저사양 휴대 단말 환경에서 웹브라우저의 성능 개선 방안

김상헌*, 김지인*, 고석주**
* 경북대학교 대학원 전자전기컴퓨터학부, ** 경북대학교 IT 대학 컴퓨터학부
e-mail : sheon.kim7@gmail.com

Performance Improvement of Web Browsers for Mobile Phones

Sang-Heon Kim*, Ji-In Kim*, Seok-Joo Koh**
*School of Electrical Engineering and Computer Science, Kyungpook National University
**College of IT Engineering, Kyungpook National University

요 약

최근 무선 인터넷 서비스의 활성화와 함께 모바일 환경에서 브라우저의 사용이 급증하고 있으며, 모바일 사용자들은 PC 수준의 품질을 요구하고 있다. 이에 따라 모바일 웹브라우저도 WAP 방식이 아닌 풀브라우징(full browsing) 방식으로 전환되고 있다. 하지만, 모바일 단말 환경에서 웹브라우저의 성능에는 많은 제약사항이 존재하며, 부족한 메모리, 저 사양의 CPU, 낮은 네트워크 속도, 그리고 브라우저의 엔진 문제 등으로 인해 여전히 사용자들의 브라우저 로딩에 대한 체감 속도는 낮은 편이다. 본 논문에서는 저사양 휴대 단말 환경에서의 브라우저 로딩 속도를 개선할 수 있는 방안을 제시한다. 제안 방식에서는 텍스트와 이미지 등 데이터 타입을 분류하여 부하가 적게 걸리는 텍스트 레이아웃을 먼저 보여줌으로써 사용자의 체감속도를 향상시키고, 아울러 이미지가 커서 렌더링(rendering) 시간이 오래 걸리는 경우 이미지를 축소하거나 화질을 낮추는 방식으로 렌더링 부하를 줄여서 페이지 로딩 시간을 단축시키는 방법을 사용한다. 실험 결과, 제안 기법을 사용하는 경우 현재 사용하는 방법에 비해 이미지가 적은 Web 페이지의 경우 1st drawing 77.04%, full drawing 5.47%, 이미지가 많은 페이지의 경우 26.32%의 로딩 시간을 단축시킬 수 있음을 확인하였다

1. 서론

모바일 환경에서 브라우저의 사용이 많아지게 되면서 모바일 단말기에 탑재되는 브라우저의 성능 또한 중요해지고 있다. 사용자들은 PC 처럼 '풀브라우징' 성능을 갖는 브라우저를 요구하게 되었으며 모바일 환경에서 브라우저는 요구되는 성능을 제공하기 위해 많은 진화를 거듭하고 있다.

저사양 무선 휴대단말기에서 일반적으로 사용하는 브라우저는 Access 社의 Netfront, Openwave, Opera, Nokia web, Safari, Obigo 브라우저 등을 들 수 있다. 각각의 독립적인 렌더링(rendering) 엔진을 가지고 있으며, 특화된 기능을 지원하고 있다. 저사양 피쳐폰의 모바일 환경은 제한된 메모리(1Gx512MB, 512x256MB 등), 작은 크기의 LCD Resolution(128x160, 240x400 등), 낮은 클럭을 갖는 ARM7, ARM9, ARM11 의 CPU, 제한된 성능을 갖는 모바일용 브라우저 등 그 특수성 때문에 WEB 대신 규격을 축소한 WAP 규격을 이용해 왔다. 점차 휴대폰 기능이 발전하고, 사용자들의 요구가 늘어나고 있으며 WAP 환경도 진화되어 현재는 WAP2.0 을 지원하며, HTTP 프로토콜을 함께 사용할 수 있게 되었다. 현재 모바일 환경에서는 WAP 브라우징이 아닌 웹브라우징 능력을 갖춘 브라우저가 요구되고 있다.

하지만, 저사양 모바일 환경에서 브라우저는 웹 페이지를 로딩할 때 제한된 메모리, 낮은 속도의 CPU 등으로 인해 많은 문제가 발생하게 되었으며 그 중 브라우저 속도에 대해 가장 큰 이슈가 현재 부각되고 있다. 이는 저사양 모바일 브라우저의 활성화를 가로막는 요인이 되고 있다. 이를 해결하기 위해 다양한 연구가 진행되고 있으며 본 논문에서는 저사양 브라우저의 로딩 알고리즘을 변환하여 로딩 속도를 개선할 수 있는 방안을 제시 하고자 한다. 제안 방식에서는 텍스트와 이미지 등 데이터 타입을 분류하여 부하가 적게 걸리는 텍스트 레이아웃을 먼저 보여줌으로써 사용자의 체감속도를 향상시키고, 아울러 이미지가 커서 렌더링 시간이 오래 걸리는 경우 사업자의 Proxy 를 사용하는 실제 망인경우 저사양 User Agent(이하 UA)로 변경하여 사업자 서버로부터 작은 이미지를 받고, 일반 사이트인 경우 이미지를 축소하거나 화질을 낮추는 방식으로 렌더링 부하를 줄여서 페이지 로딩 시간을 단축시키는 방법을 사용한다

