(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2016년12월20일
(11) 등록번호 10-1688339
(24) 등록일자 2016년12월14일

(51) 국제특허분류(Int. Cl.)
G06F 17/21 (2006.01) *G06F 17/22* (2006.01)
G06F 17/30 (2006.01) *G06F 9/455* (2006.01)
(52) CPC특허분류
G06F 17/21 (2013.01)
G06F 17/2247 (2013.01)
(21) 출원번호 10-2015-7006571
(22) 출원일자(국제) 2013년06월17일
심사청구일자 2016년06월22일
(85) 번역문제출일자 2015년03월13일
(65) 공개번호 10-2015-0045471
(43) 공개일자 2015년04월28일
(86) 국제출원번호 PCT/US2013/046099
(87) 국제공개번호 WO 2014/028115
국제공개일자 2014년02월20일
(30) 우선권주장
61/683,999 2012년08월16일 미국(US)
(뒷면에 계속)
(56) 선행기술조사문헌
US20120005429 A1

(73) 특허권자
퀄컴 인코포레이티드
미국 92121-1714 캘리포니아주 샌 디에고 모어하우스 드라이브 5775
(72) 발명자
베버 미하엘
미국 92121 캘리포니아주 샌디에고 모어하우스 드라이브 5775
레샤디 모하메드 에이치
미국 92121 캘리포니아주 샌디에고 모어하우스 드라이브 5775
카스카발 게오르게 씨
미국 92121 캘리포니아주 샌디에고 모어하우스 드라이브 5775
(74) 대리인
특허법인코리아나

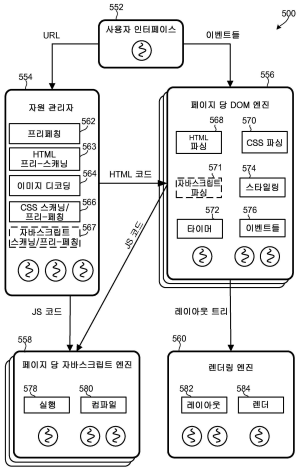
전체 청구항 수 : 총 15 항 심사관 : 이복현

(54) 발명의 명칭 **웹 브라우저들에서 스크립트들의 프리-프로세싱**

(57) 요 약

양태들은 웹 문서 (HTML 페이지) 를 병렬로 프로세싱함으로써 웹페이지를 로딩/렌더링하는 브라우저 시스템들 및 방법들을 포함한다. 스캐너 프로세스는 웹 문서를 스캔하고, 스크립트들을 식별하고, 스크립트들의 다운로딩을 개시한다. 스크립트들이 다운로드될 때, HTML 파서는 각 스크립트에 대한 식별자를 생성하고, 스크립트들 및 연관도니 식별자들을 스크립트 엔진에 전송한다. 스크립트 엔진은 스크립트들의 실행 순서와 상이할 수도 있는 순서로 실행을 위해 스크립트들을 파싱하고, 분석하고, 컴파일하고, 그렇지 않으면 준비한다.

대 표 도 - 도5



(52) CPC특허분류
G06F 17/30899 (2013.01)
G06F 9/45529 (2013.01)

(30) 우선권주장
61/684,594 2012년08월17일 미국(US)
13/722,066 2012년12월20일 미국(US)

명 세 서

청구범위

청구항 1

HTML 문서에 포함된 스크립트들을 준비하는 방법으로서,

복수의 스크립트들을 발견하기 위해 상기 HTML 문서를 스캔하는 단계 (752);

상기 복수의 스크립트들을, 실행을 위해 준비되도록 스크립트 엔진에 전송하는 단계 (760);

상기 스크립트 엔진이 실행을 위해 상기 복수의 스크립트들을 준비하는 동안 상기 HTML 문서를 파싱하는 단계 (762);

상기 복수의 스크립트들로부터 다음의 실행될 스크립트를 식별하는 단계 (764);

상기 식별된 다음의 실행될 스크립트에 대응하는 정보를 상기 스크립트 엔진에 전송하는 단계 (766);

상기 HTML 문서의 파싱을 중단하는 단계 (768);

상기 식별된 다음의 실행될 스크립트가 실행되었음을 표시하는 통지를 수신하는 단계 (768); 및

상기 통지를 수신하는 것에 응답하여 상기 HTML 문서의 파싱을 재시작하는 단계 (770) 를 포함하는, HTML 문서에 포함된 스크립트들을 준비하는 방법.

청구항 2

제 1 항에 있어서,

상기 식별된 다음의 실행될 스크립트에 대응하는 정보를 상기 스크립트 엔진에 전송하는 단계는 상기 식별된 다음의 실행될 스크립트를 상기 스크립트 엔진에 전송하는 단계를 포함하는, HTML 문서에 포함된 스크립트들을 준비하는 방법.

청구항 3

제 1 항에 있어서,

상기 복수의 스크립트들의 각각에 대한 식별자를 생성하는 단계 (758) 를 더 포함하며,

상기 복수의 스크립트들을 스크립트 엔진에 전송하는 단계 (760) 는 상기 복수의 스크립트들 및 식별자들을 상기 스크립트 엔진에 전송하는 단계를 포함하고, 그리고

상기 식별된 다음의 실행될 스크립트에 대응하는 정보를 상기 스크립트 엔진에 전송하는 단계 (766) 는 상기 다음의 실행될 스크립트의 식별자를 상기 스크립트 엔진에 전송하는 단계를 포함하며,

특히 상기 복수의 스크립트들의 각각에 대한 식별자를 생성하는 단계 (758) 는 적어도 하나의 스크립트를 URI (uniform resource identifier) 와 연관시키는 단계를 포함하거나, 또는

특히 상기 복수의 스크립트들의 각각에 대한 식별자를 생성하는 단계 (758) 는 적어도 하나의 스크립트에 대한 서명을 생성하는 단계를 포함하거나, 또는

특히 상기 복수의 스크립트들의 각각에 대한 식별자를 생성하는 단계 (758) 는 적어도 하나의 스크립트의 텍스트를 포함하는 적어도 하나의 식별자를 생성하는 단계를 포함하는, HTML 문서에 포함된 스크립트들을 준비하는 방법.

청구항 4

제 1 항에 있어서,

상기 복수의 스크립트들을 발견하기 위해 HTML 문서를 스캔하는 단계 (752) 는 제 1 프로세서 (204) 에서 상기 HTML 문서를 스캔하는 단계를 포함하고, 그리고

상기 스크립트 엔진이 실행을 위해 상기 복수의 스크립트들을 준비하는 동안 상기 HTML 문서를 파싱하는 단계 (762) 는 제 2 프로세서 (206) 에서 상기 HTML 문서를 파싱하는 단계를 포함하는, HTML 문서에 포함된 스크립트들을 준비하는 방법.

청구항 5

제 1 항에 있어서,

상기 복수의 스크립트들을 발견하기 위해 HTML 문서를 스캔하는 단계 (752) 는 프로세서 (202) 에서 실행중인 제 1 프로세스에 의해 상기 HTML 문서를 스캔하는 단계를 포함하고, 그리고

상기 스크립트 엔진이 실행을 위해 상기 복수의 스크립트들을 준비하는 동안 상기 HTML 문서를 파싱하는 단계 (762) 는 상기 프로세서 (202) 에서 실행중인 제 2 프로세스에 의해 상기 HTML 문서를 파싱하는 단계를 포함하며,

특히 상기 스크립트 엔진이 실행을 위해 상기 복수의 스크립트들을 준비하는 동안 상기 HTML 문서를 파싱하는 단계 (762) 는 상기 스크립트 엔진이 제 2 스크립트를 파싱하고, 분석하고, 컴파일링하는 것과 병렬로 상기 스크립트 엔진이 제 1 스크립트를 파싱하고, 분석하고, 컴파일링하는 동안 상기 HTML 문서를 파싱하는 단계를 포함하는, HTML 문서에 포함된 스크립트들을 준비하는 방법.

청구항 6

제 1 항에 있어서,

상기 스크립트 엔진이 실행을 위해 상기 복수의 스크립트들을 준비하는 동안 상기 HTML 문서를 파싱하는 단계 (762) 는 상기 스크립트 엔진이 상기 복수의 스크립트들이 실행되는 실행 순서와 상이한 준비 순서로 실행을 위해 상기 복수의 스크립트들을 준비하는 동안 상기 HTML 문서를 파싱하는 단계를 포함하는, HTML 문서에 포함된 스크립트들을 준비하는 방법.

청구항 7

제 1 항에 있어서,

상기 복수의 스크립트들로부터 다음의 실행될 스크립트를 식별하는 단계 (764) 는 정의된 실행 순서에 기초하여 상기 다음의 실행될 스크립트를 식별하는 단계를 포함하는, HTML 문서에 포함된 스크립트들을 준비하는 방법.

청구항 8

컴퓨팅 디바이스 (1200, 1300, 1400) 로서,

복수의 스크립트들을 발견하기 위해 HTML 문서를 스캔하는 수단;

상기 복수의 스크립트들을, 실행을 위해 준비되도록 스크립트 엔진에 전송하는 수단;

상기 스크립트 엔진이 실행을 위해 상기 복수의 스크립트들을 준비하는 동안 상기 HTML 문서를 파싱하는 수단;

상기 복수의 스크립트들로부터 다음의 실행될 스크립트를 식별하는 수단;

상기 식별된 다음의 실행될 스크립트에 대응하는 정보를 상기 스크립트 엔진에 전송하는 수단;

상기 HTML 문서의 파싱을 중단하는 수단;

상기 식별된 다음의 실행될 스크립트가 실행되었음을 표시하는 통지를 수신하는 수단; 및

상기 통지를 수신하는 것에 응답하여 상기 HTML 문서의 파싱을 재시작하는 수단을 포함하는, 컴퓨팅 디바이스 (1200, 1300, 1400).

청구항 9

제 8 항에 있어서,

상기 식별된 다음의 실행될 스크립트에 대응하는 정보를 상기 스크립트 엔진에 전송하는 수단은 상기 식별된 다음의 실행될 스크립트를 상기 스크립트 엔진에 전송하는 수단을 포함하는, 컴퓨팅 디바이스 (1200, 1300,

1400).

청구항 10

제 8 항에 있어서,

상기 복수의 스크립트들의 각각에 대한 식별자를 생성하는 수단을 더 포함하며,

상기 복수의 스크립트들을 스크립트 엔진에 전송하는 수단은 상기 복수의 스크립트들 및 식별자들을 상기 스크립트 엔진에 전송하는 수단을 포함하고, 그리고

상기 식별된 다음의 실행될 스크립트에 대응하는 정보를 상기 스크립트 엔진에 전송하는 수단은 상기 다음의 실행될 스크립트의 식별자를 상기 스크립트 엔진에 전송하는 수단을 포함하며,

특히 상기 복수의 스크립트들의 각각에 대한 식별자를 생성하는 수단은 적어도 하나의 스크립트를 URI (uniform resource identifier) 와 연관시키는 수단을 포함하거나, 또는

특히 상기 복수의 스크립트들의 각각에 대한 식별자를 생성하는 수단은 적어도 하나의 스크립트에 대한 서명을 생성하는 수단을 포함하거나, 또는

상기 복수의 스크립트들의 각각에 대한 식별자를 생성하는 수단은 적어도 하나의 스크립트의 텍스트를 포함하는 적어도 하나의 식별자를 생성하는 수단을 포함하는, 컴퓨팅 디바이스 (1200, 1300, 1400).

청구항 11

제 8 항에 있어서,

상기 복수의 스크립트들을 발견하기 위해 HTML 문서를 스캔하는 수단은 제 1 프로세서 (204) 에서 상기 HTML 문서를 스캔하는 수단을 포함하고, 그리고

상기 스크립트 엔진이 실행을 위해 상기 복수의 스크립트들을 준비하는 동안 상기 HTML 문서를 파싱하는 수단은 제 2 프로세서 (206) 에서 상기 HTML 문서를 파싱하는 수단을 포함하는, 컴퓨팅 디바이스 (1200, 1300, 1400).

청구항 12

제 8 항에 있어서,

상기 복수의 스크립트들을 발견하기 위해 HTML 문서를 스캔하는 수단은 프로세서에서 실행중인 제 1 프로세스에 의해 상기 HTML 문서를 스캔하는 수단을 포함하고, 그리고

상기 스크립트 엔진이 실행을 위해 상기 복수의 스크립트들을 준비하는 동안 상기 HTML 문서를 파싱하는 수단은 상기 프로세서 (202) 에서 실행중인 제 2 프로세스에 의해 상기 HTML 문서를 파싱하는 수단을 포함하며,

특히 상기 스크립트 엔진이 실행을 위해 상기 복수의 스크립트들을 준비하는 동안 상기 HTML 문서를 파싱하는 수단은 상기 스크립트 엔진이 제 2 스크립트를 파싱하고, 분석하고, 컴파일링하는 것과 병렬로 상기 스크립트 엔진이 제 1 스크립트를 파싱하고, 분석하고, 컴파일링하는 동안 상기 HTML 문서를 파싱하는 수단을 포함하는, 컴퓨팅 디바이스 (1200, 1300, 1400).

청구항 13

제 8 항에 있어서,

상기 스크립트 엔진이 실행을 위해 상기 복수의 스크립트들을 준비하는 동안 상기 HTML 문서를 파싱하는 수단은 상기 스크립트 엔진이 상기 복수의 스크립트들이 실행되는 실행 순서와 상이한 준비 순서로 실행을 위해 상기 복수의 스크립트들을 준비하는 동안 상기 HTML 문서를 파싱하는 수단을 포함하거나, 또는

상기 복수의 스크립트들로부터 다음의 실행될 스크립트를 식별하는 수단은 정의된 실행 순서에 기초하여 상기 다음의 실행될 스크립트를 식별하는 수단을 포함하는, 컴퓨팅 디바이스 (1200, 1300, 1400).

청구항 14

제 8 항 내지 제 13 항 중 어느 한 항에 있어서,

상기 다양한 수단들은 프로세서 실행가능 명령들로 구성된 프로세서 (202) 에 의해 구현되는, 컴퓨팅 디바이스 (1200, 1300, 1400).

청구항 15

프로세서 (202) 로 하여금 실행될 경우 제 1 항 내지 제 7 항 중 어느 한 항의 방법의 단계들을 수행하게 하도록 구성된 프로세서 실행가능 소프트웨어 명령들을 저장한 비일시적 컴퓨터 판독가능 저장 매체.

청구항 16

삭제

청구항 17

삭제

청구항 18

삭제

청구항 19

삭제

청구항 20

삭제

청구항 21

삭제

청구항 22

삭제

청구항 23

삭제

청구항 24

삭제

청구항 25

삭제

청구항 26

삭제

청구항 27

삭제

청구항 28

삭제

청구항 29

삭제

청구항 30

삭제

청구항 31

삭제

청구항 32

삭제

청구항 33

삭제

청구항 34

삭제

청구항 35

삭제

청구항 36

삭제

청구항 37

삭제

청구항 38

삭제

청구항 39

삭제

청구항 40

삭제

청구항 41

삭제

청구항 42

삭제

발명의 설명

기 술 분 야

[0001] 관련 특허 출원들

[0002] 본 출원은 2012년 8월 17일자로 출원된 발명 명칭이 "Pre-Processing of Scripts in Web Browsers"인 미국 특허 가출원 제61/684,594호와 2012년 8월 16일자로 출원된 발명 명칭이 "Pre-Processing of Scripts in Web Browsers"인 미국 특허 가출원 제61/683,999호를 우선권 주장한다.

[0003] 발명의 분야

[0004] 본 발명은 웹 브라우저에서 HTML 문서들을 렌더링하는 방법들, 시스템들, 및 디바이스들에 관한 것이고, 더 상세하게는 웹 브라우저 동작들을 병렬화하는 방법들에 관한 것이다.

배 경 기 술

[0005] 무선 통신 기술들과 모바일 전자 디바이스들 (예컨대, 셀룰러 폰들, 테블릿들, 랩톱들 등) 은 지난 몇 년 동안 인기와 사용으로 성장하였다. 증가된 소비자 요구들에 맞추기 위해, 모바일 전자 디바이스들은 더욱 피처 (feature) 가 풍부하게 되고, 이제 일반적으로, 모바일 디바이스 사용자들이 그들의 모바일 디바이스들 상에서 복잡하고 파워 집중적인 소프트웨어 애플리케이션들 (예컨대, 웹 브라우저들, 비디오 스트리밍 애플리케이션들 등) 을 실행하는 것을 허용하는 다수의 프로세서들, 시스템-온-칩 (system-on-chip; SoC) 들, 및 다른 리소스들을 포함한다. 이들 및 다른 개선들로 인해, 스마트폰들과 태블릿 컴퓨터들은 인기리에 성장하였고, 많은 사용자들에 의해 선택되는 플랫폼으로서 랩톱들과 데스크톱 머신들을 대체하고 있다.

[0006] 모바일 디바이스 사용자들은 이제 그들의 모바일 디바이스 상의 브라우저 애플리케이션들을 통해 인터넷에 액세스함으로써 쉽고 편리하게 많은 그들의 매일의 태스크들을 달성할 수 있다. 모바일 디바이스들이 인기리에 계속 성장하므로, 현대의 모바일 디바이스들의 멀티프로세싱 능력들을 더 잘 이용할 수 있는 웹 브라우저들은 소비자들에게 바람직한 것이 될 것이다.

발명의 내용

과제의 해결 수단

[0007] 다양한 양태들은 HTML 문서에 포함된 스크립트들을 준비하는 방법들을 포함하며, 그 방법들은, 복수의 스크립트들을 발견하기 위해 HTML 문서를 스캔하는 단계, 복수의 스크립트들을 스크립트 엔진에 전송하는 단계, 스크립트 엔진이 실행을 위해 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 단계, 복수의 스크립트들로부터 다음의 실행될 스크립트를 식별하는 단계, 식별된 다음의 실행될 스크립트에 대응하는 정보를 스크립트 엔진에 전송하는 단계, HTML 문서의 파싱을 중단하는 단계, 식별된 다음의 실행될 스크립트가 실행되었음을 표시하는 통지를 수신하는 단계, 및 통지를 수신하는 것에 응답하여 HTML 문서의 파싱을 재시작하는 단계를 포함할 수도 있다. 일 양태에서, 식별된 다음의 실행될 스크립트에 대응하는 정보를 스크립트 엔진에 전송하는 단계는 식별된 다음의 실행될 스크립트를 스크립트 엔진에 전송하는 단계를 포함할 수도 있다.

[0008] 일 양태에서, 그 방법은 복수의 스크립트들의 각각에 대한 식별자를 생성하는 단계를 포함할 수도 있다. 추가의 양태에서, 복수의 스크립트들을 스크립트 엔진에 전송하는 단계는 복수의 스크립트들 및 식별자들을 스크립트 엔진에 전송하는 단계를 포함할 수도 있고, 식별된 다음의 실행될 스크립트에 대응하는 정보를 스크립트 엔진에 전송하는 단계는 다음의 실행될 스크립트의 식별자를 스크립트 엔진에 전송하는 단계를 포함할 수도 있다. 추가의 양태에서, 복수의 스크립트들의 각각에 대한 식별자를 생성하는 단계는 적어도 하나의 스크립트를 URI (uniform resource identifier) 와 연관시키는 단계를 포함할 수도 있다. 추가의 양태에서, 복수의 스크립트들의 각각에 대한 식별자를 생성하는 단계는 적어도 하나의 스크립트에 대한 서명을 생성하는 단계를 포함할 수도 있다. 추가의 양태에서, 복수의 스크립트들의 각각에 대한 식별자를 생성하는 단계는 적어도 하나의 스크립트의 텍스트를 포함할 수도 있는 적어도 하나의 식별자를 생성하는 단계를 포함할 수도 있다.

[0009] 추가의 양태에서, 복수의 스크립트들을 발견하기 위해 HTML 문서를 스캔하는 단계는 제 1 프로세서에서 HTML 문서를 스캔하는 단계를 포함할 수도 있고, 스크립트 엔진이 실행을 위해 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 단계는 제 2 프로세서에서 HTML 문서를 파싱하는 단계를 포함할 수도 있다. 추가의 양태에서, 복수의 스크립트들을 발견하기 위해 HTML 문서를 스캔하는 단계는 프로세서에서 실행중인 제 1 프로세스에 의해 HTML 문서를 스캔하는 단계를 포함할 수도 있고, 스크립트 엔진이 실행을 위해 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 단계는 프로세서에서 실행중인 제 2 프로세스에 의해 HTML 문서를 파싱하는 단계를 포함할 수도 있다.

[0010] 추가의 양태에서, 스크립트 엔진이 실행을 위해 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 단계는 스크립트 엔진이 제 2 스크립트를 파싱하고, 분석하고, 컴파일링하는 것과 병렬로 스크립트 엔진이 제 1 스크립트를 파싱하고, 분석하고, 컴파일링하는 동안 HTML 문서를 파싱하는 단계를 포함할 수도 있다. 추가의 양태에서, 스크립트 엔진이 실행을 위해 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 단계는 스크립트 엔진이 복수의 스크립트들이 실행되는 실행 순서와 상이한 준비 순서로 실행을 위해 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 단계를 포함할 수도 있다. 추가의 양태에서, 복수의 스크립트들로부터

다음의 실행될 스크립트를 식별하는 단계는 정의된 실행 순서에 기초하여 다음의 실행될 스크립트를 식별하는 단계를 포함할 수도 있다.

[0011] 추가의 양태들은, 복수의 스크립트들을 발견하기 위해 HTML 문서를 스캔하는 수단, 복수의 스크립트들을 스크립트 엔진에 전송하는 수단, 스크립트 엔진이 실행을 위해 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 수단, 복수의 스크립트들로부터 다음의 실행될 스크립트를 식별하는 수단, 식별된 다음의 실행될 스크립트에 대응하는 정보를 스크립트 엔진에 전송하는 수단, HTML 문서의 파싱을 중단하는 수단, 식별된 다음의 실행될 스크립트가 실행되었음을 표시하는 통지를 수신하는 수단, 및 통지를 수신하는 것에 응답하여 HTML 문서의 파싱을 재시작하는 수단을 포함할 수도 있는, 컴퓨팅 디바이스를 포함한다.

[0012] 일 양태에서, 식별된 다음의 실행될 스크립트에 대응하는 정보를 스크립트 엔진에 전송하는 수단은 식별된 다음의 실행될 스크립트를 스크립트 엔진에 전송하는 수단을 포함할 수도 있다. 추가의 양태에서, 컴퓨팅 디바이스는 복수의 스크립트들의 각각에 대한 식별자를 생성하는 수단을 포함할 수도 있다. 추가의 양태에서, 복수의 스크립트들을 스크립트 엔진에 전송하는 수단은 복수의 스크립트들 및 식별자들을 스크립트 엔진에 전송하는 수단을 포함할 수도 있고, 식별된 다음의 실행될 스크립트에 대응하는 정보를 스크립트 엔진에 전송하는 수단은 다음의 실행될 스크립트의 식별자를 스크립트 엔진에 전송하는 수단을 포함할 수도 있다. 추가의 양태에서, 복수의 스크립트들의 각각에 대한 식별자를 생성하는 수단은 적어도 하나의 스크립트를 URI (uniform resource identifier) 와 연관시키는 수단을 포함할 수도 있다. 추가의 양태에서, 복수의 스크립트들의 각각에 대한 식별자를 생성하는 수단은 적어도 하나의 스크립트에 대한 서명을 생성하는 수단을 포함할 수도 있다. 추가의 양태에서, 복수의 스크립트들의 각각에 대한 식별자를 생성하는 수단은 적어도 하나의 스크립트의 텍스트를 포함하는 적어도 하나의 식별자를 생성하는 수단을 포함할 수도 있다.

[0013] 추가의 양태에서, 복수의 스크립트들을 발견하기 위해 HTML 문서를 스캔하는 수단은 제 1 프로세서에서 HTML 문서를 스캔하는 수단을 포함할 수도 있고, 스크립트 엔진이 실행을 위해 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 수단은 제 2 프로세서에서 HTML 문서를 파싱하는 수단을 포함할 수도 있다. 추가의 양태에서, 복수의 스크립트들을 발견하기 위해 HTML 문서를 스캔하는 수단은 프로세서에서 실행중인 제 1 프로세스에 의해 HTML 문서를 스캔하는 수단을 포함할 수도 있고, 스크립트 엔진이 실행을 위해 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 수단은 프로세서에서 실행중인 제 2 프로세스에 의해 HTML 문서를 파싱하는 수단을 포함할 수도 있다.

[0014] 추가의 양태에서, 스크립트 엔진이 실행을 위해 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 수단은 스크립트 엔진이 제 2 스크립트를 파싱하고, 분석하고, 컴파일링하는 것과 병렬로 스크립트 엔진이 제 1 스크립트를 파싱하고, 분석하고, 컴파일링하는 동안 HTML 문서를 파싱하는 수단을 포함할 수도 있다. 추가의 양태에서, 스크립트 엔진이 실행을 위해 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 수단은 스크립트 엔진이 복수의 스크립트들이 실행되는 실행 순서와 상이한 준비 순서로 실행을 위해 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 수단을 포함할 수도 있다. 추가의 양태에서, 복수의 스크립트들로부터 다음의 실행될 스크립트를 식별하는 수단은 정의된 실행 순서에 기초하여 다음의 실행될 스크립트를 식별하는 수단을 포함할 수도 있다.

[0015] 추가의 양태들은 동작들을 수행하기 위한 프로세서 실행가능 명령들로 구성된 프로세서를 포함할 수도 있는 컴퓨팅 디바이스를 포함하며, 그 동작들은, 실행을 위해 준비될 복수의 스크립트들을 발견하기 위해 HTML 문서를 스캔하는 것, 복수의 스크립트들을 스크립트 엔진에 전송하는 것, 스크립트 엔진이 실행을 위해 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 것, 복수의 스크립트들로부터 다음의 실행될 스크립트를 식별하는 것, 식별된 다음의 실행될 스크립트에 대응하는 정보를 스크립트 엔진에 전송하는 것, HTML 문서의 파싱을 중단하는 것, 식별된 다음의 실행될 스크립트가 실행되었음을 표시하는 통지를 수신하는 것, 및 통지를 수신하는 것에 응답하여 HTML 문서의 파싱을 재시작하는 것을 포함할 수도 있다.

[0016] 일 양태에서, 프로세서는, 식별된 다음의 실행될 스크립트에 대응하는 정보를 스크립트 엔진에 전송하는 것이 식별된 다음의 실행될 스크립트를 스크립트 엔진에 전송하는 것을 포함하게 하는 동작들을 수행하기 위한 프로세서 실행가능 명령들로 구성될 수도 있다. 추가의 양태에서, 프로세서는, 복수의 스크립트들의 각각에 대한 식별자를 생성하는 것을 더 포함하는 동작들을 수행하기 위한 프로세서 실행가능 명령들로 구성될 수도 있고, 프로세서는, 복수의 스크립트들을 스크립트 엔진에 전송하는 것이 복수의 스크립트들 및 식별자들을 스크립트 엔진에 전송하는 것을 포함할 수도 있게 하고, 식별된 다음의 실행될 스크립트에 대응하는 정보를 스크립트 엔진에 전송하는 것이 다음의 실행될 스크립트의 식별자를 스크립트 엔진에 전송하는 것을 포함할 수도 있게

하는 동작들을 수행하기 위한 프로세서 실행가능 명령들로 구성될 수도 있다.

[0017] 추가의 양태에서, 프로세서는 복수의 스크립트들의 각각에 대한 식별자를 생성하는 것이 적어도 하나의 스크립트를 URI (uniform resource identifier) 와 연관시키는 것을 포함할 수도 있게 하는 동작들을 수행하기 위한 프로세서 실행가능 명령들로 구성될 수도 있다. 추가의 양태에서, 프로세서는, 복수의 스크립트들의 각각에 대한 식별자를 생성하는 것이 적어도 하나의 스크립트에 대한 서명을 생성하는 것을 포함할 수도 있게 하는 동작들을 수행하기 위한 프로세서 실행가능 명령들로 구성될 수도 있다. 추가의 양태에서, 프로세서는, 복수의 스크립트들의 각각에 대한 식별자를 생성하는 것이 적어도 하나의 스크립트의 텍스트를 포함하는 적어도 하나의 식별자를 생성하는 것을 포함할 수도 있게 하는 동작들을 수행하기 위한 프로세서 실행가능 명령들로 구성될 수도 있다.

[0018] 추가의 양태에서, 프로세서는, 복수의 스크립트들을 발견하기 위해 HTML 문서를 스캔하는 것이 프로세서에서 실행중인 제 1 프로세스에 의해 HTML 문서를 스캔하는 것을 포함할 수도 있게 하고, 스크립트 엔진이 실행을 위해 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 것이 프로세서에서 실행중인 제 2 프로세스에 의해 HTML 문서를 파싱하는 것을 포함할 수도 있게 하는 동작들을 수행하기 위한 프로세서 실행가능 명령들로 구성될 수도 있다. 추가의 양태에서, 프로세서는, 실행을 위해 복수의 스크립트들을 준비하는 것이 제 2 프로세스가 제 2 스크립트를 파싱하고, 분석하고, 컴파일링하는 것과 병렬로 제 1 스크립트를 파싱하고, 분석하고, 컴파일링하는 것을 포함할 수도 있게 하는 동작들을 수행하기 위한 프로세서 실행가능 명령들로 구성될 수도 있다.

[0019] 추가의 양태에서, 프로세서는, 스크립트 엔진이 실행을 위해 병렬로 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 것이 스크립트 엔진이 복수의 스크립트들이 실행되는 실행 순서와 상이한 준비 순서로 실행을 위해 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 것을 포함할 수도 있게 하는 동작들을 수행하기 위한 프로세서 실행가능 명령들로 구성될 수도 있다. 추가의 양태에서, 프로세서는, 복수의 스크립트들로부터 다음의 실행될 스크립트를 식별하는 것이 정의된 실행 순서에 기초하여 다음의 실행될 스크립트를 식별하는 것을 포함할 수도 있게 하는 동작들을 수행하기 위한 프로세서 실행가능 명령들로 구성될 수도 있다.

[0020] 추가의 양태들은, 프로세서로 하여금 HTML 문서에 포함된 스크립트들을 준비하기 위한 동작들을 수행하게 하도록 구성된 프로세서 실행가능 소프트웨어 명령들을 저장한 비일시적 컴퓨터 판독가능 저장 매체를 포함하며, 그 동작들은, 복수의 스크립트들을 발견하기 위해 HTML 문서를 스캔하는 것, 복수의 스크립트들을 스크립트 엔진에 전송하는 것, 스크립트 엔진이 실행을 위해 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 것, 복수의 스크립트들로부터 다음의 실행될 스크립트를 식별하는 것, 식별된 다음의 실행될 스크립트에 대응하는 정보를 스크립트 엔진에 전송하는 것, HTML 문서의 파싱을 중단하는 것, 식별된 다음의 실행될 스크립트가 실행되었음을 표시하는 통지를 수신하는 것, 및 통지를 수신하는 것에 응답하여 HTML 문서의 파싱을 재시작하는 것을 포함한다. 일 양태에서, 저장된 프로세서 실행가능 소프트웨어 명령들은, 프로세서로 하여금, 식별된 다음의 실행될 스크립트에 대응하는 정보를 스크립트 엔진에 전송하는 것이 식별된 다음의 실행될 스크립트를 스크립트 엔진에 전송하는 것을 포함할 수도 있게 하는 동작들을 수행하게 하도록 구성될 수도 있다.

