월 12일
심사완료 : 2001년 2월 21일

1. 서 론

무선 인터넷은 기존 인터넷 환경의 공간적 제약을 극복하여 이동 단말을 사용해 언제 어디서나 인터넷으로 접속하여 다양한 정보검색과 전자상거래를 가능케 한다. 이러한 무선 인터넷 서비스를 지원하기 위해 무선 환경에서의 운용에 적합한 무선 응용 프로토콜(Wireless Application Protocol)[1]이 제안되었고, 이는 향후 무선 인터넷 표준 기술언어로 유력한 무선 마크업 언어(Wireless Markup Language)[2]를 기반으로 한다.

무선선로에서의 높은 에러 발생률과, 오랜 지연, 협소한 대역폭 등을 극복하고 무선 인터넷 서비스를 원활히 지원하기 위해서는 가능한 한 전송되는 데이타의 양을 감소시킬 필요가 있다. 이를 위하여 무선 응용 프로토콜 포럼에서는 이동 단말과 게이트웨이(gateway)간에 바이너리(binary) 무선 마크업 언어를 사용해 전송하도록 한다. 또한 이동 단말기의 낮은 CPU 처리속도, 메모리의 부족, 작은 화면 표시 창, 짧은 전지 수명 등의 제한된 자원으로 사용자가 만족할 만한 서비스를 하기 위해서는 저장할 데이타의 양을 최소화하여 메모리 소모를 줄이는 한편, 게이트웨이에서 가능한 한 많은 처리과정을 거쳐 이동 단말기에서의 처리량을 줄여야 한다.

본 논문에서는 텍스트로 구성된 무선 마크업 언어 문서를 바이너리 형식으로 변환, 압축시켜 무선 선로에서 전송되는 데이타 량을 줄이는 한편, 문서의 구조화 및 검증 처리를 담당하는 무선 마크업 언어 컴파일러의 구현에 대해 기술한다. 무선 마크업 언어 표준안에는 확장 마크업 언어(eXtensible Markup Language) well-formedness를 기반으로 텍스트 문서를 바이트 코드로 변환하는 기능만이 요구사항으로 명시되어 있다. 그러나, 본 논문에서는 이러한 기본적인 기능 외에 무선 마크업 언어 1.2 문서 형태 정의(Document Type Definition)[2]를 기반으로 문서의 내용 및 구조를 검사하는 기능을 구현하여, 오류 데이타 전송으로 인한 무선 선로에서의 대역폭 낭비를 사전에 방지하는 한편 이동 단말에서의 처리량을 감소하는 효과를 가져오도록 하였다. 이러한 기능의 구현은 무선 응용 프로토콜 포럼에서 제안한 무선 응용 프로토콜 바이너리 확장 마크업 언어(WAP Binary XML)[3]과 무선 마크업 언어 표준안, 그리고 월드 와이드 웹 컨소시엄(World Wide Web Consortium)의 확장 마크업 언어 표준안[4]을 기반으로 하였다.

본 논문 구성을 보면, 제 2장에서는 무선 응용 프로토콜의 개요[5] 및 무선 마크업 언어와 관련된 기존의 연구 사례를 조사하여 기술한다. 제 3장에서는 본 논문에서 구현한 컴파일러를 세부적으로 분석한다. 이 장에서는 텍스트 문서를 토큰화하여 바이너리로 표현한 방식과 토큰화를 위해 구현된 어휘분석기의 동작원리를 설명한다. 또한 문서의 검증을 위해 구현된 파서 모듈 부분에서는 확장 마크업 언어 1.0과 무선 마크업 언어 1.2 문서 형태 정의를 기반으로 작성된 LALR(1) 기반의 문법 설계를 상세히 기술한다. 다음으로 제 4장에서는 구현된 컴파일러가 표준안에서 정의한대로 정확하게 결과물을 생성해 내는지, 그리고 실제 테스트 배드를 구축하여 올바로 동작하는가를 실험한다. 실험은 세 가지 형태로 이루어진다. 첫 번째로는 구현한 컴파일러에 의해 생성된 바이너리 문서를 표준안에 따라 수작업으로 비교 검토한다. 두 번째는 *wml-tools*의 *wmld*[6]라는 무선 마크업 언어 디컴파일러(decompiler)를 이용해 컴파일러에 의해 생성된 바이너리 문서를 디컴파일해서 원래의 문서를 복원해 내는지를 본다. 세 번째는 시뮬레이션 환경을 구축하여 실제 상용 무선 응용 프로토콜 플랫폼에 본 논문에서 구현한 컴파일러를 통합하여 제대로 동작하는지를 실험한다. 마지막으로 제 5장에서 결론을 맺는다.

2. 무선 응용 프로토콜 개요 및 기존 사례 연구

2.1 무선 응용 프로토콜 모델

무선 응용 프로토콜은 무선 인터넷 서비스를 지원하기 위한 국제적 개방 통신 규약으로 인터넷망과 이동통신망의 접속 점에 게이트웨이를 두어 두 프로토콜 체계 사이의 중계역할을 하도록 한다.

<그림 1>에서 보여지듯이 무선 응용 프로토콜에서의 프로그래밍 모델[1]은 월드 와이드 웹 프로그래밍 모델과 유사하다. 이는 친숙한 프로그래밍 모델이며 증명된 모델로써 기존 기술을 이용할 수 있어서 응용 개발자들에게는 이점으로 작용한다. 또한 무선환경의 특성을 반영하기 위해 최적화와 확장성이 이루어지도록 하였으며, 가능한 한 기존의 표준을 수용하여 무선 응용 프로토콜 기술의 출발점으로써 사용되도록 하였다. 무선 응용 프로토콜의 컨텐트(content)와 응용은 월드 와이드 웹의 형식과 유사하다. 컨텐트는 월드 와이드 웹 통신규약[7] 기반의 표준 통신 규약을 사용하여 전송되며 이동 단말기에 위치한 마이크로 브라우저는 이러한 컨텐트를 브라우징 하는데 있어 표준 웹 브라우저와 유사하다. 이동 단말기에서 인터넷상의 서버와 통신이 이루질 수 있도록 하기 위한 표준 컴포넌트들이 정의되어 있다.

서버의 컨텐트를 확인하기 위해 월드 와이드 웹 표준

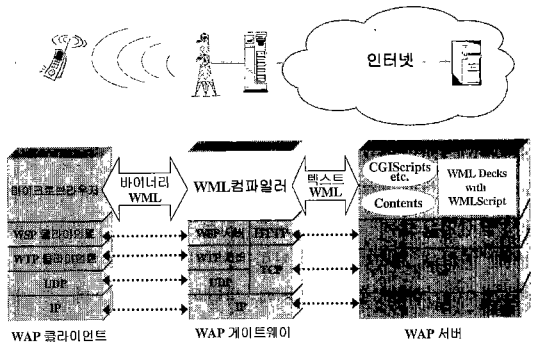


그림 1 무선 응용 프로토콜 스택

URL[8]을 사용하며 모든 컨텐트는 월드 와이드 웹 분류 방법과 일관된 특정 타입으로 주어져, 사용자 에이전트가 특정 타입을 기반으로 컨텐트를 정확하게 처리할 수 있도록 해준다. 또한 컨텐트 형식은 월드 와이드 웹 기술을 기반으로 하며 마크업, 캘린더 정보, 전자 사업 카드 객체, 그림, 스크립트 언어의 디스플레이를 포함한다.

무선 응용 프로토콜은 무선 도메인과 월드 와이드 웹 사이를 연결하기 위해 프락시 기술을 이용한다. 전형적으로 무선 응용 프로토콜 게이트웨이는 프로토콜 게이트웨이와 컨텐트 인코더(encoder)/디코더(decoder) 두 가지 기능으로 이루어져 있다. 프로토콜 게이트웨이는 무선 응용 프로토콜의 요청을 월드 와이드 웹 프로토콜에 적합하도록 변환하며 컨텐트 인코더는 무선 선로에서 전송되는 데이타 량을 줄이기 위해 컨텐트를 간소한 바이너리 형식으로 변환하는 기능을 가진다. 만일, 클라이언트의 마이크로브라우저에서 인터넷상의 서버에 위치한 무선 마크업 언어로 작성된 문서를 요청하면, 무선 마크업 언어 컨텐트 인코더는 텍스트로 작성된 덱을 바이너리 형식으로 변환시켜서 클라이언트로 보내게 된다. 이러한 모델은 이동 단말 사용자가 마이크로브라우저를 이용해 다양한 컨텐트를 브라우징 할 수 있게 하며, 응용 개발자에게는 월드 와이드 웹 기술을 사용하여 이동 단말에서 동작할 수 있는 응용 서비스를 개발할 수 있도록 해준다.

2.2 무선 마크업 언어 개요

무선 단말장치는 일반 데스크탑 PC나 노트북과는 달리 낮은 CPU 처리속도, 메모리의 부족, 작은 화면크기, 제한된 사용자 입력 기능, 짧은 전지 수명 등 자원이 제한되어 있으므로, 텍스트 위주의 필요한 정보만을 가지고 화면을 구성해야 한다. 이러한 특성을 고려하여 무선 응용 프로토콜 포럼에서는 기존의 HTML의 일부 기능에 무선 단말 장치를 고려한 기능을 추가하여 새로이 무선 마크업 언어를 정의하였다. 무선 마크업 언어는 확장 마크업 언어 기반 마크업 언어로서, 셀룰라 폰과 페이저 등을 포함한 무선 단말 장치에 적합한 내용과 사용자 인터페이스를 정의한다.

무선 마크업 언어는 다음과 같은 기능을 지원할 수 있는 요소들을 정의한다.

첫째, 텍스트와 이미지를 화면에 다양하게 보여줄 수 있는 방법을 제공한다. 둘째, 화면의 크기를 고려하여 정보를 여러 개의 단위로 나누게 되는데 이 단위를 카드(card)라 하며, 여러 개의 카드로 구성된 하나의 전송단위를 덱이라 한다. 따라서 카드는 선택 메뉴나 텍스트 화면과 같은 사용자와의 상호작용을 위한 단위로 네비게이션 단위가 되며, 덱은 URL로 구별되는 컨텐트의 전송 단위라는 점에서 HTML 페이지와 유사하다. 셋째, 덱 내에 있는 카드를 링크시켜 카드와 덱 사이의 이동을 명시적으로 관리할 수 있다. 또한 장치의 이벤트를 처리할 수 있으며 이는 네비게이션 목적으로 사용되기도 하고 스크립트를 실행하기도 한다. HTML에서와 같은 하이퍼텍스트 기능도 지원한다. 넷째, 하나의 변수를 여러 개의 카드에서 계속 사용할 수 있다. 예를 들면, 하나의 카드를 통해 받아들인 변수 값을 저장하고 있다가 다른 카드가 실행 될 때 이를 치환하여 사용하는 등 서버와의 통신을 하지 않고도 필요한 처리를 할 수 있으므로 네트워크 자원을 효율적으로 사용할 수 있게 한다.

무선선로에서는 높은 에러 발생률, 오랜 지연, 협소한 대역폭 등의 특성이 있다. 이를 극복하고 무선 인터넷 서비스를 원활히 지원하기 위해서는 가능한 한 전송되는 데이타의 양을 감소시킬 필요가 있다. 이를 위해 무선 채널 상에서는 무선 마크업 언어 문서의 전송크기를 감소시킨 바이너리 형식이 제안되었으며 이는 기능이나 의미정보의 손실 없이 압축되어야 한다. 바이너리 무선 마크업 언어는 게이트웨이와 이동 단말기 사이의 데이타 전송 형식으로써, 텍스트 기반 무선 마크업 문서의 압축된 바이너리 표현이며 무선 응용 프로토콜 포럼에서 제안한 WBXML를 기반으로 한다.

