2001년 한국정보처리학회 추계 학술발표논문집 제8권 제2호

WAP 브라우저에서의 WMLScripts 해석기의 설계 및 구현

유승범, 김민수, 장지산, 신동규 ,신동일
세종대학교 컴퓨터공학과
{bummy, mskim, shindk, dshin}@gce.sejong.ac.kr

Design and Implementation of WMLScript Interpreter

Seung-Bum Yoo, Min-Soo Kim, Ji-San Chang, Dong-Kyoo Shin, Dong-il Shin
Department of Computer Engineering, Sejong University

요 약

무선 이동 통신 인구의 증가와 무선 서비스에 대한 수요는 증가하고 있으며 이에 맞추어 무선휴대 장치의 성능도 빠르게 발전을 하고 있다. WAP 포럼에서 제정한 WML(Wireless Markup Language)을 사용한 무선 컨텐츠의 제작이 활발하게 이루어지고 있는 가운데 보다 더 동적인 WML 무선 컨텐츠의 요구가 증가하고 있다. 그로 인해 개발자가 실제와 같은 상황에서 WML과 WML Script 컨텐츠를 편리하게 테스팅 할 수 있는 환경이 필요하게 되었다. 따라서 본 논문에서는 CP(Content Provider)들이 작성된 WML(Wireless Markup Language) 컨텐츠(Content)를 쉽고 편리하게 브라우징하면서 테스팅 하기 위하여 필수적으로 요청되는 WAP(Wireless Application Protocol) 에뮬레이터 및 무선컨텐츠 제작에 필수적인 WMLScripts 해석기의 설계 및 구현에 대해 기술한다.

1. 서 론

WAP(Wireless Application Protocol)[1]을 기반으로 한 무선 인터넷의 급속한 성장으로 인해 휴대용 전화기나 PDA(Personal Digital Assitant)와 같은 무선 단말기를 이용한 다양한 정보 습득이 언제 어디서나 가능하게 되었고, 그로 인해 휴대전화 및 관련 휴대용 무선장비를 이용하는 웹 컨텐츠 이용자의 급속한 증가를 가져왔다. WML(Wireless Markup Language)[2]는 이러한 웹 컨텐츠를 개발하기 위해 만들어졌으며, 휴대용 무선장비에서의 효과적인 웹 접근을 위한 언어인 HDML(Handheld Device Markup Language)[3]을 기반으로 만들어 졌다. HDML의 기본 구조인 데크(Deck)와 카드(Card)단위의 동작 스키마를 채택하여 데크와 카드들 간의 하이퍼링크(hyper link)등 HDML의 작동원리를 그대로 승계하고 있으며, 현재 WML 버전 1.2까지 발표되었다. 이용자에게 유용하고 매력적인 방법으로 진보된 서비스를 제공하면서, 이용자의 요구를 충족시키고 다양한 종류의 컨텐츠 개발을 손쉽게 할 수 있는 각종 어플리케이션이 필요하게 되었다. 이러한 무선 컨텐츠를 제공하는 사이트를 Browsing 하거나 Web 서버 내에 있는 WML 및 WMLScript[4] 문서의 오류를 검증하고 개발자가 그 오류를 다른 도구를 이용하여 수정하기 쉽게 하면서, PC상에서 이동통신 단말기와 동일한 현실감으로 무선 사이트에 접속할 수 있도록 하는 Emulator가 절실하게 요구된다.

본 논문의 구성은 다음과 같다. 2장에서는 User Agent의 전체적인 동작구조와 각각의 핵심모듈에 대해 알아보고 3장에서는 WMLScript 해석기의 전체적인 개요와 설계 및 구현에 대해 살펴보고 4장에서는 WAP 에뮬레이터 인터페이스 부분의 전체적인 동작구조에 대해 살펴보고 5장에서는 Wap 에뮬레이터의 전체적인 구조에 대해 살펴보고 마지막으로 6장에서는 결론 및 향후 과제에 대해서 기술한다.

2. User Agent의 설계 및 구현

2.1 User Agent의 동작 구조

사용자의 요청에 의해 UI가 WAP Stack을 거쳐 WAP 게이트웨이로부터 받아온 데이터는 byte code로 되어 있다. 이것을 디코딩, 파싱, 오브젝트 생성을 통해 브라우저 화면에 그린다.