[0021] 추가의 양태에서, 저장된 프로세서 실행가능 소프트웨어 명령들은, 프로세서로 하여금, 복수의 스크립트들의 각각에 대한 식별자를 생성하는 것을 포함하는 동작들을 수행하게 하도록 구성될 수도 있고, 저장된 프로세서 실행가능 소프트웨어 명령들은, 프로세서로 하여금, 복수의 스크립트들을 스크립트 엔진에 전송하는 것이 복수의 스크립트들 및 식별자들을 스크립트 엔진에 전송하는 것을 포함할 수도 있게 하고, 식별된 다음의 실행될 스크립트에 대응하는 정보를 스크립트 엔진에 전송하는 것이 다음의 실행될 스크립트의 식별자를 스크립트 엔진에 전송하는 것을 포함할 수도 있게 하는 동작들을 수행하게 하도록 구성될 수도 있다. 추가의 양태에서, 저장된 프로세서 실행가능 소프트웨어 명령들은, 프로세서로 하여금, 복수의 스크립트들의 각각에 대한 식별자를 생성하는 것이 적어도 하나의 스크립트를 URI (uniform resource identifier) 와 연관시키는 것을 포함할 수도 있게 하는 동작들을 수행하게 하도록 구성될 수도 있다.

[0022] 추가의 양태에서, 저장된 프로세서 실행가능 소프트웨어 명령들은, 프로세서로 하여금, 복수의 스크립트들의 각각에 대한 식별자를 생성하는 것이 적어도 하나의 스크립트에 대한 서명을 생성하는 것을 포함할 수도 있게 하는 동작들을 수행하게 하도록 구성될 수도 있다. 추가의 양태에서, 저장된 프로세서 실행가능 소프트웨어 명령들은, 프로세서로 하여금, 복수의 스크립트들의 각각에 대한 식별자를 생성하는 것이 적어도 하나의 스크립트의 텍스트를 포함하는 적어도 하나의 식별자를 생성하는 것을 포함할 수도 있게 하는 동작들을 수행하게 하도록 구성될 수도 있다.

[0023] 추가의 양태에서, 저장된 프로세서 실행가능 소프트웨어 명령들은, 프로세서로 하여금, 복수의 스크립트들을 발

견하기 위해 HTML 문서를 스캔하는 것이 제 1 프로세스에 의해 HTML 문서를 스캔하는 것을 포함할 수도 있게 하고, 스크립트 엔진이 실행을 위해 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 것이 제 2 프로세스에 의해 HTML 문서를 파싱하는 것을 포함할 수도 있게 하는 동작들을 수행하게 하도록 구성될 수도 있다. 추가의 양태에서, 저장된 프로세서 실행가능 소프트웨어 명령들은, 프로세서로 하여금, 실행을 위해 복수의 스크립트들을 준비하는 것이 제 2 프로세스가 제 2 스크립트를 파싱하고, 분석하고, 컴파일링하는 것과 병렬로 제 1 스크립트를 파싱하고, 분석하고, 컴파일링하는 것을 포함할 수도 있게 하는 동작들을 수행하게 하도록 구성될 수도 있다.

[0024] 추가의 양태에서, 저장된 프로세서 실행가능 소프트웨어 명령들은, 프로세서로 하여금, 스크립트 엔진이 실행을 위해 병렬로 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 것이 스크립트 엔진이 복수의 스크립트들이 실행되는 실행 순서와 상이한 준비 순서로 실행을 위해 복수의 스크립트들을 준비하는 동안 HTML 문서를 파싱하는 것을 포함할 수도 있게 하는 동작들을 수행하게 하도록 구성될 수도 있다. 추가의 양태에서, 저장된 프로세서 실행가능 소프트웨어 명령들은, 프로세서로 하여금, 복수의 스크립트들로부터 다음의 실행될 스크립트를 식별하는 것이 정의된 실행 순서에 기초하여 다음의 실행될 스크립트를 식별하는 것을 포함할 수도 있게 하는 동작들을 수행하게 하도록 구성될 수도 있다.

도면의 간단한 설명

[0025] 본 출원서에 통합되고 본 출원서의 일부를 구성하는 첨부 도면들은 본 발명의 예시적인 양태들을 도시한다. 위에서 주어진 전반적인 설명 및 아래에 주어지는 상세한 설명과 함께, 도면들은 개시된 양태들을 제한하지 않도록 본 발명의 특징들을 설명하기 위해 사용된다.

도 1은 다양한 양태들을 구현하는 컴퓨팅 디바이스들에서 사용될 수도 있는 일 예의 시스템-온-칩 (SOC) 아키텍처를 도시하는 구성요소 블록도이다.

도 2는 다양한 양태들을 구현하는데 사용될 수도 있는 일 예의 멀티코어 프로세서 아키텍처를 도시하는 기능 블록도이다.

도 3a는 HTML 문서를 찢기 위한 일 양태의 브라우저 방법을 도시하는 프로세스 흐름도이다.

도 3b는 일 양태의 브라우저 시스템에서 예의 논리적 컴포넌트들, 정보 흐름들, 동작들, 및 변환들을 도시하는 기능 및 프로세스 흐름도이다.

도 4는 일 양태의 브라우저 시스템에서 예의 논리적 컴포넌트들, 기능적 컴포넌트들, 정보 흐름들, 및 서브시스템들을 도시하는 기능 블록도이다.

도 5는 일 양태에 따라 병렬 브라우저 인프라스트럭처를 구현하는 양태의 브라우저 시스템을 도시하는 기능 블록도이다.

도 6은 페이지 로딩/렌더링 동작들 보다 앞서 자원들을 발견하고 프리-페치하기 위해 HTML 문서를 프로세싱하는 일 양태의 브라우저 방법을 예시하는 프로세스 흐름도이다.

도 7a는 문서 자원들의 사용을 예측하기 위해 추론 (speculation) 기법들 및 휴리스틱 (heuristic) 들을 사용하는 일 양태의 브라우저 방법을 도시하는 프로세스 흐름도이다

도 7b는 자원들을 병렬로 추론적으로 프리-페치하는 일 양태의 브라우저 방법을 도시하는 프로세스 흐름도이다.

도 7c는 스크립트들을 병렬로 프리프로세싱하는 일 양태의 브라우저 방법을 예시하는 프로세스 흐름도이다.

도 8은 프리-페치된 자원들을 프로세싱하는 일 양태의 브라우저 방법을 도시하는 프로세스 흐름도이다.

도 9는 다양한 양태들과 함께 사용하기에 적합한 CSS 엔진에서의 예의 기능적 컴포넌트들을 도시하는 기능 블록도이다.

도 10은 여러 노드들 상의 규칙 매칭 및 캐스케이딩 동작들을 병렬로 수행하는 일 양태의 스타일링 방법을 도시하는 프로세스 흐름도이다.

도 11a는 다양한 양태들에서의 사용에 적합한 일 예의 문서 오브젝트 모델 (document object model; DOM) 트리의 예시도이다.

도 11b는 도 11a에 예시된 DOM 트리에 대응하는 태스크 지향성 비순환 그래프 (task directed acyclic graph;

DAG) 의 예시도이다.

도 12는 다양한 양태들과 함께 사용하기에 적합한 일 예의 모바일 디바이스의 구성요소 블록도이다.

도 13은 다양한 양태들과 함께 사용하기에 적합한 일 예의 서버의 구성요소 블록도이다.

도 14는 다양한 양태들을 구현하는데 적합한 랩탑 컴퓨터의 구성요소 블록도이다.

발명을 실시하기 위한 구체적인 내용

[0026] 다양한 양태들이 첨부 도면들을 참조하여 상세히 설명될 것이다. 가능한 곳이라면 어디서든, 동일한 참조 번호들이 도면들 전체를 통해서 동일하거나 유사한 부분들을 참조하는데 사용될 것이다. 특정한 예들 및 구현예들에 대한 언급들은 예시의 목적을 위한 것이고, 발명 또는 청구항들의 범위를 제한하려는 의도는 아니다.

[0027] 웹 브라우저들은 다수의 표준들을 구현하고, 레거시 거동을 지원해야만 하며, 매우 동적이고 상호작용하는 복잡한 소프트웨어 애플리케이션들이다. 웹 브라우저 설계자들은 일반적으로 양호한 사용자 경험을 제공하기 위해 (긴 네트워크 레이턴시들의 존재시에도) 페이지 로드들에 대한 빠른 응답 시간, (예컨대, 웹 애플리케이션들에 대한 상호작용을 가능하게 하기 위한) 높은 성능, 및 높은 사용자 인터페이스 응답들의 최적의 혼합을 달성하는 것을 목표로 한다.

[0028] 다양한 양태들은 현대식 멀티프로세서 모바일 디바이스 아키텍처들에 의해 인에이블되는 동시성/병렬성을 이용하는 기법들을 통해 빠른 응답 시간, 높은 성능, 및 높은 사용자 인터페이스 응답을 달성하도록 구성된 웹 브라우저들, 브라우저 방법들, 및 브라우저 시스템들을 제공한다.

[0029] HTML (Hyper-Text Markup Language) 코드는 JavaScript® 코드를 내장하는 것 ("인라인 스크립트들" 이라 불림) 및 JavaScript® 코드로의 링크들을 포함하는 것 ("외부 스크립트들" 이라 불림) 일 수도 있다. HTML 문서를 정확히 프로세싱하기 위해, 인라인 및 외부 스크립트들 양자는 통상적으로 HTML 표준들에 의해 정의된 특정 순서로 실행된다. 즉, 그 표준들은 스크립트들의 최종 실행 순서가 유지될 것을 요구한다.

[0030] 다양한 양태 방법들 및 브라우저들은 병렬로 및/또는 순서없이 스크립트들을 다운로딩하고, 파싱하고, 분석하고, 컴파일링하며, 표준들에 의해 요구되는 최종 실행 순서로 스크립트를 실행하도록 구성될 수도 있다.

[0031] 일반적으로, HTML 문서에 포함된 (즉, 내장되거나 링크된) 스크립트들 모두는 실제로 실행되지 않으며, 사전에 실행을 위한 스크립트들 모두를 준비하는 것은 전력 및 프로세싱 자원들을 소비할 수도 있다. 다양한 양태들은 실행을 위해 준비될 스크립트들을 지능적으로 선택한다.

[0032] 다수의 스크립트들이 병렬로 다운로드되고, 파싱되고, 분석되고, 컴파일링될 때, 스크립트들이 실행을 위해 준비되는 순서는 HTML 표준들에 의해 정의된 특정 실행 순서와 상이할 수도 있다. 스크립트가 실행을 위해 준비되지 않지만, HTML 표준들에 의해 정의된 특정 실행 순서에서 다음 스크립트라면, 브라우저는 스크립트가 HTML 문서의 임의의 추가의 프로세싱을 수행하기 전에 실행을 위해 준비될 때까지 대기하도록 요구될 수도 있다. 다양한 양태들은 이러한 대기 시간을 활용하여 (HTML 표준들에 의해 조정되지 않는) 실행을 위한 다른 스크립트들 또는 리소스들을 준비한다. 다수의 스크립트들 및 자원들은 병렬로 및/또는 다른 스크립트들의 실행 동안 준비될 수도 있다.

[0033] 단어 "예시적"은 본원에서는 "예, 사례, 또는 예시로서 역할을 한다"는 의미로 사용된다. "예시적인" 것으로서 본 명세서에서 설명된 어떤 구현예라도 다른 구현예들보다 바람직하거나 유익하다고 생각할 필요는 없다.

[0034] 용어들인 "모바일 디바이스"와 "컴퓨팅 디바이스"는, 셀룰러 전화기들, 스마트폰들, 개인 또는 모바일 멀티미디어 플레이어들, 개인휴대 정보단말들 (PDA들), 랩톱 컴퓨터들, 태블릿 컴퓨터들, 스마트북들, 팜탑 컴퓨터들, 무선 전자 메일 수신기들, 멀티미디어 인터넷 가능 셀룰러 전화기들, 무선 게이밍 제어기들, 그리고 프로그램가능 프로세서 및 메모리를 포함한 유사한 개인용 전자 디바이스들 중 임의의 하나 또는 전부를 지칭하기 위해 본원에서 교환적으로 사용된다. 다양한 양태들이 제한된 프로세싱 능력을 가질 수도 있는 모바일 디바이스들, 이를테면 셀룰러 전화기들에서 특히 유용하지만, 그 양태들은 동적 (dynamic), 스크립팅 및/또는 마크업 언어들로 작성된 스크립트들 및/또는 애플리케이션들을 실행하는 임의의 컴퓨팅 디바이스에서 일반적으로 유용하다.

[0035] 용어 "시스템 온 칩" (system on chip; SOC) 은 단일 기판 상에 통합된 다수의 자원들 및/또는 프로세서들을 포함하는 단일 집적회로 (IC) 칩을 지칭하기 위해 본원에서 사용된다. 단일 SOC가 디지털, 아날로그, 혼합된 신호, 및 라디오-주파수 기능들을 위한 회로를 포함할 수도 있다. 단일 SOC가 또한 임의의 수의 범용 및/또는 특화된 프로세서들 (디지털 신호 프로세서들, 모뎀 프로세서들, 비디오 프로세서들 등), 메모리 블록들 (예

컨대, ROM, RAM, 플래시 등), 및 하드웨어 자원들 (예컨대, 타이머들, 전압 조정기들, 발진기들 등) 을 포함할 수도 있다. SOC들은 또한 통합된 하드웨어 자원들 및 프로세서들을 제어하기 위한, 뿐만 아니라 주변 디바이스들을 제어하기 위한 소프트웨어를 포함할 수도 있다.

[0036] 용어 "멀티코어 프로세서"는 프로그램 명령들을 읽고 실행하도록 구성된 둘 이상의 독립 프로세싱 코어들 (예컨대, CPU 코어들) 을 포함하는 단일 집적회로 (IC) 칩 또는 칩 패키지를 지칭하기 위해 본원에서 사용된다. SOC가 다수의 멀티코어 프로세서들을 포함할 수도 있고, SOC에서의 각각의 프로세서가 코어라고 지칭될 수도 있다. 용어 "멀티프로세서"는 프로그램 명령들을 읽고 실행하도록 구성된 둘 이상의 프로세싱 유닛들을 포함하는 시스템 또는 디바이스를 지칭하기 위해 본원에서 사용된다.

[0037] 본 출원에서 사용된 바와 같이, 용어들 "컴포넌트", "모듈", "시스템", "엔진", "관리자" 등은, 특정 동작들 또는 기능들을 수행하도록 구성된 컴퓨터 관련된 엔티티, 이를테면 하드웨어, 펌웨어, 하드웨어 및 소프트웨어의 조합, 소프트웨어, 또는 실행 중인 소프트웨어 (software in execution) 등을 포함하지만 그것들로 제한되지 않는 것으로 의도된다. 예를 들어, 컴포넌트는 프로세서 상에서 실행중인 프로세스, 프로세서, 오브젝트, 실행가능물 (executable), 실행 스레드 (thread of execution), 프로그램, 및/또는 컴퓨터일 수도 있지만 그것들로 제한되지 않는다. 예시로서, 컴퓨팅 디바이스 상에서 실행중인 애플리케이션 및 컴퓨팅 디바이스 둘 다가 하나의 컴포넌트라고 지칭될 수도 있다. 하나 이상의 컴포넌트들이 프로세스 및/또는 실행 스레드 내에 상주할 수도 있고 컴포넌트가 하나의 프로세서 또는 코어 상에서 국소화되고 그리고/또는 둘 이상의 프로세서들 또는 코어들 사이에서 분산될 수도 있다. 덧붙여서, 이들 컴포넌트들은 다양한 명령들 및/또는 데이터 구조들을 저장하고 있는 다양한 비일시적 컴퓨터 판독가능 매체들로부터 실행될 수도 있다. 컴포넌트들은 로컬 및/또는 원격 프로세스들, 함수 또는 프로시저 호출들, 전자 신호들, 데이터 패킷들, 메모리 읽기/쓰기들, 및 다른 알려진 컴퓨터, 프로세서, 및/또는 프로세스 관련된 통신 수법들을 통해 통신할 수도 있다.

[0038] 용어 "애플리케이션 프로그래밍 인터페이스"와 그것의 머리글자 "API"는 제 2 소프트웨어 컴포넌트와 통신하기 위해 제 1 소프트웨어 컴포넌트에 의해 사용될 수도 있는 임의의 소프트웨어 인터페이스를 지칭하기 위해 본 출원에서 일반적으로 사용된다. API는 루틴들, 프로시저들, 함수들, 메소드들, 데이터 구조들, 오브젝트 클래스들, 및 변수들을 위한 사양들을 포함할 수도 있다. API는 그 API를 다른 하이-레벨 프로그래밍 언어의 특징들 (구문론적 또는 의미론적임) 에 매핑하기 위한 설비 (facility) 들을 또한 포함할 수도 있다. 이러한 설비들 및/또는 매핑들은 그것들 자체가 API들일 수도 있고, "언어 바인딩들" 또는 "바인딩들"로서 알려져 있다.

[0039] 용어 "마크업 언어"는 프로세서가 텍스트로부터 주석들을 신택스적으로 구별할 수도 있도록 텍스트를 주석달기 하기 위한 임의의 프로그래밍 언어 및/또는 시스템을 지칭하기 위해 본 출원에서 일반적으로 사용된다. 마크업 언어들의 예들은 스크라이브 (Scribe), SGML (Standard Generalized Markup Language), HTML (Hyper-Text Markup Language), XML (Extensible Markup Language), 및 XHTML (Extensible Hyper-Text Markup Language) 을 포함한다.

[0040] 용어들인 "동적 언어" 및 "스크립팅 언어"는, 임의의 동적 언어, 스크립팅 언어, 또는 실행시간에 인터프리트되는 및/또는 컴파일되는 프로그램들 (본원에서는 "스크립트들"임) 을 작성하는데 사용된 임의의 언어를 지칭하기 위해 본 출원에서 일반적으로 그리고 교환적으로 사용된다. 이들 용어들은 관리된 실행시간에 실행되고 동적으로 컴파일되는 임의의 언어를 또한 지칭할 수도 있다. 따라서, 본 출원의 목적들을 위해, 다양한 양태들의 설명에서의 "동적 언어" 및 "스크립팅 언어"라는 용어들의 사용은, 소스 코드 또는 바이트코드로부터 인터프리트되는 언어들로, 또는 네이티브 머신 코드가 되도록 전통적으로 컴파일되는 프로그램들과 함께 실행하는 것들로 청구범위를 제한하는 것으로 해석되지 않아야 한다. 본 출원의 범위 내의 동적 및 스크립팅 언어들의 예들은, 예를 들어, JavaScript®, 펄, 파이썬, 및 루비, 뿐만 아니라 미래에 개발될 수도 있는 다른 유사한 언어들을 포함한다.

[0041] 용어들인 "스타일 시트 언어"와 "스타일 언어"는 문서의 프레젠테이션 스타일이 문서의 내용으로부터 분리될 수도 있도록 구조화된 문서들의 프레젠테이션을 표현하는 임의의 컴퓨터 언어를 지칭하기 위해 본 출원에서 일반적으로 사용된다. 스타일 시트 언어의 일 예가 CSS (Cascading Style Sheets) 인데, 그것은 마크업 언어로 작성된 문서의 프레젠테이션 시맨틱스를 기술하기 위해 통상적으로 사용된다.

[0042] 참조의 편이를 위해, 본 출원 전체를 통해, HTML은 예시적인 마크업 언어로서 사용되며, CSS는 예시적인 스타일 시트 언어로서 사용되고, JavaScript®는 예시적인 동적 스크립팅 언어로서 사용된다. 그러나, 본 출원에서의 HTML, CSS, 및 JavaScript®의 사용은 단지 예시의 목적을 위한 것이고, 청구항들에 의해 명시적으로 언급되

지 않는 한 청구항들의 범위를 특정 언어로 제한하는 것으로 해석되지 않아야 한다는 것에 주의해야 한다.

[0043] HTML은 ISO/IEC 15445 표준을 구현하는 마크업 언어이다. HTML은 소프트웨어 애플리케이션, 이를테면 웹 브라우저에 의해 디스플레이될 수 있도록 웹 페이지들을 기술하는데 사용된 마크업 태그들 (예컨대, 주석들) 의 세트로서 특징화될 수도 있다. HTML은 텍스트를 위한 구조적 시맨틱스, 이를테면 머리글 (heading) 들, 단락들, 리스트들, 링크들, 인용 (quote) 들, 및 다른 아이템들에 의해 표시됨으로써 구조화된 문서들의 생성을 허용한다.

[0044] JavaScript®는 ECMAScript 언어 표준 (ECMA International에 의해 ECMA-262 규격으로 표준화됨) 및/또는 ISO/IEC 16262 표준을 구현하는 동적, 약한 타입형 (weakly typed), 객체 지향 스크립팅 언어이다. JavaScript®는 호스트 환경 내의 계산적 (computational) 오브젝트들, 이를테면 모바일 디바이스 프로세서 상에서 실행하는 웹 브라우저들에 프로그램적 액세스를 가능하게 한다.

[0045] CSS (Cascading Style Sheets) 는 웹 사이트들의 룩 및 포맷팅을 기술하는데 사용된 스타일 언어이고, 문서의 프레젠테이션을 그것의 내용으로부터 분리하는데 사용하기 위한 것이다. 각각의 스타일 시트는 다음의 포맷을 갖는 규칙들의 순서화된 컬렉션을 포함할 수도 있다: selector {property$_1$: value;...property$_n$: value; }.

일 예로서, 다음의 CSS 코드는 직계 조상 (direct ancestor) 이 적색 배경 위에 백색 전경 (foreground) 을 사용하는 <p> 엘리먼트인 모든 <cite> 엘리먼트들을 렌더링할 것을 브라우저에게 알린다: p > cite { color: white; background-color: red; }. 웹사이트들이 수만 개의 이러한 규칙들을 포함하는 것은 드물지 않다.

[0046] HTML은 포함하는 HTML 페이지의 거동 및/또는 프레젠테이션에 영향을 미칠 수 있는 JavaScript® 코드에 대한 링크들을 내장 및/또는 포함할 수도 있다. 내장된/링크된 JavaScript® 코드는 부가적인 HTML 코드를 또한 생성할 수도 있는데, 그 부가적인 HTML 코드 (즉, JavaScript®가 내장된 HTML 코드) 는 포함하는 HTML 페이지 속에 내장될 수 있다.

[0047] JavaScript®는 기능들이 HTML 페이지의 문서 오브젝트 모델 (DOM) 과 상호작용하고, 조작하도록 그 기능들을 HTML 코드 속에 내장하는데 사용될 수도 있다. DOM은 HTML에서 오브젝트들을 표현하고 그 오브젝트들과 상호작용하는 언어-독립적인 협약이고, 포함하는 HTML 페이지에 JavaScript® 코드가 액세스하며 그 포함하는 HTML 페이지를 조작하는 것을 허용한다. DOM 트리가, 웹 페이지를 정의하는 개별 컴포넌트들의 컴포넌트들, 상대적 구조, 관계들, 및 거동을 식별하기 위해 그 웹 페이지를 렌더링하는 일부로서 통상적으로 생성된다.

[0048] HTML은 CSS 코드를 포함 (예컨대, 내장 및/또는 그것에 링크) 할 수 있다. 별개의 파일들로서 특정된 CSS 코드는 원격 서버들 상에 저장될 수도 있다. 종래의 CSS 프로세싱 엔진들 (예컨대, WebKit 또는 파이어폭스) 은 메인 브라우저 스레드에서 CSS를 순차적으로 파싱하고, 높은 정도의 병렬성 또는 동시성을 지원하지 않는다. 예를 들어, CSS 코드가 HTML 문서 속에 내장된 경우, HTML 파서는 CSS 엔진이 HTML 문서의 헤더에서의 스타일 엘리먼트들을 파싱하기까지 HTML 문서의 나머지 부분들을 파싱할 수 없다. HTML 문서가 여러 CSS 파일들에 대한 링크들을 포함하는 경우, 종래의 CSS 프로세싱 엔진들은 모든 링크된 CSS 파일들을 순차적으로 파싱할 것이다. 이들 및 다른 이유들로, 종래의 CSS 프로세싱 엔진들은, 특히 큰 CSS 파일들 (이것이 일반적임) 의 경우에 심각한 둔화를 유발할 수도 있다.

[0049] 다양한 양태 방법들 및 브라우저들은 페이지 로드들, 웹 애플리케이션들, 및 네트워크 통신들의 속도 및 효율성을 개선하기 위해 현대의 모바일 디바이스들에서 사용가능한 병렬성을 이용한다.

[0050] 다양한 양태들은, 정보의 불완전한 세트로부터 요청될 가능성이 있는 자원들을 식별하는 추론/예측 기법들을 사용하여 웹 문서 (HTML 페이지) 를 프리프로세싱하고, 웹 문서의 적절한 렌더링을 위해 요청될 높은 확률을 갖도록 결정된 자원들을 요청/프리-페치함으로써 웹페이지를 로딩/렌더링하는 브라우저 시스템들 및 방법들을 포함한다. 이들 자원들의 프리-페칭은 웹 브라우저 (와 이에 따라 모바일 디바이스) 가 이용가능한 대역폭을 더 잘 이용하며, 전송 레이턴시들을 중첩시키고, 문서 로드 시간들을 개선하는 것을 가능하게 할 수도 있다.

[0051] 최근에, 모바일 전자 디바이스들 (예컨대, 셀룰러 폰들, 태블릿들, 랩톱들 등) 은 더욱 피처가 풍부하게 되고, 이제 일반적으로, 모바일 디바이스 사용자들이 그들의 모바일 디바이스들 상에서 복잡하고 파워 집중적인 소프트웨어 애플리케이션들 (예컨대, 웹 브라우저들, 비디오 스트리밍 애플리케이션들 등) 을 실행하는 것을 허용하는 다수의 프로세서들, 시스템-온-칩 (SoC) 들, 다수의 메모리들, 및 다른 리소스들을 포함한다. 이들 및 다른 개선들로 인해, 스마트폰들과 태블릿 컴퓨터들은 인기리에 성장하였고, 많은 사용자들에 의해 선택되는 플랫폼으로서 랩톱들과 데스크톱 머신들을 대체하고 있다. 모바일 디바이스 사용자들은 그들의 모바일 디바이스의 웹 브라우저를 통해 인터넷에 액세스함으로써 쉽고 편리하게 많은 그들의 매일의 태스크들을 이제 달성할

수 있다.

[0052] 다양한 양태들은 고속 프로세서들 및 멀티프로세서 모바일 디바이스 아키텍처들에 의해 인에이블되는 동시성/병렬성을 이용할 뿐만 아니라 자원들의 추론적 프로세싱 및 프리-페칭을 사용하고, 이에 따라 네트워크 레이턴시를 은닉하고 전체 사용자 경험을 개선시킴으로써 빠른 응답 시간, 높은 성능, 및 높은 사용자 인터페이스 응답을 달성하도록 구성된 브라우저 방법들 및/또는 웹 브라우저들을 제공한다.

[0053] 웹 브라우저들은 다수의 표준들을 구현하고, 레거시 거동을 지원해야만 하며, 매우 동적이고 상호작용하는 복잡한 애플리케이션들이다. 웹 브라우저 설계자들은 일반적으로 (예컨대, 양호한 사용자 경험을 제공하기 위해) (긴 네트워크 레이턴시들의 존재시에도) 페이지 로드들에 대한 빠른 응답 시간, (예컨대, 웹 애플리케이션들에 대한 상호작용을 가능하게 하기 위한) 높은 성능, 및 높은 사용자 인터페이스 응답들의 최적의 혼합을 달성하는 것을 목표로 한다.

[0054] 웹 브라우저들에서 동시성을 이용하는 것은 비교적 새로운 접근법이다. 대부분의 기존의 브라우저들 (예컨대, 파이어폭스 및 WebKit 기반 크롬 및 사파리 브라우저들) 은 상호작용성을 도울 이벤트 구동 모델 (event driven model) 들을 사용하는 순차적 엔진들로서 근본적으로 아키텍처링된다. 모바일 디바이스 및/또는 브라우저 서브시스템들 사이의 다수의 의존성들로 인해 (그리고 많은 기존의 데이터 구조들이 스레드 안정적이지 않기 때문에) 이들 기존의 솔루션들은 높은 정도의 병렬성 또는 동시성을 제공하지 못한다.

[0055] 크롬과 WebKit2는 각각의 브라우저 탭에 대해 별개의 프로세스들을 생성하는데, 그 브라우저 탭은 상이한 웹 사이트들 간에 얼마간의 격리를 제공하지만 다수의 코어들을 사용하는 책임을 운영 체제에게 준다. 덧붙여서, 이들 프로세스들은 메모리 및 시동 오버헤드 양쪽 모두의 측면들에서 헤비급이다. 이처럼, 이들 솔루션들은 개개의 페이지 로드들을 스피드업하지 않거나 또는 네트워크 통신들의 효율을 개선하지 않고, 동일한 애플리케이션의 다수의 인스턴스들을 실행하는 것에 관해 단순히 병렬성을 지원한다. 이러한 탭-레벨 병렬성은 모바일 브라우저들의 요구를 해결하지 못하여서, 단일-탭 성능이 종종 부적합하고 사용자들이 많은 탭들을 한 번에 열지 못한다.

[0056] OP 및 OP2 브라우저들은 ("웹 인스턴스"라 지칭되는) 웹 페이지 당 프로세스들의 새로운 컬렉션을 생성할 수도 있고, 브라우저 컴포넌트들 (예컨대, 네트워킹) 은 상이한 프로세스들에서 실행할 수도 있다. 그러나, 이들 솔루션들은, 모든 다른 기존의 브라우저 솔루션들처럼, 본질적으로 순차적이다. 예를 들어, 네트워크 동작이 파싱 동작과는 별개의 프로세스에서 수행될 수도 있지만, 네트워크 프로세스는 여전히 파싱 프로세스를 기다려야만 하는데 (반대의 경우도 마찬가지임) 각각의 동작이 다른 동작에 의존하기 때문이다. 다시 말하면, OP 및 OP2 브라우저들이 다수의 프로세스들 또는 스레드들의 사용을 허용하지만, 이들 솔루션들은 웹페이지를 렌더링함에 있어서 높은 정도의 병렬성을 달성하지 못하는데, 그 솔루션들이 웹페이지들을 다운로딩, 프로세싱, 및 렌더링하기 위해 브라우저 프로세싱 알고리즘들의 직렬/순차적 성질을 다루지 못하기 때문이다.

[0057] 다양한 양태들은, 기존의 브라우저 프로세싱 알고리즘들의 직렬/순차적 성질을 극복하며, 고속 프로세서들 및 멀티프로세서 모바일 디바이스 아키텍처들의 멀티-스레드 실행 및 병렬 프로세싱 능력들을 이용하고, 브라우저 성능을 개선하며 네트워크 레이턴시를 감소시키고 모바일 디바이스들의 사용자들에 대한 사용자 경험을 개선하기 위해 병렬성을 널리 이용하도록 구성된 고-성능 웹 브라우저를 포함한다.

[0058] 다양한 양태들은 시스템-온-칩 (SOC) 을 포함하는 다수의 단일 프로세서 및 멀티프로세서 컴퓨터 시스템들 상에 구현될 수도 있다. 도 1은 다양한 양태들을 구현하는 컴퓨팅 디바이스들에서 사용될 수도 있는 일 예의 시스템-온-칩 (SOC; 100) 아키텍처를 도시한다. SOC (100) 는 다수의 이종 프로세서들, 이를테면 디지털 신호 프로세서 (DSP; 102), 모뎀 프로세서 (104), 그래픽 프로세서 (106), 및 애플리케이션 프로세서 (108) 를 포함할 수도 있다. SOC (100) 는 이종 프로세서들 (102, 104, 106, 108) 중 하나 이상에 접속된 하나 이상의 코프로세서들 (110) (예컨대, 벡터 코-프로세서) 을 또한 포함할 수도 있다. 각각의 프로세서 (102, 104, 106, 108, 110) 는 하나 이상의 코어들을 포함할 수도 있고, 각각의 프로세서/코어는 다른 프로세서들/코어들과는 독립적인 동작들을 수행할 수도 있다. 예를 들어, SOC (100) 는 제 1 유형의 운영 체제 (예컨대, FreeBSD, LINUX, OS X 등) 를 실행하는 프로세서와 제 2 유형의 운영 체제 (예컨대, 마이크로소프트 윈도우 8) 를 실행하는 프로세서를 포함할 수도 있다.