2.3 무선 마크업 언어 관련 기존 사례 연구

wml-tools v0.0.4의 wmlc(wml compiler)[6]는 텍스트 기반 무선 마크업 언어 문서를 바이너리 형식으로 변환하는 기능을 제공한다. 이 컴파일러는 C 언어로 구현되어 있으며 확장 마크업 언어 well-formedness 검증 기능을 제공한다. 텍스트 문서를 태그, 속성, 속성 값, 사용자 데이타 등으로 인식하여 파싱하는데, 이때 무선 마크업 언어 토큰 테이블에 정의된 토큰이 인식되

면 바이너리 형식으로 변환하는 방식을 취하고 있다. 또한 각각의 사용자 데이타를 비롯해 모든 토큰에 사용될 수 있는 문자집합은 US-ASCII 뿐이며, 문서 내에 포함된 변수를 처리하지 못한다. Kannel: Open Source WAP and SMS gateway[9]의 무선 마크업 언어 컴파일러도 역시 C언어로 구현되어 있으며 확장 마크업 언어 well-formedness 검증 기능을 제공한다. 차이점이 있다면 변수를 인식해 처리할 수 있다는 것이다. 이들 컴파일러들은 텍스트 문서가 확장 마크업 언어의 well-formedness 제약을 충족하는지를 검사할 뿐 무선 마크업 언어의 문서 형태 정의를 기반으로 형식 및 내용이 적절히 구조화되어 작성되었는지는 검사하지 않는다. 기존의 무선 마크업 언어 관련 사례에서는 확장 마크업 언어의 형식만을 검사하므로 무선 마크업 언어의 형식 및 내용 구성에 부적합한 문서도 해당 토큰에 대한 바이트 코드를 생성하여 무선 단말기로 전송한다. 마이크로브라우저는 무선 마크업 언어 표준안에 따라 브라우징 처리를 하며, 이때 문서 형태 정의에 따른 제약사항에 부적절하게 생성된 문서에 대해서는 오류를 발생시킨다. 문서 형태 정의에 부적합한 데이타의 전송은 무선 선로의 대역폭 낭비는 물론 이동 단말기에서의 불필요한 처리과정에 따른 오버헤드를 발생시킨다. 그러나, 게이트웨이에서 문서를 검증하여 오류가 있을 경우에 프로토콜 헤더에 오류 메시지를 지정하여 보내게 되면 무선 선로에서의 불필요한 데이타 전송을 방지할 수 있을 뿐만 아니라, 마이크로브라우저에서도 헤더 정보만을 사용해 문서의 상태를 판별할 수 있게 되므로 문서 검증과 같은 처리를 하지 않아도 될 것이다.

본 논문에서는 무선 구간에서의 전송 데이타 량을 감소하기 위해 텍스트 문서를 간소한 바이너리 형식으로 변환하는 한편, 자원의 제약이 많은 이동 단말에서의 처리부담을 줄이기 위해 문서에 대한 처리를 게이트웨이에서 가능한 한 많이 하도록 설계하였다. 이를 위해 기존 사례에서 제공하는 확장 마크업 언어 well-formedness 뿐만 아니라, 문서 형태 정의를 만족하는지를 검증해 주는 확장 마크업 언어 validation 검증 기능을 제공한다. 확장 마크업 언어 1.0 표준안에는 모든 확장 마크업 언어 문서들이 관련 문서 형태 정의를 가지고 있고, 그에 따른 제약조건에 따라 컴파일 된다면 유효한 문서로 인정한다고 되어있다[4]. 본 논문에서 구현된 무선 마크업 언어 컴파일러는 무선 마크업 언어 1.2 문서 형태 정의를 기반으로 문법을 설계하여 텍스트 문서의 내용 및 구성에 대한 적합성을 검사한다. 따라서 문서가 본 논문에서 제시한 문법에 따라 파싱 된다면 확장 마크업 언어 validation을 만족하게 되고 이는 확장 마크업 언어 well-formedness 역시 만족하게 됨을 의미한다. 다음으로 본 논문에서는 어휘분석기 생성기(Flex)와 파서 생성기(Bison)를 사용해 컴파일러를 설계하였다. 이러한 구현방법은 향후 확장 태그나 무선 마크업 언어 버전이 업그레이드 될 때 변경된 부분에 해당하는 문법만 변경함으로써 실행 코드를 자동 생성할 수 있기 때문에 변화에 유연하게 대처할 수 있는 이점이 있다. 또한 Bison이라는 툴을 사용해 개발되었기 때문에 본 논문에 설계된 문법은 LALR(1) context-free 문법에 맞게 작성된 것이라고 할 수 있다[10].

이 절에서는 텍스트 무선 마크업 언어 문서를 바이너리 형식으로 변환한 사례를 조사하였다. 노키아(Nokia)[11], 폰닷컴(Phone.com)[12]등의 상용 게이트웨이에서는 요청한 문서를 바이너리로 변환하여 보내오는 것을 볼 수 있어 이들 시스템에서도 무선 마크업 언어 컴파일러를 구현했음을 알 수 있으나, 기술이 공개되지 않아 자세한 구현 내용은 알 수 없다. 반면 공개된 사례로는 wml-tools v0.0.4의 wmlc(wml compiler)[6]와 Kannel: Open Source WAP and SMS gateway[9]의 무선 마크업 언어 컴파일러가 있어 이들 사례를 연구하였다. 초기에 이들 컴파일러들은 모두 변수를 지원하지 않았으나 최근에 다시 조사한 바에 따르면, Kannel: Open Source WAP and SMS gateway Version 0.12.2[9]에서는 변수를 지원하는 것으로 나타났다. 본 논문에서도 변수를 정확히 인식해 처리하고 있다.

3. 설계 및 구현

3.1 요구 조건

무선 마크업 언어 컴파일러가 논의된 배경은 제약이 많은 무선 구간에서의 전송 데이타 량을 줄이기 위한 것이다. 본 논문에서 구현된 컴파일러는 명명된 토큰을 바이트 코드로 변환하여 데이타 량을 감소하는 효과를 가져오는 한편, 문서의 내용 및 구조에 오류가 있는 컨텐트를 검사하여 무선 선로의 대역폭 낭비 및 무선 단말기에서의 불필요한 처리 과정에 따른 오버헤드를 방지한다. 본 논문에서는 확장 마크업 언어의 well-formedness와 validation 검증 및 무선 마크업 언어의 validation 검증을 위한 구현 기술을 소개한다. 확장 마크업 언어의 well-formedness와 validation 검증을 위한 기능은 완벽히 구현되었으며, 무선 마크업 언어 validation 기능은 모두 구현되지 않아 일부 항목은 이동 단말의 마이크로브라우저에서 처리되어야 한다.

모든 확장 마크업 언어 문서들은 일정한 형식을 갖추어야 하며[4], 확장 마크업 언어 well-formedness란 적절하게 구조화된 마크업 구문을 가리킨다[13]. 확장 마크업 언어 1.0 표준안에서는 모든 확장 마크업 언어 문서들은 관련 문서 형태 정의를 가지고 있고, 이에 따라 컴파일 된다면 유효한 문서로 인정한다[4]. 또한 문서 형태 정의에 유효한 문서는 확장 마크업 언어 well-formedness도 만족한다. 본 논문에서 구현한 컴파일러는 무선 마크업 언어 1.2 문서 형태 정의를 기반으로 문법을 설계하였으므로 문서가 본 논문에서 제시한 문법에 따라 컴파일 된다면 확장 마크업 언어 validation을 만족하게 되고 이는 확장 마크업 언어 well-formedness 역시 만족하게 됨을 의미한다.

3.2 텍스트 덱에서 바이너리 덱으로의 변환 기법

무선 마크업 언어 컴파일러는 문서의 전송크기를 줄여서, 상대적으로 낮은 대역폭을 갖는 무선선로에서의 트래픽을 감소시키며, 또한 상대적으로 낮은 성능을 갖고 있는 이동 단말기에서의 브라우징을 간단하게 하도록 설계되었다. 무선 마크업 언어 컴파일러는 기능이나 의미정보의 손실 없이 전송 데이타를 압축해야 하며 텍스트 기반 무선 마크업 문서를 간소한 바이너리로 표현하기 위해 WBXML 기반의 바이너리 형식을 사용한다.

이 절에서는 본 논문에서 구현된 컴파일러의 인코딩 및 압축 기법을 기술한다. 문서를 텍스트 형식에서 바이너리 형식으로 변환하는 과정을 컴파일러의 구조와는 상관없이 논리적인 측면에서 서술한다. 이를 위해 바이너리 형식에서 사용되는 데이타 타입에 대해 설명하고, 변환 전후의 문서 양식을 전체적인 구조와 관련해 비교하여 각각의 토큰들이 변환되는 방법에 대해 분석한다.

3.2.1 바이너리 무선 마크업 언어 데이타 타입

구현된 무선 마크업 언어 컴파일러는 텍스트 기반의 문서를 압축하여 표현하기 위해 새로운 데이타 타입[2,3]을 사용한다. 마크업의 대부분은 확장 토큰으로 변환되며 보통 1-바이트의 16진수로 표현되고, 그 외 전역 데이타, 문자열 등의 데이타를 압축하기 위해서는 다양한 데이타 타입이 필요하다. <표 1>에서는 압축된 바이너리 문서에서 사용되는 데이타 타입을 나열하였다.

표 1 바이너리 무선 마크업 언어 토큰 데이타의 종류

데이타 타입	설 명
bit	1-비트 데이타.
byte	8-비트의 알 수 없는 데이타.
u_int8	8-비트의 부호 없는 정수.
mb_u_int32	32-비트의 부호 없는 정수.

비트(bit) 데이타는 시작태그에서의 내용과 속성, 플래그(flag)와 같이 토큰에 의미를 부가하는데 사용된다. 문자열 데이타나 알 수 없는 데이타는 8-비트의 바이트(byte) 타입으로 표현하며 확장 토큰은 모두 8-비트의 부호 없는 정수 형태인 u_int8로 표현한다. mb_u_int32은 정수 값을 여러 개의 바이트로 표현한 것이다. 다중-바이트 정수는 최상위 비트가 연속 플래그이고 나머지 7-비트들은 스칼라 값으로 표현되는 여러 개의 옥텟(octets)으로 구성된다. 연속 플래그는 하나의 옥텟이 일련의 다중-바이트의 끝이 아님을 가리킨다. 하나의 정수 값은 일련의 N 개의 옥텟으로 인코딩 된다. 처음 N-1 옥텟은 연속 플래그를 1로 지정하고, 마지막 옥텟은 연속 플래그 값이 0 이 된다.

3.2.2 생성된 바이너리 덱의 물리적 구조

컴파일러에 의해 생성된 바이너리 무선 마크업 언어 문서는 크게 문서의 선언부와 원래의 텍스트 기반 덱의 내용으로 구성된다. 선언부는 WBXML 버전, 문서의 공개 식별자, 문자집합, 문자열 테이블로 구성되며, 다음으로 마크업과 사용자 데이타들로 구성된 덱의 내용이 나온다.

