효율적인 무선 웹 서비스를 위한 통합 브라우저 연구

박홍성, 전성염
강원대학교 제어계측공학과

Study on Integrated Browser for Efficient Wireless Web Service

HongSeong Park, SeongYum Jeon
Dept. of Control and Instrumentation Engineering, wangwon Univ.
E-mail : hspark@control.kangwon.ac.kr , jsalt@control.kangwon.ac.kr

요 약

본 논문에서는 일반적인 HTML 브라우저의 구조 및 WML 브라우저의 구조를 연구하고, 각 브라우저와 다루는 마크업 언어의 차이점들을 고려하여 서로 다른 마크업 언어를 모두 처리할 수 있는 통합 브라우저의 구조를 제안하고, 설계시 고려할 부분 및 각 브라우저의 모듈을 공통으로 사용할 수 있는 구조를 설계하여, 이를 무선 인터넷 서비스 시스템에 적용한다.

1. 서론

현재 무선 인터넷에서는 다양한 포맷의 컨텐츠와 이를 전송하기 위한 다양한 프로토콜들이 아직 정형화 되지 않은 상태에서 무선상의 컨텐츠 서비스를 지원하고 있다. WAP에서 제안된 마크업 언어인 WML은 게이트웨이에서 인코딩 되어 무선구간에서 축소된 컨텐츠 전송을 가능하게 하며 또한 무선서비스에 적합한 통신을 지원한다. HTML은 무선상의 서비스에 적합한 컨텐츠 포맷을 지원하지는 못하지만 그 보급면에서 무선상에서의 지원을 필요로 한다.. 또한 무선 인터넷 프로토콜 표준으로 자리잡은 WAP은 무선상의 서비스에 최적화되어 과다한 오버헤드와 응답을 축소하였으며, HTTP는 현재까지 유선상의 다양한 서비스에서 잘 활용되고 있으나, 이를 무선상에 적용하는데 있어서는 과다한 오버헤드 등의 다양한 문제가 존재한다. 하지만 유선상의 컨텐츠 서버에 직접적인 연결을 하거나 유선의 프로토콜과의 보다 원활한 호환을 위해 HTTP 프로토콜의 지원을 필요로 한다.

현재 무선상에서 지원되는 컨텐츠들은 단일한 규격의 컨텐츠만을 지원하는 Agent로 인해 Web 정보에 있어 유무선 간의 실질적인 공유가 어려운 상황이다.

유무선간 정보의 공유를 위해 클라이언트는 컨텐츠의 종류와 상관없이 혹은 어떤 서버가 어떤 프로토콜을 지원하는가에 상관없이 사용자에게 정보를 보여줄 수 있어야 한다.

현재는 Contents Provider가 같은 내용의 컨텐츠를 다른 마크업 언어로 다시 제작하는 방식에 의존하거나 혹은 gateway에서 클라이언트 Agent가 정보를 처리할 수 있는 형식으로 컨버젼하는 등의 다양한 방법이 적용되고 있다.. 그러나 gateway에 이러한 기능들을 위탁할 경우 만일 gateway에서 적절한 컨버젼을 지원하지 못한다면 사용자는 이 정보를 적절하게 활

용하지 못할 것이다. 그리고 각 마크업 랭귀지의 특성의 차이를 해결해도 Contents provider 의 의도대로의 완전한 컨버젼은 사실상 불가능한 것으로 보인다. 또 이런 컨버젼이 반드시 필요하지 않다면 경우에 따라 Web Sever와 HTTP 프로토콜을 사용하여 직접 연결할 수도 있어야 할 것이다. HTTP 자체를 현재 무선상에서 그대로 사용하는 데는 문제가 있다. 그러나 현재 추세를 고려하여 볼 때 무선상 서비스에 적합하도록 대부분의 브라우저에서 수정된 HTTP를 사용하게 될 것이다. 본 논문에서는 WTP/WSP 프로토콜 스택과 TCP/HTTP를 상황에 따라 능동적으로 사용하는 Dual stack을 제안할 것이다. 하지만 프리젠테이션 계층에서 사용할 Markup language에 좀더 초점을 두고자 한다. 즉 통합 브라우저가 처리해야 할 컨텐츠인 Encoded WML, Not Encoded WML, HTML등을 주로 다룰 것이며, WAP 과 HTTP 기반에서 WML 과 HTML contents를 함께 지원할 수 있는 브라우저의 구조를 제안할 것이다.

