(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
18 October 2012 (18.10.2012)

WIPO | PCT

(10) International Publication Number
WO 2012/141838 A1

(51) International Patent Classification:
G06F 3/00 (2006.01)

(21) International Application Number:
PCT/US2012/029361

(22) International Filing Date:
16 March 2012 (16.03.2012)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/474,835 13 April 2011 (13.04.2011) US

(71) Applicant *(for all designated States except US)*: **THE BOARD OF TRUSTEES OF THE UNIVERSITY OF ILLINOIS** [US/US]; 506 South Wright Street, Urbana, IL 61801 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **KING, Samuel** [US/US]; 203 W. Pennsylvania Avenue, Urbana, IL 61801 (US). **MAI, Haohui** [CN/US]; 202 N. Coler Avenue, Apt. C, Urbana, IL 61801 (US). **TANG, Shuo** [CN/US]; 610 Oakland Avenue, Apt. 206, Urbana, IL 61801 (US).

(74) Agent: **SCHOEDEL, Lisa, M.**; Mcdonnell Boehnen Hulbert & Berghoff LLP, 300 South Wacker Drive, Suite 3100, Chicago, IL 60606 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

WO 2012/141838 A1

(54) Title: WEB BROWSING

(57) Abstract: A method and system for web browsing is disclosed. A server receives a request for a web page from a browser located on a client device. The server retrieves the web page and decomposes the web page into at least two mini-pages. The server sends the mini-pages to the client device. The client device receives the mini-pages and the browser processes the mini-pages in parallel and reassembles them into a single display on the client device.

WEB BROWSING

CROSS REFERENCE TO RELATED APPLICATIONS

Pursuant to the provisions of 35 U.S.C. § 119(e), this application claims priority to United States Provisional Application Serial No. 61/474,835 filed April 13, 2011, the entire contents of which are incorporated herein by reference.

GOVERNMENTS RIGHTS STATEMENTS

This invention was made with government support under contract number N00014-0901-0743 awarded by Office of Naval Research. The government has certain rights in the invention.

FIELD

The present invention relates generally to web browsing, and more particularly, relates to decomposing web pages into sub-pages and rendering the sub-pages in parallel.

BACKGROUND

A web browser, such as Foxfire, Chrome, and Safari, requests a web page from a web server. A Document Object Model (DOM) represents the state of the web browser. JavaScript code can access browser states through the DOM. The JavaScript code is the main executable portion of a web page.

Web browsing is slow, especially on mobile platforms. One source of overhead for web-based applications is the bandwidth of the network that includes the web server. Another source of overhead for web-based applications is the processing power of the client device, such as a mobile telephone. Thus, performance problems exist due to bottlenecks that occur at both the server-side and the client-side.

Recent trends towards multi-core mobile platforms present an opportunity to improve web browser performance on mobile devices. A multi-core platform is a platform that includes a multi-core processor, which is capable of parallel processing. Unfortunately, years of sequential optimizations, the sheer size of modern browsers (e.g., Foxfire has over three million lines of code), and the fundamentally single-

threaded event-driven programming model of modern browsers make it challenging to redesign today's browsers into multi-threaded parallel applications.

SUMMARY

A method and system for web browsing is disclosed. For example, a method includes a client device receiving at least two mini-pages. A browser on the client device processes each of the mini-pages in parallel and displays a single display on the client device using the processed mini-pages.

As another example, a method includes receiving at a server a request for a web page. The method further includes retrieving the web page, decomposing the web page into a set of mini-pages, and sending the set of mini-pages to a client device that requested the web page.

As another example, a system for decomposing a web page into a set of mini-pages includes a processor, data storage, and machine language instructions stored in the data storage. When executed by the processor, the machine language instructions obtain information regarding format of a web page, scan through the information to determine where to split the web page, and splits the web page into mini-pages.

These as well as other aspects and advantages will become apparent to those of ordinary skill in the art by reading the following detailed description, with reference where appropriate to the accompanying drawings. Further, it is understood that this summary is merely an example and is not intended to limit the scope of the invention as claimed.

BRIEF DESCRIPTION OF THE DRAWINGS

Presently preferred embodiments are described below in conjunction with the appended drawing figures, wherein like reference numerals refer to like elements in the various figures, and wherein:

Figure 1 is a block diagram depicting communication between a client device and a server, according to an example;

Figure 2 is a flow diagram of a method of displaying a web page on the client device depicted in Figure 1, according to an example;

Figure 3 is a flow diagram of a method of decomposing the web page on the server depicted in Figure 1, according to an example;

Figure 4 depicts one example of how a web page can be sliced into mini-pages;

Figure 5 is an example algorithm for decomposing web pages; and

Figure 6 is a flow diagram of a method of rendering the web page on the client device depicted in Figure 1, according to an example.

DETAILED DESCRIPTION

Figure 1 is a block diagram 100 depicting communication between a client device 102 and a server 104. The client device 102 and the server 104 may communicate via a network (not shown), such as a local area network (LAN), wide area network (WAN), wireless network (Wi-Fi), or the Internet, for example. In addition, client device 102 and the server 104 may be coupled to each other using wired or wireless communications. For example, communication links between the client device 102 and the server 104 may include wired connections, such as a serial or parallel bus, or wireless links, such as Bluetooth, IEEE 802.11 (IEEE 802.11 refers to any IEEE 802.11 version), or other wireless based communication links.

The communication between the client device 102 and the server 104 may be generally described as follows. The client device 102 requests a web page from the server 104. The server 104 retrieves the requested web page and associated resources and decomposes the web page into mini-pages. The server 104 sends the mini-pages to the client device 102. The client device 102 processes the mini-pages in parallel and reassembles them into a single display. The communication between the client device 102 and the server 104 is further described with respect to Figure 2.