바이너리 무선 마크업 언어 문서는 WBXML 응용 문서이며 최초의 바이트에 WBXML 버전을 포함해야 한다. 버전 바이트는 메이저(major) 버전보다 1 작은 값을 상위 4-bit으로 표현하고, 마이너(minor) 버전을 하위 4-bit으로 표현한다. 예를 들어, 버전 2.7에 대한 선언은 0x17로 압축된다. 본 논문에서는 1.2 버전을 지원하므로, 0x02로 표현된다. 텍스트 무선 마크업 언어는 확장 마크업 언어의 응용이며 외부 문서 형태 정의를 가지고 있다. WBXML 형식에서는 확장 마크업 언어 문서의 공개 식별자를 포함하도록 하고 있다. 이 선언은 0 이상의 미리 정의된 다중-바이트 양수 값으로 표현된다. 만일 미리 정의된 식별자가 아니라면 문자열로 인코딩될 수 있다. 무선 마크업 언어 1.2에 대한 공개 식별자 "-//WAPFORUM//DTD WML 1.2//EN"는 0x09로 정의되어 있다. 문자집합은 다중-바이트 양의 정수 값으로 인코딩되며, 만일 문서의 인코딩 방식을 알 수 없을 경우에 문자 인코딩을 결정하기 위해 전송 메타-정보가 사용되어야 한다. 만일 전송 메타-정보를 사용할 수 없을 경우 디폴트 인코딩은 UTF-8(a transformation format of Unicode and ISO 10646)[14]이 된다. 문서의 타입 선언 후, 바로 문자열 테이블을 포함해야 한다. 최소한 문자열 테이블은 길이 필드를 제외한 문자열 테이블의 길이를 표현하는 mb_u_int32로 구성된다. (예를

들어, 2-바이트의 문자열을 가진 문자열 테이블은 길이 필드 값이 2로 인코딩 된다.) 만일 길이가 0이 아니면, 뒤에 하나 이상의 문자열이 나오게 된다. 다음으로 <wml>과 같은 요소로 구성된 실제 데이타를 포함하는 덱이 토큰화되어 바이너리로 표현된다. 요소 및 속성의 시작과 같은 마크업은 대부분 1-바이트 형식인 u_int8로 압축되며, 속성 값은 인라인(inline) 문자열로 인코딩 된다. 파싱 된 데이타(pcdata)는 인라인 문자열로 인코딩되며, 변수는 문자열 테이블에 저장되도록 하였다.

3.2.3 생성된 바이너리 덱의 토큰 구조

WBXML에서의 토큰들은 코드영역을 오버랩 하는 집합으로 나뉜다. 특정 토큰의 의미는 사용되는 문맥에 따라 다르다. 토큰들은 다음 방식으로 코드영역이 구성된다.

토큰은 크게 전역토큰과 응용토큰으로 분류된다. 전역토큰은 모든 문맥 내에서 고정되어 있으며 모든 상황에 대해 확실히 구별된다. 전역토큰은 문자열과 같은 인라인(inline) 데이타와 다양한 종류의 제어 기능을 인코딩하는데 사용된다. 응용토큰은 문맥 의존적 의미를 가지며 태그 코드 영역과 속성 코드 영역이 오버랩 되어 있다. 주어진 토큰 값은 태그인지 속성인지에 따라 다른 의미를 갖는다. 태그 코드 영역에서 각 태그 토큰은 하나의 바이트 코드로 압축되며 특정 태그명(eg, card)을 나타낸다. 속성 코드 영역은 속성 시작과 속성 값을 각각 나타내는 두 가지 범주로 나뉜다. 생성된 바이너리 덱에서, 제어토큰은 모든 WBXML 응용 문서에서 공통적으로 사용되는 전역토큰 값이 사용되며 태그 토큰 값과 속성 토큰 값은 바이너리 무선 마크업 언어에서만 유효한 토큰 값이다. 다음은 바이너리 무선 마크업 언어에서 사용되는 토큰의 종류와 각 토큰의 예를 설명한다.

(1) 전역토큰

전역토큰은 모든 코드 영역과 코드 페이지 내에서 같은 의미와 구조를 가지며 문자열과 확장 토큰 엔터티(entity) 등으로 분류된다. 문자열 인코딩은 인라인 데이타와 문자열 테이블의 참조 형태가 있다. 인라인 문자열은 문자열이 나타난 위치에 문자열 그대로를 저장하는 방법이며 제어토큰 STR_I으로 시작하여 NULL로 끝나는 문자열로 구성된다. 반면 문자열 테이블 참조 방법은 덱 내에서 여러 번 나타나는 문자열들을 테이블에 하나만 저장하고 원래 위치에는 문자열 테이블의 인덱스를 저장한다.

본 논문에서는 변수에 한해서만 문자열 테이블을 참조하며 그 외 속성 값과 파싱 된 데이타는 인라인 문자열로 인코딩 된다. 일반적으로 무선 마크업 언어 문서는 사용자 데이타 내에 반복되는 문자열이 그리 많지 않을 것이며 반복되는 횟수 또한 적을 것이므로 문자열 테이블을 참조하는 것은 오히려 부담만 가중시키는 결과를 낳을 가능성이 크다. 그러나, 변수는 카드를 매개하는 역할을 하므로 덱 내에 반복되어 나타난다. 또한 여러 개의 카드에서 공통적인 변수 값을 가지고 있어야 하므로 덱 차원에서 관리할 필요가 있다. 변수를 치환하는 방법은 직접치환, escaped 치환, unescaped 치환의 세 가지가 있으며, 본 논문에서는 직접 참조 치환방법(EXT_T_2)을 사용한다. 직접 참조 치환 방법이란 실제 변수명은 선언부의 문자열 테이블에 위치하고, 원래 텍스트 덱 내의 변수가 위치한 곳에서는 전역 확장 토큰과 변수가 문자열 테이블에 위치한 오프셋(offset)을 인덱스로 사용해서 인코딩하는 방법이다. 이때 변수명은 텍스트 문서에 명시된 그대로 부호화한다. 즉, 어떠한 변형 없이 변수명의 문자열 그대로를 저장한다는 의미이다. 이러한 방법은 컴파일 된 바이너리 문서의 크기를 줄일 뿐만 아니라, 이동 단말의 마이크로브라우저가 문서 전체를 파싱하지 않고도 변수를 인식할 수 있도록 한다. 또한 브라우징 시에도 카드 내에 있는 변수의 인덱스를 사용해 덱 내에 있는 변수의 검색 및 변수 값의 편집을 용이하게 하며, 특정 변수의 값을 덱 내에 있는 모든 카드에서 일관되게 유지할 수 있도록 해준다.

표 2 전역토큰 대입표

토큰 명	토큰	설 명
SWITCH_PAGE	0	• 현재 토큰상태에서 코드페이지를 전환한다. 새로운 코드페이지를 나타내는 숫자가 u_int8로 표현된다.
END	1	• 속성리스트나 요소의 끝을 명시한다.
STR_I	3	• 문자열이 나타남을 명시하며, 다음에 문자열이 나온다.
EXT_T_2	82	• 변수치환-변수명을 변형하지 않는다. • 변수명은 문자열 테이블의 참조형태로 인코딩되고 다음에 mb_u_int32 형태의 토큰이 나온다.

<표 2>는 본 논문의 컴파일러가 생성한 바이너리 무선 마크업 언어 문서에서 사용되는 제어토큰들을 보여주고 있다. 토큰 값은 16진수로 표현된다. 속성 리스트와 요소를 종료하기 위해 END 토큰이 사용된다. END는 u_int8로 된 토큰이며 16진수로는 0x01의 값으로 표현된다. 현재 토큰 상태에 대해 현재 코드 페이지를 바꾸기 위해서는 SWITCH_PAGE 토큰이 사용된다.

SWITCH_PAGE 역시 u_int8의 데이타 타입을 가진 토큰이며 16진수인 0x00로 표현된다.

텍스트 데이타 내에서는 <나 &와 같은 마크업에 사용되는 문자를 사용할 수 없다. 만일 마크업에 사용되는 문자를 사용하고자 할 때는 < 혹은 &와 같은 문자 엔터티로 표현해야 한다. 무선 마크업 언어 1.2 문서 형태 정의에서는 7개의 문자 엔터티를 선언하고 있다[2]. 그러나 생성된 바이너리 문서 내에서는 이와 같은 문자는 실제로 마크업에 사용되지 않기 때문에, 본 논문에서는 문자 엔터티를 <나 &와 같은 문자로 치환하였다.

(2) 응용토큰

응용토큰은 WBXML의 응용 문서의 마크업에 사용되는 토큰을 의미한다. 응용토큰은 태그 토큰과 속성 토큰으로 구분된다.

태그 토큰은 무선 마크업 언어 1.2 문서 형태 정의에 정의된 태그에 연관된 바이너리 토큰 값들이 1-바이트로 정의되어 있다. 태그 토큰은 한 바이트 데이타 형식인 u_int8로 압축되며 <표 3>의 내용에 따라 값이 재설정된다.

각 태그 토큰의 상위 두 개의 비트의 값에 따라 마이크로브라우저는 현재 요소에 속성이나 내용이 포함되었는지의 여부를 알 수가 있다. 이는 이동 단말에 위치한 마이크로브라우저의 처리과정을 줄이게 되므로 CPU나 메모리의 성능이 낮은 이동 단말에 최적화된 기법이라 할 수 있다.

속성 토큰은 무선 마크업 언어 1.2 문서 형태 정의에 정의된 속성의 이름과 속성 값에 연관된 바이너리 토큰 값들이 1-바이트로 정의되어 있다. 속성 토큰은 u_int8의 데이타 타입으로 압축되며, 속성의 시작과 속성 값 두 가지 범주로 나누어진다. 속성의 시작은 문서 형태 정의에 선언된 속성의 종류를 나타내며, 때로 명명된 속성 값을 가질 경우에는 속성의 종류와 속성 값이 하나의 속성의 시작이 되기도 한다. 속성의 시작은 128보다 작은 값으로 압축된다.

다음으로 속성 토큰의 범주에는 속성 값이 포함된다. 본 논문에서는 무선 마크업 언어 표준안에서 정한 몇몇 명명된(well-known) 속성 값의 경우에는 128 이상의 값으로 압축하고, 그 외는 인라인 문자열로 인코딩한다. WBXML에서 속성 값은 원칙적으로 문자열 그대로 인코딩하도록 명시되어 있으나 무선 마크업 언어 표준안에서는 속성 값들 중에 사용 빈도가 높은 몇몇 문자열의 경우 확장 인코딩을 적용하여 한 바이트로 압축한다. http://, .com, accept 등이 그 예로 정의된 토큰은 얼마

표 3 바이너리 태그 형식

비트	설 명
7 (최상위 비트)	• 태그 코드 다음에 속성들이 나오는지를 가리킨다. • 이 비트 값이 0 이면, 태그는 속성을 포함하지 않는다. 이 비트 값이 1 이면, 태그는 바로 다음에 하나 이상의 속성들이 오게 된다. 속성 리스트는 END 토큰으로 끝을 명시한다.
6	• 태그가 내용을 포함하는 요소인지를 가리킨다. • 이 비트 값이 0 이면, 태그는 내용과 마침 태그를 포함하지 않는다. 이 비트 값이 1 이면 태그 다음에 내용이 오게 되며, END 토큰으로 끝을 명시한다.
5-0	• 각 태그와 연관된 바이너리 토큰 값

되지 않는다. 예를 들면 ac.kr 혹은 .de와 같은 문자열은 정의되어 있지 않다. 앞으로 이러한 속성 값들이 더 많이 정의되어야 한다. 문자열에 대한 확장 인코딩을 많이 적용할수록 압축 효과는 증가하기 때문이다.

다음은 속성이 인코딩되는 예를 보이고 있다. 아래의 속성을 예로 들면,

href="http://ccmc.knu.ac.kr"

본 논문에서는 href="http://를 잘 알려진 토큰의 하나로 인식하여 바이너리 토큰 값 0x4b로 변환한다. 이후의 속성 값에 대해서는 디폴트로 인라인 문자열로 인코딩한다. 위 속성을 인코딩한 결과는 아래와 같다.