2. 본 론

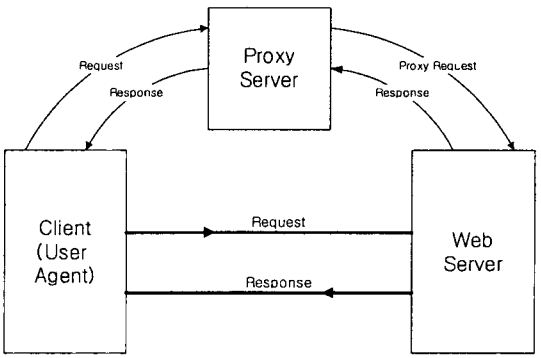
2.1 웹 브라우저의 일반적인 기능 및 목적

본 웹 브라우저는 서버에 접속하여 문서를 요구하고 서버 응답 결과를 처리하여 이를 사용자에게 보여주고 처리를 기다리는 것이 일반적인 기능이며, 웹상에 존재하는 문서나 여러 종류의 적합한 형식으로 제공된 컨텐츠를 사용자가 열어보거나 정보를 획득하는데 그 목적이 있다고 할 수 있다.

2.2 현재 무선 웹 서비스의 구성 요소들

웹 서비스 모델은 일반적으로 크게 정보를 요구하는 클라이언트와 정보를 보유하고 제공하는 웹서버, 그리고 이들 두 peer 사이에서 프로토콜 변환이나 혹은 과금, 캐싱, 방화벽 등의 부가적인 기능을 수행할 때 사용하는 proxy server등으로 구성된다. 이들 사이에 사용되는 프로토콜은 주로 HTTP가 사용되며 특별히 무선에서는 게이트웨이를 사용하는 WAP을 적용하기도 한다. HTTP와 연계되어 주로 사용되어온 마크업 언어는 HTML계열과 WAP에서 지원하는 WML 계열이 있으며, 이 두 언어 사이에는 여러가지 차이가 있다. 근래에 HTML을 무선상에 적용하기 위해 비표준으로 여러 가지 형태의 HTML의 Subset들을 새로이 만들기도 했으나, 현재는 XML 양식을 따르는 XHTML로 가고 있다. WAP의 WML 같은 경우에는 표준이지만 이미 웹을 장악하고 있는 HTML과는 많이 다른 것이 문제점으로 작용하고 있다. 이 문제를 해결하려는 시도로서 WAP Gateway가 HTML에서 WML로 컨버트를 시도하지만 완벽한 변환은 잘 되지 않고 있는 실정이다. 이는 각 언어가 표현하는 정보의 종류와 엘리먼트에서 정의한 동작이 서로 다른 것이 많고, 또 컨텐츠 프로바이더의 의도를 완벽하게 살리는데에는 한계가 존재할 수 밖에 없기 때문이다. 이 문제를 해결하기 위해 WAP2.0에서는 마크업 언어로 XHTML을 선정하여 일반 웹으로의 접근이 수월할 수 있도록 하는 시도를 진행 중이다.

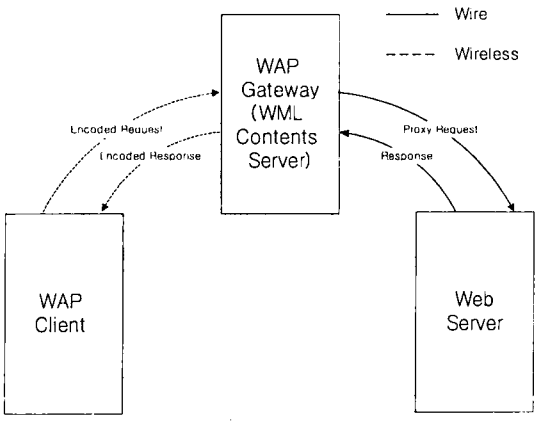
이제 HTTP 를 사용하는 웹 서비스의 프레임 워크와 WAP 프로토콜의 기본 서비스 모델을 살펴보고 그 차이점을 살펴보겠다. [그림 1]은 HTTP를 사용하는 일반적인 웹서비스의 기본 프레임워크이다.



[그림 1] 일반적인 웹 서비스 시스템(Using HTTP)

이런 경우 클라이언트와 프록시 및 웹서버는 모두 같은 HTTP 프로토콜을 사용하므로 같은 HTTP를 사용하는 각 Peer 간에 별다른 처리 없이 통신이 가능하며 이러한 서비스 모델에서 프록시 서버 혹은

게이트웨이는 선택적인 사항이 될 수도 있다.
[그림 2]는 일반적인 WAP 서비스의 개략적인 프레임 워크이다.