The client device 102 is a computing platform having a combination of hardware and software that includes a processor, memory, and programmable instructions and data stored in the memory. Preferably, the client device 102 is a multi-core mobile platform, such as a personal computer, a laptop computer, a smart phone, a tablet, or other hand-held computing device. As depicted in Figure 1, the client device 102 includes a processor 110, memory 112, an input device 114, and a display 116. The client device 102 includes other components as well, such as a power supply.

The processor 110 is a capable of parallel processing. Preferably, the processor 110 is a multi-core processor, such as a four-core ARM Cortex-A9

processor. A multi-core processor is a single computing component with two or more independent actual processors (referred to as "cores"). The multiple cores can run multiple instructions at the same time allowing for parallel processing.

The memory 112 may include transitory computer readable medium, such as computer-readable media that stores data for short periods of time like register memory, processor cache, and Random Access Memory (RAM). The memory 112 may also include non-transitory media, such as secondary or persistent long term storage, like read only memory (ROM), optical or magnetic disks, compact-disc read only memory (CD-ROM). The memory 112 may also be any other volatile or non-volatile storage systems.

The memory 112 includes a browser 118. The browser 118 is a software program operable to generate and send a request for a web page to the server 104. The browser 118 is also operable to receive a response from the server 104 and generate instructions for displaying a web page on the display 116. The browser 118 includes Inter-Process Communication (IPC) channels or other forms of thread synchronization allowing the parallel mini-page processes to communicate with each other.

The input device 114 allows a user to input information into the client device 102, such as a request for a web page. The input device 114 may be any suitable input device including a mouse, trackball, touchpad, touchscreen, stylus, and the like. In one example, the input device 114 is the display 116 and the user touches the display 116 to enter information into the client device 102. The display 116 may be a LCD, LED, or other display capable of producing textual and/or graphical output.

The server 104 is a combination of hardware and software that includes a processor, memory, and programmable instructions and data stored in the memory. The server 104 receives requests for web pages from the client device 102. In one example, the server 104 is a web server whose primary function is to deliver web pages at the request of clients. In another example, the server 104 is a proxy server that acts as an intermediary for requests from clients seeking resources from other servers, such as a web server.

As depicted in Figure 1, the server includes a processor 120 and memory 122. The processor 120 may be one or more processors suitable for receiving input signals, executing machine language instructions, and providing output signals, such as an

Intel Xeon Quad-Core processor. The memory 122 may be substantially the same as the memory 112. The server 104 includes other components as well, such as a power supply.

The memory 122 includes a decomposition algorithm 124. The decomposition algorithm 124 is a software program that divides a web page into a plurality of sub-pages, referred to herein as mini-pages. The mini-pages are formatted as browser plugins or user interface widgets. The decomposition algorithm 124 is further described with reference to Figures 3-5.

Figure 2 is a flow diagram of a method 200 of displaying a web page on the client device 102 using information received from the server 104. It should be understood that each block in this flowchart (and within other flow diagrams presented herein) may represent a module, segment, or portion of computer program code, which includes one or more executable instructions for implementing specific logical functions or steps in the process. Alternate implementations are included within the scope of the example embodiments in which functions may be executed out of order from that shown or discussed, including substantially concurrently or in reverse order, depending on the functionality involved.

At block 202, a user of the client device 102 requests a web page. In one example, the user inputs a Uniform Resource Locator (URL) into the browser 118. A URL is a character string that constitutes a reference to an Internet resource. For example, the user may enter "http://en.wikipedia.org/" into the browser or click on a link in a web page already displayed in the display 116.

At block 204, the browser 118 requests the web page from the server 104. The request is expressed in a markup language, such as Hypertext Markup Language (HTML), Extensible Markup Language (XML) with or without Extensible Style Sheets (XSL), VoiceXML, Extensible Hypertext Markup Language (XHTML), or Wireless Markup Language (WML). The request may also be made in another suitable language.

At block 206, the server 104 receives the request from the browser 118. The server 104 retrieves the requested web page and associated resources. The decomposition algorithm 124 slices the web page into at least two mini-pages. Each mini-page represents a portion of the web page. The combination of all of the mini-pages includes all of the information found in the web page and does not include any

overlapping information. Each mini-page is a complete web page that includes HTML, JavaScript, Cascading Style Sheets (CSS), and so on. The document object model (DOM) is spread across multiple mini-pages.

In one example, the server 104 is a proxy server. The server 104 receives the request from the browser 118 and retrieves the web page from a web server where the web page is stored. The web page is then decomposed on the proxy server. In this example, the web page is decomposed "on the fly." For web sites with mostly static content, such as wikipedia.org and nytimes.com, the proxy server caches the decomposed web site to reduce processing time for subsequent requests of the same web page. For web sites with mostly dynamic content, such as facebook.com or gmail.com, the proxy server decomposes the web page for each new request.

In another example, the server 104 is a web server and a web developer has previously decomposed the web page and stored the mini-pages on the server 104. In this example, the server 104 does not need to decompose the web page on the fly. Instead, the server 104 simply retrieves the previously stored mini-pages.

At block 208, the server 104 transmits the mini-pages to the client device 102. The response including the mini-pages is also expressed in a markup language or other appropriate language.

At block 210, the client device 102 receives the mini-pages from the server 104. The browser 118 processes the mini-pages in parallel. While the mini-pages are processed in parallel, each mini-page uses traditional browsing components for processing. The browser 118 synchronizes global data structures, such as the DOM, and propagates user interface (UI) and JavaScript events to maintain web semantics. The mini-page processing is further described with respect to Figure 6.