0x4b 0x03 'c' 'c' 'm' 'c' '.' 'k'
'n' 'u' '.' 'a' 'c' '.' 'k' 'r' 0x00

3.2.4 기타

본 논문에서는 문서의 유효성 검증에 필요한 부가적인 사항을 검사한다. %length;로 정의된 속성 값은 NMTOKEN 타입을 가지게 되지만 뒤에 %가 있는 NMTOKEN도 유효한 값으로 인정한다. 예를 들어 height="50"과 height="50%" 둘 다 가능하다. 또한 %vdata;로 정의된 속성 값의 타입은 CDATA이지만 변수가 포함되는지 검사한다. 그 외에, 문서 형태 정의에 속성 값의 타입이 정의되지 않은 경우에 속성 값은 모두 %vdata;로 처리한다. 주석문과 요소간의 공백은 모두 필요 없다. 주석문과 무의미한 공백은 모두 제거된다. 단, 데이타 내에 있는 의미 있는 공백은 반드시 필요하지는 않지만 제거할 수 없으므로 모두 [\b]로 치환되도록 하였다.

3.3 시스템 구조의 설계 및 구현

3.3.1 동작 원리

본 논문에서 구현된 무선 마크업 언어 컴파일러는 텍스트 무선 마크업 언어 문서를 입력으로 받아들여 바이너리 무선 마크업 언어 문서를 생성하는 도구로써, 컴파일러의 필요조건인 문서의 인코딩 기능과 확장 마크업 언어 well-formedness 검증 기능을 지원하며 부가적으로 확장 마크업 언어 validation 검증 기능을 지원한다.

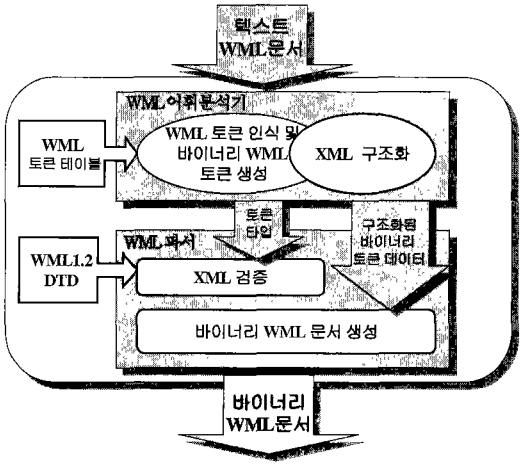


그림 2 무선 마크업 언어 컴파일러 동작 구조

무선 마크업 언어 컴파일러는 어휘분석기와 파서 두 개의 컴포넌트로 설계되었으며 <그림 2>과 같이 동작한다. <그림 2>에서 보여지듯이 어휘분석기는 텍스트 무선 마크업 언어를 기반으로 토큰을 인식하며, 파서는 어휘분석기에서 인식된 토큰 타입을 사용해 문법을 파싱하여 확장 마크업 언어의 well-formedness를 비롯해 validation 검증을 수행한다. 파싱이 완료되면, 어휘 분석기에서 토큰을 인식하면서 만들어 둔 각각의 바이너리 값들을 재구성하여 WBXML 문법에 적합한 완성된 바이너리 무선 마크업 언어 문서를 생성하게 된다. 어휘분석기는 표준안에 명시된 무선 마크업 언어 토큰 테이블과 확장 마크업 언어의 형식을 기반으로 문서의 토큰을 인식하여 파서로 토큰의 타입을 반환한다. 이 과정에서 인식된 토큰에 해당하는 바이너리 값이 생성되는데, 주석문과 같은 불필요한 토큰들에 대한 타입은 파서로 반환되지 않는다. 파서는 어휘분석기에서 반환한 토큰의 타입을 사용해 문법을 파싱해 나간다. 파서의 문법은 확장 마크업 언어 1.0의 형식 및 무선 마크업 언어 1.2 문서 형태 정의에 선언된 내용을 바탕으로 설계되었다. 텍스트 문서는 확장 마크업 언어 1.0 기반 응용문서이므로 파서의 문법구조[15]는 확장 마크업 언어의 BNF 문법 구조를 기반으로 틀을 만들어 무선 마크업 언어 1.2 문서 형태 정의를 기반으로 내용을 채워 넣는 방식으로 설계되었다. 또한 파서의 문법은 WBXML의 문법도 고려하여 설계되었다.

3.3.2 어휘분석기 모듈

(1) 토큰화 개요

텍스트 무선 마크업 언어 문서는 크게 마크업과 파싱된 데이타에 따라 어휘 분석의 형태가 달라진다. wml이라는 토큰을 인식했을 때 이 토큰이 마크업에 사용되었다면 하나의 덱을 인식하는 요소의 시작 태그로 인식될 것이고, 파싱 된 데이타에 사용되었다면 wml이라는 덱 내의 문자 데이타로 인식될 것이다. 만일 주석문 내에 존재하게 되면 무시될 것이다. 즉, 하나의 토큰을 인식하고자 할 때 우선 이 토큰이 마크업인지 파싱 된 데이타인지, 주석문인지가 구분되어야 한다.

어휘분석기는 하나의 토큰이 인식될 때 내부적으로 이 토큰이 주석문, 마크업 그리고 데이타 상태인지를 구분하여 각각이 독립적인 어휘분석기로 동작되어진다. 뿐만 아니라, 변수와 속성 값들을 정확하게 인식하도록 하기 위해 위의 어휘분석기 내에서도 변수와 속성 값 각각에 대한 상태를 구분하도록 하여 독립적인 어휘분석기가 동작하도록 하였다. 이는 정확한 데이타의 인식뿐만 아니라, 확장 마크업 언어 형식을 기반으로 문서를 인식하므로 인코딩 된 데이타를 구조화시키는데 용이하다.

(2) 내부구조

무선 마크업 언어 컴파일러의 어휘분석기는 텍스트 문서의 파싱 상태에 따라 각각의 어휘분석기를 두어 입력 문서를 토큰화하고 있다. 어휘분석기는 텍스트 문서를 확장 마크업 언어 well-formedness 정의에 따라 구

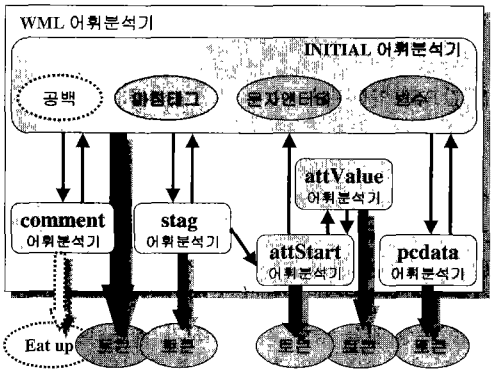


그림 3 어휘분석기 내부 구조

조화한다. <그림 3>에서는 각각의 어휘분석기를 각 상태변화와 관련하여 설명하고 있다.

텍스트 무선 마크업 언어 문서를 입력할 때의 초기상태를 INITIAL이라 한다. INITIAL 어휘분석기는 받아들인 토큰에 따라 여러 다른 어휘분석기로 상태를 변환시켜주는 하나의 중앙 관리자적인 역할을 담당한다. INITIAL 상태에서 <!-- 토큰이 인식되면 상태는 comment 어휘분석기로 넘어간다. 또한 INITIAL 상태에서 <토큰이 오면 마크업이 시작되고 나머지는 pcdata 상태로 무선 마크업 언어 텍의 데이타 부분이 된다.

comment 어휘분석기는 INITIAL 상태에서 <!-- 토큰이 인식되면 상태가 변화된다. comment 어휘분석 상태는 --> 토큰이 나올 때까지 계속된다. 이 상태에서 읽어들인 모든 토큰은 무시되고 응용프로그램으로 전달되지 않는다. 물론 comment의 시작과 끝을 표현하는 <!--과 --> 토큰 역시 무시된다. comment 어휘 분석 상태가 종료되면 상태는 다시 INITIAL로 전환된다.

INITIAL 상태에서 < 토큰이 인식되면 마크업의 시작으로 가정하여, 다음 토큰으로 시작 태그가 와야 하는데 이 어휘 분석 상태를 stag 어휘분석기로 두었다. 각 요소에 해당하는 시작 태그가 인식되면 해당하는 시작 태그 토큰의 타입을 파서로 반환한다.

표 4 무선 마크업 언어 요소의 네 가지 대표적 형태

형태	예
Stag	<tag/>
Stag Attributes END	<tag attrStart= "attValue"/>
Stag Contents Etag	<tag> contents </tag>
Stag Attributes END *Contents Etag*	<tag attrStart="attValue"> *contents* </tag>

무선 마크업 언어의 각 요소는 <표 4>에서와 같이 네 가지 형태를 띄게 된다. 다음 각각의 형태에 따라 시작 태그의 값이 변경된다.

stag에서 > 또는 공백과 함께 > 토큰이 인식되면 상태는 다시 INITIAL로 전환된다. 이 경우는 <표 4>에서 세 번째 요소의 형태에 속하므로 이 시작 태그는 내용을 포함하는 요소이다. 텍스트 텍에서 <태그명> </태그명>로 구성된 요소도 이 형태에 포함된다. 이 경우, 시작 태그는 내용을 포함하고 속성을 가지고 있지 않은 태그로 구분되어 시작 태그의 내용을 지정하는 비트가 1로 저장된다. 이 상태에서 공백과 함께 시작 태그의 끝을 표현하는 마크업 문자가 오지 않고 공백만 오게 되면, 어휘 분석 상태는 attStart 어휘분석기로 넘어가게 된다. 다음으로 /> 혹은 공백과 함께 /> 토큰이 인식되면 상태는 INITIAL로 넘어간다. 그리고 시작 태그는 아무런 변경 없이 태그 식별자 그 자체로 저장된다. 즉 이 경우는 <태그명/>의 형태를 가지는 요소로 속성도 내용도 포함하지 않는 시작 태그가 된다.

attStart 어휘분석기는 반드시 stag 어휘 분석 상태에서 넘어오게 된다. 이 상태에서는 속성의 시작 토큰이 오게 되는데, 속성의 시작이 type="accept"와 같이 속성의 값과 함께 읽어 들이거나, id="과 같이 속성의 종류를 나타내는 이름만이 인식되어 질 수 있다. 전자의 경우, 상태는 여전히 attStart 상태이지만, 후자의 경우에는 속성 값이 와야 하므로 상태는 attValue 어휘분석기로 넘어가게 된다. attValue는 무선 마크업 언어 1.2 문서 형태 정의에 명시된 각각의 속성 값의 선언에 따라 실제 구현상에서는 각 속성 값이 타입에 따라 정확하게 인식되도록 하기 위해 id, href, vdata, cdata, nmtoken 등의 독립적인 어휘 분석 상태로 구성하였다. 그러나 어휘 분석 상태의 전환이나 다른 기능들은 똑같이 작용하므로 이후 속성 값에 대한 어휘분석기는 모두 편의상 attValue로 통칭한다.

attStart 어휘 분석 상태에서 > 혹은 공백과 함께 > 토큰이 인식되면 상태는 INITIAL로 전환된다. 이 경우는 <표 4>에서 네 번째 요소의 형태에 속하므로 이 시작 태그는 속성과 내용을 모두 포함하는 요소이다. 따라서 시작 태그의 상위 두 개의 비트가 모두 1로 재 설정된다. 다음으로 /> 혹은 공백과 함께 /> 토큰이 인식되면 마크업이 끝나고 역시 INITIAL 상태로 전환된다. 이 경우는 <표 4>에서 두 번째 요소의 형태이며 요소는 내용을 가지고 있지 않을 것이 예상되므로 시작 태그는 속성 플래그만 1로 재 설정된다.