[그림 2] 일반적인 WAP 서비스 시스템(Using WAP)

이 시스템에서 각 peer 들의 프로토콜 스택은 다소 차이가 있다. 즉 WAP만을 지원하는 클라이언트와 HTTP만을 지원하는 웹서버, 그리고 이들 사이에서 프로토콜과 컨텐츠에 변화를 주며 중계역할을 하는 WAP Gateway로 구성된다.
이중 WAP Gateway는 각 peer 에 적합한 프로토콜로 변환을 수행하는 중계자 역할을 하며, 바로 WML 컨텐츠 서비스를 지원하는 컨텐츠 서버로서의 역할을 수행하기도 한다.

2.3 현재 무선 웹 서비스 시스템의 문제들.

클라이언트와 WAP Gateway 사이는 무선 구간이며, 여기에 유선에서 사용되는 TCP 기반의 HTTP를 그대로 사용하려 했을 때 TCP의 특성에서 오는 Congestion 문제와 HTTP의 과다한 오버헤드 및 클라이언트의 메모리나 작은 화면 등의 여러 제약에 대한 문제가 발생하게 되었다. 이에 대한 해결을 위해 무선 구간에서 WAP protocol 사용이 대두되었다.
그러나 무선 구간에서만 적용되는 이 WAP 프로토콜의 사용은 또 다른 문제를 불러왔다. 유선의 Web Server에서는 HTTP만을 지원하므로 WAP을 지원하지 않는 Web서버와 HTTP를 지원하지 않는 클라이언트(WAP User Agent)는 직접적인 통신이 불가능하여 필연적으로 Gateway를 통한 프로토콜 변환을 해야 했고, WML 브라우저의 컨텐츠 지원의 한계로 인해 컨텐츠를 HTML에서 WML로 변환하는 기능들도 필요하게 되었던 것이다.

그러나 현재 Web Server에서 WAP 프로토콜을 지원하는 경우는 거의 없으며, WAP 클라이언트가 브라우징 할 수 있는 컨텐츠 포맷인 WML을 전송하는 것이 필요하나 이러한 서비스를 제공하는 컨텐츠 프로바이더 역시 많지 않다.

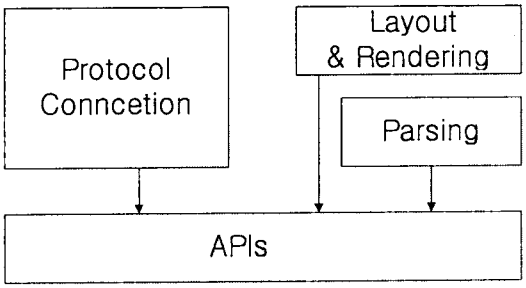
현재 HTML의 정보량은 WML과는 비교할 수 없을만큼 다양하고 많은 상태이다. 이것을 WML Browser에서 보기 위해서는 컨텐츠 프로바이더가 HTML 컨텐츠를 WML로 새로이 작성하거나 혹은 Gateway에서 프로토콜의 처리 이외에도 HTML을 WML로 변환해 주어야만 이를 브라우징 할 수 있다. 이러한 문제를 해결하기 위해서 많은 컨버팅 기술들이 나왔으나 모두 컨텐츠 프로바이더와 게이트웨이의 부하를 가중시키고 있으며, 또 변환된 컨텐츠마저 컨텐츠 프로바이더의 의도에 맞는 형태로 변환되기란 상당히 어려웠다. 결국 게이트웨이에서의 문제 해결은 이에 대한 궁극적인 해결안으로 보이지는 않는다.

2.4 브라우저 구성 및 기능

본 논문에서 설계한 통합 브라우저는 클라이언트에서의 문제 해결을 시도한 것이며 다음 세가지 부분에 초점을 맞춘다. 첫번째는 HTTP 프로토콜과 WAP 프로토콜의 Dual Stack을 가진 하나의 User Agent가 원활하게 동작하기 위한 설계 구조이며, 두번째는 HTML과 Not Encoded WML , Encoded WML, WML Script, Image 를 적절히 지원하게 하는 구조이고, 세번째는 이들이 올라가는 단말기의 환경을 고려해야 하므로 프로그램 사이즈를 줄이기 위 각 컨텐츠 처리시 공통 사용 가능한 모듈들을 활용하는 것이다.
[그림 3]은 웹브라우저가 가질 수 있는 일반적인 모

듈을 나타낸 것이다.