2. 관련연구

최근에는 웹 브라우징 성능을 측정하기 위하여 웹 브라우저의 개발을 위한 연구[1], 브라우저 기반의 네트워크 시스템의 개선[2], 브라우저의 모바일 기기의 웹브라우징 성능 요인 분석[3], 웹 브라우저의 성능향상을 위해 브라우저를 병렬화 하는 것에 대한 연구[4],

웹 사이트의 종류에 따른 웹 페이지 구성요소에 대한 분류를 수행한 연구들이 진행되어 왔다. 일반적인 브라우저의 성능에 대해 네트워크적인 연구 개선 효과와 브라우저의 엔진 성능에 대한 연구가 병행되어 왔으나, 실제 저사양 단말의 제한적인 환경에서 영향을 주는 요소에 대한 연구는 부족한 실정이다. 또한 실제 사업자 망에서 네트워크 성능에 영향을 받으며 테스트 하기란 현실적으로 많은 어려움이 있다.

3. 브라우저 로딩 속도 고속화 방안

본 논문에서는 브라우저 로딩 시 발생하는 로딩 지연을 개선할 수 있는 방안을 제시하고자 한다. 구글과 같은 작은 크기의 페이지를 로딩할 경우 저사양 휴대 단말에서는 페이지 로딩에 크게 문제가 되지 않았으나, Web 페이지인 큰 사이즈의 www.go.com 와 같이 CSS, 자바 스크립트, 이미지 등이 복합적으로 구성되어 있는 경우 단말의 브라우저는 페이지를 로딩하는데 많은 부하가 걸리며, 로딩시간이 많이 소요된다. 이때 브라우저는 부하로 인해 폰의 동작이 심하게 느려지면서 입력이 안 되는 현상이 발생하기도 하고 심지어 단말에서는 부족한 메모리로 인해 사용자에게 "Not enough memory"라는 팝업창을 보여주고 브라우저 로딩을 강제로 중지하기도 한다.

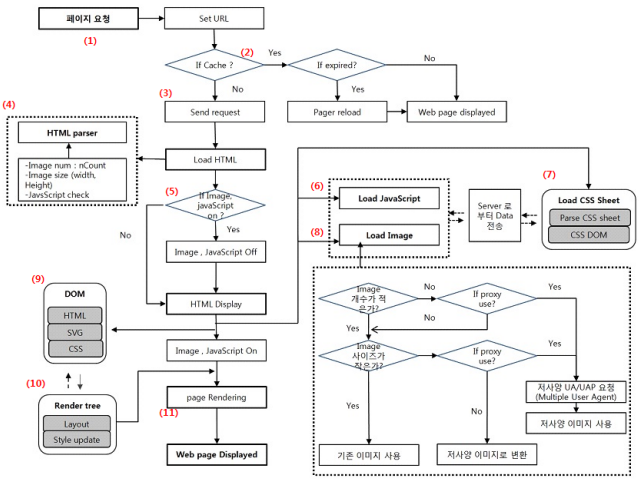
또한 특정 사업자(Vodafone, Orange, T-Mobile, 허치슨 등)에서는 브라우저의 로딩 시간에 있어 특정 페이지를 로드 하는데 최소한의 제한 시간을 두고 사업자의 요구사항에 맞춰줄 것을 요구하기도 하지만, 이는 제한적인 하드웨어 사양과 네트워크로 인해 요구사항을 맞추기가 어려운 것이 현실이다.

속도에 영향을 미치는 여러 가지 모듈 중 이미지는 많은 부분 로딩 시간에 영향을 준다. 이미지의 사이즈가 큰 경우에 데이터의 로딩이 완료되고 화면에 그리기 위해서는 많은 용량의 메모리가 필요하다. 사용가능한 메모리만큼 화면에 표시하고 기다렸다가 다시 그만큼을 그리는 반복동작을 해야 한다. 예를 들어 1100 x 759 사이즈의 WBMP 파일을 디코딩할 경우 Temp Buffer 로 브라우저에 할당된 메모리가 사용된다. 브라우저에서 점유하고 있는 메모리를 제외한 상태에서 해당 이미지 디코딩 시에 필요한 가용 메모리를 분석해 사용한다.