At block 212, the browser 118 renders the web page on the display 116. The browser 118 combines displays from the separate mini-pages into a single display window.

Figure 3 is a flow diagram of a method 300 of decomposing a web page. In one example, the server 104 runs the method 300. In another example, a web developer designs a web page and before deploying it, runs it through the method 300. The web developer then integrates the results from the method 300 directly into web pages stored on the server 104. In yet another example, a web proxy fetches raw web content from web sites and then runs the web page through the method 300.

At block 302, the decomposition algorithm 124 obtains information regarding the format of the web page. In one example, the decomposition algorithm 124 obtains information regarding the format of the web page by rendering the web page. In another example, the decomposition algorithm 124 obtains information regarding the format of the web page by creating a bit map.

At block 304, the decomposition algorithm 124 scans through the information obtained at block 302 to determine where to split the web page. In one example, the decomposition algorithm 124 uses visual cues of the web page to identify where to slice a web page. In many cases, a web developer divides a web page into separate visual containers that are mostly independent and separating the visual containers preserves the visual semantics of the web page.

At block 306, the decomposition algorithm 124 splits the web page into mini-pages. Figure 4 shows an example of a web page 400 split into four mini-pages 402-408. Of course, the decomposition algorithm 124 may split the web page 400 into a different number of mini-pages.

In the example of using visual cues, the decomposition algorithm 124 slices the web pages into the separate visual containers as divided by the web developer. In this example, the decomposition algorithm 124 slices a web page based on a minimum bounding box of visual elements on the web page. The bounding boxes do not overlap and respect the layout of the original page.

The decomposition algorithm 124 may spread the JavaScript code among the mini-pages. In another example, the decomposition algorithm 124 places all JavaScript code in a single "main" mini-page. Although running all JavaScript in a single mini-page prevents the browser 118 from running the JavaScript code in parallel, it maintains all of the JavaScript variables and namespaces in the single mini-page and the browser 118 executes the JavaScript code sequentially, making it straightforward to maintain current JavaScript semantics. As a result, the browser 118 can process legacy web-based applications.

In another example, a web developer uses an application programming interface (API) for decomposing web pages manually, which could also be used to parallelize processing of the JavaScript code. The web developer identifies independent JavaScript code and places the independent code in its own mini-page.

The browser 118 executes this JavaScript code in parallel with the other mini-pages, but without maintaining JavaScript namespaces or states across mini-pages.

The decomposition algorithm 124 may optimize the mini-pages. For example, the decomposition algorithm 124 may also use the visual cues for load balancing the mini-pages. In this example, the decomposition algorithm 124 may use the size of the screen area that a mini-page occupies to approximate the processing time. To load balance, the decomposition algorithm 124 creates mini-pages that share roughly an equal amount of the screen.

Figure 5 shows an example decomposition algorithm 500 to compute a partition *S*. In this example, the decomposition algorithm 500 divides a DOM tree (T) that represents the original web page into k mini-pages. The decomposition algorithm 500 slices k-1 mini-pages out of the DOM tree and places the remaining DOM elements into a main mini-page. The idea is to keep dividing web page segments into smaller segments until they are small enough to be pulled out as individual mini-pages considering various constraints.

The decomposition algorithm 500 is designed considering the following constraints. One constraint is maintaining consistent states of the DOM. Preferably, all elements in a slice have the same sibling before and after slicing, so that the mini-page can be plugged into the main mini-page as a whole during the merging to maintain the semantics of JavaScript. This can be represented as: $\forall i, j, (e_i \neq e_j)$ $\vee((q_i < p_j) \vee((q_j < p_i))$.

Another constraint is respecting the layout of both the main mini-page and the other mini-pages. Preferably, the mini-pages are rectangular. The bounding box for a mini-page cannot contain elements from other mini-pages or the main mini-page or the layout of the web page may be potentially different. This can be represented as: $\forall i, j, bb(s_i) \cap bb(s_j) = \Phi$.

Another constraint is to keep "sticky" nodes (i.e., <SCRIPT>) in the main mini-page. To comply with this constraint, the nodes in mini-pages do not contain SCRIPT tags. This can be represented as: $\forall i, \neg sticky(s_i)$. Another constraint is to balance the load between the *k* parts and minimize synchronization (i.e., merging). This can be represented as: $max\{R(s_0), max_i\{R(s_i)\}\}$.

The algorithm 500 takes the segment with highest rank ($r(e)$) and divides it iteratively. The rank $r(e)$ is the sum of weight for all descendants of the node *e* plus

the weight of node *e* (*w*(*e*)). The line *adjustRequired(p)* tests whether the non-overlapping constraint is satisfied if *p* is added into the result *S,* and *adjust* further slices *p* in order to satisfy the non-overlapping constraint. *makeSegment* cuts nodes that have rank higher than *r(root)/k* or nodes containing sticky descendants into a single segment, and it cuts the other parts of the segments that have rank roughly equal to *r(root)/k. makeMinipage* cuts a segment into a mini-page.

In one example, the weight of each node is equal to the time that the web browser 118 spends on processing the node. In another example, the weight of each node is the size of the screen that is occupied by the element. In yet another example, the rank of each node is the normalized size of the screen that it occupied, and the weight of a node (*w*(*e*)) is calculated by subtracting the rank of all its children. The algorithm 500 stops when the rank of the segment gets too small.

The *adjust()* and *adjustRequired()* functions are implemented using the HitTest framework in the browser. The HitTest framework enables the algorithm 500 to query the set of elements inside a rectangle. The algorithm 500 calculates the bounding box and then queries whether any elements exist in the main mini-page or in the other mini-pages to make sure that the segment meets the non-overlapping constraint.