[0059] SOC (100) 는 또한 센서 데이터, 아날로그-디지털 변환들, 무선 데이터 송신들을 관리하기 위한, 그리고 웹 브라우저에서 렌더링하기 위해 인코딩된 오디오 및 비디오 신호들을 프로세싱하는 것과 같은 다른 특수 동작들을 수행하기 위한 아날로그 회로 및 커스텀 회로 (114) 를 포함할 수도 있다. 그 SOC (100) 는 시스템 컴포넌

트들 및 자원들 (116), 이를테면 전압 조정기들, 발진기들, 위상 잠금 루프들, 주변 브리지들, 데이터 제어기들, 메모리 제어기들, 시스템 제어기들, 액세스 포트들, 타이머들, 그리고 컴퓨팅 디바이스 상에서 실행 중인 소프트웨어 클라이언트들 (예컨대, 웹 브라우저) 과 프로세서들을 지원하는데 사용되는 다른 유사한 컴포넌트들을 더 포함할 수도 있다.

[0060] 시스템 컴포넌트들 및 자원들 (116) 및/또는 커스텀 회로 (114) 는 주변 디바이스들, 이를테면 카메라들, 전자 디스플레이들, 무선 통신 디바이스들, 외부 메모리 칩들 등과 인터페이싱하는 회로를 포함할 수도 있다. 프로세서들 (102, 104, 106, 108) 은 하나 이상의 메모리 엘리먼트들 (112), 시스템 컴포넌트들 및 자원들 (116), 그리고 커스텀 회로 (114) 에 상호접속/버스 모듈 (124) 을 통해 상호접속될 수도 있으며, 그 상호접속/버스 모듈은 재구성가능 로직 게이트들의 어레이를 포함하고 및/또는 버스 아키텍처 (예컨대, CoreConnect, AMBA 등) 를 구현할 수도 있다. 통신들이 고급 상호접속들, 이를테면 고 성능 NoC들 (networks-on chip) 에 의해 제공될 수도 있다.

[0061] SOC (100) 는 SOC 외부의 자원들, 이를테면 클록 (118) 및 전압 조정기 (120) 와 통신하기 위한 입력/출력 모듈 (도시되지 않음) 을 더 포함할 수도 있다. SOC 외부의 자원들 (예컨대, 클록 (118), 전압 조정기 (120)) 은 내부 SOC 프로세서들/코어들 (예컨대, DSP (102), 모뎀 프로세서 (104), 그래픽 프로세서 (106), 애플리케이션 프로세서 (108) 등) 중 둘 이상에 의해 공유될 수도 있다.

[0062] 위에서 논의된 SOC (100) 외에도, 다양한 양태들이, 단일 프로세서, 다수의 프로세서들, 다중코어 프로세서들, 또는 그것들 중 임의의 조합을 포함할 수도 있는 매우 다양한 컴퓨팅 시스템들에서 구현될 수도 있다.

[0063] 도 2는 다양한 양태들을 구현하기 위해 사용될 수도 있는 일 예의 멀티코어 프로세서 아키텍처를 도시한다. 멀티코어 프로세서 (202) 는 (예컨대, 단일 기판, 다이, 통합된 칩 등 상에서) 가까이 근접한 둘 이상의 독립 프로세싱 코어들 (204, 206, 230, 232) 을 포함할 수도 있다. 프로세싱 코어들 (204, 206, 230, 232) 의 근접은, 신호들이 칩 밖으로 이동해야 하는 경우에 가능한 것보다 훨씬 높은 주파수/클록-레이트에서 메모리가 동작하는 것을 허용한다. 더구나, 프로세싱 코어들 (204, 206, 230, 232) 의 근접은 온 칩 메모리 및 자원들 (예컨대, 전압 레일) 의 공유, 뿐만 아니라 코어들 간의 더욱 조정된 협력을 허용한다.

[0064] 멀티코어 프로세서 (202) 는 레벨 1 (L1) 캐시들 (212, 214, 238, 240) 및 레벨 2 (L2) 캐시들 (216, 226, 242) 을 포함하는 멀티-레벨 캐시를 포함할 수도 있다. 멀티코어 프로세서 (202) 는 버스/상호접속 인터페이스 (218), 메인 메모리 (220), 및 입력/출력 모듈 (222) 을 또한 포함할 수도 있다. L2 캐시 (216, 226, 242) 는 L1 캐시들 (212, 214, 238, 240) 보다 클 (그리고 느릴) 수도 있지만, 메인 메모리 유닛 (220) 보다 작을 (그리고 실질적으로 빠를) 수도 있다. 각각의 프로세싱 코어 (204, 206, 230, 232) 는 L1 캐시 (212, 214, 238, 240) 에 대한 프라이빗 액세스를 가지는 프로세싱 유닛 (208, 210, 234, 236) 을 포함할 수도 있다. 프로세싱 코어들 (204, 206, 230, 232) 은 L2 캐시 (예컨대, L2 캐시 (242)) 에 대한 액세스를 공유할 수도 있거나 또는 독립적인 L2 캐시 (예컨대, L2 캐시 (216, 226)) 에 액세스를 할 수도 있다.

[0065] L1 및 L2 캐시들은 프로세싱 유닛들에 의해 빈번하게 액세스되는 데이터를 저장하는데 사용될 수도 있는 반면, 메인 메모리 (220) 는 프로세싱 코어들 (204, 206, 230, 232) 에 의해 액세스되고 있는 더 큰 파일들 및 데이터 단위들을 저장하는데 사용될 수도 있다. 멀티코어 프로세서 (202) 는 프로세싱 코어들 (204, 206, 230, 232) 이 L1 캐시를 먼저, L2 캐시를 그 다음에 쿼리 (query) 하고, 정보가 그 캐시들에 저장되어있지 않으면 메인 메모리에 쿼리하는 순서로 메모리로부터 데이터를 찾도록 구성될 수도 있다. 정보가 캐시들 또는 메인 메모리 (220) 에 저장되어 있지 않으면, 멀티코어 프로세서 (202) 는 외부 메모리 및/또는 하드 디스크 메모리 (224) 로부터 정보를 찾을 수도 있다.

[0066] 프로세싱 코어들 (204, 206, 230, 232) 은 서로 버스/상호접속 인터페이스 (218) 로 통신할 수도 있다. 각각의 프로세싱 코어 (204, 206, 230, 232) 는 일부 자원들에 대해 단독 제어 (exclusive control) 를 하고 다른 자원들을 나머지 코어들과 공유할 수도 있다.

[0067] 프로세싱 코어들 (204, 206, 230, 232) 은 서로 동일하거나, 이종일 수도 있고 그리고/또는 상이한 특수 기능들을 구현할 수도 있다. 따라서, 프로세싱 코어들 (204, 206, 230, 232) 은 (예컨대, 상이한 운영 체제들을 실행할 수도 있는) 운영 체제 관점 또는 (예컨대, 상이한 명령 세트들/아키텍처들을 구현할 수도 있는) 하드웨어 관점 중 어느 하나에서, 대칭적일 필요는 없다.

[0068] 멀티프로세서 하드웨어 설계들, 이를테면 도 1 및 도 2를 참조하여 위에서 논의된 것들은, 동일한 패키지 내부에, 종종 실리콘의 동일한 조각 상에 상이한 능력들의 다수의 프로세싱 코어들을 포함할 수도 있다. 대칭적

멀티프로세싱 하드웨어는 단일 운영 체제에 의해 제어되는 단일 공유 메인 메모리에 접속된 둘 이상의 동일한 프로세서들을 포함한다. 비대칭적 또는 "느슨하게 커플링된 (loosely-coupled)" 멀티프로세싱 하드웨어가, 독립적인 운영 체제에 의해 각각 제어되고 하나 이상의 공유된 메모리들/자원들에 접속될 수도 있는 둘 이상의 이종 프로세서들/코어들을 포함할 수도 있다.

[0069] 도 3a는 HTML 문서를 로딩하고 렌더링하는 일 양태의 브라우저 방법 (300) 을 도시한다. 블록 302에서, 웹 브라우저 컴포넌트가 특정 URL (uniform resource locator) 에 위치된 HTML 문서의 로딩을 요청하는 사용자 입력을 수신할 수도 있다. 블록 304에서, 웹 브라우저 컴포넌트는 인터넷을 경유하여 통신되는 잘 알려진 하이퍼텍스트 전송 프로토콜 (HTTP) 메시지들을 통해 그 URL에 위치된 웹 서버로부터 HTML 문서를 요청할 수도 있다. 블록 306에서, 웹 브라우저 컴포넌트는 그 URL에 위치된 웹 서버로부터 HTML 문서를 수신할 수도 있다. 블록 308에서, 웹 브라우저 컴포넌트는 HTML 파일에서 참조된 외부 자원들 (이미지들, 오디오, CSS 등) 을 식별/발견하기 위해 수신된 HTML 문서를 파싱할 수도 있다.

[0070] 블록 310에서, 웹 브라우저 컴포넌트는 자원들이 유지되는 네트워크 서버들로부터 식별된 외부 자원들을 요청할 수도 있는데, 그 네트워크 서버들은 HTML 문서를 제공했던 서버 또는 인터넷을 통해 액세스가능한 임의의 다른 서버를 포함할 수도 있다. 블록 312에서, 웹 브라우저 컴포넌트는 요청된 외부 자원들을 네트워크 서버로부터 수신할 수도 있다. 결정 블록 314에서, 웹 브라우저 컴포넌트는 수신된 자원들 중 임의의 것이 다른 외부 자원들을 참조하는지의 여부를 결정할 수도 있다.

[0071] 수신된 자원들이 다른 외부 자원들을 참조한다고 웹 브라우저 컴포넌트가 결정하는 경우 (즉, 결정 블록 314 = "예"), 웹 브라우저는 블록 310 ~ 블록 314에서 새로이 수신된 자원들에 의해 참조되는 그들 다른/부가적인 외부 자원들을 요청/수신할 수도 있다. 이들 동작들은 모든 참조된 외부 자원들이 다운로드되기까지 반복적으로 수행될 수도 있다.

[0072] 수신된 자원들이 임의의 부가적인 외부 자원들을 참조하지 않는다고 웹 브라우저가 결정하는 경우 (즉, 결정 블록 314 = "아니오"), 블록 316에서, 웹 브라우저는 웹페이지를 적절히 렌더링하는데 필요한 자원들을 결정하기 위해 수신된 외부 자원들을 분석할 수도 있다. 블록 318에서, 웹 브라우저는 요청된 다운로드 자원들을 사용하여 웹페이지를 렌더링할 수도 있다.

[0073] 도 3b는 일 양태의 브라우저 시스템 (350) 에서의 예의 논리적 컴포넌트들, 정보 흐름들, 동작들, 및 변환들을 도시한다. 브라우저 시스템 (350) 은, 프로세서로 하여금 인터넷으로부터 정보 및/또는 자원들을 취출하고 컴퓨팅 디바이스 (예컨대, 모바일 디바이스) 의 전자 디스플레이 상에 웹페이지들을 렌더링하기 위한 다양한 동작들을 수행하게 구성된 소프트웨어 애플리케이션/모듈일 수도 있다.

[0074] 브라우저 시스템 (350) 은 외부 모듈들 (380) 과의 상호작용성을 제공하기 위해 다양한 스테이지들에서 그리고/또는 다양한 동작들 동안 (예컨대, 페이지 로드 동작들 등의 동안 및 후에) 웹 페이지와 상호작용하도록 구성된 스크립팅 컴포넌트 (362) 를 포함할 수도 있다. 외부 모듈들 (380) 은 사용자 I/O 모듈들 (예컨대, 마우스, 키보드 등) 및/또는 애플리케이션 모듈들 (예컨대, 플러그-인들, GPS 등) 을 포함할 수도 있다. 일 양태에서, 스크립팅 (362) 컴포넌트는 JavaScript® 코드를 컴파일하고 및/또는 실행하도록 구성된 JavaScript® 엔진을 포함할 수도 있다.

[0075] 블록 354에서, 브라우저 시스템 (350) 은 웹 (352) 에서의 서버로부터 (예컨대, HTTP를 통해) 프로그래밍 명령들 (356) 을 요청/수신하기 위해 페치 동작을 수행할 수도 있다. 블록 358에서, 브라우저 시스템 (350) 은 HTML 코드 (360) 를 생성하기 위해 수신된 프로그래밍 명령들 (356) 을 해석/디코딩할 수도 있다. 생성된 HTML (360) 코드는 JavaScript® 코드를 포함 (즉, 그 코드에 대한 참조들을 내장 또는 포함) 할 수도 있으며, 그 코드의 실행은 포함하는 HTML 페이지 속으로 삽입하기 위한 부가적인 HTML 코드 (예컨대, JavaScript®가 포함된 HTML 코드) 를 생성할 수도 있다. 이러한 생성된 HTML 코드는 HTML 페이지의 거동 및/또는 프레젠테이션에 영향을 미칠 수도 있다. 생성된 HTML (360) 코드는 스타일 시트들 및/또는 CSS 코드를 또한 포함할 수도 있다.

[0076] 블록 364에서, 브라우저 시스템 (350) 은 HTML 문서의 문서 오브젝트 모델 (DOM; 366) 을 생성하기 위해 HTML (360) 코드 (및 내장된/참조된 JavaScript® 코드) 를 파싱할 수도 있다. DOM (366) 은 HTML 코드에서 다양한 오브젝트들의 콘텐츠들, 관계들, 스타일들, 및 포지션들을 표현할 수도 있다. 브라우저 "패스 (pass) 들" 및 컴포넌트들 사이의 통신들은 DOM (366) 을 통해 일어날 수도 있다. "브라우저 패스"가 HTML 문서의 관련 부분들을 통한 단일 반복에 연관된 스레드, 프로세스, 또는 애플리케이션일 수도 있다. 일 실시형태에

서, 브라우저 패스가 "작업 아이템"일 수도 있다

[0077] 위에서 언급했듯이, JavaScript® 코드는 HTML 코드에 내장될 수도 있고, 동시에, 포함하는 HTML 페이지 속으로 삽입될 부가적인 HTML 코드를 생성할 수도 있다. 코드의 삽입을 가능하게 하 (고 적절한 순서를 보장하) 기 위해 두 개의 상이한 프로세스들이 JavaScript® 코드 및 포함하는 HTML 코드를 해석, 파싱, 및 실행하기 위해 요청될 수도 있다. 따라서, 일 양태에서, 블록 364의 파싱 동작들은 다수의 프로세스들 또는 애플리케이션들에 의해 수행될 수도 있다.

[0078] 블록 368에서, 브라우저 시스템 (350) 은, 예를 들어, 하나 이상의 스타일 시트들 (예컨대, CSS) 을 HTML 문서에 그리고/또는 생성된 DOM (366) 트리에 적용함으로써 수정된 DOM 트리 (370) 를 생성하기 위해 스타일 동작들을 수행할 수도 있다.

[0079] 블록 372에서, 브라우저 시스템 (350) 은 레이아웃 동작들을 수행함으로써 페이지 레이아웃 (374) 을 "풀이 (solve)"할 수도 있다. 일 양태에서, 레이아웃 동작들은, 페이지를 디스플레이하는데 필요한 부가적인 내용이 이용가능하게 될 (예컨대, 다운로드되며, 프로세싱되고, 및/또는 DOM에 추가될) 때 페이지 레이아웃이 점진적으로 풀이되도록 수행될 수도 있다.

[0080] 블록 376에서, 브라우저 시스템 (350) 은 컴퓨팅 디바이스의 전자 디스플레이 상에 HTML 문서의 콘텐츠 (378) 를 디스플레이하기 위해 렌더 동작들을 수행할 수도 있다.

[0081] 다양한 양태들은 기존의 브라우저 프로세싱 알고리즘들의 기본적인 직렬 성질을 수정한다. 다양한 양태들은 높은 정도의 병렬성 및/또는 동시성을 지원하는 동적이고 병행하는 브라우저 시스템을 포함할 수도 있다. 다양한 양태들은 다수의 레벨들에서 동시성을 이용할 수도 있다. 다양한 양태들은 다양한 브라우저 컴포넌트들 및/또는 동작들의 프로세싱 및/또는 실행 시간들을 빠르게 하기 위해 개개의 브라우저 패스들에 대해 병렬 알고리즘들을 수행할 수도 있다. 다양한 양태들은 전체 실행 시간을 빠르게 하기 위해 브라우저 패스들을 중첩할 수도 있다.

[0082] 도 4와 도 5는 다양한 양태들에 따라 다수의 레벨들에서 동시성을 이용하기에 적합한 일 양태의 브라우저 시스템 (500) 에서의 예의 컴포넌트들, 정보 흐름들, 및 서브시스템들을 도시한다.

[0083] 도 4는 페치 관리자 컴포넌트 (502), DOM 디스패처 컴포넌트 (504), HTML 파서 컴포넌트 (506), HTML 프리-스캐너 컴포넌트 (508), 이미지 디코드 컴포넌트 (510), CSS 엔진 컴포넌트 (512), JavaScript® 엔진 컴포넌트 (514), 레이아웃 및 렌더링 엔진 컴포넌트 (516), 및 사용자 인터페이스 컴포넌트 (518) 를 포함하는 브라우저 시스템 (500) 을 예시한다. 일 양태에서, 브라우저 시스템 (500) 은 샌드박스식 (sandboxed) JavaScript® 엔진 컴포넌트 (530) 를 또한 포함할 수도 있다. 이들 컴포넌트들 (502 내지 530) 의 각각은 소프트웨어 모듈 (예컨대, 프로세서 상에서 실행중인 프로세스, 실행 스레드, 스레드 풀 (pool), 프로그램 등) 일 수도 있다.

다양한 양태들에서, 컴포넌트들 (502 내지 530) 중 임의의 것 또는 모두는 동시성을 지원하기 위해 스레드 라이브러리 (예컨대, Pthreads 등) 또는 병렬 태스크 라이브러리 (예컨대, Intel Thread Building Blocks, Cilk 등) 를 이용할 수도 있다.

[0084] 일 양태에서, 브라우저 시스템 (500) 의 컴포넌트들 (502 내지 518, 530) 은 동시성을 지원하기 위해 느슨하게 커플링되고 구성될 수도 있다.

[0085] 페치 관리자 컴포넌트 (502) 는 네트워크로부터 자원들을 페치하며, 페치된 자원들에 대한 캐시 관리를 수행하고, 네트워크로부터의 데이터의 도착에 대한 통지를 다른 브라우저 컴포넌트들로 제공하도록 구성될 수도 있다.

일 양태에서, 페치 관리자 컴포넌트 (502) 는 자원들이 HTML 문서에서 나타나는 순서로 (즉, 임의의 우선순위들을 부과하는 일 없이) 그 자원들을 페치하도록 구성될 수도 있다. 다른 양태에서, 페치 관리자 컴포넌트 (502) 는 미리 할당된 우선순위들에 기초하여 차원들에 우선순위들을 할당하고 및/또는 페치하도록 구성될 수도 있다.

[0086] DOM 디스패처 컴포넌트 (504) 는 DOM 업데이트들을 스케줄링하며, DOM 트리에 대한 액세스를 직렬화하고, 다양한 브라우저 컴포넌트들 간의 상호작용을 관리하도록 구성될 수도 있다. 다른 서브시스템들 (즉, 브라우저 인프라스트럭처의 나머지) 은 작업 아이템들 (또한 "DOM 디스패처 작업 아이템들"이라 지칭됨) 을 병행하는 DOM 디스패처 큐 속으로 디스패치할 수도 있다. DOM 디스패처 컴포넌트 (504) 는 DOM 디스패처 큐로부터 작업 아이템들 풀 (pull) 하고, 작업 아이템들을 한번에 하나씩 프로세싱하도록 구성될 수도 있다. 다양한 양태들에서, 작업 아이템들은 브라우저 패스들 및/또는 이벤트들 (예컨대, 타이머 이벤트들, 사용자 인터페이스로부

터의 이벤트들 등) 을 포함할 수도 있다.

[0087] HTML 파서 컴포넌트 (506) 는 HTML 문서의 들어오는 (예컨대, 부분적 등의) 데이터 청크 (chunk) 들을 (예컨대, DOM 디스패처 작업 아이템들 등을 통해) 수신하고, HTML 파싱 알고리즘 (예컨대, HTML5 parsing 알고리즘 등) 을 실행함으로써 DOM 트리를 구축하도록 구성될 수도 있다. HTML 파서 컴포넌트 (506) 는 HTML 문서에서 참조된 외부 자원들을 페치 관리자 컴포넌트 (502) 에 액세스가능한 페치 관리자 큐에 추가할 수도 있다. HTML 파서 컴포넌트 (506) 는 파싱 동작들 동안 적절한 시간들에서 JavaScript® 엔진 컴포넌트 (514) 를 호출함으로써 JavaScript® 코드의 실행을 또한 개시할 수도 있다.

[0088] HTML 프리-스캐너 컴포넌트 (508) 는 HTML 문서에 의해 요구되는/요청되는 외부 자원들을 빠르게 결정하기 위해 HTML 문서를 스캔하도록 구성될 수도 있다. HTML 프리-스캐너 컴포넌트 (508) 는 외부 자원들의 다운로딩 및/또는 외부 자원들에 기초한 추가의 프로세싱의 수행을 시작하기 위해 페치 관리자 컴포넌트 (502) 를 (예컨대, 통지, 메모리 쓰기 동작 등을 통해) 태스킹할 수도 있다.

[0089] 이미지 디코더 컴포넌트 (510) 는 이미지들을 디코딩하도록 구성될 수도 있다. 예를 들어, 페치 관리자 컴포넌트 (502) 가 이미지에 대한 전체 데이터를 수신한 경우, 그 페치 관리자 컴포넌트는 그 이미지를 이미지 디코더 컴포넌트 (510) 로 핸드오프 (hand off) 할 수도 있으며, 그러면 이미지 디코더 컴포넌트는 그 이미지를 나중의 사용을 위해 디코딩할 수도 있다.

[0090] CSS 엔진 컴포넌트 (512) 는 나중의 스테이지들 (예컨대, 레이아웃 및 렌더링 스테이지들) 에서의 사용을 위해 DOM 엘리먼트들의 룩 앤드 필 (look and feel) 을 계산하도록 구성될 수도 있다. 위에서 논의된 이미지 디코딩 동작들과 유사하게, 페치 관리자 컴포넌트 (502) 는 요구될 새로운 자원들을 파싱하고 발견하기 위해 CSS 스타일 시트들을 CSS 엔진에 핸드오프할 수도 있다.

[0091] 일 양태에서, CSS 엔진 컴포넌트 (512) 는 CSS 자원 프리-페처 컴포넌트 (520), CSS 파서 컴포넌트 (522), 및 DOM 스타일러 컴포넌트 (524) 를 포함할 수도 있다. CSS 자원 프리-페처 컴포넌트 (520) 는 CSS 스캐닝 및/또는 프리-페칭 동작들을 수행할 수도 있는데, 그 동작들은 어떤 외부 자원들이 CSS 문서에 의해 요구/요청되는지를 빠르게 결정하기 위해 CSS 문서를 스캔하는 것을 포함할 수도 있다. 일 양태에서, CSS 자원 프리-페처 컴포넌트 (520) 는 외부 자원들의 다운로딩 및/또는 외부 자원들에 기초한 추가의 프로세싱의 수행을 시작하도록 페치 관리자 컴포넌트 (502) 를 태스킹할 수도 있다.

[0092] CSS 파서 컴포넌트 (522) 는 CSS 코드를 읽고 메모리에서 데이터 구조들 (예컨대, CSS 규칙들) 의 컬렉션을 생성하도록 구성될 수도 있다. DOM 스타일러 컴포넌트 (524) 는 DOM 트리에서 노드들의 스타일을 결정하기 위해 CSS 파서 컴포넌트 (522) 에 의해 생성된 데이터 구조들을 사용하도록 구성될 수도 있다. 각각의 노드에 대해, CSS 엔진 컴포넌트 (512) 는 선택자들이 노드와 일치하는 규칙들을 찾기 위하여 규칙 매칭 동작들을 수행할 수도 있다. 이러한 규칙 매칭 동작들은 노드마다 많은 (및 때때로 모순되는) 규칙들을 반환할 수도 있다. 다양한 양태들에서, CSS 엔진 (512) 은 규칙들에 가중치들을 할당하고 최대 가중치를 갖는 규칙들을 선택하기 위해 캐스케이딩 동작들을 사용하도록 구성될 수도 있다.

[0093] JavaScript® 엔진 컴포넌트 (514) 는 JavaScript® 코드를 컴파일하고 실행하도록 구성될 수도 있다. 페치 관리자 (502) 는 JavaScript® 스크립트들을 다운로드하고 그것들을 컴파일되도록 JavaScript® 엔진 컴포넌트 (514) 에 전송할 수도 있다. HTML 파서 (506) 및/또는 DOM 디스패처 (504) 는 JavaScript® 엔진 컴포넌트 (514) 가 스크립트들을 실행할 것을 요청할 수도 있다.

[0094] JavaScript® 엔진 컴포넌트 (514) 는 컴파일 태스크들/동작들을 위한 스레드 풀을 포함할 수도 있고, 다수의 스크립트들 (JavaScript® 코드) 을 병렬로 컴파일하도록 구성될 수도 있다. JavaScript® 시맨틱스로 인해, 일 양태에서, 스크립트들의 실행은 메인 엔진 스레드에서 순차적으로 수행될 수도 있다. 일 양태에서, JavaScript® 엔진 컴포넌트 (514) 는, HTML 파서 (506) 또는 DOM 디스패처 (504) 가 (예컨대, 사용자 인터페이스 이벤트들에 대해) 컴파일되지 않은 스크립트를 실행하기 위해 JavaScript® 엔진 컴포넌트 (514) 를 요청하는 경우, JavaScript® 엔진 컴포넌트 (514) 는 요청된 스크립트를 실행하는 것을 시도하기 전에 스크립트들의 컴파일을 자동으로 개시하고 컴파일의 결과들을 기다리도록 구성될 수도 있다.

[0095] 다양한 양태들에서, JavaScript® 엔진 컴포넌트 (514) 는 (예컨대, JavaScript® 코드의 적응적 컴파일 및 실행을 지원하기 위해) 라이트 (light) 컴파일러 (526) 와 전체 (full) 컴파일러 (528) 를 포함할 수도 있다. 라이트 컴파일러 (526) 는 드물게 재사용된 JavaScript® 코드에 대한 실행가능 코드를 생성하도록 구성될 수도 있고 그리고/또는 페이지 로드에 최적화될 수도 있다. 전체 컴파일러 (528) 는 심하게 재사용된

JavaScript® 코드에 대한 더 높은 품질의 코드를 생성하도록 구성될 수도 있고 그리고/또는 상호작용성 및 웹 애플리케이션들에 최적화될 수도 있다. 다양한 양태들에서, 전체 컴파일러 (528) 의 더 느린 코드 생성이 재사용된 코드의 다수의 실행들 간에 상각 (amortizion) 될 수도 있다. 라이트 컴파일러 (526) 에 비해, 전체 컴파일러 (528) 는 반복적인 웹 애플리케이션들에 대해 상당한 스피드업을 달성할 수도 있다. 예를 들어, 전체 컴파일러 (528) 를 사용하여, N-바디 시뮬레이션 웹 애플리케이션이 6 배만큼 더 빠르게 실행될 수도 있다.

[0096] 샌드박스식 JavaScript® 엔진 컴포넌트 (530) 는 기본 JavaScript® 엔진 컴포넌트 (514) 로부터 분리되는 격리된 JavaScript® 엔진일 수도 있다. 샌드박스식 JavaScript® 엔진 컴포넌트 (530) 는 모든 컴포넌트들, 피처들, 및 기능성 JavaScript® 엔진 컴포넌트 (514) 를 포함할 수도 있다.

[0097] 레이아웃 및 렌더링 엔진 컴포넌트 (516) 는 스타일링된 DOM 트리를 가시 웹 페이지로 변환하도록 구성될 수도 있다. 일 양태에서, 레이아웃 및 렌더링 엔진 컴포넌트 (516) 는 사용자가 업데이트된 HTML 문서를 보고 그 문서와 상호작용하도록 모바일 디바이스의 전자 디스플레이 상에서 DOM 및/또는 CSS 스타일 시트들에 변경들을 반영하도록 구성될 수도 있다. DOM 및/또는 CSS에 대한 변경들은 페치 관리자 컴포넌트 (502) 가 새로운 자원들을 전달하는 것, HTML 파서 컴포넌트 (506) 가 DOM을 업데이트하는 것, JavaScript® 엔진 컴포넌트 (514) 계산의 결과 등으로 인한 것일 수도 있다.

[0098] 일 양태에서, 레이아웃 및 렌더링 엔진 (516) 은 DOM 정보의 스냅샷을 취하고 레이아웃 및/또는 렌더 동작들을 비동기적으로 수행하도록 구성될 수도 있다. 다른 양태에서, 레이아웃 및 렌더링 엔진 (516) 은 (예컨대, JavaScript®가 레이아웃 정보를 쿼리하는 API들을 사용하는 경우) 레이아웃 및/또는 렌더 동작들을 비동기적으로 호출하도록 구성될 수도 있다.

[0099] 사용자 인터페이스 컴포넌트 (518) 는 브라우저 시스템 (500) 과 모바일 디바이스 사용자 사이의 상호작용들을 관리하도록 구성될 수도 있다. 사용자 인터페이스 컴포넌트 (518) 는 사용자 상호작용들 (예컨대, 모바일 디바이스의 전자 디스플레이 상의 링크를 터치하는 것) 을 DOM 디스패처 큐에서의 배치를 위해 작업 아이템들을 생성하는 함수/메소드 호출들 (예컨대, 자바 네이티브 인터페이스 또는 "JNI" 메소드 호출들) 로 해석할 수도 있다.

[0100] 일 양태에서, 모든 전술한 컴포넌트들 (502 내지 518, 530) 은 각각의 웹페이지에 대해 한번 인스턴스화될 수도 있다. 다른 양태에서, 페치 관리자 컴포넌트 (502) 와 레이아웃 및 렌더링 엔진 컴포넌트 (516) 는 전역적일 수도 있는 반면, 다른 컴포넌트들 (예컨대, 504, 506, 508, 510, 512, 514, 및 518) 은 각각의 웹페이지 또는 HTML 문서에 대해 한 번 인스턴스화될 수도 있다.

[0101] 도 5는 위에서 논의된 양태의 브라우저 시스템 (500) 에서의 예의 서브시스템들 및 정보 흐름들을 도시한다. 구체적으로, 도 5는 브라우저 시스템 (500) 이 사용자 인터페이스 서브시스템 (552), 자원 관리자 서브시스템 (554), 페이지 당 (per-page) DOM 엔진 서브시스템 (556), 페이지 당 JavaScript® 엔진 서브시스템 (558), 및 렌더링 엔진 서브시스템 (560) 을 포함할 수도 있다는 것을 도시한다.

[0102] 서브시스템들 (555~560) 의 각각은 동시성을 지원하기 위해 느슨하게 커플링되고 구성될 수도 있다. 서브시스템들 (552~560) 은 소프트웨어 모듈들 (예컨대, 프로세서 상에서 실행중인 프로세스, 실행 스레드, 프로그램 등) 로서 구현될 수도 있다. 서브시스템들 (552~560) 의 동작들은 도 4를 참조하여 위에서 논의된 컴포넌트들 중 하나 이상에 의해 및/또는 임의의 단일 또는 멀티프로세서 컴퓨팅 시스템 상에서 수행될 수도 있다.

[0103] 일 양태에서, 자원 관리자 서브시스템 (554) 과 렌더링 엔진 서브시스템 (560) 은 한 번 인스턴스화될 수도 있고 (예컨대, 전역적일 수도 있고), 페이지 당 DOM 엔진 서브시스템 (556) 과 페이지 당 JavaScript® 엔진 서브시스템 (558) 은 각각의 웹페이지 또는 HTML 문서에 대해 한 번 인스턴스화될 수도 있다.