기존에는 큰 사이즈의 이미지 데이터로 인해 패킷 전송 시 트래픽에 부하가 걸려 브라우저 로딩 시간에 영향을 주는 문제가 있다. WI-FI 나 네트워크 상태가 좋은 경우에 대해서는 큰 문제가 되지 않으나 네트워크 상태가 좋지 않을 경우 성능에 아주 큰 영향을 주게 된다. 실제 망에서는 네트워크 상태가 좋지 않은 경우가 빈번하게 발생한다.

본 논문에서는 웹 브라우저의 로딩 속도 개선을 위해 두 가지 방법을 제안한다. 첫째, 사용자에게 HTML 로 구성된 텍스트 레이아웃을 먼저 보여줌으로써 사용자가 느끼는 체감 속도를 향상시키고, 화면 렌더링은 화면이 drawing 된 후 재 구성한다. 둘째, 이미지가 커서 렌더링 시간이 오래 걸리는 경우 이미지를 축소하거나 화질을 낮추는 방식으로 렌더링 부하를 줄여서 페이지 로딩 시간을 단축시키는 방법을 제시한다. 사업자 사이트의 경우 이미지의 크기를 판단하여 저사양 단말의 User Agent 를 보냄으로써 사업자 서버로부터 작은 이미지를 전송 받아 부하를 줄이고, 일반 Web 페이지의 경우 이미지가 일정개수 이상 큰 페이지가 있을 경우, 텍스트는 그대로 전송 받고 부하가 많이 걸리는 이미지는 저사양 이미지로 변경하여 큰 페이지 로딩 시에 시간을 단축하여 브라우저 로딩 시간을 개선하고자 한다. 이미지 개수가 지정한 개수 보다 적어 이미 원본 데이터를 전송 받았는데 이미지 크기가 커서 페이지 렌더링에 시간이 많이 걸리는 경우 이미지를 축소하거나 이미지의 질을 낮추어 화면 렌더링시 부하를 줄여 페이지 로딩시간을 단축하여 속도 개선 효과를 얻는다

그림 1 은 제안하는 렌더링 방법 및 이미지 변환 과정을 보여준다.



(그림 1) 제안된 브라우저 로딩 알고리즘 순서도

(1) 일반적으로 웹페이지의 경우 HTML, CSS, 자바스크립트, 이미지 등으로 페이지가 구성된다. 단말은 HTTP 의 GET/POST method 를 통해 서버에 데이터를 요청하게 되고 서버는 최초 데이터로 일반적으로 HTML 로 구성된 테스트를 전송하게 된다.

(2) 페이지 로딩 시 Header 의 정보와 단말에 저장된 Cache 를 통해 페이지 유무를 파악한다. 이는 불필요한 패킷 데이터 전송을 방지하고 사용자로 하여금 빠른 화면을 보여주기 위한 것이다. 만약 Cache 에 페이지 정보가 존재하고 expired 되지 않았다면 화면에 페이지를 출력한다. HTTP Header 에서 Last Modified Time 과 Max-age 의 값을 구분하여 페이지가 expired 되었는지 판단하여 페이지 reload 유무를 판단한다.

(3) Cache 에 페이지가 존재하지 않는다면 Server 로부터 HTML page 의 정보를 요청한다.

(4) 서버로부터 전송된 문서는 HTML 파일 형식이며, HTML 파일에 대한 토큰화(tokenize) 및 파싱을 수행한다. HTML 의 페이지에는 레이아웃 정보와 페이지내의 CSS, 자바스크립트 유무, 이미지 크

기, 이미지 개수 등의 정보가 포함된다. 지정한 개수 이상과 지정한 크기 이상이 될 경우 이미지의 크기 및 개수 정보는 별도의 DB 에 저장하여 이미지 데이터 요청 시 활용한다.

(5) 자바스크립트와 이미지의 플래그를 파악하여 우선적으로 HTML 페이지를 렌더링하여 보여줌으로써 사용자의 체감 속도를 높일 수 있다.

(6) HTML 텍스트 페이지를 화면에 보여주고, 자바스크립트의 정보를 서버에 요청하여 데이터를 전송 받는다.