Figure 6 is a flow diagram of a method 600 of rendering the web page on the client device 102. The method 600 allows the browser 118 to parallelize a wide range of browser algorithms and computations while working with legacy web application semantics. The user of the client device 102 may notice the increased speed of display, but not the separate mini-page processing.

At block 602, the browser 118 receives a set of mini-pages and at block 604, the browser 118 processes the mini-pages in parallel. For each of the mini-page processes, the browser 118 runs the same code, but is presented with different data. The browser 118 synchronizes global data structures, such as the DOM tree, and propagates UI and JavaScript events to maintain correct web semantics. Each mini-page builds up its own DOM structure in parallel, but shares global data structures and display states.

Preferably, the decomposition algorithm 124 slices web pages in a manner such that the JavaScript code does not need partially-formed DOM states from remote

mini-pages. However, if necessary to maintain DOM consistency, the browser 118 merges mini-pages when the JavaScript code accesses remote DOM states.

When the JavaScript code accesses remote DOM states, the remote mini-page serializes its entire DOM and sends the contents back to the requesting JavaScript code. The JavaScript code then inserts this DOM into the DOM of its own mini-page, creating a local and consistent view of the DOM. Also, the browser 118 terminates the remote mini-page, leaving the local copy as the sole copy of the newly inserted DOM states. To handle the mini-page merging, the browser 118 includes modifications to the Node methods *firstChild*, *lastChild*, *previousSibling*, and *nextSibling*; the Element methods *firstElementChild*, *lastElementChild*, *previousElementSibling*, and *nextElementSibling*; and the DOM API calls *getElementById* and *getElementsByTagName*.

In a traditional sequential web page, the browser builds up the DOM data structure as it parses the HTML of the page. If the parser encounters JavaScript code, it executes this code with the current state of the DOM. After the JavaScript code finishes, the parser continues building up the DOM data structure.

Here, each mini-page builds up its own DOM structure in parallel, so when the JavaScript code executes, the browser 118 ensures that the JavaScript code accesses the correct DOM state. To ensure that the JavaScript code accesses the correct DOM state, the browser 118 inspects the program counter of the JavaScript code at the base of the current call stack.

If the JavaScript code is earlier in the page than the DOM of a mini-page, the browser 118 skips the mini-page when searching for DOM elements because from the JavaScript code's perspective that later portion of the DOM does not yet exist. If the JavaScript code is later in the page than the DOM of a mini-page, the browser 118 blocks the JavaScript request until the mini-page finishes forming its DOM.

At block 606, the browser 118 combines displays from separate mini-pages into a single display. A display widget, which may be the main mini-page, hosts the display for the other mini-pages and embeds the mini-pages as plugins using *<EMBED>* tags. The browser 118 renders mini-page content to a bitmap and passes this bitmap to the display widget using shared memory. The display widget then displays this bitmap on the display 116.

To handle UI events, the main mini-page receives events from a window manager. If the destination of the event is a mini-page, the main mini-page forwards the UI event to the appropriate mini-page. The mini-page processes the event.

Some UI events cause DOM events (e.g., *onclick*) and programmers can add JavaScript event handlers that run in response to these events. If a mini-page detects a DOM event that has a registered listener, the mini-page routes the event to the main mini-page where the event handler JavaScript code resides. For example, if a user clicks on a link in a mini-page, the main mini-page sends the UI mouse event to the mini-page. After the mini-page processes the UI mouse event and determines that the click was on a link, the mini-page forwards the *onclick* DOM event for that link back to the main mini-page, which runs the appropriate event handlers. This DOM event-delivery mechanism ensures that DOM events originating in mini-pages are routed to the main mini-page.

The client device 102 may also provide feedback to the server 104 regarding the results of the decomposition algorithm 124. For example, if the browser 118 detects merging for a web page, the browser 118 notifies the server 104 about this merging on subsequent requests to the same web page. The decomposition algorithm 124 prevents the merged mini-page from forming a separate mini-page in future executions of the decomposition algorithm 124.

In addition to parallel processing, the client device 102 experiences performance improvements due to fewer DOM nodes and processing fewer CSS rules. CSS rules are a mechanism that enables web developers to separate the display attributes of a web page from the web page itself. For example, a web developer uses the CSS rule *hl em { color : red }* to specify that any emphasized text (em) within a heading (h1) should be red.

The browser 118 includes a CSS selector matching algorithm. To implement the CSS rules, the CSS selector matching algorithm searches through all of the DOM nodes to find selector matches to apply the appropriate rules. Because mini-pages have fewer DOM nodes than a web page, the CSS selector matching algorithm has a smaller set of nodes to process and this processing happens in parallel. Furthermore, not all CSS rules apply to all mini-pages. For example, if a mini-page does not have any heading elements, then the browser 118 skips a rule for setting the color of

emphasized text within a heading. As a result, the browser 118 executes a subset of CSS rules during individual mini-page processing.

As another example of a performance improvement, the browser 118 uses a Bloom filter to improve the speed of calls to the DOM *getElementById* function. The JavaScript code can use the *getElementById* function to get a pointer to a specific DOM element and the *getElementById* function checks each mini-page for the requested element. To avoid this remote procedure call, the browser 118 uses the Bloom filter to quickly check if the requested ID is contained within a mini-page, avoiding cross mini-page communication.