[0104] 사용자 인터페이스 서브시스템 (552) 은, 사용자 상호작용들 (예컨대, 모바일 디바이스의 전자 디스플레이 상의 링크를 터치하는 것) 을 DOM 디스패처 큐에서의 배치를 위해 작업 아이템들을 생성하는 함수/메소드 호출들로 해석하는 것, 페이지 당 JavaScript® 엔진 서브시스템 (558) 의 올바른 인스턴스에 대한 이벤트들을 검출하고 및/또는 전송하는 것, 그리고/또는 URL (uniform resource locator) /URI (uniform resource identifier) 정보를 (예컨대, 메모리 쓰기 동작, 함수 호출 등을 통해) 자원 관리자 서브시스템 (554) 으로 전송하는 것을 포함한 브라우저 시스템 (550) 으로 사용자 상호작용들을 관리하기 위한 다양한 동작들을 수행하도록 구성될 수도 있다.

[0105] 자원 관리자 서브시스템 (554) 은 프리-페칭 동작들 (562), HTML 프리-스캐닝 동작들 (563), 이미지 디코딩 동

작들 (564), CSS 스캐닝/프리-페칭 동작들 (566), 및 JavaScript 스캐닝/프리-페칭 동작들 (567) 을 수행하도록 구성될 수도 있다. 예로서, 이들 동작들은 페치 관리자 (502), HTML 프리-스캐너 (508), 이미지 디코더 (510), CSS 엔진 (512), 및/또는 JavaScript 엔진 (514, 530) 컴포넌트들에 의해, 또는 도 4에 관련하여 위에서 논의된 컴포넌트들의 임의의 조합에 의해 수행될 수도 있다.

[0106] 프리-페칭 동작들 (562) 은, URL/URI에 대응하는 웹 서버로부터 자원들 및/또는 프로그래밍 명령들을 요청/수신하는 것, HTML을 생성하기 위해 수신된 프로그래밍 명령들을 해석하거나 또는 디코딩하는 것, 및 생성된 HTML 코드를 (예컨대, 메모리 쓰기 동작 등을 통해) 페이지 당 JavaScript® 엔진 서브시스템 (558) 의 올바른 인스턴스로 전송하는 것을 포함할 수도 있다.

[0107] 생성된 HTML 코드는 JavaScript® 코드, CSS 코드, 이미지들, 및 다양한 다른 자원들을 내장하고 및/또는 참조할 수도 있다. HTML 문서에서 가장 일반적으로 참조되는 자원들은 이미지들, CSS 스타일 시트들, 및 JavaScript® 소스들이다. 스타일 시트들과 JavaScript® 소스들은 추가의 외부 자원들을 또한 참조할 수도 있다. 일 양태에서, 생성된 HTML 코드는 HTML 문서에 의해 식별된 모든 참조들 (내장된 또는 참조된 스타일 시트들 및 JavaScript® 소스들을 포함함) 이 미리 (예컨대, 프리-페칭 동작들 (562) 의 일부로서) 페치될 수도 있도록 스캔될 수도 있다.

[0108] HTML 프리-스캐너 동작들 (563) 은 요구된/요청된 외부 자원들을 빠르게 발견하기 위해 생성된 HTML 코드를 스캔하는 것과, 외부 자원들의 다운로딩 및/또는 발견된 외부 자원들에 기초한 추가의 프로세싱의 수행을 시작할 수도 있음을 페치 관리자 및/또는 프리-페처에게 알리는 것을 포함할 수도 있다. 일 양태에서, 외부 자원들의 다운로딩은 위에서 논의된 프리-페칭 (562) 동작들의 일부로서 수행될 수도 있다. 일 양태에서, HTML 프리-스캐너 동작들 (508) 과 프리-페칭 동작들 (562) 은 (예컨대, 별개의 스레드들/프로세스들에서) 동시에 수행될 수도 있다.

[0109] 이미지 디코딩 동작들 (564) 은 렌더링 엔진 서브시스템 (560) 에 의한 나중의 사용을 위해 이미지들을 디코딩하는 것을 포함할 수도 있다. 이미지 디코딩 동작들 (564) 은 이미지를 위한 완전한 데이터 세트가 (예컨대, 프리-페칭 (562) 동작들의 일부로서 수행된 메모리 쓰기 동작 등을 통해) 다운로드되었다는 결정에 응답하여 그리고/또는 (예컨대, 페치 관리자 (502) 컴포넌트로부터의) 통지의 수신에 응답하여 수행될 수도 있다. 일 양태에서, 이미지 디코딩 동작들 (564) 은 HTML 프리-스캐너 동작들 (563) 및 프리-페칭 동작들 (562) 과 동시에 수행될 수도 있다.

[0110] CSS 스캐닝/프리-페칭 동작들 (566) 은 CSS 스타일 시트들에 의해 요청되는 요구된/요청된 외부 자원 요청을 빠르게 발견하기 위해 생성된 HTML 코드에 내장된 (또는 그 코드에 의해 참조된) CSS 스타일 시트들을 스캔하는 것을 포함할 수도 있다. 일 양태에서, CSS 스캐닝/프리-페칭 동작들 (566) 은 발견된 외부 자원들의 다운로딩을 시작할 수도 있음을 페치 관리자 및/또는 프리-페처에게 알리는 것을 포함할 수도 있다. 일 양태에서, CSS 스캐닝/프리-페칭 동작들 (566) 은 발견된 외부 자원들의 다운로딩을 개시하는 것을 포함할 수도 있다. 일 양태에서, CSS 스캐닝/프리-페칭 동작들 (566) 은 페치 관리자 컴포넌트 (502) 가 하나 이상의 CSS 스타일 시트들을 CSS 엔진 컴포넌트 (512) 로 전송하는 것에 응답하여 (예컨대, CSS 자원 프리-페처 (520)) 에 의해) CSS 엔진 컴포넌트 (512) 에서 수행될 수도 있다. 일 양태에서, CSS 스캐닝/프리-페칭 동작들 (566) 은 이미지 디코딩 동작들 (564), HTML 프리-스캐너 동작들 (563), 및 프리-페칭 동작들 (562) 과 동시에 수행될 수도 있다.

[0111] 페이지 당 DOM 엔진 서브시스템 (556) 은 HTML 파싱 동작들 (568), CSS 파싱 동작들 (570), 타이머 동작들 (572), 스타일링 동작들 (574), 및 이벤트들 (576) 을 관리하기 위한 동작들을 수행하도록 구성될 수도 있다. 일 양태에서, 페이지 당 DOM 엔진 서브시스템 (556) 의 동작들은 다른 서브시스템들 (552, 554, 558, 560) 의 동작들과 동시에 수행될 수도 있다.

[0112] HTML 파싱 동작들 (568) 은 수신된 HTML 코드를 파싱하는 것, 실질적인 내용으로부터 HTML 마크업 태그들을 분리하는 것, 및/또는 수신된 HTML 코드의 DOM을 생성하는 것을 포함할 수도 있다. HTML 파싱 동작들 (568) 은 식별된 외부 자원들이 페치 관리자 (502) 에 의해 그리고/또는 프리-페칭 동작들 (562) 의 일부로서 다운로드될 수도 있도록 HTML 문서에서 참조된 외부 자원들을 식별하는 것을 또한 포함할 수도 있다. HTML 파싱 동작들 (568) 은 (예컨대, JavaScript®가 발견되는 때 등의) HTML 코드의 파싱 동안 (예컨대, 실행 동작 (578) 을 호출함으로써) JavaScript® 코드의 실행을 개시하는 것을 더 포함할 수도 있다.

[0113] CSS 파싱 동작들 (570) 과 스타일링 동작들 (574) 은 생성된 DOM 트리에 하나 이상의 CSS 스타일 시트들을 적용

하는 것 (또는 CSS 스타일 시트들에 기초하여 수정된 DOM 트리를 생성하는 것) 을 포함할 수도 있다. 다양한 양태들에서, HTML 파싱 동작들 (568), CSS 파싱 동작들 (570), 및 스타일링 동작들 (574) 중 임의의 것 또는 모두는 동시에 수행될 수도 있다.

[0114] 타이머 동작들 (572) 은 타이머들 및/또는 타이머 클래스들 (예컨대, System.Timers) 에 관련한 이벤트들 및/또는 조건들을 관리거나 또는 그것들에 응답하는 것을 포함할 수도 있다.

[0115] 이벤트들 동작들 (576) 은 다양한 이벤트들, 이를테면 타이머 이벤트들 및 사용자 인터페이스 이벤트들 (예컨대, 사용자가 모바일 디바이스의 전자 디스플레이 상의 링크를 터치하는 것에 응답하여 생성된 이벤트) 을 관리하는 것을 포함할 수도 있다.

[0116] 페이지 당 JavaScript® 엔진 서브시스템 (558) 은 JavaScript® 실행 동작들 (578) 과 JavaScript® 컴파일 동작들 (580) 을 수행하도록 구성될 수도 있다.

[0117] 다양한 양태들에서, 페이지 당 DOM 엔진 서브시스템 (556) 및/또는 자원 관리자 서브시스템 (554) 은, HTML 코드에 내장된 (또는 그 HTML 코드에 의해 참조된) JavaScript® 코드를 컴파일 및/또는 실행을 위한 페이지 당 JavaScript® 엔진 (558) 의 올바른 인스턴스로 (즉, 실행 (578) 및 컴파일 (580) 동작들을 통해) 전송하도록 구성될 수도 있다. 양태에서, JavaScript® 엔진 (558) 은 생성된 DOM 트리를 JavaScript® 컴파일 및/또는 실행 동작들 (578, 580) 의 결과들에 기초하여 업데이트/수정할 수도 있다.

[0118] 렌더링 엔진 서브시스템 (560) 은 레이아웃 동작들 (582) 과 렌더 동작들 (584) 을 수행하도록 구성될 수도 있다. 예를 들어, 렌더링 엔진 서브시스템 (560) 은, 페이지 당 DOM 엔진 서브시스템 (556) 으로부터의 DOM 트리 및/또는 레이아웃 트리를 (예컨대, 메모리 쓰기들, 호출들, 통지들 등을 통해) 수신하며, 페이지 레이아웃을 (레이아웃 동작 (582) 을 통해) 풀이하고, 컴퓨팅 디바이스의 전자 디스플레이 상에 (렌더 동작 (584) 을 통해) 내용을 디스플레이할 수도 있다. 일 양태에서, 레이아웃 동작들 (582) 을 수행하는 것은, 부가적인 콘텐츠가 렌더링 엔진 서브시스템 (560) 에 이용가능하게 될 (예컨대, 다운로드될, 프로세싱될, 및/또는 DOM 트리에 추가될) 때 페이지 레이아웃을 점진적으로 풀이하는 것을 포함할 수도 있다. 다양한 양태들에서, 레이아웃 동작들 (582) 및/또는 렌더 동작들 (584) 중 임의의 것 또는 모두는 동시에 수행될 수도 있다.

[0119] 도 4 및 도 5를 참조하여 위에서 논의된 바와 같이, HTML 파서 (506) 및/또는 CSS 파서 (522) 는, HTML 문서를 렌더링하는데 필요한/요구된 외부 자원들 (이미지들, 오디오, CSS, JavaScript® 등) 을 발견하고, 발견된 자원들이, 이를테면 페치 관리자 (502) 를 통해 및/또는 프리-페치 동작들의 일부로서 다운로드될 것을 요청할 수도 있다.

[0120] 모바일 디바이스들은 HTML 및 CSS 코드/콘텐츠에서 발견된 자원들을 다운로드하는 경우 높은 레이턴시 시간들을 경험할 수도 있다. 예를 들어, HTML5 사양에서의 특이성들로 인해, HTML 파서는 HTML 문서의 나머지 부분들을 계속 파싱하기 전에 실행을 완료하기 위해 스크립트 엘리먼트 (예컨대, <script> 블록블) 를 기다려야만 한다. 따라서, 웹 페이지가 스크립트 엘리먼트 뒤의 외부 자원을 참조하면, 그 외부 자원을 페치하는 동작은 실행을 완료하기 위해 스크립트 엘리먼트를 기다리는 동작과 중첩될 수 없다. 이는 종종, 웹페이지를 다운로드하고 디스플레이하는데 필요한 시간을 증가시킨다.

[0121] 다양한 양태들에서, 브라우저 시스템 (500) 은 실행을 완료하기 위해 스크립트 엘리먼트를 기다리는 일 없이 새로운 자원들을 발견하기 위해 스크립트 엘리먼트들보다 앞에 추론적으로 파싱하도록 구성될 수도 있다 이들 양태들에서, 브라우저 시스템 (500) 은 (예컨대, JavaScript®가 새로운 내용을 document.write API 등을 통해 DOM 트리 속으로 삽입한 경우) 추론적 파싱의 결과들의 일부를 버리도록 강제될 수도 있다.

[0122] 일 양태에서, 브라우저 시스템 (500) 은, 요청된/요구된 자원들을 가능한 한 일찍 발견하고 병렬로 페치/다운로드될 다수의 자원들을 요청하기 위해 공격적인 자원 프리-페칭 동작들을 수행하도록 구성될 수도 있다. 이런 방식으로, 다양한 양태들은 브라우저 시스템 (500) 이 추론적 파싱의 결과들의 일부를 버리도록 강제되는 것을 방지할 수도 있고, 네트워크 레이턴시들을 감추며, 이용가능한 대역폭 중 더 많은 이용가능한 대역폭을 이용하고, 도착할 자원들을 기다리는데 소비되는 전체 시간을 감소시킬 수도 있다.

[0123] 브라우저 시스템 (500) 은, 샌드박스식 실행을 통한 추론적 자원 프리페칭을 포함할 수도 있는 공격적 자원 프리-페칭 동작들을 수행하도록 구성될 수도 있다. 다양한 양태들에서, 이들 공격적 자원 프리-페칭 동작들은 HTML 프리-스캐닝 동작들 (563), CSS 프리-페칭 동작들 (566), 또는 양쪽 모두의 일부로서 수행될 수도 있다.

[0124] 도 4와 도 5를 참조하면, 공격적 자원 프리-페칭 동작들의 추진 (furtherance) 으로 수행된 HTML 프리-스캐닝

동작들 (563) 은 HTML 문서에서 모든 "id", "클래스", 및/또는 "스타일" 속성들을 획득하는 것, HTML 문서에서 참조된 외부 자원들을 빠르게 발견하는 것, 및 네트워크로부터의 발견된 자원들의 다운로딩을 트리거하는 것을 포함할 수도 있다. HTML 프리-스캐너 (508) 는, HTML 파서 (506) 로부터 요청되는 실질적인 또는 계산 집약적인 임의의 프로세싱 (예컨대, DOM 트리의 구축) 을 수행하는 일 없이, 자원들을 발견하기 위하여 HTML을 "대략적으로 파싱"할 수도 있다. 이들 복잡한 파싱 동작들을 포기함으로써, HTML 프리-스캐닝 동작들 (563) 은 HTML 파싱 동작들 (568) 과 동시에 (그리고 그 동작들보다 앞서) 수행될 수도 있고, 실행을 완료하기 위해 스크립트 엘리먼트들을 기다릴 필요가 없다.

[0125] 일 양태에서, 네트워크 패킷들은, 그것들이 도착하면, HTML 프리-스캐너 (508) 및 HTML 파서 (506) 에 독립적으로 전송될 수도 있다. 일 양태에서, 도착할 자원들을 기다리는데 소비된 시간은 HTML 프리-스캐닝 동작들 (563) 을 (비-추론적) HTML 파싱 (570) 동작들과는 병렬로 수행함으로써 더욱 감소될 수도 있다.

[0126] 위에서 논의된 바와 같이, 웹 브라우저 시스템 (500) 은 CSS 문서를 빠르게 스캔하도록 구성된 CSS 파서 (522) 와 CSS 프리-페칭 동작들을 수행하도록 구성된 CSS 자원 프리-페처 (520) 를 포함할 수도 있다. 일 양태에서, CSS 스타일 시트들은 CSS를 파싱하는 것을 담당하는 스레드 풀과 동시에 디스패치될 수도 있다. CSS 규칙이 추가의 외부 자원들을 포함하면, CSS 자원 파서 (520) 는 추가의 외부 자원들이 HTML 문서에서 실제로 참조되는 우도에 기초하여 추가의 외부 자원들에 대한 프리페칭을 개시할 지의 여부에 관해 결정할 수도 있다. 일 양태에서, CSS 자원 프리-페처 (520) 는 특정 범위/수의 참조된 자원들을 다운로드 (또는 그것들의 다운로딩을 개시) 하도록 구성될 수도 있다 (너무 적은 수의 자원들을 다운로드하는 것은, 나중에 DOM 트리를 스타일링하는 경우, 부가적인 레이턴시들이 초래되게 할 수도 있는 더욱 새로운 자원들이 DOM 스타일러 (524) 에 의해 발견될 것임을 의미한다).

[0127] 사이트 와이드 공통 스타일 파일을 사용함으로써 임의의 주어진 문서에 대해 실제로 요청된 것보다 더 많은 자원들을 참조하는 것이, 예를 들어, 웹사이트들 중에서 일반적 관행이다. 모든 포함된 자원들을 다운로드하는 것은 과도한 대역폭을 소비하고 페이지 로딩을 느려지게 할 수도 있다. 다양한 양태들에서, CSS 파서 (522) 는 CSS 규칙이 매칭될 가능성이 있는지의 여부를 결정하기 위해 HTML 프리-스캐너 (508) 에 의해 발견된 "id" 및 "클래스" 속성들을 채용하도록 구성될 수도 있다. CSS 규칙 선택기에서 참조된 속성 값들의 모두가 HTML 프리-스캐너 (508) 에 의해 보이면/평가되면, 그 CSS 규칙이 적어도 하나의 DOM 트리 엘리먼트를 매칭할 가능성이 있다고 결정될 수도 있고, 브라우저 시스템 (500) 은 그 CSS 규칙에 대응하는 자원들의 다운로딩을 개시할 수도 있다. 이 "CSS 규칙" 휴리스틱은 매우 효과적이고, 잘못된 결정들은 브라우저 시스템 (500) 의 동작들에 대해 상당한 부정적 영향을 가지지 않는다. 손실된 자원들은 자원을 다운로드하는데 필요한 레이턴시를 대가로 DOM 스타일링 페이즈 동안 (DOM 스타일러 컴포넌트 (524) 를 통해) 발견될 수도 있다.

[0128] 일 양태에서, HTML 프리-스캐너 (508) 는 JavaScript®를 실행하지 않고 발견될 수도 있는 자원들을 식별 및/또는 발견하도록 구성될 수도 있다.

[0129] 위에서 논의된 바와 같이, HTML5 사양에서의 특이성들, 이를테면 HTML 파서는 그것이 계속 파싱할 수 있기 전에 실행을 완료하기 위해 스크립트 엘리먼트 (예컨대, <script> 블록들) 를 기다리는 것이 필요하기 때문에, 모바일 디바이스들은 HTML 및 CSS 코드/내용에서 발견된 자원들을 다운로드하는 경우 높은 레이턴시 시간들을 경험할 수도 있다. 덧붙여서, 현대의 웹 문서들 (예컨대, HTML 페이지들, HTML 문서들 등) 은 다수의 외부 자원들을 참조할 수도 있고, 각각의 외부 자원은 다른 외부 자원들에 대한 참조들을 포함할 수도 있다. 예를 들어, HTML 문서들은 다양한 외부 자원들, 이를테면 이미지들, 오디오, CSS (Cascading Style Sheets), 및 JavaScript®에 대한 참조들을 통상적으로 포함하고, 참조된 자원들 (예컨대, CSS, JavaScript®) 은 부가적인 외부 자원들 (예컨대, 이미지들, 오디오 등) 에 대한 참조들을 더 포함할 수도 있다.

[0130] 문서 로드 시간 (즉, 문서의 요청부터 그 문서가 스크린 상에 디스플레이될 준비가 되기까지의 시간) 은 입력/출력 비용들 (예컨대, 요청된 자원들의 네트워크 전송들) 에 의해 지배된다. 모든 요청된 자원들을 로드하는데 필요한 최소 문서 로드 시간은 자원 스토리지와 컴퓨팅 디바이스 사이의 접속의 대역폭에 의해 제한된다. 또한, 문서 자원들을 디스플레이하는 디바이스로 전송하는 것은 레이턴시 비용을 발생시킨다. 다양한 양태들이 이용가능한 대역폭을 더 잘 이용하며, 전송 레이턴시들을 중첩하고, 문서 로드 시간들을 개선하기 위해 가능한 한 일찍 자원 전송들을 시작하도록 구성될 수도 있다.

[0131] 위에서 언급했듯이, 참조된 외부 자원들의 모두가 주어진 웹페이지를 렌더링하기 위해 요청 (또는 심지어 사용) 되지 않으므로, 참조된 자원들의 모두를 재귀적으로 다운로드하는 것은 상당한 양의 대역폭 및 전력을 낭비할 수도 있다. 덧붙여서, 자원들 중 임의의 것이 즉시 이용가능하지 않은 경우, 페이지가 적절히 렌더링될 수

있기 전에 브라우저는 자신이 그들 자원들을 수신하고 분석하기까지 기다려야만 한다. 이는 웹페이지를 로드 및/또는 렌더링하는데 필요한 시간의 양 (예컨대, 문서 로드 시간) 을 증가시키고, 사용자 경험을 저하시킨다.

[0132] 종래의 솔루션들은 페이지가 액세스될 때 다운로드되어야만 하는 정보를 줄이기 위해 메모리 내에 웹 페이지들의 부분들을 캐싱하는 것과 같은 기법들을 사용하여 웹 페이지들의 렌더링을 빠르게 하는 것을 시도한다. 그러나, 종래의 솔루션들을 사용하면, 먼저 전체 문서 (즉, 웹페이지) 를 분석하며, 문서 및 서브문서들에서 참조되는 자원들의 대부분 (만약 아니라면 전부) 을 요청하고 수신하고, 수신된 자원들을 분석하는 일 없이는, 웹 페이지를 처음으로 렌더링하는데 필요한 외부 자원들을 웹 브라우저가 식별할 수 없다. 따라서, 종래의 솔루션들을 사용하면, 문서에 의해 요청된 자원들의 정확한 세트는 전체 문서가 완전히 분석된 후까지 결정될 수 없다.

[0133] 기존의 솔루션들의 이들 한계들을 극복하기 위해, 다양한 양태들이 전체 문서가 분석되기 전에 웹 페이지 또는 문서를 렌더링하는데 필요한 자원들을 식별하기 위해 추론/예측 기법들을 이용할 수도 있다.

[0134] 일반적으로, (정보의 완전한 세트에 기초하여) 자원이 필요한지의 여부를 추론적으로 예측하는 것은 다음의 4 가지 가능한 결과들 중 하나가 초래되게 한다: 트루 포지티브; 트루 네거티브; 폴스 포지티브; 및 폴스 네거티브. 트루 포지티브 결과는 자원이 추론적으로 다운로드되었고 나중에 실제로 필요했던 경우이다. 트루 네거티브 결과는 자원이 추론적으로 다운로드되지 않았으나 필요하지 않았던 경우이다. 폴스 포지티브 결과는 필요하지 않은 자원이 추론적으로 다운로드된 경우이고 (이는 대역폭 및 에너지를 낭비함), 폴스 네거티브 결과는 자원이 추론적으로 다운로드되지 않지만 필요한 경우이다 (따라서 추론적인 프리프로세싱으로부터 이 자원에 관해 아무것도 얻은 것이 없다).

[0135] 트루 포지티브 및 트루 네거티브 결과들은 유익하고 바람직한데, 이러한 결정들이 페이지 로드 시간들을 감소시킴으로써 사용자 경험을 개선하기 때문이다. 그러나, 폴스 포지티브 및 폴스 네거티브 결과들은 유익하지 않다. 예를 들어, 문서 (예컨대, HTML 문서) 의 렌더링 동안에 요청된 자원에서는 폴스 네거티브이 초래될 수도 있는데, 이는 자원들이 이용가능하기까지 문서 로드 시간들을 연장시키는 것일 수도 있다. 문서를 적절히 렌더링하는 브라우저에 대해 자원이 필요하지 않으므로, 컴퓨팅 및 네트워크 자원들 (대역폭, 프로세싱 등) 의 낭비이다.

[0136] 다양한 양태들은 폴스 포지티브 및 폴스 네거티브 다운로드 결정들의 수를 최소화하면서도 트루 네거티브 및 트루 포지티브들의 수를 최대화하기 위해 휴리스틱들에 기초하여 추론적 자원 다운로딩 동작들을 수행하도록 구성된 웹 브라우저 시스템들을 포함한다.

[0137] 도 6은 페이지 로딩/렌더링 동작들에 앞서 웹페이지의 적절한 렌더링에 필요한 외부 자원들 (이미지들, 오디오, CSS, JavaScript® 등) 을 발견하기 위해 HTML 문서를 프로세싱하고 발견된 자원들을 프리-페치하는 일 양태의 브라우저 방법 (600) 을 도시한다. 방법 (600) 의 동작들은 적절히 구성된 웹 브라우저를 실행하는 단일 또는 멀티프로세서 컴퓨팅 시스템의 프로세서에 의해 수행될 수도 있다.

[0138] 도 6을 참조하면, 블록 602에서, 구조 정보를 위한 HTML 문서 및/또는 CSS 문서들을 스캔하기 위해 그리고/또는 자원들을 발견하기 위해 웹 브라우저가 스캔 동작 (예컨대, HTML 프리-스캐너 (508), CSS 엔진 (512) 등을 통해) 을 개시 또는 호출할 수도 있다. 일 양태에서, 스캔 동작은 HTML 프리-스캐닝 동작들 (563) 의 일부로서 수행될 수도 있다. 일 양태에서, 스캔 동작은 CSS 스캐닝 동작들 (566) 의 일부로서 수행될 수도 있다. 다양한 양태들에서, 스캔 동작은 HTML 및 CSS 파싱 동작들 (568, 570) 과 동시에, 그리고 그 동작들과는 독립적으로 실행될 수도 있다. 다양한 양태들에서, 스캔 동작은 프로세스, 스레드, 애플리케이션, 작업 아이템, 및/또는 브라우저 패스에 의해 수행될 수도 있다.

[0139] 블록 604에서, 스캔 동작 (예컨대, HTML 및/또는 CSS 스캐닝 동작 (563, 566)) 은 발견된 자원들 중 어느 것이 요청될 가능성이 있는지를 결정 (즉, 예측, 추론) 할 수도 있다. 블록 606에서, 스캔 동작은 요청될 확률이 높은 것으로 결정된 자원들의 다운로딩을 시작하기 위해 (예컨대, 메모리 쓰기 동작 등을 통해) 자원 요청들을 브라우저 페치 컴포넌트에게 (예컨대, 페치 관리자 (502) 에게) 발행할 수도 있다. 일 양태에서, 블록 606의 일부로서, 둘 이상의 자원 요청들이 병렬로 또는 동시에 발행 (또는 전송) 될 수도 있다. 일 양태에서, 각각의 자원 요청은 새로운 프로세스가 생겨나게 할 수도 있고 그리고/또는 상이한 실행 스레드에 의해 프로세싱될 수도 있다. 블록 608에서, 스캔 동작은 부가적인 요청된 자원들을 발견하기 위해 HTML 문서 및/또는 CSS 문서들의 스캐닝을 계속할 수도 있다. 블록들 (604~608) 에서의 동작들은 모든 외부 자원들이 발견되고

및/또는 전체 HTML 문서가 스캔되기까지 반복될 수도 있다.

[0140] 블록 610에서, 웹 브라우저는 자원 요청 (예컨대, 블록 606의 스캔 동작에 의해 발행된 자원 요청) 에 의해 식별된 하나 이상의 자원들을 다운로드하기 위해 (예컨대, 페치 관리자 (502) 를 통해) 페치 동작을 개시하거나 또는 호출할 수도 있다.

[0141] 블록 612에서, 웹 브라우저는 외부 자원들에 대한 부가적인 참조들을 발견하기 위해 다운로드된 자원들을 스캔할 수도 있다. 블록 612의 일부로서, 웹 브라우저는 스캐닝 동작들을 수행하기 위해 새로운 프로세스 또는 실행 스레드를 개시하거나 또는 호출할 수도 있다. 일 양태에서, 블록 612의 일부로서, 웹 브라우저는 CSS 스캐닝 동작 (566) 을 개시하거나 또는 호출할 수도 있다. 일 양태에서, 블록 612의 일부로서, 웹 브라우저는 HTML 스캐닝 동작 (563) 을 개시하거나 또는 호출할 수도 있다.

[0142] 블록 614에서, 웹 브라우저는 다운로드된 자원들의 스캐닝에 기초하여 요청될 가능성이 있는 발견된 자원들을 결정 (즉, 예측, 추론) 할 수도 있다. 블록 616에서, 웹 브라우저는 요청될 확률이 높은 것으로 결정된 자원들의 다운로딩을 시작하기 위해 (예컨대, 메모리 쓰기 동작들 등을 통해) 부가적인 자원들의 요청들을 브라우저 페치 컴포넌트에게 (예컨대, 페치 관리자 (502) 에게) 발행할 수도 있다. 일 양태에서, 이들 부가적인 자원 요청들의 각각은 다른 프로세스들이 생겨나게 할 수도 있고 그리고/또는 상이한 프로세스 또는 실행 스레드에 의해 프로세싱될 수도 있다. 블록들 (610~616) 에서의 동작들은 모든 외부 자원들이 발견 및/또는 다운로드되기까지 반복될 수도 있다. 일 양태에서, 블록들 (602~608) 의 동작들은 블록들 (610~616) 의 동작들과 병렬로 수행될 수도 있다.

[0143] 종래의 HTML 파서들와는 달리, 도 6을 참조하여 위에서 논의된 스캔 동작들은 스캔된 HTML 문서에 대한 오류 정정을 수행하지 않거나 또는 조우된 JavaScript® 코드를 실행하지 않는다. 이는 스캔 동작들이 빠르게 수행되는 것을 가능하게 한다. 또한, 종래의 HTML 파서들과는 달리, 위에서 논의된 스캔 동작들은 (예컨대, 독립적인 스레드들 또는 프로세스들 등에서) 병렬로 또는 동시에 실행될 수도 있는데, 이는 다양한 양태들이 현대 컴퓨팅 디바이스들에서 우세한 멀티프로세서 아키텍처들을 더욱 충분히 이용하는 것을 가능하게 한다. 덧붙여, 위에서 논의된 스캔 프로세스들은 HTML 문서 (예컨대, CSS 문서들) 에서 참조된 자원들을 스캔할 수도 있는데, 이는 또한 종래의 HTML 파서들에서는 수행되지 않는다.

[0144] 일반적으로, 스캔 동작 (예컨대, HTML 프리-스캐닝 동작들 (563), CSS 스캐닝 동작들 (566) 등) 이 HTML 문서의 구조만을 스캔하면, 예를 들어, 문서에 구조적 에러들이 있지 않는 한 (스캐너가 오류 정정을 수행하지 않기 때문임) 또는 파싱된 대로의 문서를 개조한 문서에 내장된 JavaScript® 코드가 있지 않는 한 (스캐너가 JavaScript®를 실행하지 않기 때문임), 요청되는 (즉, 트루 포지티브들만을 생성하는) 자원들에 관해 올바르게 추정할 가능성이 있다.

[0145] 일 양태에서, 트루 포지티브들 및 트루 네거티브들의 수를 최대화하기 위해, 스캔 동작들 (예컨대, HTML 프리-스캐닝 동작들 (563), CSS 스캐닝 동작들 (566) 등) 은 HTML 문서의 초기 스캔 동안에 획득된 정보를 사용하여 획득될 가능성이 있는 자원들을 식별할 수도 있다.