(7) HTML 파싱을 수행 중에 발견되는 추가적인 HTML, CSS(cascade style sheet), 자바 스크립트, 이미지 등에 대해 Server 에 데이터 정보를 요청하게 되는데, 다운로드가 되지 않은 경우 서버에 다시 요청하게 된다. CSS 파일의 경우 CSS 문서 파일을 통해 스타일 정보가 추출되어 CSS DOM 구조를 통해 컨텐츠 정보를 활용하게 된다.

(8) 이미지의 개수가 적을 경우, 이미지 사이즈가 작은 경우 기존과 동일하게 서버에 이미지를 요청하고 서버로부터 이미지를 전송 받는다. 이미지 개수가 많고 사업자 proxy 를 사용하여 사업자 홈페이지에 접속하는 경우 저사양 휴대폰 단말의 UA(User Agent)/UAprof(User Agent Profile)를 사용하여 저사양 단말의 이미지를 전송 받는다. 또한 이미지 개수가 적고 이미지 사이즈가 클 경우 사업자 proxy 를 확인하고 일반적인 사이트인 경우 저사양 이미지로 변환하여 렌더링에 대한 CPU 부하를 줄인다. 만약 이미지 크기가 지정한 값보다 클 경우 저사양 휴대폰 단말의 UA/UAProf 로 변환하여 서버에서 저사양 이미지를 전송 하도록 유도한다. 이미지 개수가 작아서 휴대폰 단말로 전송했는데, 이미지의 크기가 큰 경우 일정 사이즈가 넘은 이미지에 대해서는 내부 디코딩을 통해 이미지를 축소하여 화면에 보여 주는데 걸리는 시간을 단축한다. 페이지에 큰 이미지가 존재할 경우 상당히 많은 부하가 발생됨으로 그 이미지를 최소한의 사이즈로 축소하게 된다.

(9) HTML 파싱 단계를 걸쳐 HTML 태그 등과 같은 HTML 구성 요소로 분리되어 DOM(document object mode) 트리 형태의 자료 구조가 생성된다.

(10) DOM 트리와 함께 render tree 가 생성된다. 렌더링에 필요한 다양한 컨텐츠에 대한 처리를 진행하여 자료구조의 갱신(layout, style update)을 수행하게 된다.

(11) 화면 재구성을 통해 페이지를 화면에 drawing 하게 된다.

4. 실험 및 성능분석

본 논문에서 제안하는 기법의 성능 실험을 위해 인도 New Delhi 에서 Airtel 사업자 페이지와, 표 1 과 같은 성능의 휴대단말기를 사용하였다.

<표 1> 무선 휴대 단말기 사양

Category	휴대 단말기 1	휴대 단말기 2
Base band	Infineon	Qualcomm
Data	GPRS Class 10 (4+1/3+2 slots), 32 – 48 kbps EDGE Class 10, 236.8 kbps 3G No WLAN No BluetoothYes, v2.1 with A2DP Infrared portNo	GPRS/EDGE class 12, HSDPA 7.2 Mbps WLAN No BluetoothYes, v2.1 with A2DP Infrared portNo
RF Band	2G Network GSM 900 / 1800 / 1900 – SIM 1 GSM 900 / 1800 / 1900 – SIM 2	GSM 850/900/1800/1900 MHz, UMTS 900/2100 MHz
LCD Type	240 x 320 pixels, 2.6 inch	240 x 320 pixels2,8-inch TFT, QVGA resolution
WAP	Netfront Browser WAP2.0/XHTML,HTML HTTP, OTA provisioning supported WAP Push service supported	WebKit Browser 0.8 WAP2.0/XHTML,HTML HTTP, OTA provisioning supported WAP Push service supported

표 2 와 3 은 표 1 의 휴대 단말기들을 사용하여 www.go.com 페이지를 대상으로 제안 기법을 적용한 페이지 로딩 시간을 보여준다. 휴대 단말기 종류에 관계 없이 제안 기법을 통해 페이지 로딩 시간이 단축됨을 알 수 있다.