속성에 따라 속성 값의 타입이 다르고 이들 타입은 서로 완전히 독립적이지 않고 서로 중첩되는 요소를 지니고 있다. 따라서 이를 하나의 attValue 어휘분석기에서 인식하게 되면 충돌이 발생하여 타입에 따라 토큰을 정확하게 구분하기 어렵다. 그래서 attStart에서 이들 각각을 구분하여 인식할 수 있도록 하였다. 만일 속성 값의 타입이 문서 형태 정의에 정의되어 있지 않으면 vdata로 간주하여 토큰을 읽어 들인다. vdata 타입에는 변수가 올 수 있다. 변수는 $ 문자를 사용해 인식이 가능하므로 예약된 문자열(reserved word)를 변수로 사용해도 정확한 값의 인식이 가능한 것이다. 예를 들어, $password 과 password는 명확히 구분된다. 만일, 사용자 데이타로

$값을 사용하려면 $$로 표기하여 변수를 나타내는 $ 문자와 구분할 수도 있다. 속성 값의 종류에 따라 각각 다른 어휘분석기가 동작하게 되고, 속성 값이 모두 인식되면 attStart 상태로 돌아가게 된다. 여러 개의 속성들이 나열된 경우 각각의 속성은 공백을 통해 구분된다.

pcdata 어휘분석기는 INITIAL 상태에서 주석문에 사용되는 <!--와 마크업의 시작에 사용되는 < 가 아닌 토큰이 인식되면 이는 파싱 된 데이타로 인식하여 pcdata 어휘분석 상태로 전환된다. 즉, 일반 문자열, 변수, 문자 엔터티가 오게 되면 pcdata 상태로 전환하도록 하였다. pcdata 어휘분석기에서는 공백은 데이타로 취급되므로 무시하지 않고 모두 [/b]로 치환된다. pcdata 어휘분석기에서 변수가 인식되면 문자열 테이블에 변수의 이름을 저장하고 실제 덱에서는 전역 확장 토큰 EXT_T_2와 문자열 테이블의 인덱스를 저장한다. 문자 엔터티는 그와 연관된 실제 문자로 치환되어서 일반 문자열과 함께 저장된다. pcdata 상태에서 변수와 일반문자열들은 각각 독립적인 토큰으로 인식되어 파서로 반환된다. 지면 관계상 구현된 각각의 속성 값에 대한 자세한 설명은 생략한다.

(3) 바이너리 토큰으로의 변환

어휘분석기에서는 텍스트 무선 마크업 언어 문서를 토큰화하며 표준안에 정의된 토큰 테이블을 기반으로 해당하는 바이너리 값을 생성한다. 이들 바이너리 값들은 완성된 문서가 아니라 문서를 헤더와 문자열 테이블, 덱 등으로 구조화되어 각각 따로 관리된다.

<그림 3>에서처럼 주석문과 공백은 제거되고 문자 엔터티는 문자로 변환되어 저장된다. 이처럼 각각의 어휘분석기에서 모든 토큰이 직접 바이너리 토큰으로 변환되는 것이 아니라 일부는 무시되거나 다른 처리과정을 거쳐 저장된다.

시작 태그 토큰에 대한 바이너리 값은 다른 토큰과 마찬가지로 정의되어 있다. 그러나 시작 태그 토큰은 다른 토큰과는 달리 요소가 속성 및 내용을 포함하는지의 여부에 따라 값이 달라진다. 이를 처리하기 위해 어휘분석기는 시작 태그를 읽어들이게 되면 일단은 토큰 테이블에 정의되어 있는 바이너리 토큰 값을 저장한 후에 태그의 형식이 결정되면 속성과 내용에 해당하는 비트를 1로 재 설정한다.

속성 값은 보통은 문자열로 저장하지만 많이 사용되는 속성 값, 즉 명명된 속성 값은 바이너리 토큰 값이 정의되어 있어 확장 토큰 형식을 사용해 압축한다. 속성 값을 문자열로 저장할 때 본 논문에서는 인라인 문자열 저장방식을 택하고 있다. 형태는 <표 5>와 같다.

표 5 속성 값 저장 형식

STR_I	문자열	NULL

속성 값에서 변수가 인식되어 질 경우가 있다. 이 경우에는 VAR 토큰으로 인식되어 토큰의 타입이 파서로 반환된다. 변수는 덱 내의 어디에서나 참조할 수 있어야 되므로 문서의 선언부 다음에 위치한 문자열 테이블에 저장한다. 각 변수의 문자열은 NULL로 끝나게 된다. 실제 변수가 위치한 덱에서는 <표 6>의 형식으로 저장된다. 문자열 테이블에는 전체 덱 내의 모든 변수가 연속하여 저장되며, 어휘분석기에서는 변수가 저장될 때마다 마지막으로 저장된 오프셋 값을 가지고 있어서 다음 변수가 문자열 테이블에 저장될 때 변수의 처음 오프셋 값을 알 수 있다.

표 6 덱 내에서의 변수 참조

EXT_I_2	문자열 테이블 오프셋

파싱 된 데이타는 속성 값과 마찬가지로 인라인 문자열 저장방식을 취한다. 어휘분석기는 이 파싱 된 데이타를 효율적으로 관리하기 위해 임시 버퍼를 가지고 있다. pcdata 어휘분석기는 구현상에서 일반 문자열과 공백, 문자 엔터티를 각각의 토큰으로 인식한다. 그러나 실제 파서로 반환될 때 이들은 모두 하나의 PCDATA 토큰으로 인식되어 반환된다. 연속된 공백의 경우에는 하나의 [/b]로, 문자 엔터티는 해당 문자로 변환되어 임시 버퍼에 저장되었다가 하나의 토큰으로 묶여서 파서로 넘겨진다. 이와는 달리 파싱 된 데이타에 나타나는 변수는 VAR 토큰으로 인식되어 반환된다. 그 외 속성의 시작과 같이 제어토큰이 필요 없는 토큰은 표준안에 정의된 바이트 코드로 압축된다.

3.3.3 파서 모듈

무선 마크업 언어 파서는 입력 문서의 확장 마크업 언어의 well-formedness와 validation을 검사하여 완성된 바이너리 무선 마크업 언어 문서를 생성하기 위한 모듈이다. 파서는 문서가 확장 마크업 언어 1.0과 무선 마크업 언어 1.2 문서 형태 정의에 유효한 문서인가를 검증해야 하며 바이너리 무선 마크업 언어 문서를 구성하기 위해서는 WBXML의 양식을 적용해 문서를 생성해야 한다. 이러한 이유로 파서의 문법구조는 확장 마크업 언어 1.0과 무선 마크업 언어 1.2 문서 형태 정의를 기반으로 WBXML를 고려하여 작성되었다.

(1) 텍스트 덱의 문법구조

텍스트 무선 마크업 언어 문서의 물리적인 구조는 확장 마크업 언어 1.0 표준안에 따라 작성된다. 프롤로그와 루트 요소, 그 외 주석문, 처리명령, 공백 등이 올 수 있다. 여기서 프롤로그는 모든 텍스트 무선 마크업 언어 문서에 공통적으로 적용되므로 이에 대한 문법은 생략하였다. 주석문과 공백은 모두 어휘분석기에서 처리하므로 역시 포함시키지 않았다.

확장 마크업 언어 1.0 표준안에서는 하나의 요소는 내용을 포함하지 않는 시작 태그만으로 이루어진 형태와 시작 태그, 내용 그리고 마침 태그로 이루어진 형태, 두 가지로 분류한다. 여기서 시작 태그는 태그명과 속성들의 리스트로 구성되어 있다.

또한 요소의 내용으로 주석문도 포함되어 있다. 본 논문에서는 각 요소를 그 요소 내에 속성이나 내용을 포함하는가의 여부에 따라 여섯 가지로 룰이 구분된다. 각 룰은 <, 태그명, 속성, >, 내용, </, 태그명, > 등의 터미널(terminal)과 넌터미널(nonterminal)로 구분되고, 시작 태그에 나타나는 공백들 중 속성들 간의 공백을 제외한 나머지는 어휘분석기에서 다른 토큰과 함께 인식하도록 하였다. 예를 들면, S? '>'는 어휘분석기에서 {whitespace}*'>'로 인식하여 파서로는 > 문자만을 반환하도록 하였다.

속성에서 Name Eq '\"'은 어휘분석기에서 하나의 속성 시작 토큰으로 인식되어 파서로 반환되어 지며 attValue'\"' 역시 하나의 속성 값 토큰으로 처리된다. 만일, 속성 값이 명명되어 있다면 속성명과 속성 값이 하나의 토큰으로 인식될 것이다. 속성 값에 참조(reference)가 나타나면 이는 1-바이트의 문자로 변환되어 속성 값 전체가 문자열로 인식될 것이다. 주석문은 어휘분석기가 인식하여 생략하므로 파서에서는 처리하지 않는다.

(2) 문법설계

모든 텍스트 무선 마크업 언어 문서는 공통의 확장 마크업 언어 버전과 무선 마크업 언어 1.2 문서 형태 정의에 대한 공개 식별자를 가지므로 어휘분석기에서 이를 인식하고 파서에서는 단지 WBXML 버전인지 문서 형태 정의에 대한 공개 식별자인지만을 구분해 주면 된다. 물론, 순서는 지켜져야 한다. 덱은 <wml>이 루트 요소가 된다. 요소의 형식은 확장 마크업 언어의 문법구조를 기반으로 하며 요소의 내용은 무선 마크업 언어 1.2 문서 형태 정의를 따르고 있다. 문서 형태 정의에는 각 요소들에 대한 정의와 요소에 대한 속성의 종류와 속성 값의 타입 및 요소에 포함되어야 할 내용 등이 선언되어 있다.

본 논문에서는 무선 마크업 언어 1.2 문서 형태 정의에 정의된 이러한 내용을 기반으로 문서의 유효성 여부를 검사한다.

파서의 문법구조는 LALR(1) 문법[15]이며 무선 마크업 언어 1.2 문서 형태 정의에 정의된 모든 엔터티들과 요소들의 속성 및 속성 값, 그리고 내용에 관해 각각을 파싱할 수 있도록 설계하였다.

다음은 무선 마크업 언어 요소 중 가장 일반적인 문법형태를 따르고 있는 <p> 요소의 문서 형태 정의 선언에 따른 문법이다. 지면 관계상 파서의 전체 문법을 나열하진 못하였다.

```
1    <!ELEMENT p (%fields; | do)*>
2    <!ATTLIST p
3    align %TAlign; "left"
4    mode %WrapMode; #IMPLIED
5    xml:lang NMTOKEN #IMPLIED
6    %coreattrs;
7    >
```

위의 내용은 무선 마크업 언어 1.2 문서 형태 정의 중 <p>요소와 속성에 대해 정의한 부분이다. 라인 1은 엔터티 %fields;와 <do> 요소를 내용으로 포함하는 <p> 요소를 선언하고 있다. 라인 2에서 7까지는 <p> 요소의 속성들을 나열하고 있다. <p> 요소는 속성으로 align, mode, xml:lang와 엔터티 %coreattrs;을 가질 수 있다. align의 타입은 엔터티 %TAlign;이며 디폴트로 left 값을 가진다. 다음으로 mode의 타입은 엔터티 %WrapMode;이며 디폴트값은 없고, 꼭 필요한 요소는 아니다.

xml:lang 역시 반드시 요구되는 속성은 아니며, 타입은 NMTOKEN 이다. %coreattrs;는 많이 사용되는 속성들을 따로 정의해 놓은 것으로 id와 class를 정의하고 있다. 이러한 내용을 검증하기 위한 파서의 문법은 아래와 같다.

```
1   p: '<' TAG_P "/>"
2   | '<' TAG_P p_attributes "/>"
3   | '<' TAG_P '>' "</" TAG_P '>'
4   | '<' TAG_P '>' p_contents "</" TAG_P '>'
5   | '<' TAG_P p_attributes '>'    "</" TAG_P '>'
6   | '<' TAG_P p_attributes '>' p_contents "</" TAG_P '>'
7   ;
8  p_attributes: p_attr
```

```
9    | p_attributes  S        p_attr
10   ;
11 p_attr: p_attr_align
12   | mode
13   | xml_lang
14   | ENTITY_coreattrs
15   ;
16 p_attr_align: ATTR_ALIGN_LEFT
17   | ATTR_ALIGN_RIGHT
18   | ATTR_ALIGN_CENTER
19   ;
20 p_contents: p_content
21   | p_contents    p_content
22   ;
23 p_content: ENTITY_fields | do
24   ;
25 mode: ATTR_MODE_NOWRAP
26   | ATTR_MODE_WRAP
27   ;
28 xml_lang: ATTR_XML_LANG nmtoken ;
29 ENTITY_coreattrs: id    | class ;
30 id: ATTR_ID        TOK_ID  ;
31 class: ATTR_CLASS        cdata   ;
```

각 요소는 <, 태그명, 속성, >, 내용, </, 태그명, >등의 터미널과 넌터미널을 구성한다. 하나의 요소를 파싱하기 위해서는 이들 터미널과 넌터미널의 조합으로 이루어진 룰이 존재해야 한다. 본 논문에서는 라인 1-7에서와 같이, 각 요소의 전체적인 형태를 여섯 가지로 나누어 룰을 작성하였다. 라인 1은 시작태그만을 가진 형태로 속성과 내용 모두 포함하지 않는다. 라인 2에서는 속성만을 포함한 형태이며, 라인 4는 내용만을 포함할 경우를 라인 6은 속성과 내용 모두를 포함하는 경우를 나타낸다. 특이하게 라인 3에서는 실제 요소는 데이타를 포함하지 않지만, 태그의 형태는 내용을 포함하는 것으로 취급된다. 라인 5의 경우도 마찬가지이다.