[0146] 도 7a는 추론적 다운로딩을 위한 문서 자원들을 발견하기 위해 추론 기법들 및 휴리스틱들을 사용하는 일 양태의 브라우저 방법 (700) 을 도시한다. 문서 자원들은 이미지들, CSS 파일들, JavaScript® 스크립트들 등을 포함할 수도 있다. 브라우저 방법 (700) 은 HTML 문서 스캐너 및 복수의 CSS 문서들의 스캐너들이 병렬로 실행하며, 요청될 가능성이 있는 자원들을 지능적으로 식별하며, 추론적 자원 요청들 및/또는 프리-페칭 동작들로 생기는 폴스 네거티브들의 수를 줄이는 것을 가능하게 한다. 일 양태에서, 브라우저 방법 (700) 은 폴스 포지티브들의 수를 최소화하기 위해 휴리스틱 (예컨대, "CSS 규칙" 휴리스틱) 을 이용할 수도 있다.

[0147] 브라우저 방법 (700) 의 블록 702에서, HTML 문서 스캐너 (예컨대, HTML 프리-스캐너 (508)) 는 자원들을 발견하기 위해 HTML 문서의 스캐닝을 시작하고 HTML 문서 포함 HTML 엘리먼트들과 연관된 (또는 그 HTML 엘리먼트들에 의해 언급된) 모든 URL/URI들과, HTML "id", "클래스", 및/또는 "스타일" 속성들을 획득할 수도 있다. HTML 문서 스캐너는 HTML 파서와는 독립적일 수도 있고 그리고/또는 그 HTML 파서와는 병렬로 실행할 수도 있다.

[0148] 블록 704에서, HTML 문서 스캐너는 HTML 문서에 포함된 URL/URI들 및/또는 HTML 엘리먼트들에 의해 참조된 외부 자원들과 조우할 수도 있다. 블록 706에서, HTML 문서 스캐너는 HTML 문서에서 참조된 조우된 자원들을 다운로드하기 위한 요청을 (예컨대, 페치 관리자에게) 발행할 수도 있다. 일 양태에서, HTML 문서 스캐너는 예컨대, 외부 자원들이 스캐너 등에 의해 조우될 때) 각각의 조우된 외부 CSS 자원의 다운로딩 및/또는 파싱을

호출하도록 구성될 수도 있다. 일 양태에서, 외부 CSS 자원의 다운로딩은 CSS 문서 스캐너 (예컨대, CSS 엔진 (512) 등) 로 하여금 CSS 문서의 스캐닝을 개시하게 할 수도 있다.

[0149] 블록 708에서, HTML 문서 스캐너는 HTML id, 클래스, 및 스타일 속성들과 조우하고 그리고/또는 그것들을 수집할 수도 있다. 블록 710에서, HTML 문서 스캐너는 조우된/수집된 정보 (즉, 수집된 id, 클래스, 및 스타일 속성들에 관한 정보) 를 CSS 문서 스캐너로 전송할 수도 있다. 일 양태에서, 수집된 정보를 전송하는 것은 모든 조우된 및/또는 식별된 HTML id, 클래스, 및 스타일 속성을 CSS 문서 스캐너로 전송하는 것을 포함할 수도 있다.

[0150] 블록 712에서, HTML 문서 스캐너는 부가적인 자원들을 발견하기 위해 HTML 문서의 스캐닝을 계속할 수도 있다. 결정 블록 714에서, HTML 문서 스캐너는 HTML 문서의 스캐닝을 완료하였는지의 여부를 결정할 수도 있다. HTML 문서의 스캐닝을 완료하였다고 HTML 문서 스캐너가 결정하는 경우 (즉, 결정 블록 714 = "예"), 블록 716에서, HTML 문서 스캐너는 HTML 문서의 스캐닝을 완료하였다는 것을 (예컨대, 메모리 쓰기 동작, 메소드 호출, 통지 등을 통해) CSS 문서 스캐너 (예컨대, CSS 엔진 (512), CSS 스캐닝 동작들 (566) 을 수행하는 프로세스 등) 에게 통지할 수도 있다. HTML 문서의 스캐닝이 아직 완료되지 않았다고 HTML 문서 스캐너가 결정하는 경우 (즉, 결정 블록 714 = "아니오"), 블록 702에서, HTML 문서 스캐너는 부가적인 자원들을 발견하기 위해 HTML 문서의 스캐닝을 계속할 수도 있다.

[0151] 브라우저 방법 (700) 의 블록 719에서, CSS 문서 스캐너는 외부 자원들에 대한 CSS 문서의 스캐닝을 시작할 수도 있다. 블록 719에서의 CSS 문서 스캐너의 개시는 페치 관리자에 의해 획득된 CSS 문서의 가용성에 의해 (예컨대, 블록 706 등의 일부로서 수행된 동작들에 응답하여) 트리거될 수도 있다. 일 양태에서, CSS 문서들의 스캐닝은 HTML 문서의 스캐닝 (예컨대, 블록들 (702~716) 의 동작들) 과 병렬로 수행될 수도 있다. 따라서, HTML 문서 스캐너가 HTML 문서를 계속 스캔하는 동안 CSS 문서 스캐너는 수신된 CSS 문서들에서 참조된 외부 자원들을 식별하기 위해 (예컨대, 다운로드를 위해 부가적인 CSS 문서들을 식별하는 등을 위해) 수신된 CSS 문서들을 스캔할 수도 있다. 게다가, 병렬로 실행되는 다수의 CSS 문서 스캐너들이 있을 수도 있다 (예컨대, 다수의 CSS 문서들이 다운로드되는 경우임).

[0152] 블록 720에서, CSS 문서 스캐너는 HTML 문서 스캐너로부터 HTML id, 클래스, 및/또는 스타일 속성들에 관한 정보를 수신할 수도 있다. 블록 721에서, CSS 문서 스캐너는, 수신된 정보가 HTML 문서에 의해 요청될 및/또는 사용될 가능성이 있는 것으로서 (수신된 HTML id, 클래스, 및/또는 스타일 속성들에 연관된) CSS 규칙 및/또는 외부 자원을 마킹 또는 식별하는지의 여부를 결정할 수도 있다. 양태에서, 블록 721의 일부로서, CSS 문서 스캐너는 CSS 규칙과 연관된 모든 HTML id, 클래스, 및/또는 스타일 속성이 HTML 문서 스캐너에 의해 이미 조우되었는지의 여부를 결정할 수도 있다.

[0153] 결정 블록 722에서, CSS 문서 스캐너는 (수신된 HTML id, 클래스, 및/또는 스타일 속성들에 연관된) CSS 규칙 및/또는 외부 자원이 HTML 문서에 의해 요청될 및/또는 사용될 가능성이 있는지의 여부를 결정할 수도 있다. 일 양태에서, 결정 블록 722의 일부로서, CSS 문서 스캐너는 HTML 문서에 의해 언급된 모든 URL/URI과, HTML id, 클래스, 및/또는 스타일 속성이 이미 조우되었는지의 여부를 결정할 수도 있다.

[0154] CSS 규칙 및/또는 외부 자원이 HTML 문서에 의해 요청될 및/또는 사용될 가능성이 있다고 CSS 문서 스캐너가 결정하는 경우 (즉, 결정 블록 722 = "예"), 블록 724에서, CSS 문서 스캐너는, 이를테면 메모리 쓰기 동작을 수행하고 및/또는 페치 관리자 (502) 에게 통지함으로써 당해 CSS 규칙에 의해 참조된 다운로드될 자원들을 즉시 요청할 수도 있다.

[0155] 일 양태에서, CSS 문서 스캐너는, HTML 문서에 의해 언급된 모든 URL/URI, 및 HTML id, 클래스, 및/또는 스타일 속성이 이미 조우되었다고 결정되는 경우, CSS 규칙 및/또는 외부 자원이 요청될 가능성이 있다고 결정할 수도 있다.

[0156] CSS 규칙 및/또는 외부 자원이 HTML 문서에 의해 요청되고 및/또는 사용될 가능성이 없다고 CSS 문서 스캐너가 결정하는 경우 (즉, 결정 블록 722 = "아니오"), 블록 723에서, CSS 문서 스캐너는 자원 참조들의 리스트에서의 CSS 규칙에 관한 정보 (예컨대, 수신된 HTML id, 클래스, 및/또는 스타일 속성들) 를 메모리에 저장할 수도 있다. 블록 725에서, CSS 문서 스캐너는, 필요하다면 (예컨대, 스캔될/프로세싱될 부가적인 엘리먼트들이 있는 경우 등에서), CSS 문서의 스캐닝을 계속할 수도 있다.

[0157] 블록 726에서, CSS 문서 스캐너는 HTML 문서 스캐너로부터 HTML 문서 스캐너가 HTML 문서의 스캐닝을 완료하였음을 나타내는 통지를 수신할 수도 있다. 블록 727에서, CSS 문서 스캐너는 메모리에 저장된 자원 참조의

리스트로부터 CSS 규칙에 관한 정보를 취출하고 취출된 정보를 평가할 수도 있다.

[0158] 결정 블록 728에서, CSS 문서 스캐너는, 취출된 정보가 HTML 문서에 의해 요청되고 있는 (또는 요청될 가능성이 있는) CSS 규칙 및/또는 외부 자원을 마킹/식별하는지의 여부를 결정할 수도 있다. 양태에서, 결정 블록 728의 일부로서, CSS 문서 스캐너는 취출된 CSS 규칙과 연관된 모든 HTML id, 클래스, 및/또는 스타일 속성이 HTML 문서 스캐너에 의해 이미 조우되었는지의 및/또는 프로세싱되었는지의 여부를 결정할 수도 있다.

[0159] CSS 규칙 및/또는 외부 자원이 HTML 문서에 의해 요청될 및/또는 사용될 가능성이 있음을 취출된 정보가 마킹/식별하고 있다고 CSS 문서 스캐너가 결정하는 경우 (즉, 결정 블록 728= "예"), 블록 729에서, CSS 문서 스캐너는 당해 CSS 규칙에 대응하는 자원들의 다운로딩을 요청할 수도 있다. 이런 방식으로, HTML 문서 및 CSS 문서들을 동시에 스캐닝함으로써 초래된 폴스 네거티브들의 수는 최소화될 수도 있다. 덧붙여서, 다양한 양태들은 거의 없거나 또는 없는 데이터 전송 비용의 증가, 뿐만 아니라 프로세서 및 네트워크 인터페이스/라디오의 감소된 이용으로 인한 더 적은 소비 전력과 함께 문서 로드 시간들을 감소시킬 (따라서, 응답성을 증가시킬) 수도 있다.

[0160] 도 7a로 되돌아가서, 취출된 정보가 HTML 문서에 의해 요청된 (또는 요청될 가능성이 있는) 것으로서 외부 자원을 마킹 또는 식별하지 않는다고 CSS 문서 스캐너가 결정하는 경우 (즉, 결정 블록 728= "아니오"), 블록 721에서, CSS 문서 스캐너는 메모리로부터 다음의 규칙을 취출할 수도 있다. 블록들 (720~722) 의 동작들은 HTML 문서 스캐너에 의해 메모리에 저장된 모든 CSS 규칙들이 평가되기까지 반복될 수도 있다.

[0161] 다양한 양태들에서, 위에서 설명된 CSS 규칙보다 더욱 정밀한 휴리스틱들이 성능을 개선하기 위해 HTML 문서 스캐너 및/또는 CSS 문서 스캐너에 의해 사용될 수도 있다. 예를 들어, 일 양태에서, HTML 문서 스캐너는 HTML 문서를 수정할 수 있는 커맨드들 및/또는 URL들에 대해 내장된 JavaScript® 코드를 스캔하도록 구성될 수도 있다. 마찬가지로, 일 양태에서, CSS 문서 스캐너는, CSS 문서 스캐너가 더욱 잠재적인 폴스 포지티브들을 식별하고 거절하는 것을 허용할 수도 있는, 각각의 조우된 ID에 연관된 HTML 태그들에 관한 계층적 정보를 기록하도록 구성될 수도 있다.

[0162] 종래의 브라우저들에서, HTML 파서는 외부 자원들의 모두를 식별하고 그것들을 서버들로부터 네트워크를 통해 요청하는 것을 일반적으로 담당한다. 위에서 논의된 바와 같이, 이들 자원들이 HTML 문서에서 명시적으로 특정되는 경우, 다양한 양태들은 이들 자원들을 프리-페치하고 그 요청을 페이지 로드에서 종래의 브라우저들보다 훨씬 빨리 발행할 수도 있다. 덧붙여서, 다양한 양태들은 자원들을 병렬로 프리-페치 및/또는 프로세싱할 수도 있다.

[0163] 소프트웨어 개발자들은 특정 애플리케이션-디바이스 조합 (예컨대, 웹 브라우저-모바일 디바이스 조합) 에 대해 요청될 것인 자원들을 동적으로 결정하기 위해 스크립트들 (예컨대, JavaScript® 코드®) 를 점점 더 사용하고 있다. 예를 들어, 스크립트들은 다운로드될 것인 자원들을 식별하기 위해 클라이언트 (예컨대, 브라우저) 및 컴퓨팅 디바이스에 관련한 다양한 팩터들을 평가할 수도 있다. 이러한 스크립트들은 본질적으로는, 평가된 팩터들에 기초하여 자원 (예컨대, 이미지들, CSS, 다른 JavaScript® 등) 에 대해 동적으로 URL을 구축한다. 따라서, HTML 문서는 HTML 문서에서 명시적으로 식별되지 않은 자원들을 요청할 수도 있는데, 그러한 자원들은 HTML 문서에 포함된 JavaScript® 코드를 실행함으로써만 결정될 수도 있다.

[0164] JavaScript® 코드가 포함하는 HTML의 상태, 거동, 및/또는 프레젠테이션 (과 HTML 코드 자체) 을 변경할 수도 있으므로, HTML 파서에게는 조우된 JavaScript® 코드 (또는 스크립트들) 를 순차적으로 및/또는 HTML 사양들에 의해 정의된 순서화 규칙들을 추종함으로써 실행할 것이 요청된다. 예를 들어, HTML 파서가 스크립트 태그 (즉, 클라이언트-측 스크립트, 이를테면 JavaScript® 스크립트를 정의하는데 사용된 <script> 태그) 와 조우하는 경우, HTML 파서는 HTML 문서의 나머지 부분들의 파싱을 계속할 수도 있기 전에 다운로드되고 실행될 스크립트를 기다려야 한다. 그 결과, 모든 자원 요청들은 JavaScript® 스크립트 (즉, <script> 태그들 내부의 JavaScript® 코드) 의 실행 내에서 직렬화될 (즉, 차례로 수행될 것이 요청될) 수도 있다. 또한, 웹페이지를 적절한 렌더링하기 위해 요청될 것인 자원들을 정적으로 예측하기 위한 HTML 문서 스캐닝 동작들 (예컨대, HTML 프리-스캐닝 동작들 (563) 등) 이 더욱 곤란할 수도 있다.

[0165] 다양한 양태들은 샌드박스식 JavaScript® 엔진 (530) 에서 자원들을 추론적으로 프리-페치함으로써 이들 및 다른 제한들을 극복할 수도 있는데, 이는 브라우저 시스템 (500) 이 HTML 문서에서 명시적으로 요청되지 않은 자원들을 다른 브라우저 동작들 (예컨대, HTML 파싱) 및 다른 자원 요청들과는 병렬로 발견하고 다운로드하는 것을 가능하게 한다. 이들 양태들은 브라우저 상태를 의도적이지 않게 수정하는 일 없이 브라우저 시스템

(500) 이 다수의 JavaScript® 스크립트들을 병렬로 실행하는 것을 또한 가능하게 할 수도 있다.

[0166] 다양한 양태들은 스크립트들 (예컨대, JavaScript® 코드) 을 그것들이 발견되자마자 다른 브라우저 동작들 (예컨대, HTML 프리-스캐닝 (563), HTML 파싱 (568) 등) 및/또는 다른 스크립트들과는 병렬로 실행할 수도 있다.

웹페이지의 정상적인 프로세싱과의 간섭을 피하기 위하여, 스크립트들은 다른 브라우저 컴포넌트들로부터 (예컨대, 기본 JavaScript® 엔진의 동작들에 영향을 미치지 않기 위해서) 격리 및/또는 분리되는 샌드박스식 JavaScript® 엔진 (530) 에서 실행될 수도 있다. 샌드박스식 JavaScript® 엔진 (530) 에서 스크립트들을 실행하는 것은 스크립트들의 병렬 실행 동안 시스템이 브라우저 상태를 의도적이지 않게 수정하는 것을 방지한다. 일 양태에서, 각각의 스크립트는 샌드박스식 JavaScript® 엔진 (530) 의 별개의 인스턴스 (예컨대, 스레드) 에서 실행될 수도 있다.

[0167] 다양한 양태들은 브라우저 클라이언트와 JavaScript® 엔진 (530) 사이의 API를 수정할 수도 있다.

[0168] 일반적으로, 스크립팅 엔진들 (예컨대, JavaScript® 엔진 (514, 530, 558)) 은 브라우저 동작들 (예컨대, DOM 조작, 네트워크 액세스 등) 을 호출하기 위해 브라우저 API (즉, 스크립트들이 브라우저 동작들을 호출하는 것을 가능하게 하는 인터페이스) 에 바인딩들 (즉, 언어들을 매핑하기 위한 API) 을 제공한다.

[0169] 일 양태에서, JavaScript® 엔진 (530) 은 네트워크로부터 자원들을 요청하는 브라우저 API들을 모니터링할 수도 있다. JavaScript® 엔진 (530) 은 자원 요청들이 상이한 브라우저 컴포넌트, 이를테면 프리-페처 컴포넌트로 재지향되게 하기 위해 바인딩들을 수정 (또는 스크립팅 엔진을 위한 바인딩들의 별개의 세트를 제공) 할 수도 있다. 이런 방식으로, 자원 요청들 및/또는 수집된 정보는 추가의 프로세싱을 위해 프리-페처 컴포넌트로 직접 전해질 수도 있다.

[0170] 샌드박스식 JavaScript® 엔진은 JavaScript® 코드를 통해 스캔하고 그 코드의 선택된 부분들만을 실행하고 및/또는 외부 자원들을 발견하는 것에 가장 관련된 동작들을 선택할 수도 있다. 스크립트가 요청할 수도 있는 자원들을 발견하는 것에만 스캐닝 동작이 관련되므로, 스캐닝 동작은 HTML 사양 규칙들에 의해 바인딩되지 않고, 조우된 코드의 모두를 동작/실행할 필요가 없다. 조우된 코드의 모두를 완전히 실행하지 않음으로써, JavaScript® 스캐닝 동작들은 샌드박스식 JavaScript® 엔진에 의해 빠르게 수행될 수도 있다.

[0171] 샌드박스식 JavaScript® 엔진은 JavaScript® 스캐닝 동작들을 추가로 스피드업하기 위해 휴리스틱들을 적용할 수도 있다. 예로서, 이러한 휴리스틱들은 총 실행 시간 (예컨대, 스크립트 또는 동작 당 최대 10ms를 소비하는 등임), 루프 반복들의 수 (예컨대, 루프의 처음 10회 반복들만을 프로세싱하는 것 등임), 되풀이 깊이, 지원되는 피처들, 추상적 해석 등을 제한하는 것을 포함할 수도 있다.

[0172] 다양한 양태들은 JavaScript® 스캐닝 동작들을 추가로 스피드업하기 위해 오브젝트 및 데이터 구조들 (예컨대, 해시 테이블들, 어레이들 등) 의 사이즈들을 제한할 수도 있는데, 이러한 구조들이 일반적으로는 자원 의존성들에 영향을 미치지 않기 때문이다.

[0173] 소프트웨어 개발자들은 그들의 코드 내에 공통 패턴들, 프레임워크들, 및/또는 서비스들 (본원에서는 총칭하여 "패턴들") 을 종종 사용한다. 다양한 양태들은 코드에서 이러한 공통성들/패턴들을 (예컨대, 파싱, 분석, 컴파일 등의 동안) 검출하고, 자원들을 발견하는 것에 관련한 패턴들 (또는 그 패턴들에 의해 식별된 JavaScript® 코드의 부분들) 만을 실행할 수도 있다. 일 양태에서, 완전한 컴파일 및 보수적인 코드 생성 대신, 샌드박스식 JavaScript® 엔진은 가장 공통의 패턴들을 (예컨대, 공격적 컴파일러 최적화들을 통해) 타켓으로 구성될 수도 있다. 패턴들은 매우 다양한 알려진 패턴 인식 기법들, 이를테면 코드에서 키워드들을 검출하는 것 (이것은 비교적 간단한 동작임) 그리고/또는 페이지 및/또는 스크립트의 구조를 분석하는 것 (이것은 상대적으로 복잡한 동작임) 을 사용하여 검출될 수도 있다.

[0174] 도 7b는 샌드박스식 JavaScript® 엔진에서 스크립트들의 병렬 프로세싱에 의해 자원들을 병렬로 추론적으로 프리-페치하는 일 양태의 방법 (730) 을 도시한다. 방법 (730) 의 동작들은 본원에서 논의되는 다른 브라우저 동작들과 병렬로 수행될 수도 있다.

[0175] 방법 (730) 의 블록 732에서, HTML 문서 스캐너 (예컨대, HTML 프리-스캐너 (508)) 는 구조 정보를 위해 그리고/또는 자원들을 발견하기 위해 HTML 문서의 스캐닝을 시작할 수도 있다. 블록 734에서, HTML 문서 스캐너는 JavaScript® 스크립트와 조우하고, 조우된 스크립트를 즉시 실행하기 위해 조우된 스크립트를 (예컨대, 메모리 쓰기 동작, 재지향된 자원 요청, 수정된 바인딩들 등을 통해) 샌드박스식 JavaScript® 엔진으로 전송할 수도 있다. 블록 732에서, HTML 문서 스캐너는 구조 정보를 위해 그리고/또는 자원들을 발견하기 위해 HTML 문서를 계속 스캔할 수도 있다. 일 양태에서, HTML 문서 스캐너는 스크립트의 조우에 응답하여 샌드박스식

JavaScript® 엔진을 생성 (또는 생기게) 할 수도 있다.

[0176] 블록 735에서, 샌드박스식 JavaScript® 엔진은 자원들을 발견하기 위해 스크립트의 스캐닝을 시작할 수도 있다. 블록 736에서, 샌드박스식 JavaScript® 엔진은 스크립트 (또는 스크립트에 포함된 JavaScript® 코드의 부분들) 를 추론적으로 실행할 수도 있다. 스크립트의 추론적 실행은 외부 자원들의 발견에 관련될 가능성이 가장 큰 코드의 부분들 및/또는 동작들만을 실행하는 것을 포함할 수도 있다. 다양한 양태들에서, 추론적 실행 동작들은 다른 브라우저 동작들 (예컨대, HTML 프리-스캐닝 (563), HTML 파싱 (568) 등) 과는 병렬로 그리고/또는 (추론적이든 아니든) 다른 스크립트들의 실행과는 병렬로 수행될 수도 있다.

[0177] 일 양태에서, 스크립트의 추론적 실행은 자원들의 발견에 관련된 패턴에 대응하는 JavaScript® 코드의 부분들만을 실행하는 것을 포함할 수도 있다.

[0178] 일 양태에서, 블록 736의 일부로서, 샌드박스식 JavaScript® 엔진은 휴리스틱들에 기초하여 (예컨대, 실행 시간을 줄이기 위해) JavaScript® 코드의 추론적 실행을 수행할 수도 있다. 이러한 휴리스틱들은 총 실행 시간, 루프 반복들의 수, 되풀이 깊이, 지원되는 피처들, 및/또는 코드의 추상적 해석의 제한을 포함할 수도 있다.

[0179] 일 양태에서, 블록 736의 일부로서, 샌드박스식 JavaScript® 엔진은 스크립트의 추론적 실행으로부터 생성된 데이터 구조들 (예컨대, 해시 테이블들, 어레이들 등) 의 사이즈들을 제한할 수도 있다. 완전한 데이터 구조들은 다운로딩을 위해 추가의 자원들을 식별하게 하지 않을 수도 있으며, 그래서 큰 데이터 구조를 충분히 생성/차지하는데 필요한 프로세싱 시간이 바이패스될 수 있다.

[0180] 블록 738에서, 샌드박스식 JavaScript® 엔진은 HTML 문서를 렌더링하기 위하여 필요하지만 HTML 문서에서 명시적으로 요청되지 않은 자원을 발견할 수도 있다. 블록 740에서, 샌드박스식 JavaScript® 엔진은 발견된 자원을 취출하도록 프리-페처에 알릴 (또는 생기게 할) 수도 있다. 블록 742에서, 샌드박스식 JavaScript® 엔진은 블록 736에서 수행된 프로세싱의 결과들을 버릴 수도 있다.

[0181] 블록 744에서, 프리-페처는 블록 738에서 샌드박스식 JavaScript® 엔진에 의해 발견되는 자원들을 로케이트할 수도 있다. 블록 746에서, 프리-페처는 로케이트된 자원을 다운로드할 수도 있다. 블록 748에서, 프리-페처는 다운로드된 자원을 메모리에 저장할 수도 있다.

[0182] 위에서 논의된 바와 같이, HTML 코드는 JavaScript® 코드를 내장할 수도 있고 ("인라인 스크립트들"라고 지칭됨) JavaScript® 코드에 대한 링크들을 포함할 수도 있다 ("외부 스크립트들"이라고 지칭됨). HTML 문서를 올바르게 프로세싱하기 위하여, 인라인 및 외부 스크립트들 양쪽 모두는 HTML 표준들에 의해 정의된 특정 순서로 실행되어야만 한다.

[0183] 다수의 스크립트들이 병렬로 다운로드되며, 파싱되며, 분석되고, 컴파일되므로, 스크립트들이 실행할 준비가 될 순서는 HTML 표준들에 의해 정의된 특정 실행 순서와는 상이할 수도 있다. 스크립트가 실행할 준비가 되지 않지만 HTML 표준들에 의해 정의된 특정 실행 순서에서의 다음의 스크립트이면, HTML 문서의 임의의 부가적인 프로세싱을 수행하기 전에 스크립트가 실행할 준비가 되기까지 브라우저가 기다릴 필요가 있을 수도 있다. 다양한 양태들은 (HTML 표준들에 의해 조정되지 않는) 실행에 대해 다른 스크립트들 또는 자원들을 준비하기 위해 이 대기 시간을 이용한다. 다수의 스크립트들 및 자원들은 병렬로 및/또는 다른 스크립트들의 실행 동안 준비될 수도 있다.

[0184] 덧붙여서, HTML 문서에 포함된 (즉, 내장된 또는 링크된) 스크립트들의 모두가 실제로는 실행되지 않고, 실행을 위한 모든 스크립트들을 미리 준비하는 것은 전력 및 프로세싱 자원들을 낭비할 수도 있다. 다양한 양태들은 실행을 위해 준비될 스크립트들을 지능적으로 선택할 수도 있다.

[0185] 예로서, HTML 프리-페처가 모든 참조된 스크립트들을 (비순차로) 발견하고 다운로드할 수도 있고 HTML 파서는 나중에 그것들의 실행을 올바른 순서로, 그리고 HTML 문서를 프로세싱하는 올바른 시점 (point in time) 에 조정할 수도 있다.

[0186] 스크립트들의 최종 실행 순서는 일반적으로 유지되어야만 한다. 그러나, 스크립트들의 다운로딩, 파싱, 분석, 및 컴파일에 연관된 모든 동작들은 병렬로 및/또는 비순차적으로 수행될 수도 있다.

[0187] 일 양태에서, HTML 문서에 포함된 스크립트들은 병렬로의 (즉, 서로에 대해서임) 및 비순차적으로의 (즉, HTML 표준들에 의해 정의된 특정 실행 순서에 대해서임) 실행을 위해 준비될 수도 있다. 이는 고유 식별자 및/또는 서명을 생성 및/또는 각각의 스크립트와 연관시킴으로써 달성될 수도 있다. 서명들은 스크립트의 내용에

기초할 수도 있다. 다양한 양태들에서의 사용에 적합한 서명들 및 사이닝 (signing) 프로세스들의 예들은 파일 오프셋들 (인라인 스크립트들을 위한 것임), 메시지-다이제스트 알고리즘 (예컨대, MD5), 보안 해시 알고리즘 (secure hash algorithm; SHA), 스크립트의 URL, 스크립트의 URI, 브라우저 캐시 키들, 및/또는 다양한 알려진 사이닝 프로세스들 중 임의의 것을 포함한다.

[0188] 도 7c는 병렬 실행을 위해 HTML 문서에 포함되는 스크립트들을 지능적으로 준비하기 위한 일 양태의 브라우저 방법 (750) 을 도시한다. 방법 (750) 의 동작들은 다른 브라우저 동작들과는 병렬로 프로세서에 의해 수행될 수도 있다.

[0189] 블록 752에서, HTML 스캐너/프리-페처가 구조 정보를 위해 그리고/또는 자원들 (이미지들, CSS, 스크립트들 등) 을 발견하기 위해 HTML 문서를 스캔할 수도 있다. 블록 754에서, HTML 스캐너/프리-페처는 HTML 문서에서 하나 이상의 스크립트들을 발견하고, (HTML 스캐너와는 병렬로 실행하는) HTML 파서에게 발견된 스크립트들을 알려줄 수도 있다. 블록 756에서, HTML 스캐너/프리-페처는 외부 스크립트들의 다운로딩을 개시할 수도 있다.

[0190] 블록 758에서, HTML 파서는 각각의 발견된 스크립트 (인라인 및 외부 스크립트들 양쪽 모두) 에 대해 식별자 (또는 서명) 를 생성하고 및/또는 각각의 발견된 스크립트와 식별자를 연관시킬 수도 있다. 일 양태에서, HTML 파서는 발견된 스크립트의 텍스트를 그것의 식별자로서 설정할 수도 있다. 일 양태에서, HTML 파서는 외부 스크립트들의 URL/URI와 외부 스크립트들을 연관시킬 수도 있고 (즉, 외부 스크립트들의 URL/URI를 외부 스크립트들의 서명으로서 설정할 수도 있고), 인라인 스크립트들에 대한 서명들을 컴퓨팅하기 위해 다이제스트 및/또는 해시 알고리즘을 수행할 수도 있다. 스크립트의 URL/URI가 이용가능하지 않으며, 고유한 것이 아니고 그리고/또는 다르게는 스크립트를 고유하게 식별하지 않으면, 블록 758의 일부로서, HTML 파서는 당해 스크립트를 식별하기 위해 서명을 생성하고 사용할 수도 있다.

[0191] 블록 760에서, HTML 파서는 스크립트들 및 그것들의 연관된 식별자들 또는 URL/URI를 HTML 파서와는 병렬로 (예컨대, 별개의 스레드에서) 실행하는 JavaScript® 엔진으로 전송할 수도 있다. 블록 762에서, HTML 파서는 다양한 HTML 파서 동작들, 이를테면 다른 스크립트들을 발견하기 위한 HTML의 파싱을 수행할 수도 있다.

[0192] 블록 772에서, JavaScript® 엔진은 HTML 파서로부터 스크립트들 및 연관된 식별자들, 서명들, 또는 URL/URI를 수신할 수도 있다. 블록 774에서, JavaScript® 엔진은 수신된 스크립트들을 실행을 위해 준비 (예컨대, 파싱, 분석, 및/또는 컴파일) 할 수도 있다. 준비 동작들은 모든 수신된 스크립트들에 걸쳐 비순차적으로 및/또는 병렬로 수행될 수도 있다 (즉, 다수의 스크립트들이 한꺼번에 준비될 수도 있다). 일 양태에서, 블록 774의 일부로서, JavaScript® 엔진은 코드의 실행 없이 호출 그래프를 검출하기 위해 (예컨대, 추상화 해석을 통해) 휴리스틱들을 채용하며, 호 그래프에 기초하여 실행될 가능성이 가장 큰 스크립트들 (또는 기능들) 을 식별하고, 실행할 가능성이 있다고 결정된 스크립트들만을 실행을 위해 준비할 수도 있다. 블록 776에서, JavaScript® 엔진은 스크립트의 준비 동안에 생성된 정보 (예컨대, 컴파일된 코드 등) 와 그 스크립트의 식별자, 서명 또는 URL/URI를 연관시킬 수도 있다.