<표 2> 제안 방식을 통한 페이지 로딩시간 측정(단위: 초)

	기존 방식		제안 방식		차 이	
	1st drawing	full drawing	1st drawing	full drawing	1st drawing	full drawing
1	43.82	44.30	10.47	43.55	-33.35	-0.75
2	46.14	46.62	9.63	44.45	-36.51	-2.17
3	45.07	45.55	10.89	43.99	-34.18	-1.56
4	46.68	47.16	9.67	43.12	-37.01	-4.04
5	46.51	46.99	9.59	45.88	-36.92	-1.11
6	46.69	47.17	9.29	43.23	-37.40	-3.94
7	46.75	47.23	9.54	41.88	-37.21	-5.35
8	44.86	45.34	9.98	42.32	-34.88	-3.02
9	44.83	45.31	10.01	44.33	-34.82	-0.98
10	44.95	45.43	9.92	43.12	-35.03	-2.31
평균	45.63	46.11	9.90	43.59	-35.73	-2.52

측정: 휴대 단말기 1
Site: http://www.go.com
Browser: Netfront 4 Browser

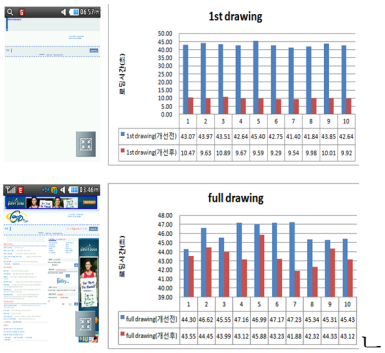
<표 3> 제안 방식을 통한 페이지 로딩시간 측정(단위: 초)

	기존 방식		제안 방식		차이	
	1st drawing	full drawing	1st drawing	full drawing	1st drawing	full drawing
1	11.45	35.09	9.23	32.01	-2.22	-3.08
2	13.43	32.86	9.87	30.11	-3.56	-2.75
3	12.56	36.68	10.4	31.8	-2.16	-4.88
4	15.59	33.45	9.78	30.47	-5.81	-2.98
5	14.44	34.15	10.77	31.17	-3.67	-2.98
6	13.54	35.56	9.45	32.15	-4.09	-3.41
7	16.45	39.44	9.32	34.65	-7.13	-4.79
8	13.45	35.1	8.45	31.77	-5.00	-3.33
9	12.56	34.15	11.2	31.45	-1.36	-2.70
10	15.1	36.98	9.84	34.55	-5.26	-2.43
AVR	13.86	35.35	9.83	32.01	-4.03	-3.33

측정: 휴대 단말기 2
Site: http://www.go.com
Browser: WebKit Browser 0.8

그림 2 는 Netfront 브라우저를 사용하는 경우 1st drawing 에 대한 로딩 속도를 나타내고 있다. 표 2 와 그림에서 정리된 바와 같이 1st drawing 시 기존 방식에 비해 제안 방식 후가 평균 속도로 -35.73 초가 빨라 졌음을 알 수 있다. www.go.com 의 사용자는 최초 브라우저 화면을 평균 10 초 때에 볼 수 있다. 즉, 사용자가 느끼는 체감 속도는 많은 개선 효과를 볼 수 있다. full drawing 인 경우 -2.52 초를 보인다. 이는 최초 drawing 후 파싱, 이미지 전송, 화면 rendering, 자

바스크립트, 스타일등 여러 가지 화면구성에 필요한 처리를 하고, 성능에 영향을 주는 부분들에 대해 처리를 하면서 기존 대비 향상됨을 알 수 있다



(그림 2) 제안 방식을 통한 1st drawing / full drawing 로딩시간 측정(초)

표 4 는 표 2 와 표 3 의 실험 결과를 토대로 제안방식의 페이지 로딩시간 감소율을 정리한 결과이다. 최초 화면 로딩이 기존 대비 1st drawing 시 78.31%, 28.22%의 속도 증가 현상을 보였으나, 전체 화면 로딩인 full drawing 에 대해서는 5.47%, 9.38%의 개선 현상을 보였다. 이는 1st drawing 시 부하가 걸리는 자바스크립트나 일반 렌더링이 아닌, 단순히 레이아웃을 보여줌으로써 사용자의 체감속도 현상이 발생한 것이다. 하지만 full drawing 을 할 경우 전체 화면을 구성하고 drawing 하기 위해 자바스크립트, 이미지, CSS 등을 재구성 함으로써 속도 차이가 1st drawing 에 비해 다소 크게 개선되지는 않았지만, 기존 방식에 비해서는 5.47%, 9.38% 개선 되었음을 알 수 있다.