As another example of a performance improvement, the server 104 may pre-compute results to reduce the processing required by the client device 102. The server 104 passes the pre-computed results to the client device 102, which may overcome some of the latency added by operations inherent in the use of mini-pages. For example, the server 104 includes URLs for each of the mini-pages in the header of the main HTML document that the server 104 returns to the client device 102. This placement allows the browser 118 to issue mini-pages requests without parsing and downloading the entire main HTML document.

It is intended that the foregoing detailed description be regarded as illustrative rather than limiting and that it is understood that the following claims including all equivalents are intended to define the scope of the invention. The claims should not be read as limited to the described order or elements unless stated to that effect. Therefore, all embodiments that come within the scope and spirit of the following claims and equivalents thereto are claimed as the invention.

WE CLAIM:

1. A computer-implemented method, comprising:

at a client device, receiving at least two mini-pages;

a browser on the client device processing each of the mini-pages in parallel; and

the browser displaying a single display on the client device by using the processed mini-pages.

2. The computer-implemented method of claim 1, wherein the browser synchronizes global data structures during the parallel mini-page processing.

3. The computer-implemented method of claim 1, wherein the browser propagates user interface and JavaScript events during the parallel mini-page processing.

4. The computer-implemented method of claim 1, wherein the browser merges the mini-pages when JavaScript code accesses a remote document object model state.

5. The computer-implemented method of claim 1, wherein the browser inspects a program counter of the JavaScript code to ensure that JavaScript code accesses a correct document object model state.

6. The computer-implemented method of claim 1, wherein the browser uses a display widget to host the single display of the other mini-pages.

7. A computer-implemented method, comprising:

receiving at a server a request for a web page;

retrieving the web page;

decomposing the web page into at least two mini-pages; and

sending the at least two mini-pages to a client device that requested the web page.

8. The computer-implemented method of claim 7, wherein the server is a proxy server and the proxy server retrieves the web page from a web server where the web page is stored.

9. The computer-implemented method of claim 7, wherein the server is a web server and the web server retrieves the at least two mini-pages.

10. The computer-implemented method of claim 7, wherein the at least two mini-pages includes a main mini-page that includes JavaScript code.

11. The computer-implemented method of claim 7, wherein each of the mini-pages in the at least two mini-pages is formatted as a browser plugin.

12. The computer-implemented method of claim 7, wherein each of the mini-pages in the at least two mini-pages is formatted as a user interface widget.

13. A system for decomposing a web page into a set of mini-pages, comprising:

a processor;

data storage; and

machine language instructions stored in the data storage executable by the processor to:

obtain information regarding format of a web page;

scan through the information to determine where to split the web page; and

split the web page into mini-pages.

14. The system of claim 13, wherein obtain information includes machine language instructions stored in the data storage executable by the processor to render the web page.

15. The system of claim 13, wherein obtain information includes machine language instructions stored in the data storage executable by the processor to create a bit map.

16. The system of claim 13, wherein scan through the information includes machine language instructions stored in the data storage executable by the processor to use visual cues.

17. The system of claim 13, wherein split the web page into mini-pages includes machine language instructions stored in the data storage executable by the processor to divide the web page based on a minimum bounding box of visual elements on the web page.

18. The system of claim 13, wherein split the web page into mini-pages includes machine language instructions stored in the data storage executable by the processor to place JavaScript into a single mini-page.

19. The system of claim 13, wherein split the web page into mini-pages includes machine language instructions stored in the data storage executable by the processor to place JavaScript into more than one mini-pages.

20. The system of claim 14, further includes machine language instructions stored in the data storage executable by the processor to optimize the mini-pages.

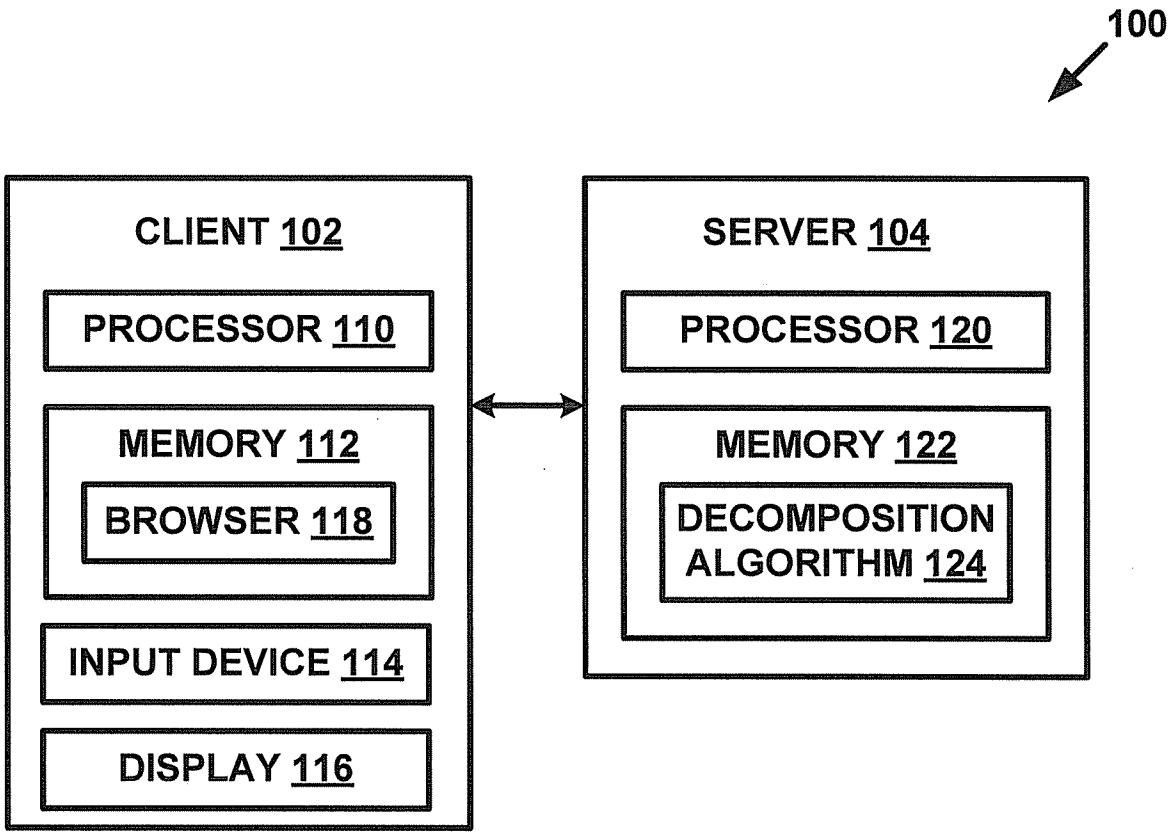
100
CLIENT 102
PROCESSOR 110
MEMORY 112
BROWSER 118
INPUT DEVICE 114
DISPLAY 116
SERVER 104
PROCESSOR 120
MEMORY 122
DECOMPOSITION ALGORITHM 124