[0193] 블록 764에서, HTML 파서는 (예컨대, HTML 표준들에 의해 정의된 실행 순서에 기초하여) 실행될 다음의 스크립트를 식별할 수도 있다. 블록 766에서, HTML 파서는 실행될 다음의 스크립트의 식별자 (예컨대, 스크립트의 텍스트, 서명, URL/URI, 등) 를 JavaScript® 엔진에 전송할 수도 있다. 블록 768에서, HTML 파서는 실행의 결과 또는 스크립트가 실행되었다는 통지를 기다릴 수도 있다. 블록 770에서, HTML 파서는 HTML 파서 동작들의 수행을 계속할 수도 있다.

[0194] 블록 778에서, JavaScript® 엔진은 HTML 파서로부터 식별자, 서명, 또는 URL/URI를 수신할 수도 있다. 블록 780에서, JavaScript® 엔진은 수신된 식별자, 서명 또는 URL/URI에 기초하여 적절한 스크립트를 식별할 수도 있다. 결정 블록 782에서, JavaScript® 엔진은, 예를 들어, 파싱, 분석, 및 컴파일 동작들의 모두가 당해 스크립트를 위해 수행되었는지의 여부를 결정함으로써, 식별된 스크립트가 즉시 실행할 준비가 되어 있는지의 여부를 결정할 수도 있다. 스크립트가 즉시 실행할 준비가 되어 있다고 JavaScript® 엔진이 결정하면 (즉, 결정 블록 782= "예"), 블록 786에서, JavaScript® 엔진은 실행의 결과들 또는 실행이 완료되었음을 HTML 파서에게 알려줄 수도 있다.

[0195] 스크립트가 즉시 실행할 준비가 아직 되어있지 않다고 결정되는 경우 (즉, 결정 블록 782= "아니오"), 블록 784에서, JavaScript® 엔진은 종래의 솔루션들을 사용하여 실행을 위한 스크립트를 준비할 수도 있다. 블록 786에서, JavaScript® 엔진은 HTML 표준들에 의해 정의된 특정 실행 순서에 따라 스크립트를 실행할 수도

있다. 이런 방식으로, 방법 (750) 은 HTML 문서에 포함된 스크립트들을 병렬로의 (즉, 서로에 대한 것임) 및 비순차적으로의 (즉, HTML 표준들에 의해 정의된 특정 실행 순서에 대한 것임) 실행을 위해 준비하고, 스크립트들은 표준들에 의해 정의된 순서로 실행된다.

[0196] 도 8은 프리-페치된 자원들을 프로세싱하는 일 양태의 브라우저 방법 (800) 을 도시한다. 블록 802에서, 웹 브라우저 컴포넌트는 (예컨대, 페치 관리자 (502) 를 통해) 발견된 자원 (예컨대, 이미지) 의 다운로딩을 개시할 수도 있는데, 발견된 자원은 다른 브라우저 동작들 (예컨대, HTML 파싱 등) 의 수행과 동시에 (또는 병렬로) 다운로드/페치될 수도 있다. 발견된 자원에 연관된 모든 데이터가 다운로드 및/또는 수신되는 경우, 블록 804에서, 다운로드된 데이터 (예컨대, 이미지 데이터) 는 디코딩을 위해 스레드 풀로 전송될 수도 있다. 일 양태에서, 디코딩 동작들은 다른 브라우저 동작들과 동시에 수행될 수도 있다.

[0197] 블록 806에서, 다운로드된 데이터 (예컨대, 이미지 데이터) 는 디코딩될 수도 있다. 블록 808에서, 디코딩된 데이터는 DOM 디스패처 큐 (queue) 에 추가될 수도 있다. 블록 810에서, DOM 디스패처 컴포넌트 (504) 는 DOM 트리 및 개별 트리 노드들 (예컨대, 이미지 데이터의 경우 "img" 트리 노드) 에 대한 업데이트들을 직렬화할 수도 있다. 블록 812에서, 자원 (예컨대, 이미지) 은 프로세싱 리스트 (예컨대, 보류중인 이미지들의 리스트) 로부터 제거될 수도 있다.

[0198] 도 9는 다양한 양태들과 함께 사용하기에 적합한 CSS 엔진 (512) 에서의 예의 컴포넌트들을 도시한다. CSS 엔진 (512) 은 다음 3 개의 주요 카테고리들의 동작들을 수행하도록 구성될 수도 있다: CSS 자원 프리페칭 동작들 (902), CSS 파싱 동작들 (904), 및 DOM 스타일링 동작들 (906).

[0199] CSS 파싱 동작들 (904) 은 CSS 코드의 판독과 메모리에서의 데이터 구조들 (예컨대, CSS 규칙들) 의 컬렉션의 생성을 포함할 수도 있다. CSS 코드는 HTML에 내장 또는 별개의 파일들로서 링크될 수도 있고, 상이한 서버들 상에 저장될 수도 있다. 전통적인 CSS 엔진들 (예컨대, WebKit 또는 파이어폭스에서의 것들) 은 메인 브라우저 스레드에서 CSS를 순차적으로 파싱할 수도 있다. 따라서, 페이지가 내장된 CSS를 사용하면, CSS 엔진이 문서의 헤더에서의 스타일 엘리먼트를 파싱하기까지, HTML 파서는 HTML 문서의 나머지를 파싱하지 않을 수 있다. 페이지가 여러 CSS 파일들을 사용하면, 비록 활용되지 않는 CPU 코어들이 있을 수도 있을지라도, 그 CSS 파일들은 순차적으로 파싱될 것이다. 이러한 CSS 파싱 직렬화 (serialization) (즉, CSS 문서들의 직렬적 프로세싱) 는 사이트가 큰 CSS 파일들을 사용하면 심각한 속도 둔화를 초래할 수도 있다. 다양한 양태들은 CSS 파싱 직렬화를 피하기 위해 비동기적 태스크들을 사용할 수도 있다.

[0200] 도 9를 참조하면, HTML 파서 (506) 는 페이지 로드 동작 동안 DOM 트리에서의 각각의 스타일 엘리먼트에 대해 CSS 파싱 (570) 태스크를 생기게 하도록 구성될 수도 있다. 마찬가지로, 페치 관리자 (502) 는 새로운 CSS 파일이 도착할때마다 CSS 파싱 (570) 태스크를 생기게 할 수도 있다. 그 결과, 다수의 CSS 파싱 (570) 태스크들은 HTML 파서 (506) 및/또는 HTML 파싱 동작들 (568) 과 동시에 실행할 수도 있다.

[0201] 스타일 시트들 (CSS) 및 규칙 (CSS 규칙들) 의 전체 순서가 스타일링 동작들 (574) 의 핵심 부분일 수도 있기 때문에, 모든 스타일 시트들 (CSS) 이 프로그래머가 의도했던 순서로 파싱되었던 듯이, 브라우저 시스템 (500) 은 전체 순서가 동일할 것을 보장하도록 구성될 수도 있다.

[0202] 다양한 양태들에서, 파싱 태스크들 또는 파싱 동작들 (568, 570) 의 각각은 고유한, 순차적 파서 ID를 수신할 수도 있다. 브라우저 시스템 (500) 은 그 다음에 문서에서의 스타일 시트들의 순서를 재생성하기 위해 당해 ID를 사용할 수도 있다.

[0203] DOM 스타일링 동작들 (906) 은 DOM 트리에서 노드들의 스타일을 결정하기 위해 CSS 엔진 (512) 이 CSS 파서 (522) 에 의해 생성된 데이터 구조들을 사용하는 것을 가능하게 할 수도 있다. 각각의 노드에 대해, CSS 엔진 (512) 은 선택자들이 노드와 일치하는 모든 규칙들을 찾기 위해 규칙 매칭 동작들을 수행할 수도 있다. 규칙매칭은 일반적으로 노드마다 많은 (그리고 때때로 모순되는) 규칙들을 반환한다. 캐스케이딩을 사용하여, CSS 엔진 (522) 은 규칙들에 가중치들을 할당하고 가장 큰 가중치를 갖는 규칙들을 선택할 수도 있다.

[0204] 노드를 스타일링함에 있어서의 마지막 단계는 캐스케이딩 알고리즘에 의해 선택된 규칙들을 사용하고 그것을 DOM에 부속시킴으로써 스타일 데이터 구조를 생성하는 DOM 스타일링 동작들 (906) 을 포함할 수도 있다. 규칙 매칭 및 캐스케이딩 동작들은 특정 의존성들이 강제되는 한 여러 노드들에 대해 병렬로 수행될 수도 있다.

[0205] 다양한 양태들은 다수의 브라우저 동작들 및/또는 패스들의 병행하는 실행 (또는 중첩) 동안 기존의 HTML 및 JavaScript® 시맨틱스를 준수/강제할 수도 있다. DOM 트리가 모든 브라우저 패스들에 의해 사용된 메인 데이터 구조일 수도 있다. 다양한 양태들에서, DOM 트리에 대한 액세스 (이는 HTML5 파서에 의해 수행될 수도

있음) 가 HTML5 사양을 준수하도록 직렬화될 수도 있다. 덧붙여서, 더 큰 병렬성을 허용하기 위해, 각각의 패스들에는 병행하는 데이터 구조를 (즉, DOM 트리에 더하여) 개인화하기 위한 액세스가 제공될 수도 있다. 일 양태에서, 이 부가적인 데이터 구조는 레이아웃 트리일 수도 있다.

[0206] 도 10은 규칙 매칭 및 캐스케이딩 동작들이 여러 노드들 상에서 병렬로 수행되는 일 실시형태의 병렬 DOM 스타일링 방법 (1000) 을 도시한다. 블록 1002에서, CSS 엔진 (512) 은 DOM 트리를 트래버스 (traverse) 하고 DOM 노드 당 2 개의 상이한 태스크들, 즉, 매칭 태스크, 및 노드 스타일링 태스크를 생기게 할 수도 있다. 블록 1004에서, 매칭 태스크는 DOM 노드에 대해 규칙 매칭 및 캐스케이딩 동작들을 수행할 수도 있다. 블록 1006에서, 스타일링 태스크는 DOM 노드를 기술하는 스타일 데이터 구조를 생성할 수도 있다. 블록 1008에서, 스타일링 태스크는 스타일 데이터 구조를 DOM 트리에 부속시킬 수도 있다.

[0207] 도 11a는 다양한 양태들에서의 사용에 적합한 일 예의 DOM 트리를 예시한다. 도 11b는 도 11a에 예시된 예의 DOM 트리에 대응하는 일 예의 태스크 지향성 비순환 그래프 (directed acyclic graph; DAG) 를 도시한다. 구체적으로는, 도 11b는 어떻게 매칭 태스크들 (삼각형들로서 나타내어짐) 이 완전히 서로 독립적이고 스타일링 태스크들 (정사각형들로서 나타내어짐) 과 독립적일 수도 있는 반면, 스타일링 태스크들이 서로 및 매칭 태스크들에 의존하는가를 예시한다. 일반적으로, 매칭 태스크들의 병렬 실행은 컴퓨팅 시스템에서의 프로세싱 코어들의 수에 의해서만 제한된다.

[0208] 위에서 언급했듯이, 스타일링 태스크들은 서로 및/또는 매칭 태스크들에 의존할 수도 있다. 각각의 스타일링 태스크는 그것이 실행할 수 있기 전에 2 개의 의존성들을 충족할 것이 필요할 수도 있다. 먼저, 동일한 노드 상에 작동하는 매칭 태스크가 실행을 완료한 후에만 스타일링 태스크가 실행할 수도 있다. 이는 스타일링 태스크가 매칭 태스크에 의해 선택된 규칙들을 사용하여 스타일 데이터 구조를 구축하기 때문이다. 둘째로, 노드의 부모 상에서 작동하는 스타일링 태스크가 실행을 완료한 후에만 노드 상에서 작동하는 스타일링 태스크가 실행할 수도 있다. 이는 노드의 스타일 속성들의 일부가 그것의 부모의 것들로부터 상속할 수도 있기 때문이다. 예를 들어, CSS 코드 p {color: inherit}는 <p> 노드들을 그 부모들과 동일한 전경색을 사용하여 렌더링할 것을 브라우저에게 지시한다.

[0209] 매칭 태스크들에 의해 수행되는 규칙 매칭 동작들은 계산, 전력, 레이턴시 등의 측면들에서 값비쌀 수도 있다. 예를 들어, 규칙 "h1 p div {color:red}"를 엘리먼트 E에 적용할지의 여부를 결정하는 것을 CSS 엔진 (512) 이 필요로 하면, 매칭 알고리즘은 E의 조상들 중 임의의 것이 <p> 엘리먼트인지와, <p>의 조상들 중 임의의 것이 <h1> 엘리먼트인지의 여부를 확인하는 것을 필요로 할 수도 있다. 이는 루트까지 계속 DOM 트리를 보행 (walk) 으로 올라가는 것을 필요로 할 수도 있는데, 이는 값비싼 동작일 수도 있다. 덧붙여서, 전형적인 웹사이트가 400,000을 초과하는 이러한 DOM 트리 보행들을 필요로 할 수도 있다.

[0210] DOM 트리 보행들의 수를 줄이기 위해, 다양한 양태들은 DOM 노드의 조상들에 관한 정보를 저장하는 블룸 필터 (bloom filter) 를 포함할 수도 있다. 블룸 필터는 루트까지의 DOM 트리 보행들의 수 (A) 를 스타일링 알고리즘에서 소비된 시간의 90%만큼 감소시켜, 스타일링 알고리즘에서 소비된 시간을 반으로 줄일 수도 있다.

[0211] 블룸 필터는 큰 데이터 구조일 수도 있고, CSS 엔진 (512) 은 각각의 스타일링 태스크를 위해 그것을 복사할 것이 요구될 수도 있다. Since 복사 비용이 성능 이득보다 훨씬 더 클 수도 있으므로, 다양한 양태들은 블룸 필터보다 작은 구조를 사용할 수도 있다. 이는 복사 동작들의 수를 감소시키고 및/또는 복사된 엘리먼트들의 사이즈를 감소시킴으로써 브라우저 성능을 개선할 수도 있다.

[0212] 위에서 설명된 바와 같이, 다양한 양태들은 CSS 파일에서 참조된 이미지가 프리패치되어야 하는지의 여부를 예측하기 위해 엘리먼트 id 및 클래스 속성들을 사용할 수도 있다. 일 양태에서, 이들 엘리먼트들 및 속성들은 그것들의 각각이 문서에서 얼마나 많은 횟수로 나타나는지를 기록하는 데이터베이스에 저장될 수도 있다. HTML 파서는 또한 이 데이터베이스에 정보를 추가할 수도 있다.

[0213] 규칙 매칭 알고리즘이 시작하기 전에, CSS 엔진 (512) 은 데이터베이스에서의 아이템들을 그것들의 빈도에 따라 정렬할 수도 있다. 브라우저 시스템 (500) 은 그 다음에 비트맵 데이터 구조 ("매칭 비트맵들"이라고 지칭됨) 에서의 각각의 아이템에 비트를 할당할 수도 있다. ID들 및 클래스들의 수가 비트맵 사이즈보다 크면, 단일 비트가 다수의 아이템들에 할당될 수도 있다. 이들 비트맵들이 작으므로, 그것들은 컴퓨팅 디바이스의 성능에 크게 영향을 미치지 않고 다수 회 복사될 수도 있다.

[0214] 규칙 매칭 동작들 동안, 각각의 스타일링 태스크는 그것의 부모로부터 매칭 비트맵을 수신할 수도 있다. 매칭 비트맵은 그것의 조상들의 ID들, 클래스들, 및 태그들을 기록할 수도 있다. 스타일링 태스크들은 결코

일치할 수 없는 규칙들을 걸러내기 위해 매칭 비트맵을 사용할 수도 있다. 그 후, 스타일링 태스크들은 그것들의 노드의 id, 클래스, 및 태그를 추가하고 그것들의 후손 (descendant) 들에 복사본을 전송할 수도 있다. 평균하여, 이러한 매칭 비트맵들은 0.024%의 폴스 포지티브들만으로 DOM 트리의 루트까지의 보행들의 90%를 회피한다.

[0215] 레이블들 및 ID들이 조우되는 순서를 매칭 비트맵들이 기록하지 않기 때문에 폴스 포지티브들이 발생할 수도 있다. 예를 들어, 규칙 "h2 h1 p {color: red}"가 특정 노드 <p>에 적용될지의 여부와, <h1> 및 <h2> 양쪽 모두가 <p>의 조상들이라는 것을 매칭 비트맵이 나타낸다고 결정하기 위해, 브라우저 시스템 (500) 은 <h2>가 <h1>의 조상인지의 여부를 체크하기 위해 DOM 트리를 보행하여 올라갈 것이 요구될 수도 있다. 그것이 그 경우가 아니면, 그것은 폴스 포지티브 상황이다. 이러한 폴스 포지티브들은 페이지가 올바르지 않게 렌더링되게 하지 않을 수도 있지만, CPU 사이클들을 낭비할 수도 있다.

[0216] 일 양태에서, 이를테면 렌더링 엔진 서브시스템 (560) 에 의한 레이아웃 및 렌더링 동작은, 스크린 상의 디스플레이를 위해 스타일링된 DOM을 비트맵 이미지로 변환하는 계산들을 수행하는 것을 포함할 수도 있다. 비트맵 이미지에 적용된 DOM 및 CSS 스타일들은 새로운 트리 구조 (레이아웃 트리라 지칭됨) 를 형성하도록 결합될 수도 있는데, 이 트리 구조에서 각각의 노드는 웹 페이지 상의 시각적 엘리먼트를 나타낸다. 각각의 DOM 노드는 0, 1, 또는 많은 레이아웃 트리 노드들로 해석될 수도 있다. 렌더링 엔진 서브시스템 (560) 은 레이아웃 트리를 입력으로서 취하고 각각의 엘리먼트가 점유하는 페이지의 지역을 컴퓨팅할 수도 있다. 각각의 엘리먼트의 스타일은 레이아웃 (예컨대, 인라인/블록 디스플레이, 플로트 (float), 폭, 높이 등) 에 대한 제약으로서 보일 수도 있다.

[0217] 렌더링 엔진 서브시스템 (560) 은 각각의 엘리먼트의 최종 폭, 높이, 및 포지션을 결정하기 위해 레이아웃 트리를 트래버스하고 (예컨대, 레이아웃 동작들 (582) 의 일부로서) 제약들을 해결할 수도 있다. 렌더링 엔진 서브시스템 (560) 은 또한, (레이아웃 엔진의 계산들의 결과들로 주석이 달릴 수도 있는) 레이아웃 트리를 통해 (예컨대, 렌더링 동작들 (584) 의 일부로서) 보행하고 CSS의 규칙들에 따라 스크린 상에 그러한 보행을 그릴 수도 있다.

[0218] 레이아웃 동작들 (582) 과 렌더링 동작들 (584) 이 밀접하게 관련되고 파이프라인 패션으로 함께 동작하므로, 일 양태에서, 그 동작들은 단일 컴포넌트, 이를테면 레이아웃 및 렌더링 엔진 (516) 에 의해 수행될 수도 있다.

[0219] 다양한 양태들에서, 렌더링 엔진 서브시스템 (560) 은 CSS 레이아웃 알고리즘이 레이아웃 트리를 상의 4 개의 패스들에서 수행되게 레이아웃 동작들 (582) 을 수행하도록 구성될 수도 있다. 각각의 패스에서, 정보는 종래의 접근법들보다 더 많이 제어되는 방식으로 트리를 통해 흘러, 레이아웃 프로세스에서 병렬성의 가능성을 드러낼 수도 있다.

[0220] 일 양태에서, 렌더링 엔진 서브시스템 (560) 은 레이아웃 트리 상에서 4 개의 패스들, 즉, 최소 또는 바람직한 폭 계산 패스, 폭 계산 패스, 블록-포맷팅 콘텍스트 흐름 패스, 및 절대 포지션 계산을 수행할 수도 있다.

[0221] 제 1 패스 (즉, 최소 또는 바람직한 폭 계산 패스) 는 최소 폭 및 바람직한 폭을 각각의 엘리먼트에 할당하기 위해 트리에서 폭들을 위로 전파하는 상향식 패스일 수도 있다. 예로서, 텍스트의 단락을 포함하는 div 엘리먼트의 경우, 최소 폭은 각각의 단어 뒤에 위치된 행 바꿈 (line break) 으로서의 폭일 수도 있고, 바람직한 폭은 어떠한 행 바꿈들도 없는 폭일 수도 있다.

[0222] 제 2 패스 (즉, 폭 계산 패스) 는 각각의 엘리먼트의 최종 폭을 계산하는 하향식 패스일 수도 있다. 엘리먼트의 스타일에 의존하여, 최종 폭은 그 엘리먼트의 부모의 폭, 또는 최소/바람직한 폭으로부터 도출될 수도 있다.

[0223] 제 3 패스 (즉, 블록-포맷팅 콘텍스트 흐름 패스) 동안, 각각의 엘리먼트는 알려진 폭을 가지고, 그것의 내용들은 그것의 높이를 계산하는데 사용될 수도 있다. 예로서, 텍스트의 단락을 포함하는 div 엘리먼트의 경우, 폭이 결정된 후, 텍스트는 그것의 내부에 위치될 수도 있고, 각각의 라인의 높이는 div의 전체 높이를 구하기 위하여 합산될 수도 있다. 전파의 방향은 복잡할 수도 있다. 내용들이 그 높이를 계산하는데 사용되는 엘리먼트들은 블록-포맷팅 콘텍스트 (block-formatting context; BFC) 들이라고 지칭될 수도 있다. 엘리먼트가 블록-포맷팅 콘텍스트인지의 여부는 그 엘리먼트의 CSS 스타일에 의해 결정될 수도 있다.

[0224] DOM 트리에서의 블록-포맷팅 콘텍스트 엘리먼트들은 DOM 상에 중첩될 수도 있는 논리적 트리를 형성할 수도 있다. 블록-포맷팅 콘텍스트 오버레이 트리는 상향식으로 보행될 수도 있고, 브라우저 시스템 (300) 은 DOM 트리의 루트에 도달할 때까지, 전체 웹페이지를 레이아웃할 것이다. 이 페이즈의 말미에, 브라우저 시스템

(500) 에는 모든 엘리먼트들의 높이, 뿐만 아니라 그것들을 포함하는 블록-포맷팅 콘텍스트 내의 그 엘리먼트들의 상대 포지션들이 알려질 것이다.

[0225] 제 4 패스 (즉, 절대 포지션 계산 패스) 는 페이지 상의 각각의 엘리먼트의 절대 포지션을 계산하기 위해 이전의 패스로부터의 각각의 블록-포맷팅 콘텍스트 내의 상대 포지션들을 사용하는 하향식 패스일 수도 있다.

[0226] 일 양태에서, 렌더링은 배경 엘리먼트들이 전경 엘리먼트들 전에 방문되도록 레이아웃 트리를 보행함으로써 달성될 수도 있다. 다양한 양태들은 각각의 엘리먼트를 그것의 스타일에 일치하는 방식으로 그래픽 버퍼 속으로 들여오고, 버퍼의 내용들을 스크린 상에 (예컨대, GUI를 통해) 디스플레이할 수도 있다. 이들 렌더링 동작들은 합성 단계들에 의해 사용된 메모리 대역폭 때문에 계산적으로 값이 비쌀 수도 있다. 다양한 양태들은 각각의 합성 단계에 의해 요구된 메모리 대역폭을 다양한 컴포넌트들/서브시스템들의 병렬 또는 동시 실행을 통해 감소시키도록 구성될 수도 있다.

[0227] 일반적으로, 레이아웃 및 렌더링 동작들의 성능은 페이지 로드 시간들로부터 사용자 인터페이스의 응답성이 이르기까지의 모든 것에 대한 그것들의 영향으로 인해 중요하다. 덧붙여서, 레이아웃 및 렌더링 동작들은 CPU 사이클 동안 JavaScript®의 실행과 같은 다른 중요한 태스크들과 경쟁한다.

[0228] 순차적 최적화들과 함께, 다양한 양태들은 레이아웃 및 렌더링 엔진의 성능을 개선하기 위해 코오스 (coarse) 및 파인 (fine) 입도 병렬성 양쪽 모두를 포함할 수도 있다. 이들 2 가지 접근법들은 상보적일 수도 있다. 코오스 레벨에서, 일 양태의 브라우저가 가능한 한 많은 작업으로 중요 경로 밖으로 및 작업자 (worker) 스레드들 속으로 이동할 수도 있다. 파인 레벨에서, 양태의 브라우저는 레이아웃 및 렌더링 알고리즘들/방법들을 병렬화할 수도 있다.

[0229] 종래의 웹 브라우저에서, DOM를 조작하는 태스크들 (예컨대 파싱 또는 JavaScript®) 은 동시에 레이아웃 및 렌더링 태스크들로서 결코 실행하지 않는데, 이는 그 두 개가 서로 간섭하지 않는 것을 보장한다. 그 반면, 다양한 양태들은 이들 2 가지 유형들의 태스크들을 중첩시킨다. 이처럼, 다양한 양태들에서, 레이아웃 트리는 DOM이 변경될 때마다 업데이트되지 않을 수도 있다.

[0230] 다양한 양태들은 레이아웃 트리 및 DOM을 분리 (또는 별개로 유지) 할 수도 있다. 레이아웃 트리에 대한 업데이트들은 레이아웃 및 렌더링 동작들이 일반적으로 발생할 시간들에서 배치 (batch) 동작으로서 수행될 수도 있으며; 종종 이는 파싱 또는 JavaScript® 실행 태스크가 완료된 후이다. 이 방식으로 업데이트들을 그룹화하는 것은, 레이아웃 트리가 DOM의 각각의 중간 상태에 대해 업데이트되지 않으므로, 브라우저 시스템 (500) 이, 변경되었지만 불필요한 작업의 수행을 피할 DOM의 부분들을 식별하기 위해 부가적인 상태 정보를 유지할 것을 필요로 할 수도 있다는 것을 의미할 수도 있다.

[0231] 다양한 양태들은 결과적으로 유용한 작업을 할 준비가 되어 있는 경우 레이아웃 트리를 업데이트할 수도 있다. 레이아웃 트리는 DOM으로부터의 별개의 엔티티일 수도 있다. 모든 DOM 변경들이 레이아웃 트리에 영향을 미치는 일 없이 수행될 수도 있다. 반대로 렌더링 엔진 서브시스템 (560) 은 일단 레이아웃 트리가 업데이트되면 어떤 식으로든 DOM에 액세스할 필요가 없다. 이는 병렬성을 가능하게 하고, 레이아웃 트리가 관례적으로 DOM에만 저장될 특정 정보를 복제해야만 한다는 것을 또한 의미한다. 특히, 레이아웃 트리는 텍스트, 이미지들, CSS 스타일들, 및 HTML 캔버스 엘리먼트들에 대한 직접 참조들을 포함할 수도 있다.

[0232] 텍스트와 이미지들은 불변이고 DOM와 안전하게 공유될 수도 있다. CSS 스타일들은 논리적으로 불변일 수도 있지만, CSS 스타일 오브젝트에서의 데이터의 양은 전체 오브젝트를 매번 복사하기에는 너무 클 수도 있다 (그리고/또는 그것들은 너무 빈번하게 업데이트될 수도 있다). 따라서, 일 양태에서, 각각의 스타일 오브젝트는 많은 더 작은 서브-스타일 오브젝트들로 내부적으로 나누어질 수도 있다. 공유된 서브-스타일들이 카피-온-라이트 (copy-on-write) 접근법을 사용하여 업데이트될 수도 있다. 비공유 서브-스타일들이 제자리에서 업데이트될 수도 있다. 따라서, 스타일 오브젝트를 복사하는 것은 동일한 서브-스타일들을 공유하는 새로운 스타일 오브젝트를 생성하는 것만을 필요로 할 수도 있는데, 이는 훨씬 더 저렴할 수도 있다. 덧붙여서, 서브-스타일들은 함께 업데이트되는 CSS 속성들이 동일한 서브-스타일에 있도록 그룹화될 수도 있는데, 이는 업데이트 발생할 경우의 서브-스타일 복사들을 최소화할 수도 있다. 이 배열은 DOM, 레이아웃, 및 렌더링 컴포넌트들이 하나의 장소에서 이루어진 변경들/다른 것들에게 가시적인 컴포넌트 없이 동일한 CSS 스타일들을 참조하는 것을 허용한다. 유사한 카피-온-라이트 접근법이 HTML 캔버스 엘리먼트들을 위해 사용될 수도 있다.

[0233] DOM 트리로부터의 레이아웃 트리의 분리는 렌더링 엔진 서브시스템 (560) 에서 코오스-입도 병렬성을 가능하게 한다. 웹 페이지가 사용자에게 처음 디스플레이될 준비가 되는 경우, 브라우저 시스템 (500) 은 레이아웃

트리를 초기화하고 그것을 렌더링 엔진 서브시스템 (560) 에 프로세싱을 위해 핸드 오프하는 작업 아이템을 생성할 수도 있다. 레이아웃 및 렌더링 동작들의 상이한 스레드들로의 분리는 브라우저 시스템 (500) 의 나머지가 앞으로 이동하는 것을 허용하여, 이를테면 JavaScript®가 실행될 수 있으며, 사용자 인터페이스 (UI) 이벤트들은 프로세싱될 수 있고, CSS 스타일링이 컴퓨팅될 수도 있다는 등등이 허용된다.

[0234] 렌더링 엔진 서브시스템 (560) 이 자신의 태스크들을 완료하고 스크린 상에 페이지를 디스플레이하는 경우, 그 서브시스템은 레이아웃 트리를 업데이트하기 위해 "LR 작업 아이템"을 제출하고, 프로세스를 완전히 다시 시작할 수도 있다. "LR 작업 아이템"만이 DOM에 대한 배타적인 액세스를 필요로 하고, 일단 트리가 업데이트되면, 다른 동작들은 병렬로 및/또는 비동기적으로 수행될 수도 있다.

[0235] 특정 JavaScript® DOM API들 (예컨대, getComputedStyle 및 offsetTop) 이 레이아웃 알고리즘이 컴퓨팅하는 결과들에 관한 정보를 필요로 할 수도 있다. 렌더링 엔진 서브시스템 (560) 은 결과들이 이용가능하기까지 일시정지 (pause) 를 필요로 할 수도 있다. 렌더링 엔진 서브시스템 (560) 이 메인 스레드에서 레이아웃을 수행하면, 그 시스템은 LR 작업 아이템 (또는 LR 스레드) 에서 수행될 계산들을 중복할 수도 있으며, 이는 시간 및 에너지를 낭비할 수도 있다.

[0236] 일 양태에서, 렌더링 엔진 서브시스템 (560) 은 레이아웃 트리가 최신 레이아웃 정보를 갖는지의 여부를 기억하도록 구성될 수도 있다. 만약 그렇다면, 동기적 레이아웃 요청이 즉시 반환될 수도 있다. 아니라면, 레이아웃 동작들은 LR 스레드에서 평상시처럼 수행될 수도 있고, 렌더링 엔진 서브시스템 (560) 은 레이아웃 프로세스가 완료된 때를 메인 스레드에 통지할 것이 요청될 수도 있다. 이는 중복 작업을 방지하면서도 가능한 한 빠르게 필요한 결과들을 전달한다.