<표 4> 제안 방식을 통한 페이지 로딩시간 측정(%)

	제안 방식 (휴대 단말기 1)		제안 방식 (휴대 단말기 2)	
	1st drawing	full drawing	1st drawing	full drawing
1	76,11	1,69	19,39	8,78
2	79,13	4,65	26,51	8,37
3	75,84	3,42	17,20	13,30
4	79,28	8,57	37,27	8,91
5	79,38	2,36	25,42	8,73
6	80,10	8,35	30,21	9,59
7	79,59	11,33	43,34	12,15
8	77,75	6,66	37,17	9,49
9	77,67	2,16	10,83	7,91
10	77,93	5,08	34,83	6,57
평균	78,31	5,47	28,22	9,38

사업자 페이지가 아닌 일반 사이트로 UA/UAProf 의 값에 영향을 받거나 받지 않을 수도 있다. 이는 생산되는 모든 단말의 UA/UAProf 를 모든 사이트가 실시간으로 등록하기란 사실상 불가능 하기 때문이다. 그렇기 때문에 일반적으로 사용되는 웹 페이지인 경우 LCD resolution 에 영향을 받지 않도록 페이지를 구성하게 된다. 일부 사이트는 UA/UAProf 에 영향을 받지만 일부 웹 사이트들은 영향을 받지 않아 두 개의 UA/UAProf 로 인해 저품질의 이미지 변환에 대한 알고리즘이 적용되지 않아 full drawing 에 대한 속도에 영향을 주지 못하였다. 반면 큰 이미지나 일정 수준 이상의 이미지를 로딩한 경우 화면에 drawing 할 경우 크기를 축소하고, 저품질로 변환하는 경우 렌더링에 영향을 줌을 알 수 있다

5. 결론

본 논문에서는 저사양 휴대 단말기의 웹 브라우저 로딩 속도를 개선하기 위한 기법을 제안하였다. 이를 위해, 휴대 단말 환경에서 브라우저 웹 페이지 로딩 시 어떤 과정을 거치는지 알아보고 어느 모듈에서 부하가 걸리는지 문제점을 찾아보았다. 분석을 통해, 성능에 영향을 미치는 자바 스크립트 프로세싱, 이미지 렌더링, 이미지 사이즈 등이 직접적으로 로딩시간에 영향을 준다는 것을 파악하였다. 이를 토대로, 본 논문에서는 텍스트와 이미지 등 데이터 타입을 분류하여 부하가 적게 걸리는 텍스트 레이아웃을 먼저 보여줌으로써 사용자의 체감 속도를 향상시키고, 아울러 이미지가 커서 렌더링 시간이 오래 걸리는 경우 이미지를 축소하거나 화질을 낮추는 방식으로 렌더링 부하를 줄여서 페이지 로딩 시간을 단축시키는 방법을 사용하였다. 또한 UA/UAProf 에 영향을 받는 페이지인 경우 실제 서버로부터 전송되는 이미지 패킷 데이터 용량이 작아지기 때문에 속도 향상을 됨을 알 수 있었다. 한편, 웹브라우저의 로딩 속도는 본 논문에서 분석한 소프트웨어적인 요소 뿐만 아니라 브라우저 엔진(CSS, JavaScript, DOM, Image, Parser 등), CPU 성능, 네트워킹 프로토콜 등의 다양한 요소에 영향을 받는다. 향후 연구로서 하드웨어 및 다양한 요인에 대한 로딩 속도 개선 방안 도출이 필요하다. 아울러, 개방형 웹 플랫폼 환경에서 다양한 이미지 축소 기법과 이에 대한 컴퓨팅 시간 및 렌더링 시간에 대한 비교 분석도 함께 이루어질 필요가 있다.

<감사의 글(Acknowledgement)>
이 논문은 한국연구재단의 기초연구사업(2011-0026529)과 정보통신산업진흥원의 대학 IT 연구센터지원사업(NIPA-2012-H0301-12-2004) 및 방송통신위원회의 산업원천기술개발사업(KCA-2011-10913-05004)의 연구결과로 수행되었음

참고문헌

[1] 한선영, 김병학, 권민순, 박종훈, 송관호, "브라우저 개발을 위한 연구", 한국통신학회 학술대회 논문집, 제 7 권 1 호, 1997, pp. 114~128.

[2] Jiann Hung Lin, et al., “Implement Browser-Based Network Management System using Visibroker,” Proceeding of Asia-Pacific Network Operations and Management Symposium, 1997

[3] 박기호, “모바일 기기의 웹브라우징 성능 요인 분석,” 정보과학회논문지: 컴퓨팅의 실제 및 레터, 제 17 권 제 2 호, 2011.

[4] C. G. Jones, et al., "Parallelizing the Web Browser," First USENIX Workshop on Hot Topics in Parallelism (HotPar09), March 2009.