FIG. 1

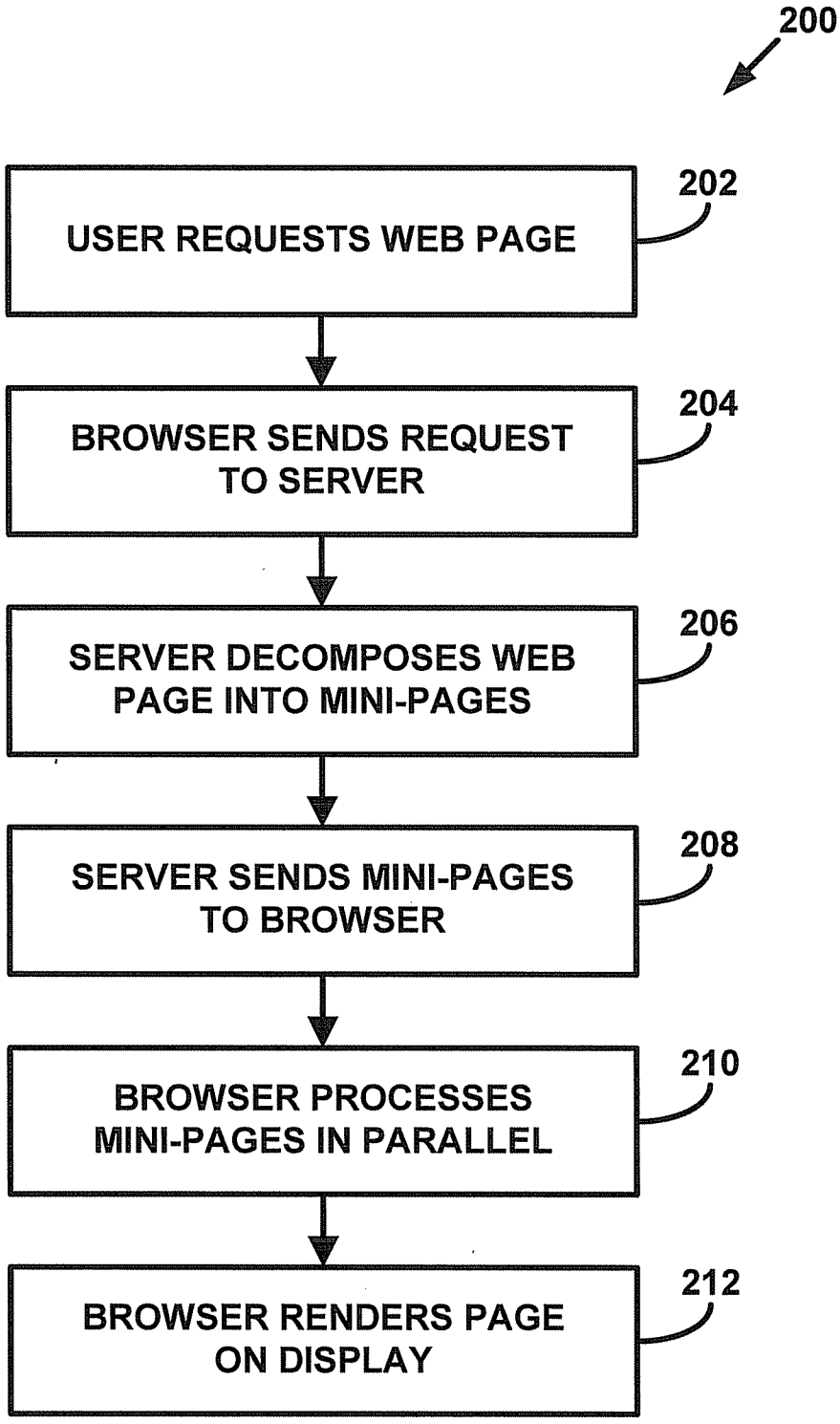
200
USER REQUESTS WEB PAGE
202
BROWSER SENDS REQUEST TO SERVER
204
SERVER DECOMPOSES WEB PAGE INTO MINI-PAGES
206
SERVER SENDS MINI-PAGES TO BROWSER
208
BROWSER PROCESSES MINI-PAGES IN PARALLEL
210
BROWSER RENDERS PAGE ON DISPLAY
212

