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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
20 December 2012 (20.12.2012)

WIPO | PCT

(10) International Publication Number
WO 2012/174033 A1

(51) International Patent Classification:
G06F 11/36 (2006.01)

(21) International Application Number:
PCT/US2012/042127

(22) International Filing Date:
13 June 2012 (13.06.2012)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
13/160,021 14 June 2011 (14.06.2011) US

(71) Applicant *(for all designated States except US)*: **GOOGLE INC.** [US/US]; 1600 Amphitheatre Parkway, Mountain View, CA 94043 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **SMITH, Corbin** [US/US]; 441 Graham Ave., Apt. 2R, Brooklyn, New York (US). **SESHADRI, Shyam** [IN/IN]; 1804A Pride, Plot 1, Sector 7, Kharghar, Mumbai (IN).

(74) Agents: **KRUEGER, Brett, A.** et al.; Honigman Miller Schwartz and Cohn LLP, 350 East Michigan Avenue, Suite 300, Kalamazoo, MI 49007-3800 (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, 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 2012/174033 A1

(54) Title: SYSTEM AND METHOD TO IN-LINE SCRIPT DEPENDENCIES

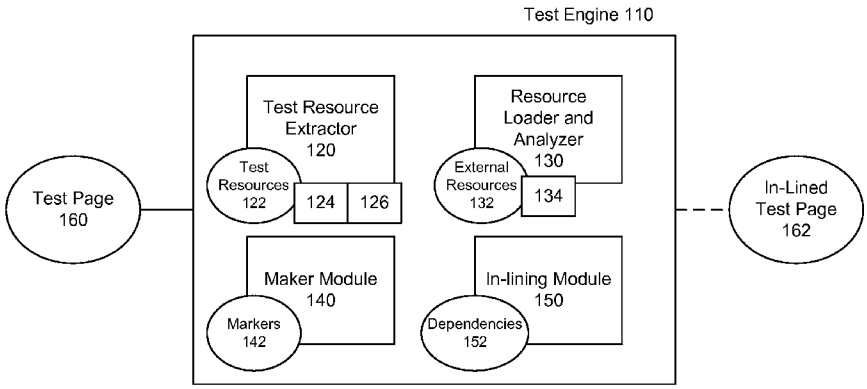

FIG. 1

(57) Abstract: Systems, methods and articles of manufacture to in-line script dependencies include extracting test resources (122) addressed in language defining a test web page (160), placing markers (142) identifying the location of each extracted test resource within the language defining the test page, iteratively loading external resources (132) associated with a path of each test resource, analyzing each test resource to identify one or more dynamically added dependencies (152), and replacing each marker with external resources and dependencies that reference their respective marker to generate updated language defining an updated test web page.

SYSTEM AND METHOD TO IN-LINE SCRIPT DEPENDENCIES

TECHNICAL FIELD

[0001] This disclosure relates to testing frameworks, such as script testing.

BACKGROUND

[0002] Unit testing is a method by which individual units of source code (or script) are tested to determine if they are fit for use. In one example, a unit may be a smallest testable part of an application. In another example, a unit may be an individual function or procedure. Unit tests are typically written and run by software developers to ensure that code behaves as intended when executed. Unit tests are generally run using testing frameworks. Testing frameworks may be used test a web page that includes script. Such testing of the web page may enable developers to determine if, for example, script in the web page loads and executes appropriately.

[0003] JsUnit is an example of a publicly available client-side (or browser-side) unit testing framework for JavaScript. A unit test using the JsUnit framework determines and retrieves the necessary files (or "dependencies") containing JavaScript code for a test to execute properly. In a first technique to load code dependencies into a browser for test, the JsUnit testing framework can place a script tag (e.g.,"<script>") in the Hyper Text Markup Language (HTML) portion of a test page to reference the dependencies. In a second technique, the JsUnit testing framework can dynamically add a script tag referencing dependencies using loaded test code. In a third technique, the JsUnit testing framework can dynamically read dependencies from a server (e.g., using an iFrame or XMLHttpRequest).