라인 8-19는 <p> 요소의 나열된 속성들을 인식하기 위한 문법이다. <p> 요소의 속성에는 align, mode, xml:lang, 엔티티 coreattrs가 있다. 속성 align과 mode는 정해진 값 이외에는 사용할 수 없다. align은 align="left", align= "right" 혹은 align="center" 중 하나가 올 수 있고, mode는 mode="wrap"와 mode="nowrap" 두 가지 형태 중 한가지를 취할 수 있다. xml:lang의 속성 값의 타입은 NMTOKEN이며, id는 ID, class는 CDATA 타입을 속성 값으로 가진다. <p>요소는 엔티티 %field;와 <do> 요소를 포함할 수 있다. 엔터티 %field;는 다시 <input>이나 <select>, <fieldset>을 비롯해 데이타와 같은 내용을 포함할 수 있다. 다른 요소들 역시 이런 방식으로 문서 형태 정의에 따라 룰을 정의한다.

(3) 최종 바이너리 파일 생성

어휘분석기에서는 인식된 토큰에 해당하는 바이너리 값들을 생성한다. 이는 텍스트 문서를 기반으로 구조화되어 있으므로 최종 바이너리 무선 마크업 언어 덱을 생성하기 위해서는 생성된 바이너리 값들을 재구성해야 한다. 완성된 바이너리 덱은 다음과 같은 구성을 가진다.

전체 바이너리 무선 마크업 언어 문서는 WBXML 버전, 무선 마크업 언어 1.2 문서 형태 정의에 대한 공개 식별자, 문자집합, 문자열 테이블과 덱으로 이루어져 있다. 버전과 공개 식별자는 모두 동일하게 적용되므로 어휘분석기에서 쉽게 생성된다. 문자집합은 텍스트 무선 마크업 언어의 문자인코딩을 나타낸다. 만일 텍스트 문서 내에 문자 인코딩에 대한 선언이 없다면, 인코딩은 UTF-8[14]로 지정하였다. 문자열 테이블은 덱 내에 있는 변수를 참조하기 위한 것인데 파싱이 완료되어야 결정이 되므로 다른 토큰과는 독립적으로 생성된다.

WBXML 표준안에서는 시작 태그가 확장토큰으로 주어지지 않으면 문자열로 그대로 주어지도록 한다. 그러나 무선 마크업 언어 1.2 표준안에서는 태그의 문자열 표현을 허용하지 않고 있어, 본 논문에서 모든 태그들은 확장토큰으로 압축된다. 속성의 시작도 마찬가지로 확장토큰만이 가능하다. 요소 내의 확장토큰의 경우, 변수를 인코딩하는데 있어 문자열 테이블을 참조하도록 하는 직접 참조 치환(EXT_T_2)방법을 사용하고 있다. 문자열의 경우에는 인라인 문자열만을 채택하고 있으며 문자열의 참조는 변수로 제한하였다.

3.4 구현환경

운영체제는 ALZZA Linux release 6.0(Kernel 2.2.12 on an i686)를 사용하였으며, 텍스트 무선 마크업 언어 문서를 확장 마크업 언어의 well-formedness를 기반으로 토큰화하기 위해 리눅스용 Flex 버전 2.5.4[16]를 이용해 어휘분석기를 작성하였다.

확장 마크업 언어 validation을 검증하기 위해서는 확장 마크업 언어1.0과 무선 마크업 언어 1.2 문서 형태 정의를 기반으로 설계된 문법이 필요하였고, 이를 위해 GNU Bison 1.27[10]을 사용해 LALR(1) context-free 문법을 가진 파서를 구현하였다. 그 외 어휘분석기와 파서에서는 C언어를 사용하였고, 파싱 과정에서 생성한 바이너리 토큰들을 재구성하기 위한 과정도 C언어로 구현하였다.

4. 무선 마크업 언어 컴파일러 기능 실험 및 결과 데모

4.1 구현 결과 및 압축효과에 대한 평가

4.1.1 실험에 사용된 데이타

본 논문에서 구현한 무선 마크업 언어 컴파일러의 기능 검증 실험에서는 모두 130개의 텍스트 무선 마크업 언어 파일이 사용되었다. 실험에 사용된 데이타의 일부는 노키아[11]와 폰닷컴[12]의 툴킷에 포함된 예제들이며 나머지는 인터넷을 통해 다운로드받은 것들이다[6].

실험에 사용된 데이타의 크기는 268 바이트에서 2968 바이트 사이로써, 평균 파일 크기는 1618 바이트이다. 실험에 사용된 데이타 중 노키아와 폰닷컴의 데이타들은 무선 마크업 언어의 다양한 기능을 보여줄 목적으로 작성된 데이타가 대부분이다. 그 외 약 100여 개의 데이타는 현재 무선 포털 사이트에서 다운로드받은 것들이며 이들 데이타는 현재 서비스되고 있는 무선 홈페이지들이다.

본 논문에서는 무선 마크업 언어의 다양한 기능을 무선 마크업 언어 컴파일러를 사용해 실험하였으며, 실제 서비스되고 있는 데이타들에 대해 컴파일 해 봄으로써, 구현된 컴파일러의 기능을 검증하였다. 다음으로 본 논문에서는 무선 마크업 언어 1.2를 지원하므로 이전 버전의 데이타의 경우에는 무선 마크업 언어 1.2로 업그레이드하여 실험에 사용하였다.

4.1.2 구현 결과 및 압축효과에 대한 평가

본 논문에서 구현된 무선 마크업 언어 컴파일러는 텍스트 문서를 이동 단말로 전송하기 전에 문서의 형식 및 내용상의 오류를 검사하여 이동 단말에서 오류 없이 브라우징 가능한 데이타만을 전송하도록 한다. 만일, 잘못 작성된 문서일 경우 에러를 리턴하고, 에러의 내용을 출력하도록 하였다. 게이트웨이는 이러한 오류에 대한 내용만을 이동 단말의 마이크로브라우저에게 알리게 되는 것이다. 구체적 방법은 무선 응용 프로토콜 게이트웨이 구현 방법에 따라 다를 수 있다.

기존의 무선 마크업 언어 관련 사례에서는 확장 마크업 언어의 형식만을 검사하므로 무선 마크업 언어의 형식 및 내용 구성에 부적합한 문서도 해당 토큰에 대한 바이트 코드를 생성하여 무선 단말기로 전송한다. 마이크로브라우저는 무선 마크업 언어 표준안에 따라 브라우징 처리를 하므로, 문서의 내용이나 구조적인 면에서 오류가 있는 컨텐트는 이동 단말에서 브라우징 할 수 없다. 문서 형태 정의에 부적합한 데이타의 전송은 무선 선로의 대역폭 낭비는 물론 이동 단말기에서의 불필요한 처리과정에 따른 오버헤드를 발생시킨다. 그러나, 게이트웨이에서 문서를 검증하여 오류가 있을 경우에 프로토콜 헤더에 오류 메시지를 지정하여 보내게 되면 무선 선로에서의 불필요한 데이타 전송을 방지할 수 있을 뿐만 아니라, 마이크로브라우저에서도 헤더 정보만을 사용해 문서의 상태를 판별할 수 있게 되므로 문서검증과 같은 처리를 하지 않아도 될 것이다.

본 논문에서는 이러한 불필요한 데이타의 전송을 사전에 방지하여 무선 선로에서 대역폭의 낭비를 없애고, 메모리, CPU 및 전력 자원이 제한된 이동 단말에서의 오류 데이타에 대한 불필요한 처리로 인한 오버헤드를 방지할 수 있다. 이를 위해 텍스트 무선 마크업 언어 문서의 확장 마크업 언어 well-formedness와 validation 검증 기능을 지원한다.

본 논문에서 구현된 무선 마크업 언어 컴파일러는 문서의 내용 및 구조를 검증하여 바이트 코드를 생성해 낸다. 이 결과 특별한 압축 알고리즘을 사용하지 않고도 원래 문서의 데이타 량이 감소하여 압축의 효과를 가져온다. 바이트 코드는 gzip, bzip2, ELS(Entropy Logarithmic Scale)와 같은 기존의 압축 알고리즘을 사용하여 압축을 했을 경우, 압축 효과가 별로 없다. 이때, 생성된 바이트 코드의 크기가 적을 경우에는 오히려 압축한 파일의 크기가 증가하게 된다[19]. 즉, 이것은 변환된 바이트 코드는 불필요한 데이타의 중복 없이 압축되어 있다는 것을 의미한다.

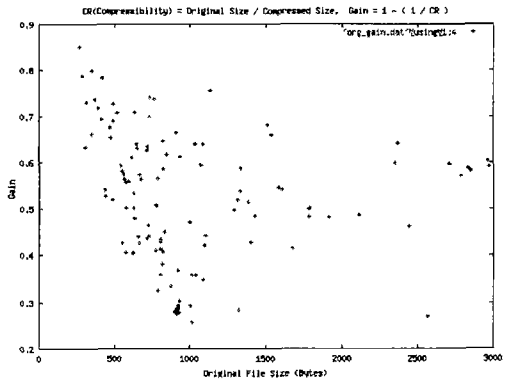
컴파일 한 결과, 데이타 량이 감소되며 바이트 코드에 대한 압축의 정도를 압축비율과 압축 효과로 나타내어 분석하였다. 4.1절에서 제시한 예제를 컴파일 한 바이트 코드에 대한 압축률은 396/77≒5이고, 압축효과는 80%이다.

무선 마크업 언어 문서 130개를 테스트한 결과, 평균 압축률은 4.5 정도이며 최대 6.7 까지 나오는 것을 볼 수 있다. 이를 압축효과로 나타내면, 평균 72%, 최대 85%이다.