FIG. 2

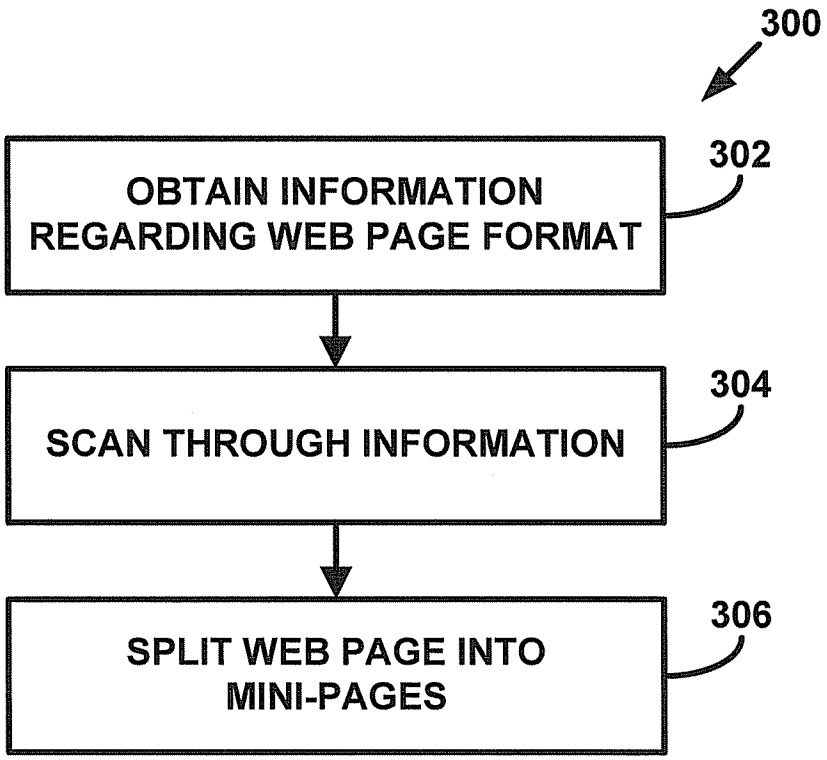
300
OBTAIN INFORMATION REGARDING WEB PAGE FORMAT
302
SCAN THROUGH INFORMATION
304
SPLIT WEB PAGE INTO MINI-PAGES
306

FIG. 3

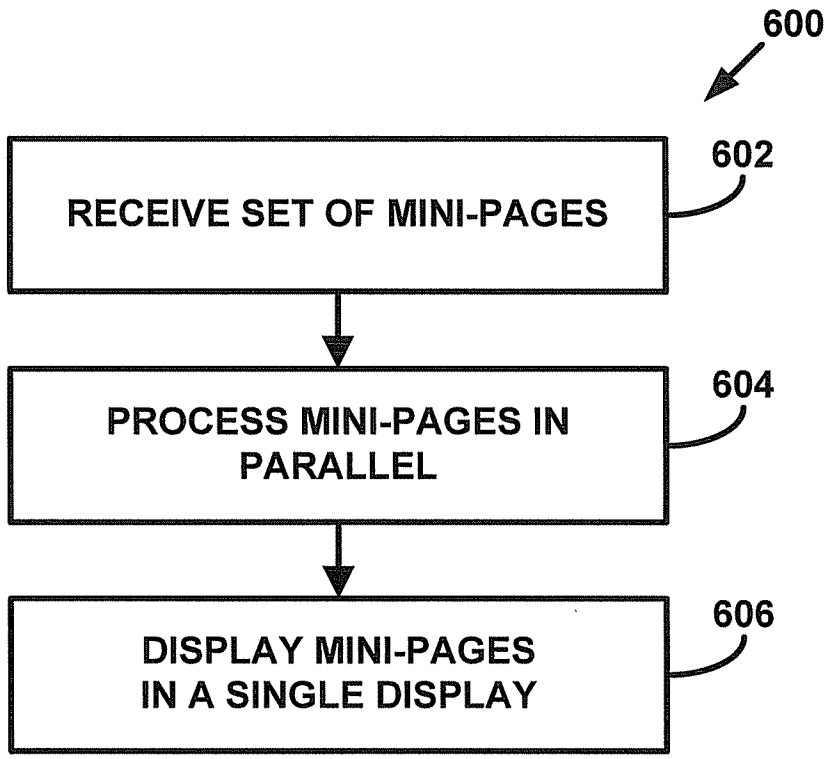
600
RECEIVE SET OF MINI-PAGES
602
PROCESS MINI-PAGES IN PARALLEL
604
DISPLAY MINI-PAGES IN A SINGLE DISPLAY
606