[0237] 병렬성에 더하여, 레이아웃 트리 및 DOM을 분리하는 다른 장점은 렌더링 엔진 서브시스템 (560) 이 웹 페이지들 간에 공유된 서비스로서 처리될 수도 있다는 것이다. 레이아웃 트리들은 그것들이 구축되었던 DOM을 다시 참조하지 않으므로, 동일한 렌더링 엔진 서브시스템 (560) 은 모든 레이아웃 트리들을 그것들의 소스에 상관없이 관리할 수도 있다. 이는 그래픽 버퍼들과 같은 값 비싼, 유한 렌더링 관련 자원들이 전체 브라우저 시스템 (500) 에서 하나의 인스턴스만을 필요로 한다는 것을 의미한다.

[0238] 레이아웃 트리에 의해 제공된 또 다른 장점은 사용자가 신속히 변경하는 페이지와 상호작용하는 경우에 사용자의 의도를 결정함에 있어서 추가된 유연성이다. 예를 들어, 사용자가 JavaScript®에 의해 스크린에서 이동하고 있는 버튼을 클릭하면, 레이아웃 및 렌더링 동작들에 시간이 걸리기 때문에 DOM을 변경하는 JavaScript®와 스크린 상에 나타나는 결과들 사이에는 지연이 있다. 사용자의 클릭이 등록될 때까지, DOM은 업데이트되었고 브라우저의 관점에서 박스의 로케이션은 변경될 수도 있다. 심지어 사용자의 마우스 포인터가 박스 바로 위에 있더라도, 클릭의 시도는 성공적이지 않을 수도 있다. 그러나, 레이아웃 트리가 DOM으로부터 분리되기 때문에, 브라우저 시스템 (500) 은 현재 작업 트리와 스크린 상에 디스플레이되었던 마지막 트리에 액세스할 수 있다. 이는, DOM의 현재 상태가 아니라, 사용자가 클릭한 경우에 사용자가 보았던 것에 기초하여 사용자가 클릭할 것을 의도했던 오브젝트를 브라우저 시스템 (500) 이 결정하는 것을 가능하게 하여, 개선된 인지 응답성 및 양호한 사용자 경험이 이루어지게 한다.

[0239] 다양한 양태들은 다양한 모바일 컴퓨팅 디바이스들에 구현될 수도 있으며, 그런 디바이스들의 일 예가 도 12에 도시된다. 구체적으로는, 도 12는 양태들 중 임의의 것과 함께 사용하기에 적합한 스마트폰/셀 전화기 (1200) 형태의 모바일 트랜시버 디바이스의 시스템 블록도이다. 셀 전화기 (1200) 는, 내부 메모리 (1202), 디스플레이 (1203), 및 스피커 (1208) 에 커플링된 프로세서 (1201) 를 포함할 수도 있다. 덧붙여, 셀 전화기 (1200) 는 프로세서 (1201) 에 커플링된 무선 데이터 링크, 및/또는 셀룰러 전화기 트랜시버 (1205) 에 접속될 수도 있는, 전자기 복사를 전송하고 수신하기 위한 안테나 (1204) 를 포함할 수도 있다. 셀 전화기들 (1200) 은 통상 사용자 입력들을 수신하기 위한 메뉴 선택 버튼들 또는 로커 스위치들 (1206) 을 또한 포함할 수도 있다.

[0240] 전형적인 셀 전화기 (1200) 는 마이크로폰으로부터 수신된 사운드를 무선 송신에 적합한 데이터 패킷들로 디지털화하고 수신된 사운드 데이터 패킷들을 디코딩하여 사운드를 생성하는 스피커 (1208) 에 제공되는 아날로그 신호들을 생성하는 사운드 인코딩/디코딩 (CODEC) 회로 (1213) 를 또한 포함한다. 또한, 프로세서 (1201), 무선 트랜시버 (1205) 및 코덱 (1213) 중 하나 이상은 디지털 신호 프로세서 (DSP) 회로 (별도로 도시되지 않음) 를 포함할 수도 있다. 셀 전화기 (1200) 는 무선 디바이스들 사이의 저전력 단거리 통신들을 위한 ZigBee 트랜시버 (즉, IEEE 802.15.4 트랜시버) (1213), 또는 다른 유사한 통신 회로 (예컨대, Bluetooth® 또는 WiFi 프로토콜들 등을 구현하는 회로) 를 더 포함할 수도 있다.

[0241] 다양한 양태들은 다양한 상업적으로 입수가능한 서버 디바이스들 중 임의의 것, 이를테면 도 13에 예시된 서버 (1300) 상에 구현될 수도 있다. 이러한 서버 (1300) 는 보통 휘발성 메모리 (1302) 및 대용량 비휘발성 메모리, 이를테면 디스크 드라이브 (1303) 에 커플링된 프로세서 (1301) 를 포함한다. 서버 (1300) 는 프로세서 (1301) 에 커플링된 플로피 디스크 드라이브, 콤팩트 디스크 (CD) 또는 DVD 디스크 드라이브 (1311) 를 또한 포함할 수도 있다. 서버 (1300) 는 다른 통신 시스템 컴퓨터들 및 서버들에 커플링된 로컬 영역 네트워크와 같은 네트워크 (1305) 와의 데이터 접속들을 확립하기 위해 프로세서 (1301) 에 커플링된 네트워크 액세스 포트들 (1306) 을 또한 포함할 수도 있다.

[0242] 다른 형태들의 컴퓨팅 디바이스들이 다양한 양태들에서 또한 유익할 수도 있다. 이러한 컴퓨팅 디바이스들은 보통, 일 예의 개인용 랩톱 컴퓨터 (1400) 를 도시하는 도 14에 예시된 컴포넌트들을 포함한다. 이러한 개인용 컴퓨터 (1400) 는 일반적으로, 휘발성 메모리 (1402) 와 대용량 비휘발성 메모리, 이를테면 디스크 드라이브 (1403) 에 커플링된 프로세서 (1401) 를 포함한다. 컴퓨터 (1400) 는 프로세서 (1401) 에 커플링된 콤팩트 디스크 (CD) 및/또는 DVD 드라이브 (1404) 를 또한 포함할 수도 있다. 컴퓨터 디바이스 (1400) 는 프로세서 (1401) 에 커플링되어 데이터 접속들을 확립하거나 또는 외부 메모리 디바이스들을 수용하는 다수의 커넥터 포트들, 이를테면 프로세서 (1401) 를 네트워크에 커플링하기 위한 네트워크 접속 회로 (1405) 를 또한 포함할 수도 있다. 컴퓨터 (1400) 는 컴퓨터 분야에서 잘 알려진 바와 같은 키보드 (1408), 마우스 (1410) 와 같은 포인팅 디바이스, 및 디스플레이 (1409) 에 더 커플링될 수도 있다.

[0243] 프로세서들 (1201, 1301, 1401) 은 아래에서 설명되는 다양한 양태들의 기능들을 포함한 다양한 기능들을 수행하는 소프트웨어 명령들 (애플리케이션들) 에 의해 구성될 수 있는 임의의 프로그램가능 마이크로프로세서, 마이크로컴퓨터 또는 다중 프로세서 칩 또는 칩들일 수도 있다. 일부 모바일 디바이스들에서, 이를테면 하나의 프로세서가 무선 통신 기능들에 전용이고 하나의 프로세서가 다른 애플리케이션들에 전용인 다수의 프로세서들 (1301) 이 제공될 수도 있다. 통상, 소프트웨어 애플리케이션들은 그것들이 액세스되어 프로세서 (1201, 1301, 1401) 속으로 로딩되기 전에 내부 메모리 (1202, 1302, 1303, 1402) 에 저장될 수도 있다. 프로세서 (1201, 1301, 1401) 는 애플리케이션 소프트웨어 명령들을 저장하기에 충분한 내부 메모리를 포함할 수도 있다.

[0244] 다양한 양태들은 임의의 수의 단일 또는 멀티 프로세서 시스템들에서 구현될 수도 있다. 일반적으로, 프로세스들은 다수의 프로세스들이 단일 프로세서에서 동시에 구동중인 것처럼 보이도록, 짧은 시간 슬라이스들에서 프로세서에서 실행된다. 프로세스가 시간 슬라이스의 단부에서 프로세서로부터 제거될 경우, 프로세스의 현재 동작 상태에 관한 정보가 메모리에 저장되어 그 프로세서는 프로세서에서의 실행에 복귀할 경우 그 동작들을 매끄럽게 재시작할 수도 있다. 이러한 동작 상태 데이터는 프로세스의 어드레스 공간, 스택 공간, 가상 어드레스 공간, 레지스터 세트 이미지 (예컨대, 프로그램 카운터, 스택 포인터, 명령 레지스터, 프로그램 상태 워드, 등), 어카운팅 정보, 허가들, 액세스 제한들, 및 상태 정보를 포함할 수도 있다.

[0245] 프로세스는 다른 프로세스들을 생겨나게 할 수도 있고, 생겨난 프로세스 (즉, 자식 프로세스) 는 생성 프로세스 (즉, 부모 프로세스) 의 허가들 및 액세스 제한들 (즉, 콘텍스트) 의 일부를 상속받을 수도 있다. 프로세스는 다른 프로세스들/스레드들과 그들의 콘텍스트 (예컨대, 어드레스 공간, 스택, 허가들 및/또는 액세스 제한들, 등) 의 전부 또는 부분들을 공유하는 프로세스들인, 다수의 경량의 (lightweight) 프로세스들 또는 스레드들을 포함하는 중량의 (heavy-weight) 프로세스들일 수도 있다. 따라서, 단일 프로세스는 단일 콘텍스트 (즉, 프로세서의 콘텍스트) 를 공유하고, 단일 콘텍스트로 액세스하고, 및/또는 단일 콘텍스트 내에서 동작하는 다수의 경량의 프로세스들 또는 스레드들을 포함할 수도 있다.

[0246] 앞서의 방법 설명들 및 프로세스 흐름도들은 구체적인 예들로서만 제공되고 다양한 양태들의 블록들이 제시된 순서로 반드시 수행되어야만 함을 요구하거나 의미하도록 의도된 것은 아니다. 당업자에 의해 이해될 바와 같이 앞서의 양태들에서의 순서의 블록들은 어떤 순서로도 수행될 수도 있다. 단어들 이를테면 "그 후", "그 다음", "다음에" 등은 블록들의 순서를 한정하려는 의도는 아니고; 이들 단어들은 단지 독자에게 방법들의 기재를 통하여 설명하는데 사용된다. 게다가, 단수형으로, 예를 들어, 관사 "a", "an" 또는 "the"의 사용에 해당하는 것으로 여겨질 수 있는 청구항 엘리먼트들에 대한 어떠한 언급이라도, 그 엘리먼트를 단수형으로 제한하는 것으로 해석되지는 않는다.

[0247] 본원에서 개시된 양태들에 관련하여 설명되는 각종 구체적인 논리 블록들, 모듈들, 회로들, 및 알고리즘 블록들은 전자적 하드웨어, 컴퓨터 소프트웨어, 또는 이것 둘의 조합들로 구현될 수도 있다. 하드웨어 및 소프트웨어의 이러한 상호교환가능성을 명백하게 예증하기 위하여, 다양한 구체적인 컴포넌트들, 블록들, 모듈들, 회로들 및 블록들은 일반적으로 그것들의 기능성의 측면에서 설명되어있다. 이러한 기능성이 하드웨어 또는

소프트웨어 중 어느 것으로 구현되는지는 전체 시스템에 부과되는 특정 애플리케이션 및 설계 제약들에 달려있다. 당업자들은 설명된 기능을 각각의 특정한 애플리케이션에 대하여 다양한 방식으로 구현할 수도 있지만, 이러한 구현 결정은 본 발명의 범위를 벗어나도록 하는 것으로 해석되지 않아야 한다.

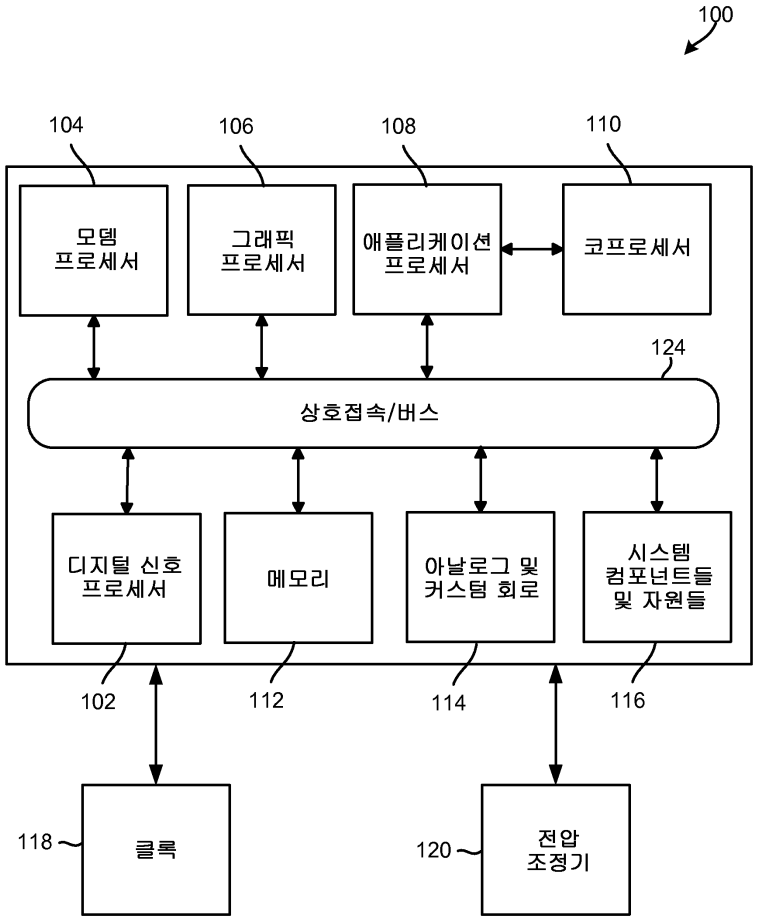
[0248] 본원에 개시된 양태들에 관련하여 설명된 다양한 예시적인 로직들, 논리 블록들, 모듈들, 및 회로들을 구현하는데 사용되는 하드웨어는 본원에서 설명된 기능들을 수행하도록 설계된 범용 프로세서, 디지털 신호 프로세서(DSP), 주문형 집적회로 (ASIC), 필드 프로그램가능 게이트 어레이 (FPGA) 또는 다른 프로그램가능 로직 디바이스, 개별 게이트 또는 트랜지스터 로직, 개별 하드웨어 컴포넌트들, 또는 그것들의 임의의 조합으로 구현되거나 또는 수행될 수도 있다. 범용 프로세서는 마이크로프로세서일 수도 있지만, 대체예에서, 이 프로세서는 임의의 종래의 프로세서, 제어기, 마이크로제어기, 또는 상태 머신 (state machine) 일 수도 있다. 프로세서는 또한, 컴퓨팅 디바이스들의 조합, 예컨대, DSP 및 마이크로프로세서의 조합, 복수의 마이크로프로세서들의 조합, DSP 코어와 협력하는 하나 이상의 마이크로프로세서들의 조합, 또는 임의의 다른 이러한 구성으로 구현될 수도 있다. 다르게는, 일부 블록들 또는 방법들이 주어진 기능에 특화된 회로에 의해 수행될 수도 있다.

[0249] 하나 이상의 예시적인 양태들에서, 설명된 기능들은 하드웨어, 소프트웨어, 펌웨어, 또는 그것들의 임의의 조합으로 구현될 수도 있다. 소프트웨어로 구현된다면, 기능들은 하나 이상의 명령들 또는 코드로서 비일시적 컴퓨터 판독가능 매체 또는 비일시적 프로세서 판독가능 매체 상에 저장될 수도 있다. 본원에서 개시된 방법 또는 알고리즘의 단계들은, 비일시적 컴퓨터 판독가능 저장 매체 또는 프로세서 판독가능 저장 매체 상에 존재할 수도 있는 프로세서 실행가능 소프트웨어 모듈로 실시될 수도 있다. 비일시적 컴퓨터 판독가능 또는 프로세서 판독가능 저장 매체들은 컴퓨터 또는 프로세서에 의해 액세스될 수도 있는 임의의 저장 매체들일 수도 있다. 비제한적인 예로, 이러한 비일시적 컴퓨터 판독가능 또는 프로세서 판독가능 매체들은 RAM, ROM, EEPROM, FLASH 메모리, CD-ROM 또는 다른 광 디스크 스토리지, 자기 디스크 스토리지, 또는 다른 자기적 저장 디바이스들, 또는 소망의 프로그램 코드를 명령들 또는 데이터 구조들의 형태로 저장하는데 사용될 수도 있는 컴퓨터에 의해 액세스될 수도 있는 임의의 다른 매체를 포함할 수도 있다. 디스크 (Disk 및 disc) 는 본원에서 사용되는 바와 같이, 콤팩트 디스크 (compact disc, CD), 레이저 디스크, 광 디스크, 디지털 다용도 디스크 (DVD), 플로피 디스크 (floppy disk) 및 블루레이 디스크를 포함하는데, disk들은 보통 데이터를 자기적으로 재생하지만, disc들은 레이저들로써 광적으로 데이터를 재생한다. 상기한 것들의 조합들은 또한 비일시적 컴퓨터 판독가능 및 프로세서 판독가능 매체들의 범위 내에 포함된다. 덧붙여, 방법 또는 알고리즘의 동작들은 코드들 및/또는 명령들의 하나 또는 임의의 조합 또는 세트로서 컴퓨터 프로그램 제품에 통합될 수도 있는 비일시적 프로세서 판독가능 매체 및/또는 컴퓨터 판독가능 매체 상에 존재할 수 있다.

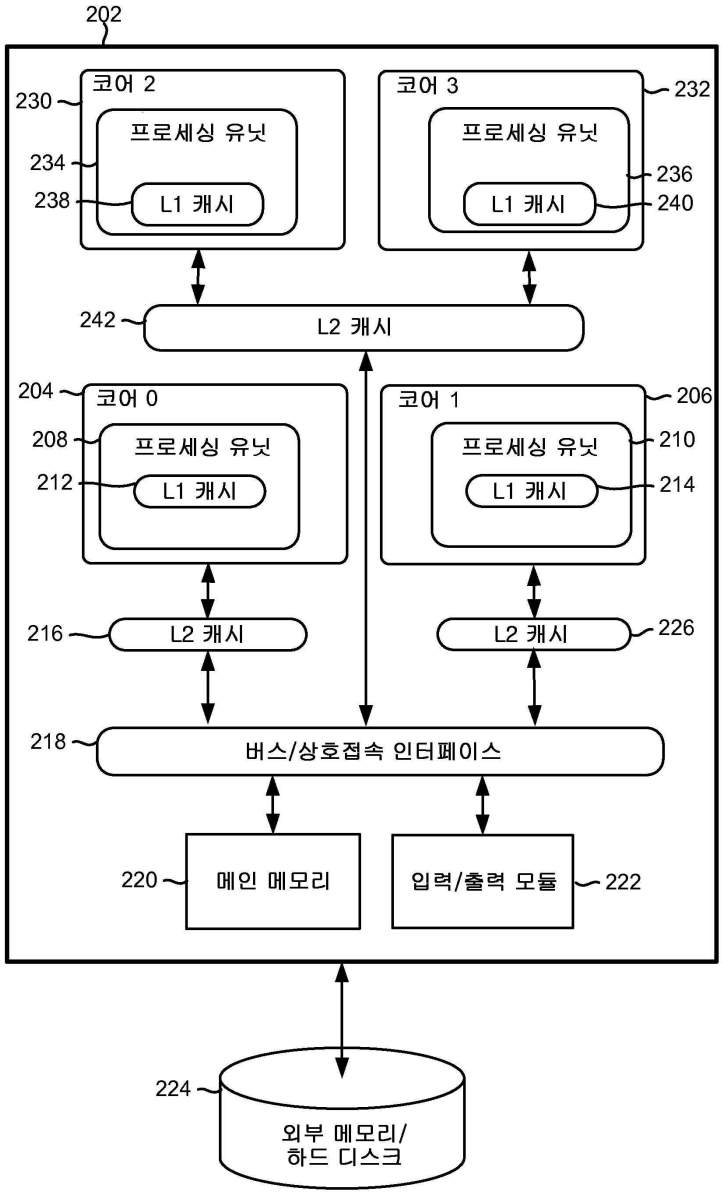
[0250] 개시된 양태들의 전술한 설명은 어떤 당업자도 본 발명을 제작하고 사용할 수 있게끔 제공된다. 이들 양태들에 대한 다양한 변형예들은 당업자들에게 쉽사리 명확하게 될 것이고, 여기에 정의된 일반 원리들은 다른 양태들에 적용될 수도 있다. 따라서, 본 발명은 본 명세서에서 보인 양태들로 한정할 의도는 아니며 다음의 청구항들 및 여기에 개시된 원리들 및 신규한 특징들과 일치하는 가장 넓은 범위를 부여하는 것을 의도한다.

도면

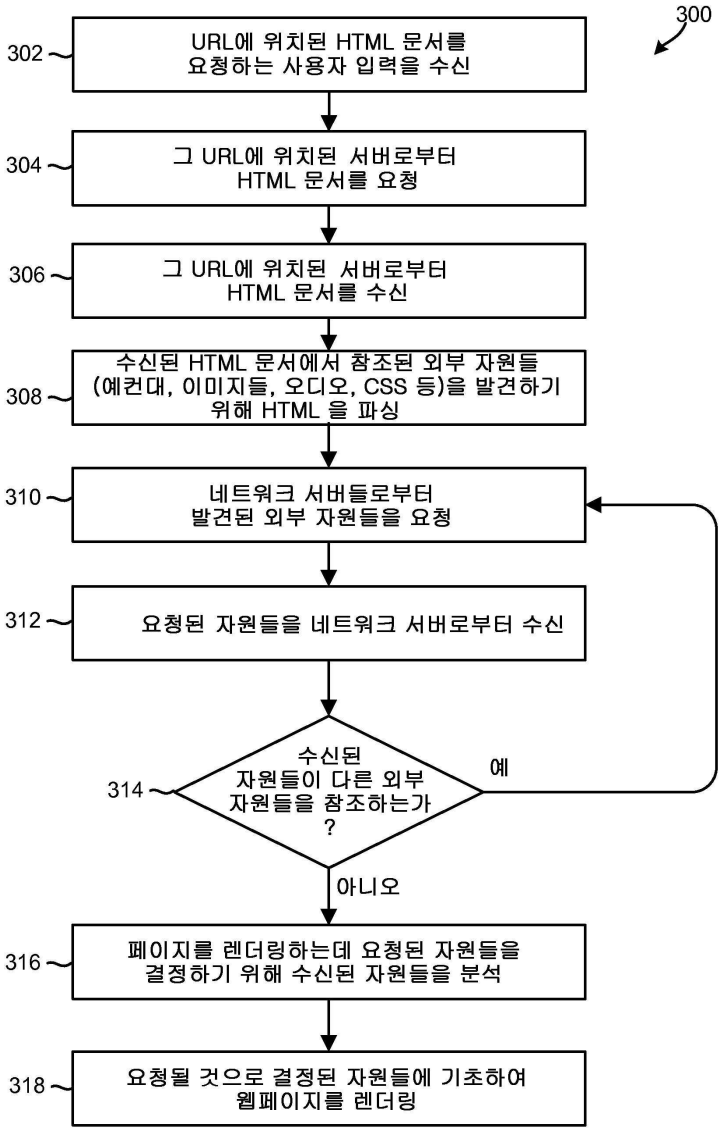
도면1


100
104
106
108
110
모뎀 프로세서
그래픽 프로세서
애플리케이션 프로세서
코프로세서
124
상호접속/버스
디지털 신호 프로세서
메모리
아날로그 및 커스텀 회로
시스템 컴포넌트들 및 자원들
102
112
114
116
118
클록
120
전압 조정기

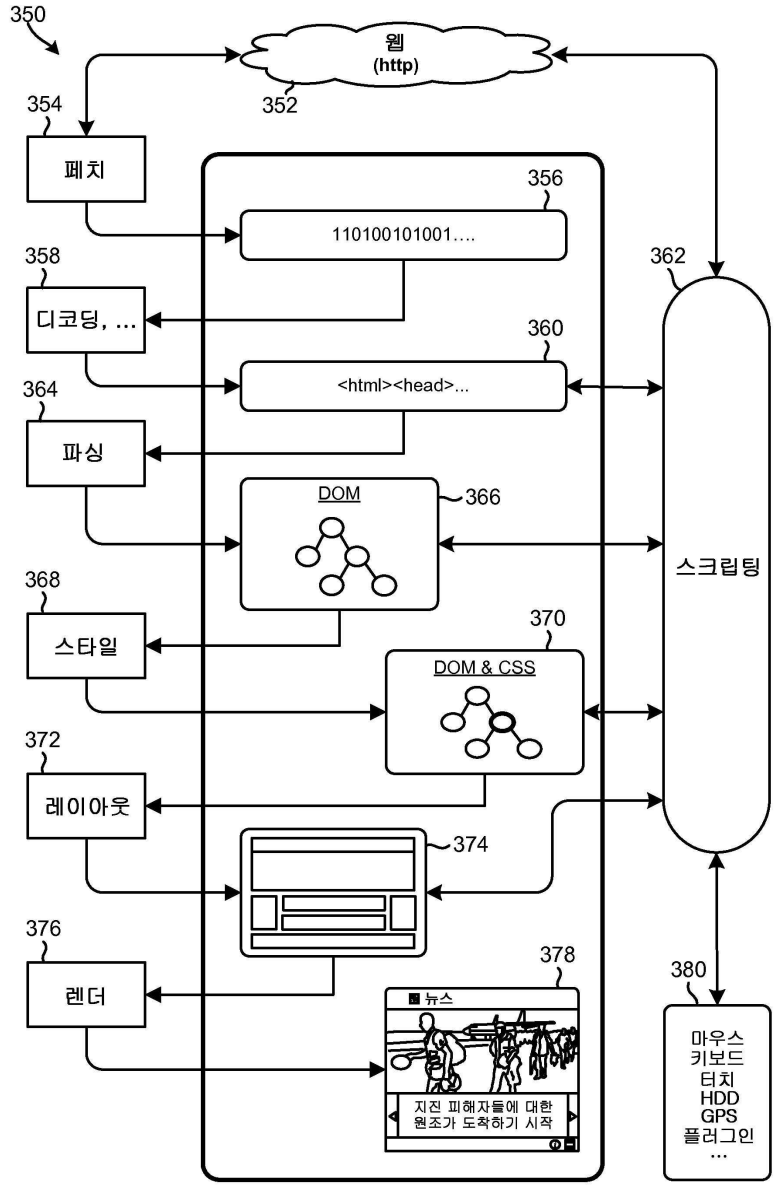
도면2


202
코어 2
230
프로세싱 유닛
234
L1 캐시
238
코어 3
232
프로세싱 유닛
236
L1 캐시
240
242
L2 캐시
204
코어 0
208
프로세싱 유닛
212
L1 캐시
코어 1
206
프로세싱 유닛
210
L1 캐시
214
216
L2 캐시
L2 캐시
226
218
버스/상호접속 인터페이스
220
메인 메모리
입력/출력 모듈
222
224
외부 메모리/
하드 디스크

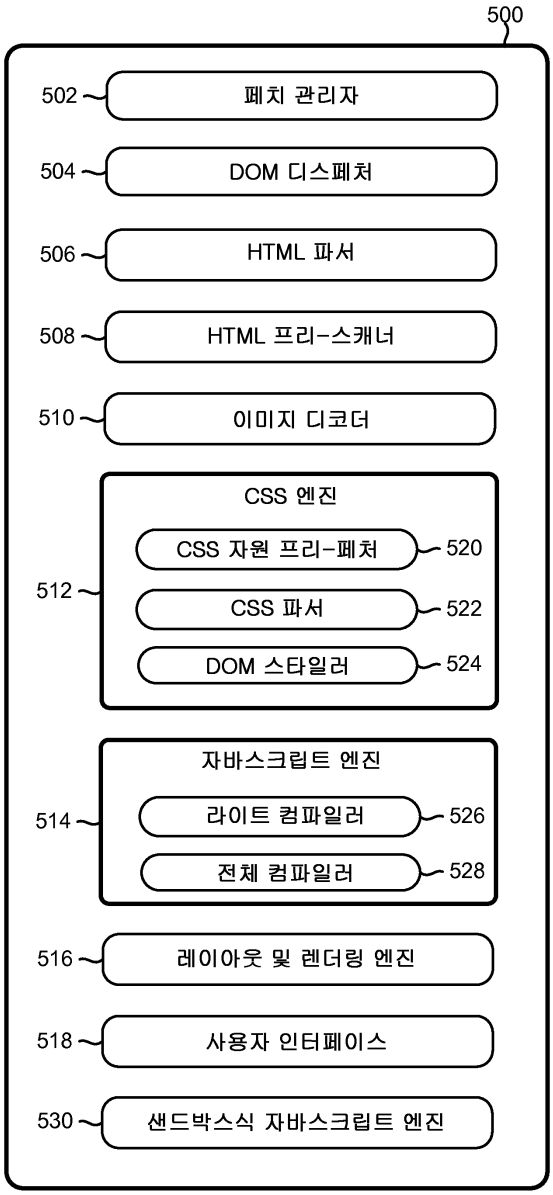
도면3a


300
302
URL에 위치된 HTML 문서를 요청하는 사용자 입력을 수신
304
그 URL에 위치된 서버로부터 HTML 문서를 요청
306
그 URL에 위치된 서버로부터 HTML 문서를 수신
308
수신된 HTML 문서에서 참조된 외부 자원들 (예컨대, 이미지들, 오디오, CSS 등)을 발견하기 위해 HTML 을 파싱
310
네트워크 서버들로부터 발견된 외부 자원들을 요청
312
요청된 자원들을 네트워크 서버로부터 수신
314
수신된 자원들이 다른 외부 자원들을 참조하는가 ?
예
아니오
316
페이지를 렌더링하는데 요청된 자원들을 결정하기 위해 수신된 자원들을 분석
318
요청될 것으로 결정된 자원들에 기초하여 웹페이지를 렌더링

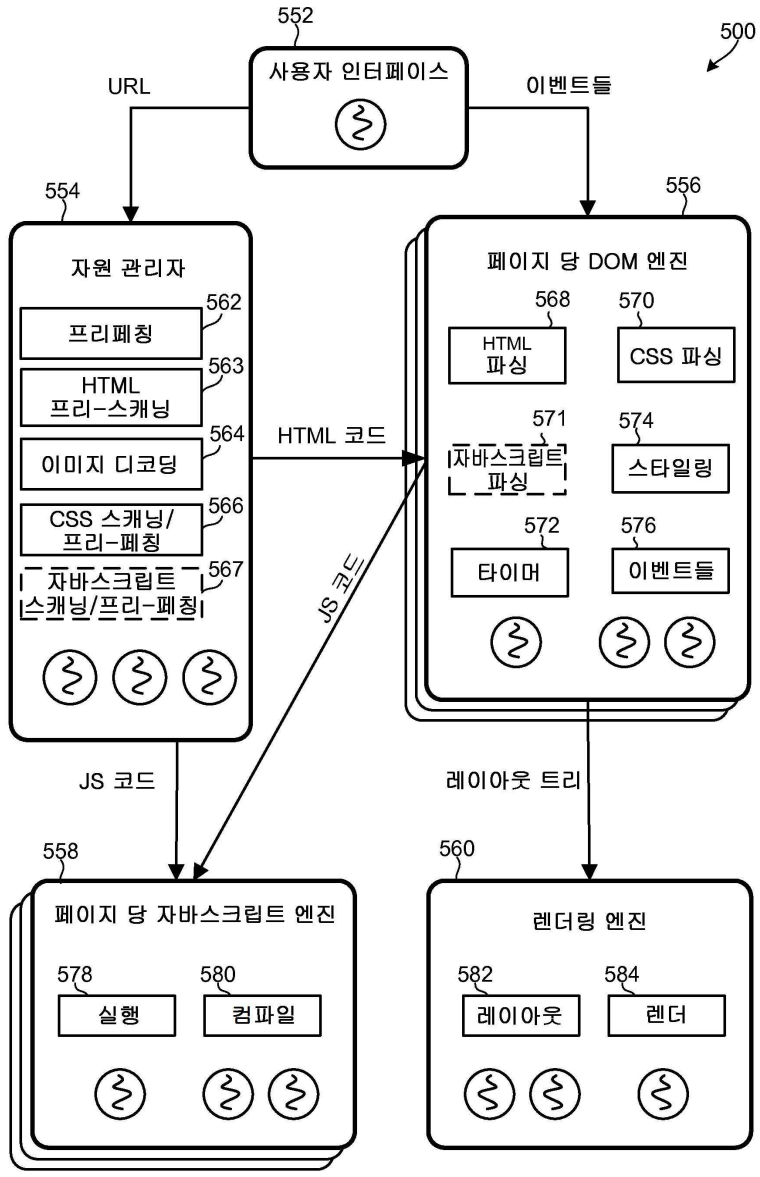
도면3b


350
웹
(http)
352
354
페치
356
110100101001....
358
디코딩, ...
360
<html><head>...
362
364
파싱
DOM
366
스크립팅
368
스타일
370
DOM & CSS
372
레이아웃
374
376
렌더
378
뉴스
지진 피해자들에 대한
원조가 도착하기 시작
380
마우스
키보드
터치
HDD
GPS
플러그인
...