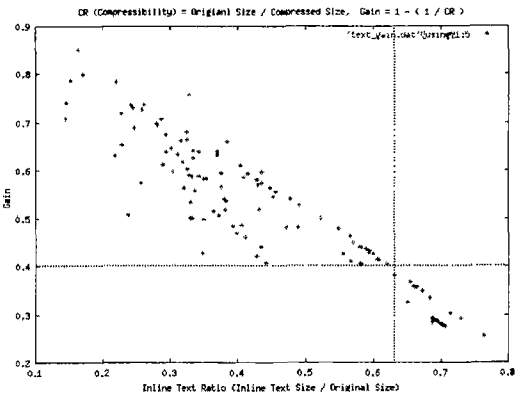
<그림 4>(a)는 텍스트 문서 크기에 따른 압축효과를 나타내며, <그림 4>(b)는 문서 내에서 일반 문자열 데이타의 비율 변화에 따른 압축효과를 나타낸다. CR은 압축비율로써, 컴파일결과 생성된 바이너리 문서 크기에 대한 텍스트 문서 크기의 비로 나타내며, 압축효과는 다음의 식으로 나타내었다.

$$\text{압축비율}(CR) = \frac{\text{텍스트 WML 데이타량}(Bytes)}{\text{바이너리 WML 데이타량}(Bytes)}$$

$$\text{압축효과}(Gain) = 1 - (\frac{1}{CR})$$



(a) 텍스트 무선 마크업 언어 파일 크기에 대한 압축 효과



(b) 일반 문자열 데이타의 비율 변화에 따른 압축효과

그림 4 무선 마크업 언어 컴파일러의 압축효과 분석

<그림 4>(a)는 압축효과가 전체 파일의 크기에 관계없이 임의로 변화하는 것을 보여주고 있고, <그림 4>(b)는 텍스트 문서에 포함된 일반 문자열의 비율이 높을수록 압축효과가 거의 선형 비례적으로 감소하는 것을 나타내고 있다. 이는 태그나 속성을 나타낸 문자열은 한 바이트로 압축되는 반면, 일반 문자열은 이러한 변환 없이 문자열 그대로를 전송하게 되어 덱 내에서 일반 문자열 데이타의 비율이 높아지면 상대적으로 압축 효과가 감소하게 되는 것이다. 그러나 일반 문자열 데이타에 대해 직접 참조 치환 방식을 적용한다고 하더라도 반복되는 문자열의 수가 많지 않을 것이므로 데이타 량의 감소는 기대하기 힘들다. 이동 단말장치는 화면의 크기가 제한되어 있어 디스플레이 가능한 문자열의 크기가 제한되어 있다. 따라서, 무선 홈페이지를 디자인할 때, 이를 고려하여 한 화면 내에서 디스플레이 가능한 정도로 텍스트 데이타를 나누게 되므로 하나의 덱 내에서 일반 문자열이 차지하는 비율은 평균 42%정도이다. <그림 4>(b)에서 보여지듯이 실험에 사용된 전체 130개 예제 문서의 3/4 정도는 압축효과가 40%이상인데, 이들은 모두 덱 내에서의 일반 문자열의 비율이 60%를 넘지 않는다. 나머지 1/4정도는 전체 파일 내에서 일반 문자열의 비율이 63%가 넘는 데이타들인데, 이 데이타에 대한 압축효과는 40%미만임을 알 수 있다.

마지막으로 이러한 압축효과가 무선환경에서 문서의 전송시간에 미치는 영향을 살펴보기로 한다. 게이트웨이는 인코딩시간을 항상 일정하게 유지할 수 있을 정도로 처리속도가 충분히 빠르다고 가정한다. 또한, 이동 단말에서의 디코딩에 대한 분석은 본 연구의 범위를 벗어나므로 디코딩시간도 항상 일정하다는 가정 하에 무선 구간에서의 전송시간을 분석하기로 한다. 무선 세션 프로토콜(Wireless Session Protocol)[17,21]에서의 서비스 데이타 유닛(SDU)의 크기는 디폴트로 1400 바이트이다. 즉, 세션 계층에서 전송할 수 있는 메시지의 최대 크기가 1400 바이트인 것이다. 이때, 서버에서 이동 단말로 1400 바이트 크기의 컨텐트를 전송할 경우 전송시간을 고려해 보자.

현재 IS-95B망에서는 단말기에서 기지국으로 인터넷 접속과 데이타통신 서비스 모두 초당 14.4Kbps의 전송속도를 지원하므로 1400 바이트의 컨텐트를 전송하는데 걸리는 시간은 $\frac{1400bytes \times 8bit}{14.4Kbps} = 778ms$ 이다. 그러나, 이를 컴파일하면 데이타의 크기가 1/6로 감소하여 83% 정도의 압축효과를 가져온다. 이렇게 감소된 데이타는 약 $\frac{778ms}{6} \fallingdotseq 130ms$ 정도로 전송시간이 줄어들기 때문에 전체 무선 트랜잭션 처리를 위한 전체 시스템 반응 속도를 크게 감소시키는 효과를 가져온다.

4.2 컴파일 예 및 분석

다음은 무선 마크업 언어 1.2 표준안에서 제시된 텍스트 덱의 예제이다.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD
  WML 1.2//EN"
      "http://www.wapforum.org/DTD/wml12.dtd">
<!-- WML Encoding Examples -->
<wml>
   <card id="abc" ordered="true">
         <p>
            <do type="accept">
```

```
            <go href="http://xyz.org/s"/>
        </do>
        X: $(X)<br/>
        Y: $(&#x59;)<br/>
        Enter name: <input type="text"
          name="N"/>
      </p>
  </card>
</wml>
```

위 예제를 표준에 따라 바이너리 무선 마크업 언어로 변환시켰을 때 무선 마크업 언어 1.2 표준안에서 제시한 결과를 아래에 보이고 있다. 모든 토큰들은 16진수로 표시되어 있고, 표준안의 예제에서 오류가 있는 토큰들은 WBXML 표준안과 무선 마크업 언어 1.2 표준안을 토대로 정정하였다.

```
02 09 6A 04 'X' 00 'Y' 00 7F E7 55 03
'a' 'b' 'c' 00 33 01 60 E8 38 01 AB 4B
03 'x' 'y' 'z' 00 88 03 's' 00 01 01 03
' ' 'X' ':' ' ' 00 82 00 26 03 ' ' 'Y'
':' ' ' 00 82 02 26 03 ' ' 'E' 'n' 't'
'e' 'r' ' ' 'n' 'a' 'm' 'e' ':' ' ' 00
AF 48 21 03 'N' 00 01 01 01 01
```

다음은 본 논문에서 구현된 무선 마크업 언어 컴파일러를 사용하여 위에 있는 텍스트 덱을 바이너리 형식으로 변환시킨 결과이다.