2.2 주요 모듈

2.2.1 Decoder

WML은 XML(extensible Markup Language)[4]의 subset으로, XML 문서를 해석하려면 DTD(Document Type Definition)와 XML 파서가 필요하다. 본 프로그램에는 Decoder의 내부에 WML

문서를 파싱할 수 있도록 파서를 내장하고 있는데, 파싱속도를 빠르게 하기 위해서 인코드(encode)된 WML문서를 바로 파싱한다. 또한 Decoder에는 Drawer가 좀 더 그리기 쉽도록 파싱한 결과를 토큰 단위로 분리해서 개개의 오브젝트로 만든다. Drawer에서는 Decoder의 작업 결과인 오브젝트를 기반으로 브라우저 상의 화면에 디스플레이하게 된다.

여기서 오브젝트의 단위는 WML 태그의 단위로 생각할 수가 있다. 브라우저 화면상에는 WML의 각 태그의 종류에 따라 브라우저 화면상에 다르게 표현되는데, 이것을 정리하면 몇 가지의 많지 않은 디스플레이 방식으로 나눌 수가 있다. 디스플레이 방식의 분류는 Drawer모듈 설명에서 자세하게 기술할 것이다.

각 오브젝트는 자신의 어트리뷰트의 값, 기본값을 저장하고 어떤 오브젝트를 포함할 수 있는지의 여부의 정보도 저장하게 된다. 예를 들어 select 태그의 child로는 option 태그가 올 수 있고, input 태그에는 child가 될 수 있는 태그가 없다. 여기에서 select 오브젝트는 option 오브젝트를 child멤버로 가지게 되고, input 오브젝트는 child멤버 없이 자신의 어트리뷰트값으로만 구성된 형태로 된다.

파싱을 하다 보면 WML문서 외에 더 필요한 것들이 있다.

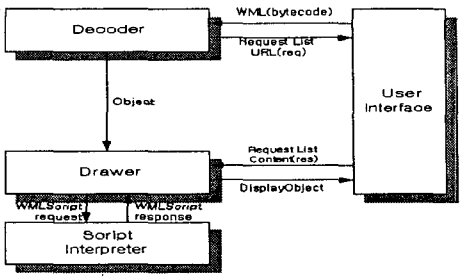
2.2.2 Drawer

Drawer는 Decoder에서 만든 오브젝트를 바탕으로 각 디스플레이 방식에 따라 오브젝트들을 다시 DisplayObject로 만든다. 위에서 언급한 바와 같이 디스플레이 방식은 몇 가지로 분류할 수가 있는데, 단순한 텍스트를 디스플레이하는 Text방식, WBMP같은 단색 이미지를 디스플레이하는 Image 방식, select태그에서 여러 option들 중 선택할 수 있게 되는 Select방식, input태그에서 사용자의 텍스트 입력을 받는 Edit방식, table태그에서 행과 열에 맞춰 선을 그려주고 각 셀에 디스플레이하는 Table방식이 있다.

WML 컨텐츠의 기본 구조는 deck과 card로 되어 있는데, HTML과 같이 하나 하나의 WML 문서 자체를 deck이라 부른다. 한 deck 안에는 여러 card가 있어서 실제 디스플레이는 card 단위로 이루어지게 되어 사용자가 다른 card로 혹은 다른 deck으로 이동할 수도 있다. 그렇기 때문에 deck이 바뀌지 않고 card만 바뀔 경우에는 Decoder에서 만드는 오브젝트는 계속 지속되지만 Drawer에서 만드는 Display Object는 바뀌어야 한다. 다시 설명하면 Decoder의 오브젝트 정보는 deck 단위로 바뀌고, Display Object는 card 단위로 바뀌게 된다. card가 바뀔 때마다 Drawer는 Display Object를 계속 재구성해주어야 한다.

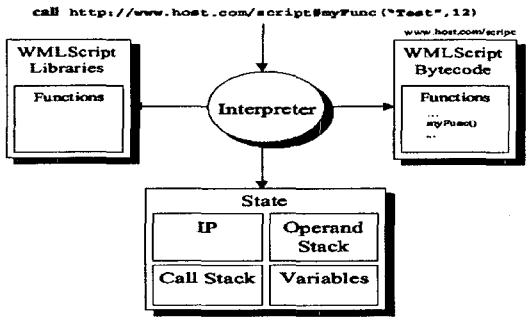
3. WMLScript 해석기의 설계 및 구현

3.1 WMLScript의 개요



[그림 1] User Agent 데이터 전송도

WMLScript interpreter의 구조는 WAP 포럼에서 제시하는 구조를 기반으로 한다. [그림1]은 WAP 포럼에서 제시하는 구조를 개괄적으로 나타내었다.