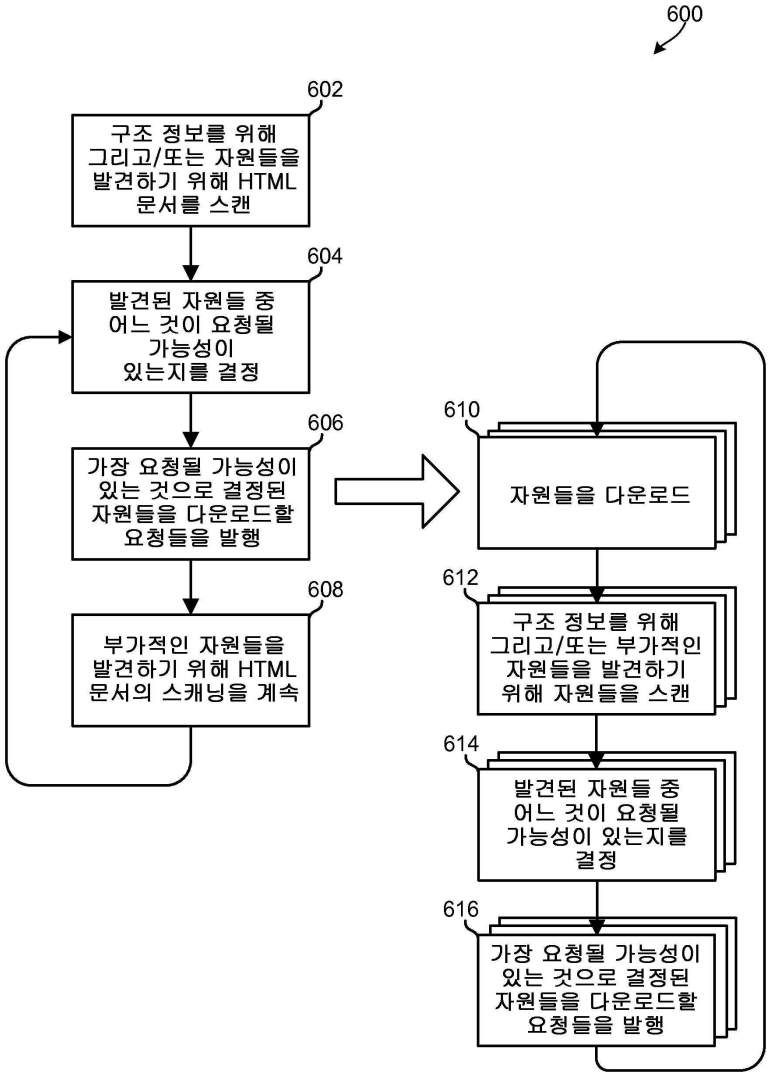
도면4


500
502
페치 관리자
504
DOM 디스페처
506
HTML 파서
508
HTML 프리-스캐너
510
이미지 디코더
512
CSS 엔진
CSS 자원 프리-페처
520
CSS 파서
522
DOM 스타일러
524
514
자바스크립트 엔진
라이트 컴파일러
526
전체 컴파일러
528
516
레이아웃 및 렌더링 엔진
518
사용자 인터페이스
530
샌드박스식 자바스크립트 엔진

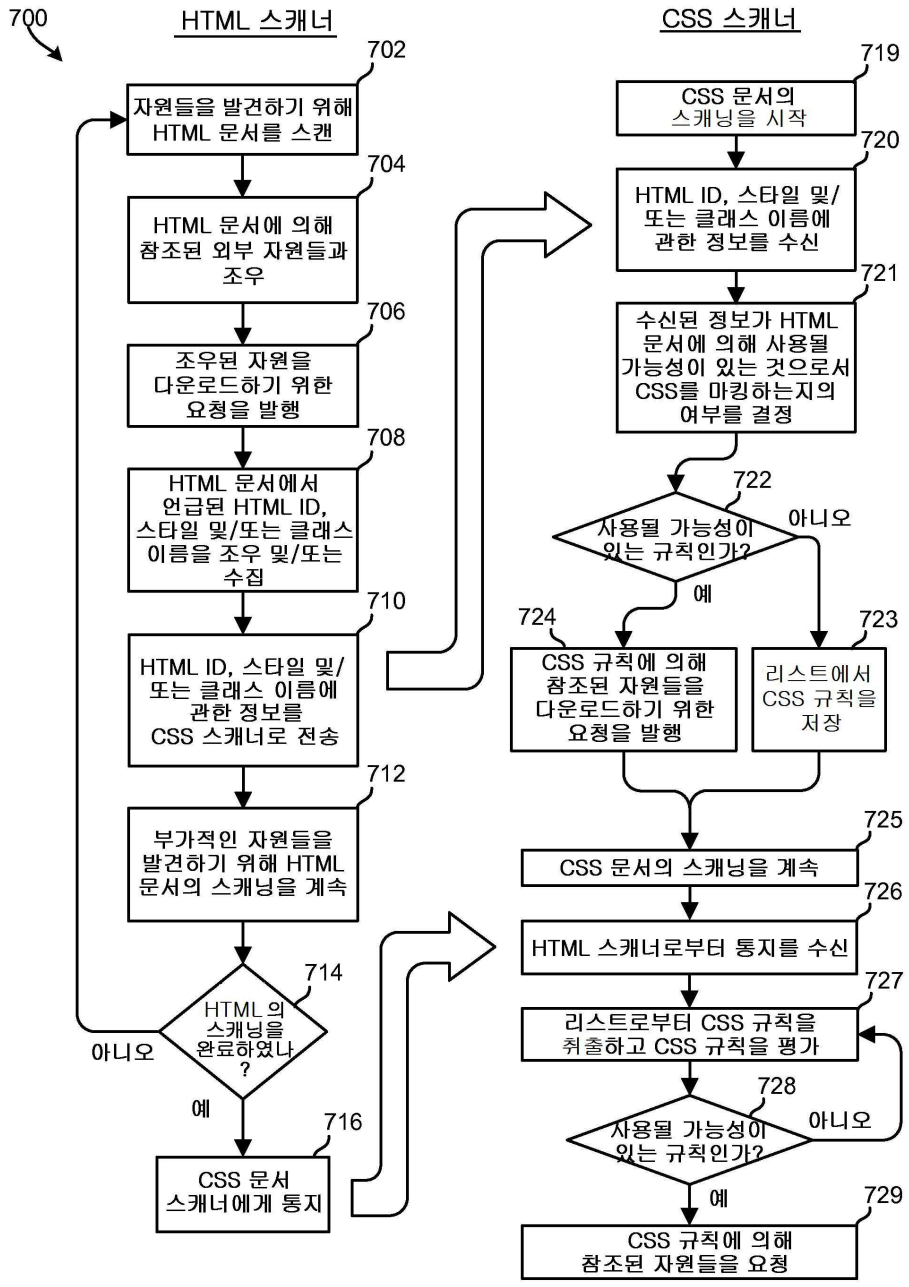
도면5


552
사용자 인터페이스
500
URL
이벤트들
554
556
자원 관리자
562
프리페칭
563
HTML
프리-스캐닝
564
이미지 디코딩
566
CSS 스캐닝/
프리-페칭
567
자바스크립트
스캐닝/프린-페칭
HTML 코드
JS 코드
JS 코드
페이지 당 DOM 엔진
568
570
HTML
파싱
CSS 파싱
571
574
자바스크립트
파싱
스타일링
572
576
타이머
이벤트들
레이아웃 트리
558
560
페이지 당 자바스크립트 엔진
578
580
실행
컴파일
렌더링 엔진
582
584
레이아웃
렌더

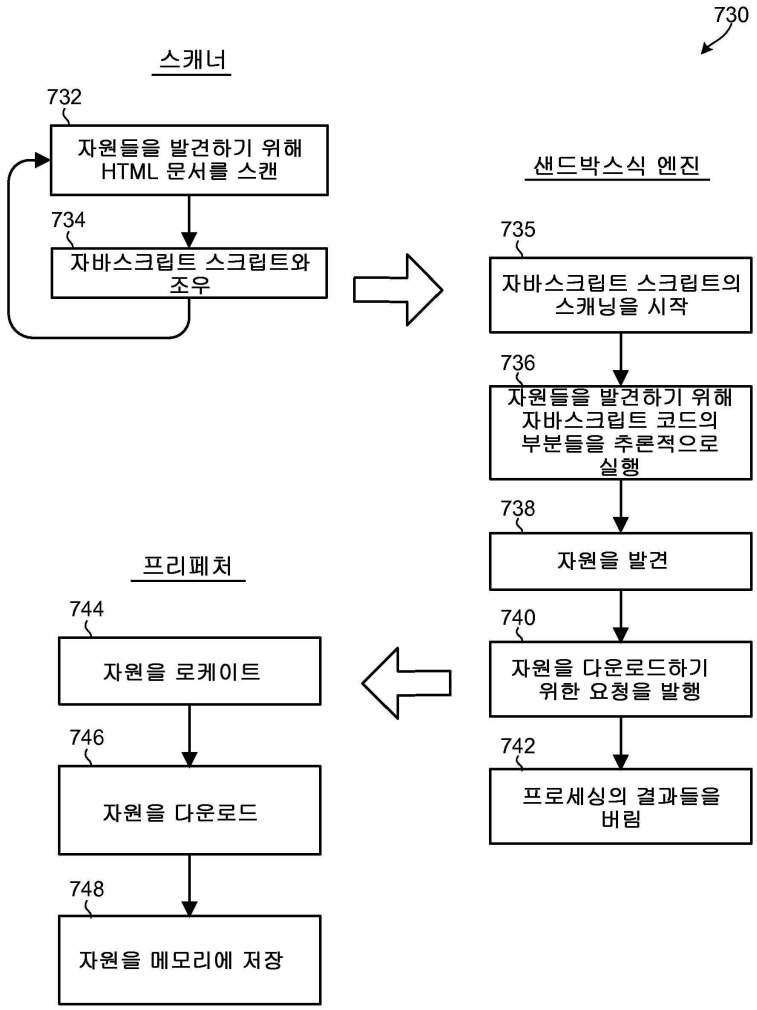
도면6


600
602
구조 정보를 위해 그리고/또는 자원들을 발견하기 위해 HTML 문서를 스캔
604
발견된 자원들 중 어느 것이 요청될 가능성이 있는지를 결정
606
가장 요청될 가능성이 있는 것으로 결정된 자원들을 다운로드할 요청들을 발행
608
부가적인 자원들을 발견하기 위해 HTML 문서의 스캐닝을 계속
610
자원들을 다운로드
612
구조 정보를 위해 그리고/또는 부가적인 자원들을 발견하기 위해 자원들을 스캔
614
발견된 자원들 중 어느 것이 요청될 가능성이 있는지를 결정
616
가장 요청될 가능성이 있는 것으로 결정된 자원들을 다운로드할 요청들을 발행

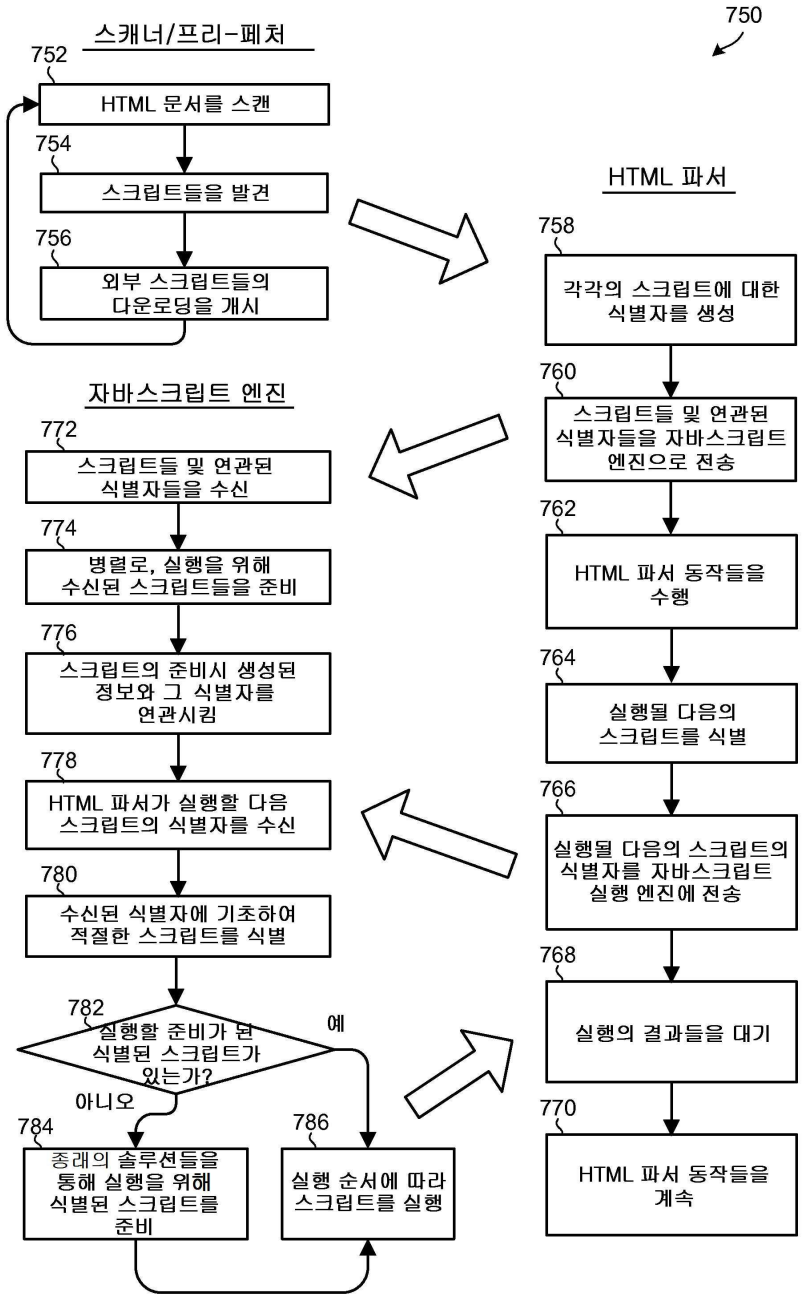
도면7a


700
HTML 스캐너
CSS 스캐너
702
자원들을 발견하기 위해 HTML 문서를 스캔
704
HTML 문서에 의해 참조된 외부 자원들과 조우
706
조우된 자원을 다운로드하기 위한 요청을 발행
708
HTML 문서에서 언급된 HTML ID, 스타일 및/또는 클래스 이름을 조우 및/또는 수집
710
HTML ID, 스타일 및/또는 클래스 이름에 관한 정보를 CSS 스캐너로 전송
712
부가적인 자원들을 발견하기 위해 HTML 문서의 스캐닝을 계속
714
HTML의 스캐닝을 완료하였나?
아니오
예
716
CSS 문서 스캐너에게 통지
719
CSS 문서의 스캐닝을 시작
720
HTML ID, 스타일 및/또는 클래스 이름에 관한 정보를 수신
721
수신된 정보가 HTML 문서에 의해 사용될 가능성이 있는 것으로서 CSS를 마킹하는지의 여부를 결정
722
사용될 가능성이 있는 규칙인가?
아니오
예
723
리스트에서 CSS 규칙을 저장
724
CSS 규칙에 의해 참조된 자원들을 다운로드하기 위한 요청을 발행
725
CSS 문서의 스캐닝을 계속
726
HTML 스캐너로부터 통지를 수신
727
리스트로부터 CSS 규칙을 취출하고 CSS 규칙을 평가
728
사용될 가능성이 있는 규칙인가?
아니오
예
729
CSS 규칙에 의해 참조된 자원들을 요청

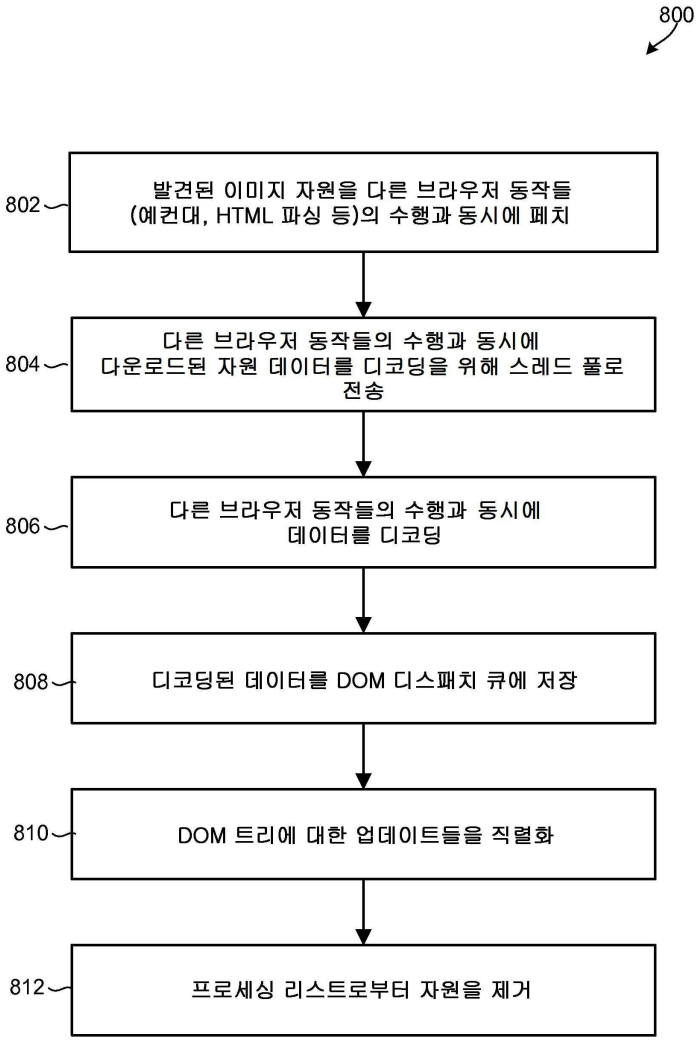
도면7b


730
스캐너
732
자원들을 발견하기 위해 HTML 문서를 스캔
734
자바스크립트 스크립트와 조우
샌드박스식 엔진
735
자바스크립트 스크립트의 스캐닝을 시작
736
자원들을 발견하기 위해 자바스크립트 코드의 부분들을 추론적으로 실행
738
자원을 발견
740
자원을 다운로드하기 위한 요청을 발행
742
프로세싱의 결과들을 버림
프리페처
744
자원을 로케이트
746
자원을 다운로드
748
자원을 메모리에 저장

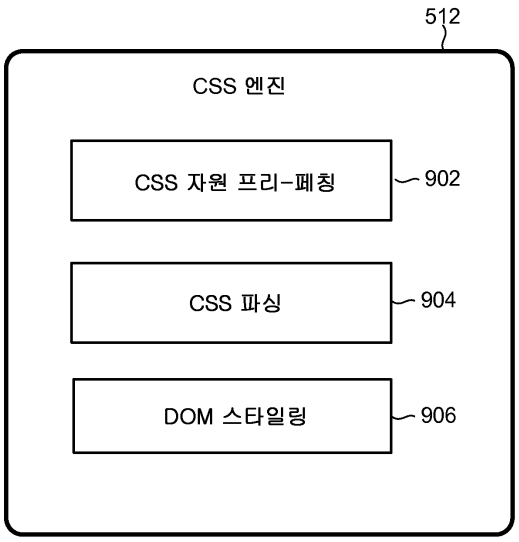
도면7c


750
스캐너/프리-페처
752
HTML 문서를 스캔
754
스크립트들을 발견
756
외부 스크립트들의 다운로딩을 개시
HTML 파서
758
각각의 스크립트에 대한 식별자를 생성
760
스크립트들 및 연관된 식별자들을 자바스크립트 엔진으로 전송
762
HTML 파서 동작들을 수행
764
실행될 다음의 스크립트를 식별
766
실행될 다음의 스크립트의 식별자를 자바스크립트 실행 엔진에 전송
768
실행의 결과들을 대기
770
HTML 파서 동작들을 계속
자바스크립트 엔진
772
스크립트들 및 연관된 식별자들을 수신
774
병렬로, 실행을 위해 수신된 스크립트들을 준비
776
스크립트의 준비시 생성된 정보와 그 식별자를 연관시킴
778
HTML 파서가 실행할 다음 스크립트의 식별자를 수신
780
수신된 식별자에 기초하여 적절한 스크립트를 식별
782
실행할 준비가 된 식별된 스크립트가 있는가?
예
아니오
784
종래의 솔루션들을 통해 실행을 위해 식별된 스크립트를 준비
786
실행 순서에 따라 스크립트를 실행

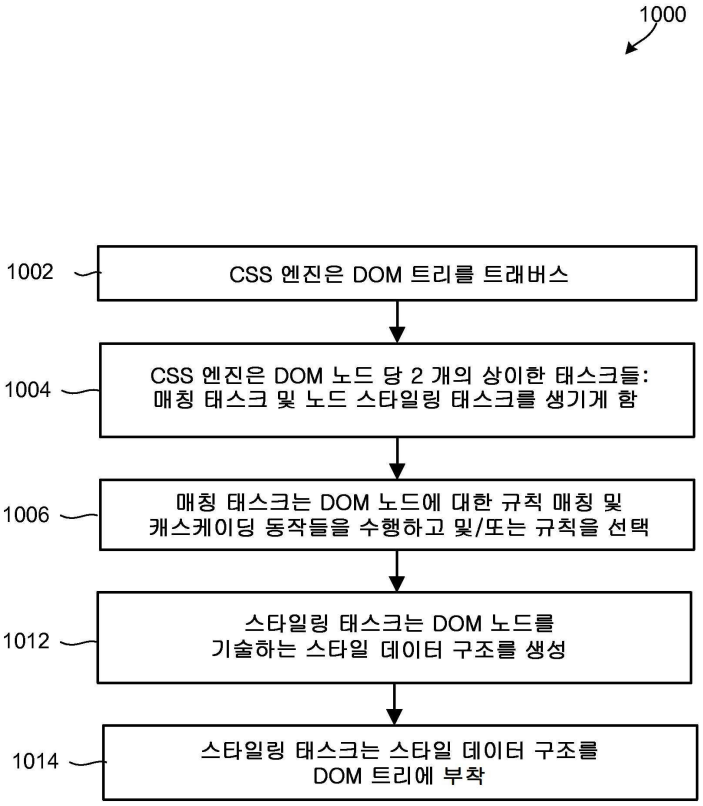
도면8


800
802
발견된 이미지 자원을 다른 브라우저 동작들 (예컨대, HTML 파싱 등)의 수행과 동시에 페치
804
다른 브라우저 동작들의 수행과 동시에 다운로드된 자원 데이터를 디코딩을 위해 스레드 풀로 전송
806
다른 브라우저 동작들의 수행과 동시에 데이터를 디코딩
808
디코딩된 데이터를 DOM 디스패치 큐에 저장
810
DOM 트리에 대한 업데이트들을 직렬화
812
프로세싱 리스트로부터 자원을 제거

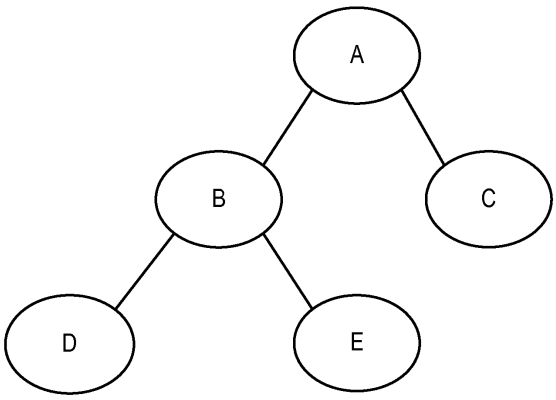
도면9


512
CSS 엔진
CSS 자원 프리-페칭
902
CSS 파싱
904
DOM 스타일링
906

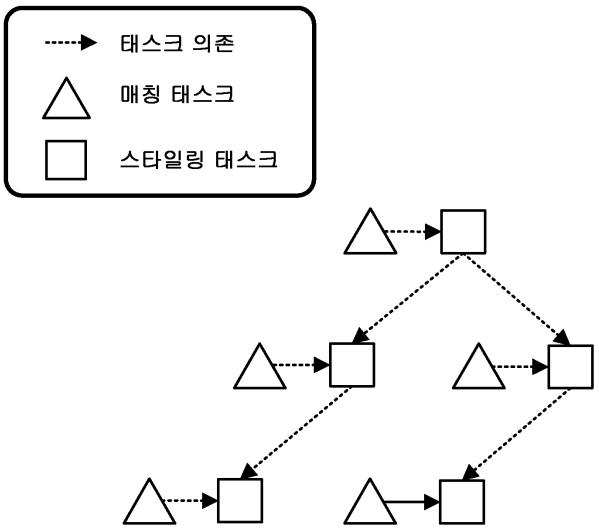
도면10


1000
1002
CSS 엔진은 DOM 트리를 트래버스
1004
CSS 엔진은 DOM 노드 당 2 개의 상이한 태스크들:
매칭 태스크 및 노드 스타일링 태스크를 생기게 함
1006
매칭 태스크는 DOM 노드에 대한 규칙 매칭 및
캐스케이딩 동작들을 수행하고 및/또는 규칙을 선택
1012
스타일링 태스크는 DOM 노드를
기술하는 스타일 데이터 구조를 생성
1014
스타일링 태스크는 스타일 데이터 구조를
DOM 트리에 부착

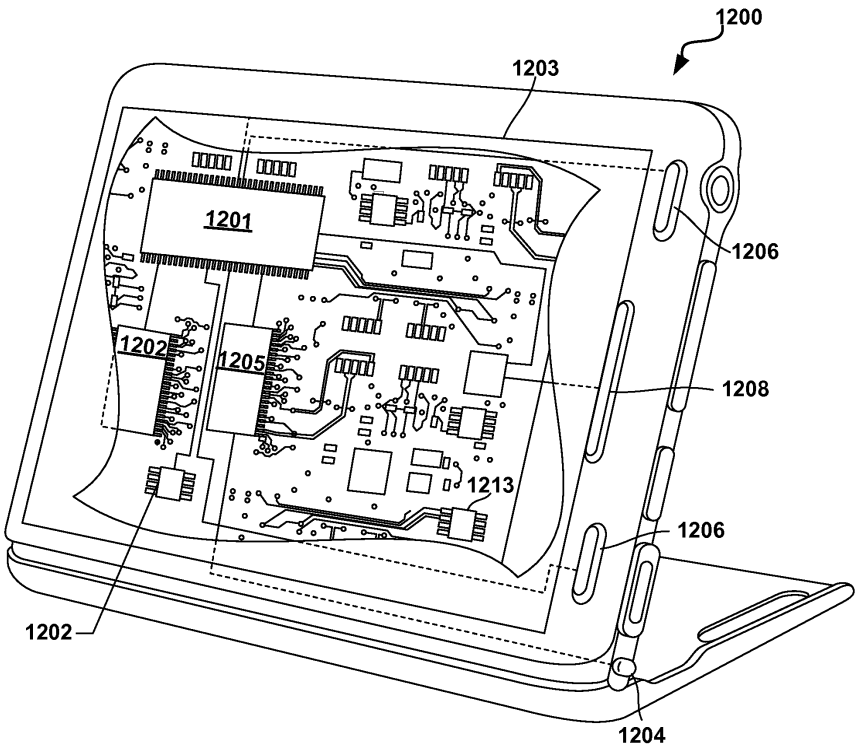
도면11a


A
B
C
D
E

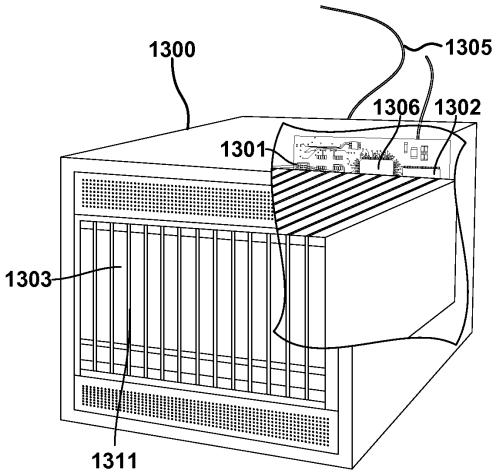
도면11b


태스크 의존
매칭 태스크
스타일링 태스크

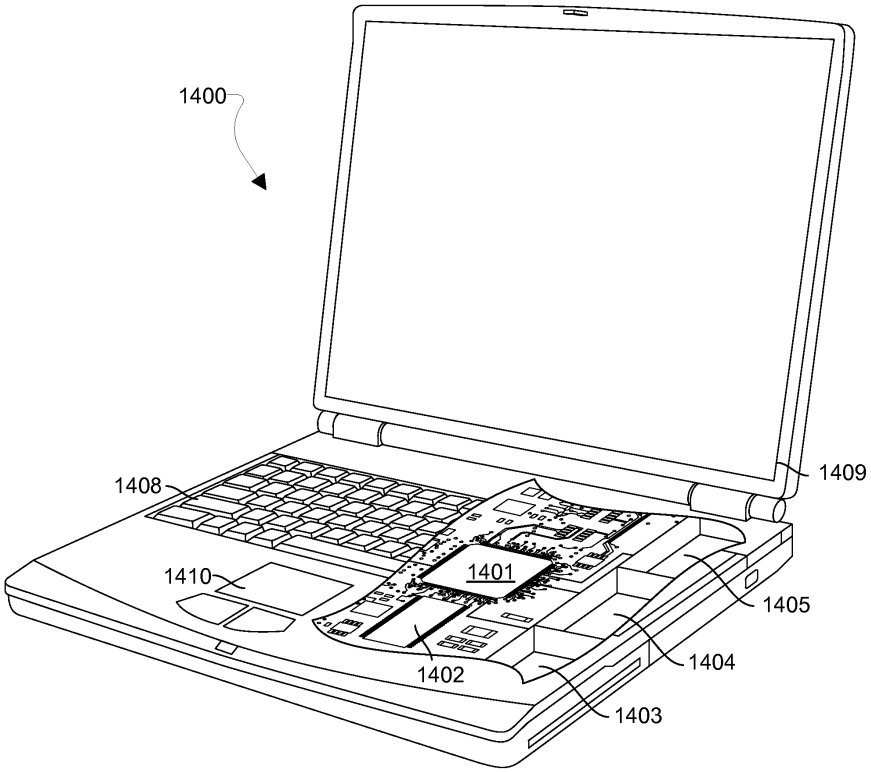
도면12


1200
1203
1201
1206
1202
1205
1208
1213
1206
1202
1204

도면13


1300
1305
1306
1302
1301
1303
1311

도면14


1400
1409
1408
1401
1410
1405
1404
1402
1403