FIG. 6

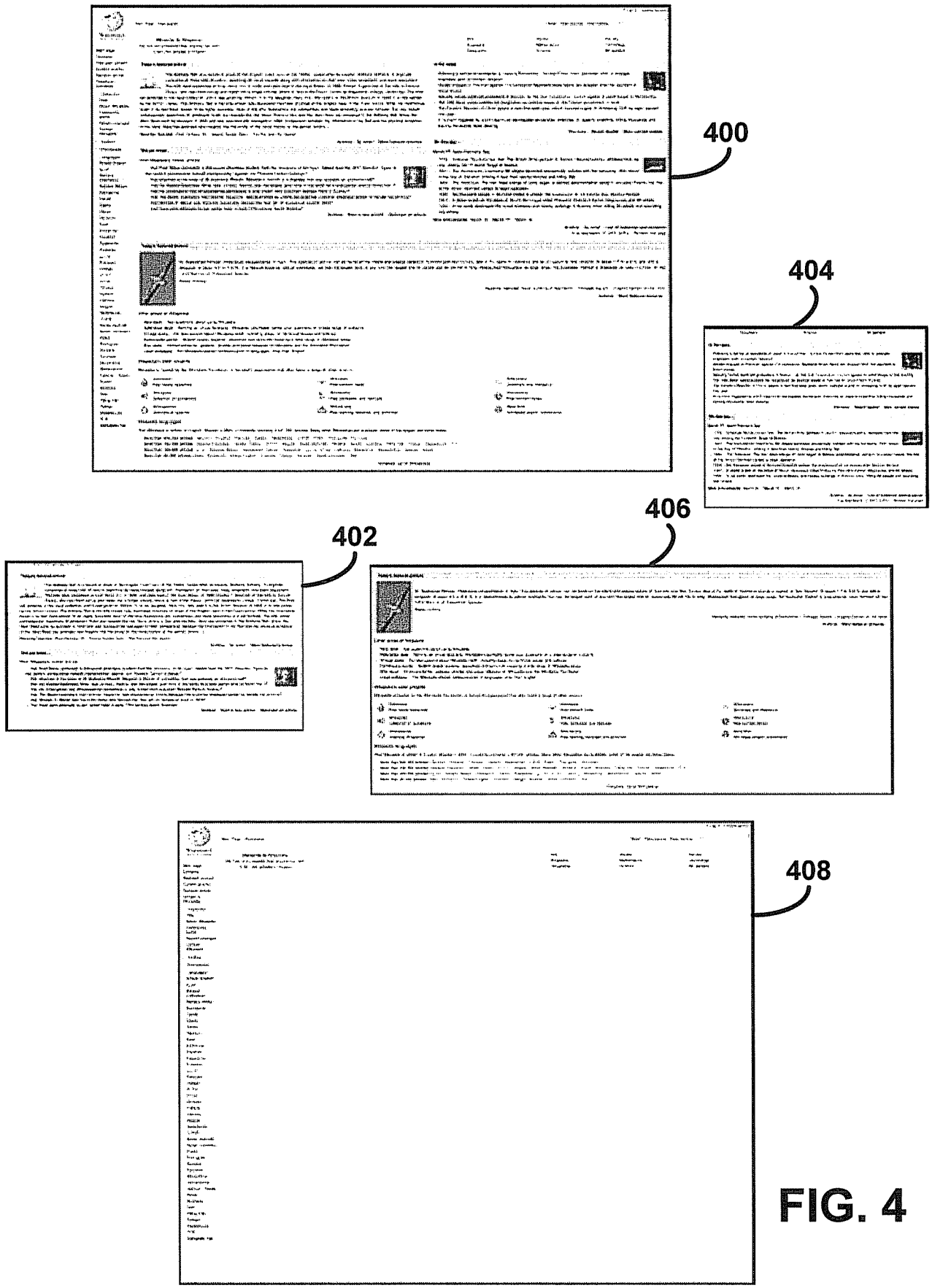

FIG. 4

500

```
slice(root, k)
q ⟵ Priority queue of segments with respect to their ranks.
Add a shadow node root' which contains only one child root
enqueue(q, (root' , 0, 1) )
while k > 1 and q is not empty do
    p ⟵ pop_front(q)
    if adjustRequired(p) then
        P ⟵ adjust(p)
        enqueue all segments in P into q
        continue
    end if
    if R(p) < α · r(root) / k then
        break
    end if
    if R(p) > r(root) / k then
        P ⟵ makeSegment(p, r (root) / k)
        enqueue all segments in P into q
    else
        if ¬ sticky(p) then
            makeMinipage (p)
            k ⟵ k - 1
        else
            P ⟵ makeSegment(p, r (root) / k)
            enqueue all segments in P into q
        end if
    end if
end while
```

FIG. 5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 12/29361

A. CLASSIFICATION OF SUBJECT MATTER
IPC(8) - G06F 3/00 (2012.01)
USPC - 715/760
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC (8) - G06F 3/00 (2012.01)
USPC - 715/760

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
USPC - 715/ 234, 277, 700, 738; 709/203 (keyword limited; search terms below)

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
PubWEST (PGPB,USPT,USOC,EPAB,JPAB), Google Scholar (Articles and Patents); search terms: divide split partition webpage website parallel mobile cellular smart phone pda segment simultaneous document object model javascript ajax render process load visual proxy server program counter dynamic cache

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X — Y	US 2008/0133722 A1 (Ramasundaram et al.) 05 June 2008 (05.06.2008) entire document, especially fig. 4, para. [0027], [0043], [0047]-[0048], [0055]-[0057], [0062], [0096]-[0097], [0100], [0102], [0104]-[0105]	1-4, 6-7, 9-12 --- 5, 8, 18-19
X — Y	US 2009/0177959 A1 (Chakrabarti et al.) 09 July 2009 (09.07.2009) entire document, especially fig. 10 para. [0031]-[0034], [0040]-[0041], [0047], [0056], [0071]	13-14, 16-17, 20 --- 15, 18-19
Y	US 2009/0187918 A1 (Chen et al.) 23 July 2009 (23.07.2009) entire document, especially para. [0034]	5
Y	US 2005/0188048 A1 (Yuan et al.) 25 August 2005 (25.08.2005) entire document, especially abstract, fig. 2	8
Y	US 2005/0041858 A1 (Celi, Jr. et al.) 24 February 2005 (24.02.2005) entire document, especilly abstract	15

☐ Further documents are listed in the continuation of Box C. ☐

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
30 May 2012 (30.05.2012)	13 JUN 2012
Name and mailing address of the ISA/US Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, Virginia 22313-1450 Facsimile No. 571-273-3201	Authorized officer: Lee W. Young PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774