[그림2] WMLScript interpreter의 구조도

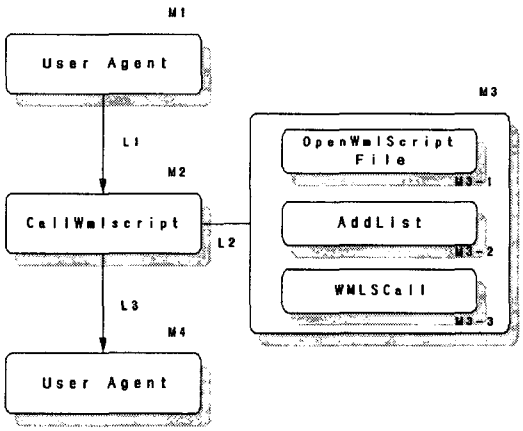
3.2 WMLScript interpreter의 설계 및 구현

WMLSript interpreter에서의 내부 데이터 및 Function의 실행은 [그림2]의 구조를 전체적인 설계 구조로 하였다.

전체적인 구조에서 연산 수행은 다음의 순서를 따른다.

- UserAgent M1이 WMLScript interpreter에게 연산에 필요한 파라미터를 전달한다.
- 파라미터들은 WMLScript interpreter의 함수 호출과 관련되는 전반부 처리기 M2에 전달된다.
- M2로 전달되는 파라미터들은 바이트 코드에 대한 연산을 수행하는 M3로 전달된다.
- M3-1은 전역 메모리 포인터에 M1이 전달하는 파라미터 중 WMLScript 문서 전체와 크기를 저장한다. 여기서 저장을 하는 것은 M2에서 데이터를 분석하여 OperandStack에 저장하기 위함이다.
- M3-2는 Variable을 List 스택에 저장하는 역할을 수행한다. 이 데이터들은 이 후에 연산에서 리스트 목록을 포인터로 전달받음으로써 동작을 처리할 수 있게 한다.
- M3-3은 연산이 수행되는 단계이다.
- M4에서는 다시 UserAgent에게 연산 후의 결과

데이터를 전달한다. 데이터의 전달은 저장되어 있는 포인터의 주소 값을 전달하는 것과 WML문서 정보를 갖고 있는 UserAgent 구조체에 데이터를 갱신 혹은 추가한다.

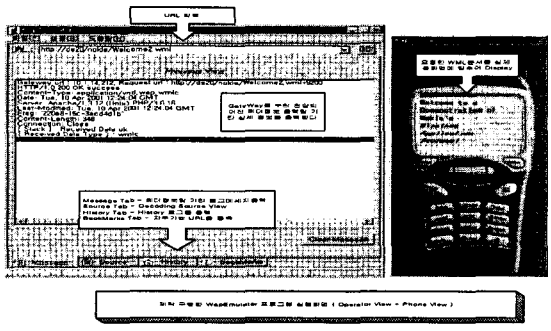


[그림 3] WMLScrpit Interpreter 전체 구조도

4. User Interface의 설계 및 구현

4.1 실행화면

사용자로부터 URL을 입력받을 수 있도록 URL을 입력받는 부분과 메시지를 출력하는 부분을 가지고 있고 또한 WAP Gateway 및 Cache의 사용여부를 설정 할 수 있도록 하는 메인 다이얼로그와 그리고, WML문서에 대한 화면 출력을 볼 수 있는 실제 폰과 같은 폰 이미지를 가지고 있는 폰 다이얼로그로 크게 나누어져있다. 그리고 내부적으로 Cache와 History를 구현 할 수 있도록 하는 부분을 포함하고 있다.



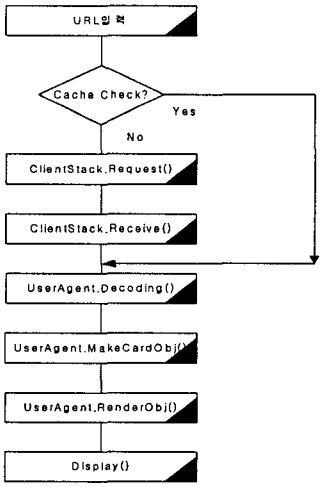
[그림 4] 프로그램 실행화면

4.2 동작 구조

구현한 Emulator는 내부적으로 다음과 같은 구조를

- Client Stack Module
- WMLScript Interpreter Module
- UserAgent Module
- User Interface Module

Client Stack Module과 WMLScript Interpreter Module, UserAgent Module은 User Interface Module에서 함수를 호출할 수 있도록 Library형태로 제작 하였다.
내부적인 함수 호출의 흐름도는 [그림3]과 같다.