```
02 09 6A 04 'X' 00 'Y' 00 7F E7 55 03
'a' 'b' 'c' 00 33 01 60 E8 38 01 AB 4B
03 'x' 'y' 'z' '.' 'o' 'r' 'g' '/' 's' 00
01 01 03 'X' ':' ' ' 00 82 00 26 03
'Y' ':' ' ' 00 82 02 26 03 'E' 'n' 't'
'e' 'r' ' ' 'n' 'a' 'm' 'e' ':' ' ' 00 AF
48 21 03 'N' 00 01 01 01 01
```

위 두 개의 결과는 구현된 무선 마크업 언어 컴파일러의 정상적인 동작의 예를 보이고 있다. 볼드(bold)체로 표시한 부분에서 차이가 나는 이유는 현재 구현에서는 일반 문자열에 대해서는 확장코드를 적용하지 않았기 때문인데, 표준에서는 속성 값을 원칙적으로 문자열로 표현하도록 하고 있어 이는 표준에 위배되지 않는다.

4.3 무선 마크업 언어 디컴파일러(decomiler)를 이용한 무선 마크업 언어 컴파일러의 인코딩 결과 확인

본 논문에서 구현된 무선 마크업 언어 컴파일러에 의해 생성된 결과물의 유효성을 검증하기 위해 wml-tools v0.0.4의 wmld(wml decompiler)[6]를 사용한다. 실험에 사용된 wmld는 바이너리 무선 마크업 언어 문서를 원래의 텍스트로 복원해 내는 툴이다. 이 툴은 컴파일 후 생성된 바이너리 무선 마크업 언어 문서 130개 모두를 디컴파일(decompile)하여 원래의 문서로 복원해 내었다. 이는 본 논문에서 구현된 컴파일러의 결과물에 대한 유효성을 검증하는 동시에, 컴파일러의 기능을 확인하기 위한 것이다.

4.4 노키아 무선 응용 프로토콜 툴킷을 이용한 무선 마크업 언어 컴파일러 기능 실험 및 결과 데모

본 논문에서 구현한 무선 마크업 언어 컴파일러의 기능을 검증하기 위해 상용 시스템과의 연동 실험을 실시하였다. 이를 위해 <그림 1>에서의 무선 응용 프로토콜 프로그래밍 모델을 바탕으로 <그림 5>의 실험 환경을 구성하였다.

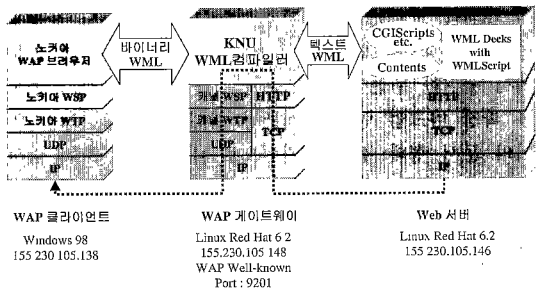


그림 5 시뮬레이션 환경구축

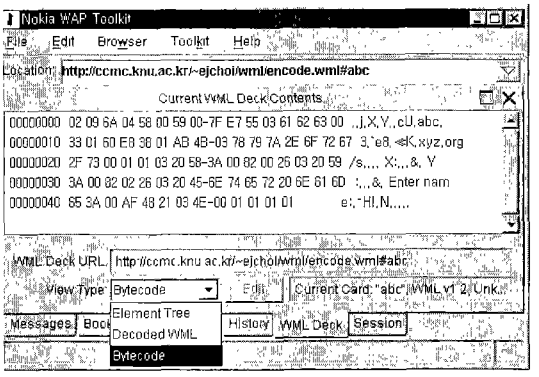
본 논문에서 구현된 무선 마크업 언어 컴파일러는 Kannel: Open Source WAP and SMS gateway Version 0.7[9]의 게이트웨이에 통합된다. 게이트웨이를 설치한 환경[17,18]은 ALZZA Linux release 6.0 (Kernel 2.2.12 on an i686)이며, 게이트웨이-웹서버-클라이언트는 LAN 환경으로 묶여 있다. 본 실험에서는 구현한 컴파일러가 생성한 결과에 대한 적합성 및 컴파일러의 기능을 시험하는 것이 목적이므로 유선 환경에서 테스트 베드를 구성하였다. 실험에 사용된 클라이언트 플랫폼은 Windows 98 상에서 노키아의 상용 제품인 Nokia WAP Toolkit Version 2.0[11]으로써, 무선 응용 프로토콜 스택을 지원하고 있으며 바이너리 무선 마크업 언어 문서를 브라우징하는 기능을 지원하고 있다. 이 실험에서는 노키아[11], 폰닷컴[12]과 WebCab.de[20]을 비롯한 무선 포털 사이트에서 제공하는 예제 130개를 웹서버에 두고 테스트하였다.

실험 절차는 다음과 같다. 노키아 마이크로브라우저에서 웹서버에 있는 encode.wml 이라는 문서를 HTTP 프로토콜을 사용해 명시적으로 요청한다. 이러한 요청을

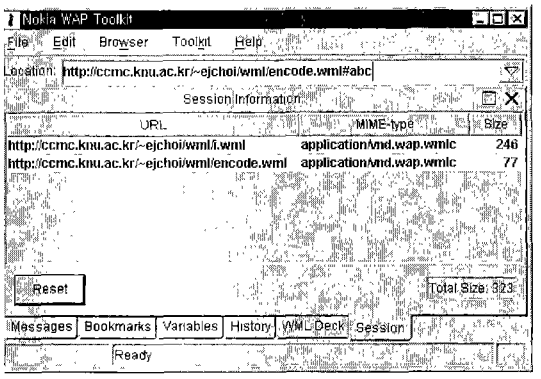
받은 게이트웨이에서는 텍스트 문서를 가져와서 무선 마크업 언어 컴파일러를 동작시켜 텍스트 무선 마크업 언어 문서를 컴파일하여 바이너리 무선 마크업 언어 문서를 생성한 후, Content-Type을 application/vnd.wap.wmlc로 지정하여 클라이언트로 보내지게 된다. 노키아 마이크로브라우저는 바이너리 무선 마크업 언어 문서를 인식하여 보여준다. 이 과정에서 무선 마크업 언어 1.2 문서 형태 정의에 부적합한 컨텐트를 받게 되면 마이크로브라우저는 컨텐트를 검사하지 않고 프로토콜의 헤더 정보만을 사용해 에러 메시지를 보여준다. 만일, 확장 마크업 언어의 well-formedness만을 검사하고 무선 마크업 언어의 문서 형태 정의에는 부적합한 컨텐트를 이동 단말로 전송할 경우, 마이크로브라우저는 문서를 검사한 후에 컴파일 오류를 보고한다. 이는 무선 선로에서의 대역폭 낭비는 물론이고, 이동 단말기에서도 불필요한 처리과정으로 인한 자원의 낭비를 가져온다. 본 논문에서 구현한 무선 마크업 언어 컴파일러는 텍스트 문서를 확장 마크업 언어 형식뿐만 아니라 무선 마크업 언어의 문서 형태 정의를 기반으로 내용 및 구조를 검사하여 이러한 불필요한 자원의 낭비를 막을 수 있도록 설계되었다. 즉, 컨텐트를 게이트웨이에서 검사하여 오류 없는 컨텐트만을 이동 단말로 전송하고 검사 중에 오류가 발생하면 부적절한 컨텐트임을 알리는 제어 메시지만을 전송하여 불필요한 자원의 낭비를 없앨 수가 있다.

본 논문에서 구현한 무선 마크업 언어 컴파일러는 무선 마크업 언어와 WBXML 표준안을 기반으로 한 기능을 모두 제공하며, 이를 토대로 생성된 결과물은 마이크로브라우저를 비롯한 무선 응용 프로토콜 컴포넌트와의 연동실험에서 <그림 6>과 같이 아무런 문제없이 디스플레이 되었다. <그림 6>에서 실험한 예제는 문서 내에 변수가 포함된 예제이다. <그림 6>(a)에서 실제 바이너리 무선 마크업 언어 데이타는 화면 중앙의 16진수 값들이고 오른 쪽의 문자들은 각 라인 내에 포함된 문자들을 나타낸 것이다.

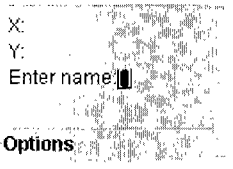
<그림 6>은 노키아 마이크로브라우저가 웹서버에 존재하는 무선 마크업 언어 문서를 요청하여 디스플레이 하는 과정을 캡쳐한 것이다. <그림 6>(a)는 게이트웨이로부터 받은 바이너리 덱의 내용을 보여주고 있으며, <그림 6>(b)는 게이트웨이로부터 받은 컨텐트에 대한 정보이며, <그림 6>(c)는 브라우저의 화면에 디스플레이 된 그림이다. <그림 6>(a)에서 보여지는 바이트 코드는 본 논문에서 구현된 무선 마크업 언어 컴파일러의 실행 결과로 생성된 바이너리 덱을 나타낸다. 노키아 마이크



(a) 무선 응용 프로토콜 게이트웨이로부터 가져온 바이너리 덱



(b) 무선 응용 프로토콜 게이트웨이로부터 가져온 컨텐트 정보



(c) 바이너리 덱을 브라우징한 화면

그림 6 노키아 무선 응용 프로토콜 툴킷을 이용한 구현 결과 화면

크로브라우저는 무선 마크업 언어 1.2 문서 형태 정의에 유효한 문서만을 화면에 브라우징 할 수 있다. 따라서, <그림 6>은 본 논문에서 구현한 컴파일러에 의해 생성된 바이너리 무선 마크업 언어 문서가 확장 마크업 언어 well-formedness를 만족하는 것은 물론, 무선 마크업 언어 1.2 문서 형태 정의에 유효한 문서임을 보여주고 있는 것이다. 또한, 이러한 실험 결과는 본 논문에서

구현한 컴파일러가 확장 마크업 언어의 well-formedness 및 validation 검증을 제대로 수행하고 있음을 보여준다.

5. 결론 및 향후 과제

본 논문에서 설계, 구현한 무선 마크업 언어 컴파일러는 텍스트 문서를 바이너리 형태로 인코딩 함으로써, 무선 선로에서 전송되는 데이타의 양을 최소화할 뿐만 아니라, 확장 마크업 언어 well-formedness와 validation 과정을 무선 응용 프로토콜 게이트웨이에서 처리함으로써, 이동 단말기에서의 문서 처리부담을 경감하는 효과를 가져온다. 또한 파서 생성기를 사용하여 컴파일러를 구현함으로써, 향후 확장 태그나 무선 마크업 언어 버전이 업그레이드될 때에도 변경된 부분에 해당하는 문법만을 재 설계함으로써 새로운 컴파일러를 자동 생성할 수 있는 장점을 가진다.

구현된 컴파일러는 어휘분석기와 파서 두 개의 컴포넌트로 설계되었는데, 어휘분석기는 텍스트 를 기반으로 토큰을 인식하며, 파서는 어휘분석기에서 인식된 토큰 타입을 사용해 문법을 파싱하여 확장 마크업 언어 well-formedness를 비롯해 validation 검증을 수행한다. 파싱이 완료되면, 어휘 분석기에서 토큰을 인식하면서 만들어 둔 각각의 바이너리 값들을 재구성하여 WBXML 문법에 적합하도록 최종 바이너리 덱을 생성하게 된다. 파서의 문법은 확장 마크업 언어 1.0과 확장 마크업 언어 1.2 문서 형태 정의를 기반으로 WBXML의 문법을 고려하여 LALR(1) context-free 문법으로 설계되었다.

본 논문에서 구현한 컴파일러의 기능을 검증하기 위해 생성한 결과물을 표준안과 비교 분석하였으며, 결과물을 디컴파일해서 원래의 문서로 복원해 내는지를 보았다. 뿐만 아니라, 실제 테스트베드를 구성하여 게이트웨이 상에서 컴파일러를 동작시키고, 바이너리 문서를 브라우징하는 기능을 갖춘 외국의 상용 제품인 노키아 무선 응용 프로토콜 툴킷과 연동실험을 실시하여 정상적으로 동작하는 것을 볼 수 있었다.

컴파일 된 데이타의 압축률은 문서에 있는 문자열의 양에 따라 다소 차이를 보이고 있다. 따라서 일반 문자열이 많은 문서의 압축효과를 높이기 위해서는 이러한 일반 문자열에 대해서도 확장 인코딩 기법을 적용할 필요가 있다. 이는 HTML 문서를 무선 마크업 언어 문서로 변환하는 HTML 필터에 특히 적용될 필요가 있을 수 있다. 향후 인터넷 컨텐츠가 다양한 무선 단말기에 적절히 표현될 수 있도록 인공 지능적인 요소를 가미한 HTML/WML 필터에 대한 연구가 필요하다.

참 고 문 헌

[1] "Wireless Application Protocol Architecture Specification," WAP Forum, 30-April-1998. URL:http://www.wapforum.org/

[2] "Wireless Markup Language," WAP Forum, 30-April-1998. URL:http://www.wapforum.org/

[3] "WAP Binary XML Content Format," WAP Forum, 30-April-1998. URL:http://www. wapforum.org/

[4] "Extensible Markup Language(XML), W3C Recommendation 10-February-1998, REC-xml-19980210," Tim Bray, et al., February 10, 1998. URL:http://www.w3.org/TR/REC-xml

[5] "Wireless Application Environment Specification," WAP Forum, 30-April-1998. URL:http://www.wapforum.org/

[6] "wml-tools v0.0.4," wml-tools, 31-January-2000. URL:http://pwot.co.uk/wml/

[7] "Hypertext Transfer Protocol - HTTP/1.1," R. Fielding, et al., January 1997. URL: http://www.ietf.org/rfc/rfc2068.txt

[8] "Uniform Resource Identifiers (URI): Generic Syntax," T. Berners-Lee, et al., August 1998. URL: http://www.ietf.org/rfc/rfc2396.txt

[9] "Kannel: Open Source WAP and SMS gateway," WapIT Ltd., June-1999. URL:http://www.kannel.org/

[10] Charles Donnelly, Richard Stallman, "Bison: The YACC-compatible Parser Generator," Free Software Foundation, Inc., 1995. URL:http://www.gnu.org/

[11] "Nokia WAP Tookit Version 2.0," Nokia, URL:http://www.nokia.com/

[12] "UP.SDK4.0," Phone.com Inc., 1999. URL:http://www.phone.com/

[13] Alex Ceponkus, Faraz Hoodbhoy, *Applied XML*, p.17, Wiley, New York, 1999.

[14] "UTF-8, a transformation format of Unicode and ISO 10646," F. Yergeau, October 1996, URL: http://www.ietf.org/rfc/rfc2044.txt

[15] Kenneth C. Louden, *Compiler construction: Principles and practice*, PWS Publishing Company, Boston, 1997.

[16] Vern Paxson, "Flex, version 2.5: A fast scanner generator," Free Software Foundation, Inc., 1995. URL:http://www.gnu.org/

[17] 김기조, 최윤석, 최은정, 임경식, "무선 응용 프로토콜 기술", 한국정보처리학회지 제7권 제3호, pp.44~56, 2000년 5월.

[18] 임경식, "무선 응용 프로토콜 게이트웨이 구현 기술", KRNET'2000, 발표자료집, pp.184~188, 2000년 6월.
[19] Ojanen E, Veijalainen J, "Compressibility of WML and WMLScript byte code: Initial results," Proceedings Tenth International Workshop on Research Issues in Data Engineering. RIDE 2000. IEEE Comput. Soc 2000, pp. 55-62, 2000.
[20] "WML Examples," WebCab.de, URL:http://webcab.de/
[21] "Wireless Session Protocol," WAP Forum, 5-November-1999. URL: http://www.wapforum.org/
[22] 최은정, 임경식, "무선 인터넷 서비스를 위한 WAP 게이트웨이용 바이너리 WML 인코더의 구현", 제10회 통신 정보 합동 학술대회(JCCI 2000) 발표논문집, pp.863~866, 2000년 5월.

최 은 정
1999년 경북대학교 컴퓨터과학과(이학사). 2001년 경북대학교 컴퓨터과학과(이학석사). 2001년 ~ 현재 한국전자통신연구원 연구원. 관심분야는 유무선 마크업 언어 및 통합 마이크로브라우저, 무선인터넷, 인터넷 정보가전, 자바 및 지니

한 동 원
1982년 2월 숭실대학교 전자공학과(학사). 1992년 2월 한남대학교 전자공학과(석사). 1982년 3월 ~ 현재 한국전자통신연구원 책임연구원, 휴대클라이언트연구팀장. 관심분야는 멀티미디어 휴대정보단말, 모빌 컴퓨팅, 인터넷 정보가전

임 경 식
1982년 경북대학교 전자공학과(공학사). 1985년 한국과학기술원 전산학과(공학석사). University of Florida 전산학과(공학박사). 1985 ~ 1998년 한국전자통신연구원 책임연구원, 실장. 1998년 ~ 현재 경북대학교 컴퓨터과학과 조교수. 관심분야는 이동 컴퓨팅, 무선 인터넷, 홈 네트워킹, 컴퓨터통신