[0004] Each of these techniques requires server, network and browser resources, resulting in a distinctly longer time for test setup. When the number of tests become non-trivial, test setup time increases due to the expenditure of resources, in browser and server, necessary to load test dependencies. Additionally, because of reduced computational and network resources, actual test execution time can increase resulting in inappropriate test failures. The second and third techniques noted above also expend

browser resources which can result in slower test execution and occasionally even complete browser failure.

[0005] Some testing frameworks may cache dependencies at a server, or rely on a browser's built-in cache to store the dependencies. Server caching only reduces the time and resources necessary to serve the dependencies to the browser, still requiring a network connection and browser resources. Furthermore, because a browser's cache is inconsistent, caching of dependencies in the browser's built-in cache does not ensure that a script test will run appropriately and without error.

BRIEF SUMMARY

[0006] One aspect of the disclosure provides a method to in-line script dependencies that includes extracting test resources addressed in language defining a test web page, placing markers identifying the location of each extracted test resource within the language defining the test page, iteratively loading external resources associated with a path of each test resource, analyzing each test resource to identify one or more dynamically added dependencies, and replacing each marker with external resources and dependencies that reference their respective marker to generate updated language defining an updated test web page. The method further includes adding each identified dependency after or before a top level parent resource, performing the analyzing and the adding until no new dependencies are identified and providing each new dependency with a reference to a parent marker associated with the top level parent resource.

[0007] In this way, implementations enable in-lining of script dependencies to reduce load and execution time of a script test.

[0008] The details of one or more implementations of the disclosure are set forth in the accompanying drawings and the description below. Other aspects, features, and advantages will be apparent from the description and drawings, and from the claims.

BRIEF DESCRIPTION OF DRAWINGS

[0009] The drawing in which an element first appears is generally indicated by the leftmost digit in the corresponding reference number.

[0010] FIG. 1 illustrates an exemplary system for in-lining script dependencies within test code.

[0011] FIG. 2A illustrates an exemplary overall operation of a testing engine.

[0012] FIG. 2B illustrates an exemplary operation performed by a testing engine.

[0013] FIG. 3 illustrates an example computer useful for implementing components of the implementations discussed.

[0014] Like reference symbols in the various drawings indicate like elements.

DETAILED DESCRIPTION

[0015] Embodiments relate to in-lining script dependencies. Embodiments can in-line (or include) test resources and their dependencies within HTML defining a test page. Because test resources (e.g., scripts) and their dependencies (e.g., external files) are retrieved prior to testing the test page and directly included within HTML defining the test page, embodiments need not expend server and browser resources to load test resources and their dependencies into memory at test run time. In this way, embodiments reduce load and execute time of a script test.

[0016] Embodiments extract test resources addressed in language defining a test page and place markers identifying the location of each extracted test resource within HTML defining the test page. The markers enable embodiments to track locations of extracted test resources while dependencies associated with the test resources are being retrieved. Furthermore, the markers also enable the embodiments to maintain a correct hierarchy of retrieved parent and child dependencies so that the retrieved parent and child dependencies are appropriately included in the correct hierarchy within HTML defining the test page.

[0017] Embodiments load external resources associated with a path of each test resource and analyze each test resource to identity dynamically added dependencies while replacing each marker with external resources and dependencies that reference their respective marker to generate updated language defining an updated (or in-lined) test page. A script test can then be run using the in-lined test page.

[0018] Embodiments further include adding each identified dependency after or before a top level parent resource and performing the analyzing of each test resource and

the adding until no new dependencies are identified. Embodiments provide each new dependency with a reference to a parent marker associated with the top level parent resource.

[0019] While the following is discussed herein with reference to illustrative embodiments for particular applications, it should be understood that the embodiments are not limited thereto. Those skilled in the art with access to the teachings provided herein will recognize additional modifications and applications within the scope thereof and additional fields in which the embodiments would be of significant utility.

[0020] System

[0021] This section describes a system for in-lining script dependencies according to an embodiment illustrated in FIG. 1. FIG. 1 is a diagram of a system 100 for in-lining script dependencies. While the following is described in terms of JavaScript and HTML, the embodiments are not limited to JavaScript and HTML, and can be applied to any other forms of executable content, code, or content defining (or mark-up) language. The embodiments are applicable to any system having generally the structure of FIG. 1, or that would benefit from the operation, methods and functions as described herein.