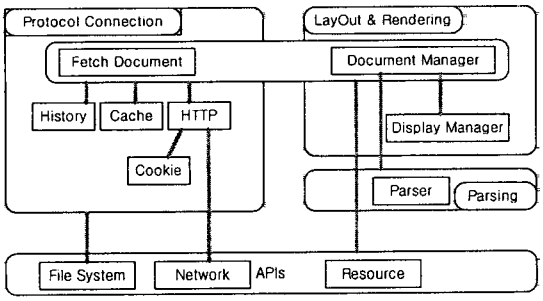
웹 브라우저는 크게 프로토콜을 처리하는 부분과, 문서 저장 및 파싱하는 부분과, 또 문서의 내용을 화면에 뿌려주고 사용자의 입력을 받아 동작을 하게 하는 부분으로 구성된다.



[그림 3] 일반 웹브라우저 구성

2.5. 일반 HTTP Browser 구조 및 모듈별 기능

[그림 4]는 HTTP 웹 브라우저의 구조이며 중요 기능만을 모듈로 표현했다. 다음 그림은 브라우저 기능에 대한 개념도로 실제 구현에 있어 각 모듈사이의 관계나 기능은 약간의 차이가 있을 수 있다.

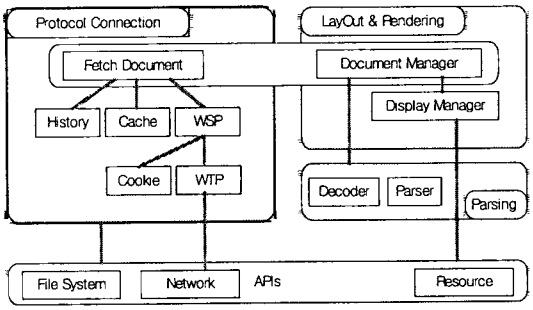


[그림 4] 일반 HTML 웹 브라우저 구조

각 모듈별 기능을 보면 먼저 Fetch Document는 서버에게 문서를 요구하여 그 응답을 수신하고 수신된 컨텐츠나 혹은 그 결과를 적절한 포맷으로 구조화 한 뒤 Document Manager에게 전달하는 기능을 한다. History 모듈은 접속한 문서의 URL을 히스토리 스택에 저장하는 기능을 하며, Cache 모듈은 서버로부터 수신한 문서를 HTTP 헤더를 참고하여 파일로 저장하는 기능과 이를 캐시 테이블로 관리하는 기능을 한다. HTTP 모듈은 HTTP 프로토콜과 관련된 요구헤더를 만들거나 혹은 응답되어온 HTTP헤더를 해석하는 기능을 담당한다. 이 과정에서 Cookie를 통해 쿠키 기능과 관련된 처리를 병행하게 된다.
이러한 처리를 다 마치고 나면 Document Manager에게 응답된 문서를 넘겨 주고, Document Manager는 먼저 문서를 Parsing 모듈로 넘겨주어 그 파싱된 결과를 다시 Document Manager를 통해 Display Manager에게 전달하여 화면에 출력하게 된다.

2.5. 일반 WAP Browser 구조 및 모듈별 기능

[그림 5]는 일반 WAP Browser의 구조이며 중요 기능만을 모듈로 표현 했다. 다음 그림은 브라우저 기능에 대한 개념도로 실제 구현에 있어 각 모듈사이의 관계는 약간의 차이가 있을 수 있다.
[그림 5]를 살펴보면 HTML 브라우저[그림 4]의 모듈과 같은 부분이 비교적 많은 것을 볼 수 있다.
명칭이 같은 모듈들은 그 기능에서 차이가 없지만 데이터 구조나 다루는 자료형이 다를 수 있으므로 이러한 부분들은 공통으로 사용하려 할 때 충분히 수정될 수 있다. HTML의 모듈과 같은 모듈은 설명을 생략한다.