[그림 5] 내부 함수 흐름도

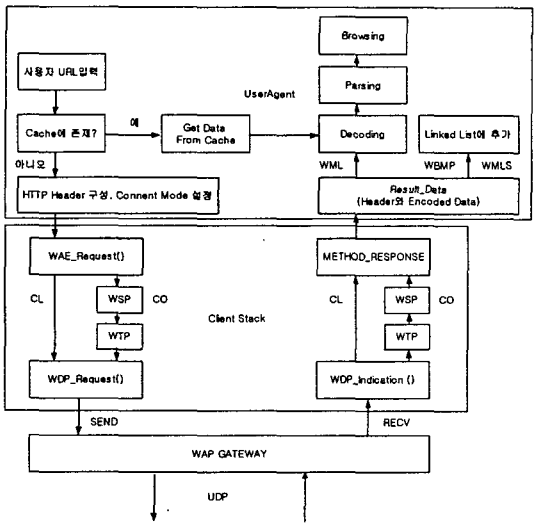
Client Stack으로부터 받아온 Data를 Decoding하고 이를 WML의 구조인 Card구조로 변환을 하여 화면에 Display할 수 있는 구조로 다시 변환을 하여 Phone Dialog에 Display한다.

5. Wap Emulator의 구조

5.1 Wap Emulator의 전체 구조

설계, 구현한 Wap Emulator의 전체 구조 및 동작 방법에 대해 살펴보자. 사용자가 테스트 하고자 하는 URL을 입력하면 그 URL로 Request를 보내기 위해 프로그램에 포함된 Client Stack을 호출하여 Wap Gateway로 UDP 소켓을 생성하여 Request를 보내게 된다. Request를 함에 있어서 Connection Oriente(CO)방식과 ConnectionLess(CL)방식 두 가지가 있는데 해당 방식에 따라 Client Stack의 단계를 거쳐 Wap Gateway로 Request를 보내게 된다. WAP Gateway는 WAP 기반 무선 인터넷을 구성하는 핵심 요소로써 프로토콜변환 및 WML Content의 암호화 및 복호화등의 작업을 담당한다. 이 WAP Gateway에서 해당 URL의 Content를 가져와 다시 Wap Browser로 Response를 주게 된다. Wap Gateway에서 Response가 들어오면 먼저 Client Stack이 받아 UserAgent로 전달한다. UserAgent는 Encoding된 Data를 Decoding작업을 수행하고 이를 화면상에 Display를 하도록 Data를 가공한다. 이때 가장 중요한 작업이 화면상에 Display를 하기 위해 데이터를 가공하는 작업인데 이 작업을 어떻게 하는가에 따라 화면에 Display하기가 수월해 진다.
이 모든 작업이 One Cycle (URL입력 ->ClientStack->

WapGateway->ClientStack->UserAgent->UserInterface) 로 수행되지만 필요한 동작을 수행 하도록 필요한 함수를 호출 하도록 하고 관리를 할 수 있는 커다란 프로그램이 존재하는데 이것이 User Interface부분이다.



[그림 6] Wap Browser의 전체 구조도

5.2 구현환경

본 WAP 브라우저는 Microsoft Visual C++ 6.0에서 구현되었다. Windows 2000상에서 테스트되었으며, WAP 브라우저와 통신한 WAP 게이트웨이는 Unix 상에서 구현되었다.

6. 결론

본 논문에서는 WML 컨텐츠 개발 과정에서 실제 휴대폰과 동일한 환경과 현실감을 제공해 개발자가 쉽게 WML 및 WMLScript 문서의 오류를 수정할 수 있는 Wap Emulator에서 User Interface부분에 대한 설계 및 구현에 대해 기술하였으며, 또한 WAP 브라우저의 파싱과 디스플레이 기능을 가진 User Agent의 설계와 구현에 대해서도 살펴보았다. 향후 위와 같은 주제들에 대한 연구 및 개발이 이루어져야 한다.

참고 문헌

[1]WAP, "Wireless Application Protocol Architecture Specification" , WAP Forum, http://www.wapforum.org/what/technical.htm

[2]WML, "Wireless Markup Language Specification" ,WAP Forum, http://www.wapforum.org/what/technical.htm

[3]Handheld Device Markup Language Specification, http://www.w3.org/TR/NOTE-Submission-HDML-spec.html

[4]WMLScript, "WMLScript Language Specification, WAP Forum, http://www.wapforum.org/what/technical