[그림 5] 일반 WAP 브라우저 구조

먼저 Request 할 때 WSP 모듈은 WSP 헤더를 인코드하여 송신하거나 혹은 수신하여 디코드하며 이 과정 중에 WAP/WSP 스펙에 정의되어진 동작을 수행한다. WTP는 이때 상대방의 수신의 신뢰성을 보

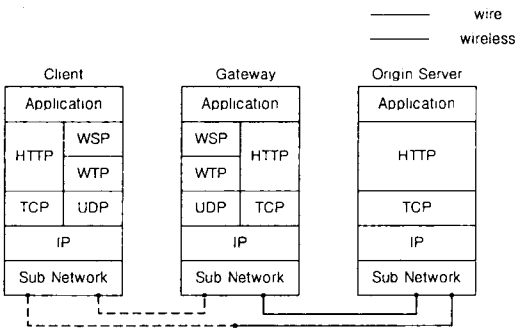
장하는 기능을 수행한다. 일단 수신된 문서가 WML인 경우 Document Manager에게 넘겨질 때 두가지 형식중 하나를 취한다. 하나는 게이트웨이에 의해 인코딩된 문서이고, 다른 하나는 인코딩 되지 않은 문서이다. Encoded WML인 경우 Document Manager는 Parser가 아닌 Decoder 모듈로 문서를 넘겨주어 처리한 후 그 결과를 Display Manager로 다시 넘겨주어 화면에 출력한다. Non Encoded WML인 경우는 Parser에서 처리되고, 그 이후로는 Encoded WML과 동일한 과정을 거친다.

2.6. 통합 브라우저가 지향하는 통신 프레임 워크

[그림 6]은 통합브라우저가 지향하는 통신 프레임워크이다. 여기서 클라이언트는 Dual Protocol Stack 을 가지게 되며, 만일 클라이언트가 WAP을 사용하여 통신할 경우에는 일반 WAP 브라우저와 동일한 동작을 한다.

또한 TCP나 HTTP의 무선 적용의 문제에도 불구하고 웹서버와의 직접 연결이 가능해지며, 이때는 게이트웨이를 거치지 않을 수도 있다.

WAP은 UDP포트를 사용하고, HTTP는 TCP 포트를 사용하므로 상위 계층에서 통신에 관련된 충돌은 생기지 않는다.

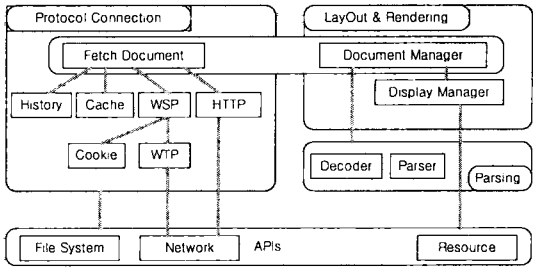


[그림 6] 통합 브라우저 통신 프레임워크

WAP으로 통신을 할 때에는 게이트웨이가 설정된 상태에서 동작해야 하며, 사용자의 설정에 따라 즉각적으로 HTTP 및 WAP 프로토콜이 적용될 수 있어야 한다. 이것은 사용자가 User Interface를 통해서 설정할 수 있다. 그리고 이 설정은 바로 통신에 적용되어야 한다. 만일 WAP 게이트웨이의 문제로 인해 연결이 성립 되지 않을 경우에는 HTTP를 통한 웹서버와의 직접 연결도 가능하다.

2.7. 통합 브라우저 구조

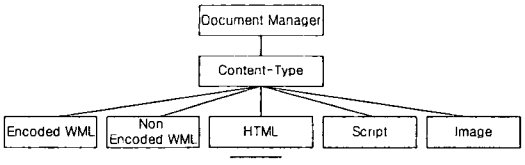
[그림 7]은 통합 브라우저의 구조이다.



[그림 7] 통합 브라우저 구조

우선 HTTP 모듈과 WSP/WTP 모듈을 모두 지니게 되며, Document Manager는 여기서 사용된 프로토콜 헤더의 미디어 타입(Content-Type)을 이용하여 수신된 문서의 종류를 구별 할 수 있다.

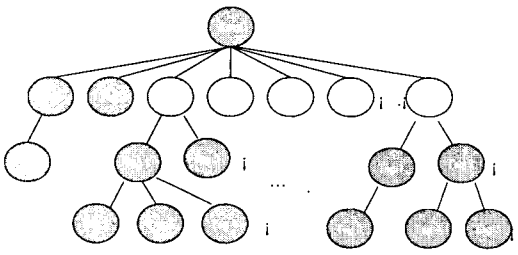
[그림 8]은 Document Manager를 보다 세부적으로 살펴 본 것이다. Document Manager 는 Fetch Document 로부터 HTTP 헤더 중 Entity Header로 넘겨받거나 혹은 WSP 헤더 중 디코드된 Entity Header를 넘겨받아 그 중 미디어 타입 정보인 Content-Type 헤더 필드를 이용하여 문서의 종류를 알아낸다.



[그림 8] Document Manager 내부 구조

문서의 종류는 크게 Encoded WML, Non Encoded WML, HTML, Script, Image 등으로 나눌 수 있으며, 이들 컨텐츠의 종류에 따른 적합한 동작을 위해 필요한 모듈을 실행한다. 수신된 문서가

Non Encoded WML 이나 HTML인 경우에는 Parser를 거치게 되고 Encoded WML 은 Decoder을 거치며 처리가 되는데 이 처리의 결과는 모두 [그림 9]와 같은 다중 트리 구조를 만들어 낸다. 수신된 문서가 WML이나 혹은 HTML인 경우 Display Manager는 이 트리를 이용하여 출력할 정보와 고유 동작 및 관련된 이벤트 정보를 추출하고 이를 위한 적절한 자료형을 만들어 출력 및 처리하게 된다.



[그림 9] Decoder와 Parser의 Output 자료 구조

결국 HTTP나 혹은 WSP 헤더의 미디어 타입을 결정하는 Content-Type은 수신된 문서가 Decoder를 거쳐야 할 지 Parser를 거쳐야 할지 혹은 이외에 Script의 동작을 위해 인터프리터를 거쳐야 할지, 혹은 이미지를 처리하기 위해 테이블로 관리하며 저장해야 할지를 결정하는 중요한 정보가 되며, 이 헤더 필드는 스크립트나 혹은 이미지를 수신할 때에도 적합한 모듈로 분기하는데 중요한 정보가 된다.

2.8. 통합 브라우저의 공통 사용 가능한 모듈들

브라우저를 통합할 때에는 공통 사용 가능한 모듈을 고려하는 것이 좋다.

URL만을 저장하고 관리하는 History 모듈과, 캐시 테이블 및 파일 저장 기능을 가진 Cache 모듈, 태그의 문법 처리와 의미분석 후 각 엘리먼트들의 관계를 표현한 트리 구조를 만드는 Parser 모듈을 공통 사용 할 수 있다. Cookie 모듈은 HTTP나 혹은 WSP의 헤더의 데이터 형태의 차이에서 오는 영향을 받으므로 추가 및 수정을 통해 공통 사용이 가능하며, Display Manager 모듈에서는 HTML 및 WML 인 경우가 확실히 분리되어 처리되지만 테이블 처리 모듈이나, 텍스트 정렬, 컨트롤 및 이미지 처리 부분의 모듈은 부분적인 공통 사용이 가능하므로 이를 고려하여야 할 것이다.

3. 결 론

본 논문에서는 일반적인 HTML 브라우저의 구조 및 WML 브라우저의 구조를 연구하여, 서로 다른 마크업 언어를 모두 처리할 수 있도록 각 브라우저의 통합을 제시하였으며, 그 결과 두개의 브라우저의 사이즈를 합한 것에 비해 통합 브라우저의 실행 파일 사이즈가 약 30%정도 축소되는 잇점도 있음을 볼 수 있었다.

본 연구를 통해 서로 다른 마크업 언어를 모두 처리하는 브라우저를 제작하고 그 구현을 통해 현재 무선 웹 서비스의 효율성을 증대시켜 정보 서비스를 받을 때의 문제를 클라이언트 측에서 해결하려는 시도를 하였으며, 이를 실현하는데 있어서 새로운 기술의 부담없이 공통 모듈을 이용하는 설계가 가능함을 보였다.

웹 서비스의 문제 해결을 위해서는 현 무선서비스 시장을 가로막고 있는 폐쇄적인 시스템을 개방할 필요가 있다. 따라서 컨텐츠를 제작하는 마크업 언어의 통일이 현 시점에서 가장 필수적인 것이다. 그러나 이러한 시스템이 실현 된다고 해도 기존 컨텐츠와의 호환을 유지하는 방안이 필요하며 이에 대한 해결안으로 통합 브라우저의 구조를 제시한다.

[참고문헌]

[1] WAP forum, WAP White Paper, 1999.6

[2] WAP forum, Wireless Application Protocol: Architecture Specification, 1998.4

[3] WAP forum, Wireless Application Protocol: WAP Wireless Markup Language version 1.2

[4] RFC2068 HTTP/1.1" 2.1997

[5] A study of three browser history mechanisms for Web navigation *Nadeem, T.; Killam, B.* Information Visualisation, 2001.