[0022] FIG. 1 is an architecture diagram of testing engine 110, according to an embodiment. As shown in FIG. 1, testing engine 110 includes test resource extractor 120, resource loader and analyzer 130, marker module 140 and in-lining module 150. FIG. 1 also illustrates test page 160 and in-lined test page 162. In an embodiment, test resource extractor 120 is configured to extract test resources 122 addressed in language (e.g., HTML) defining test page 160. Marker module 140 is configured to place markers 142 for identifying the location of each test resource 122 within the language defining test page 160. Resource loader and analyzer 130 is configured to iteratively load into memory external resources 132 associated with a path of each test resource 122, and analyze each test resource 122 to identify dynamically added dependencies 152. In an embodiment, an in-lining operation may include replacing a function call site with the body (or code) of the called function. In-lining module 150 replaces each marker 142 with resources 132 and dependencies 152 that reference their respective marker 142 to generate updated language defining in-lined test page 162. In an embodiment, testing engine 110 then performs a script test using in-lined test page 162. Because test

resources 122 (e.g., scripts) and their dependencies 152 (e.g., external files) are retrieved prior to testing the test page 160 and directly included within HTML defining in-lined test page 162, embodiments need not expend server and browser resources to load test resources and their dependencies 152 into memory at test run time. In this way, embodiments reduce load and execute time of a script test.

[0023] The operation of test resource extractor 120, resource loader and analyzer 130, marker module 140 and in-lining module 150 is discussed further below.

[0024] Testing engine 110 can be any type of processing (or computing) device having one or more processors. For example, testing engine 110 can be a workstation, mobile device, computer, cluster of computers, set-top box, or other device having at least one processor. Such a device may include, but is not limited to, a device having a processor and memory for executing and storing instructions. Such a device may include software, firmware, and hardware. Software may include one or more applications and an operating system. Hardware can include, but is not limited to, a processor, memory and user interface display. Optional input devices, such as a mouse, touch sensitive screen, or any future developed interaction method may be used.

[0025] Testing engine 110 may also be implemented within a browser. Such a browser can be implemented on any type of processing (or computing) device having one or more processors. In an embodiment, testing engine 110 may also be implemented as a Java "servlet" that is instantiated on a server (not shown) that communicates over a network with testing engine 110. Testing engine 110 may also be implemented as a standalone application or module.

[0026] In an embodiment, testing engine 110 is configured to read test page 160 and generate in-lined test page 162 where script dependencies 152 have been in-lined by testing engine 110. In an embodiment, test page 160 may be a HTML file or any other file. In an embodiment, test page 160 includes one or more test resources 122. Test resources 122 may be test functions that include script code (e.g., JavaScript) that can be executed when test page 160 is read by testing engine 110. In an embodiment, test resources 122 within test page 160 may be identified by testing engine 110 using tags (or identifiers) appearing adjacent to (or encapsulating) the test resources 122. In an example where the test resources 122 are JavaScript, such tags, may be script tags 124 (e.g.,

"<script>"). As known to persons skilled in the art, a script tag is used to define and identify a client-side script, such as JavaScript. The script tag may include scripting statements or it may point to an external script file through an "src" attribute. It is to be appreciated that the embodiments are not limited to script tags and JavaScript, and may be used with other forms of test resource identifiers or scripts known now or developed in the future.

[0027] In an embodiment, test resource extractor 120 is configured to extract test resources 122 included or addressed in language (e.g., HTML) defining test page 160. Test resources 122 may be test functions that include script code (e.g., JavaScript) that can be executed when test page 160 is read by testing engine 110 for a test. In an embodiment, test resources 122 within test page 160 may be identified by tags (or identifiers) appearing adjacent to (or encapsulating) the test resources 122. As noted above, such script tags may be embedded within HTML defining test page 160. In an embodiment, test resource extractor 120 parses test page 160 to identify and extract script tags and associated scripts.

[0028] In an embodiment, marker module 140 is configured to place markers 142 identifying the location of each extracted test resource 122 within HTML defining test page 160. A marker 142 can be any form of identifier or data structure that is used by marker module 140 to identify and maintain the location of each extracted test resource 132.

[0029] In an embodiment, resource loader and analyzer 130 is configured to load into memory external resources 132 (e.g., files, images, scripts, etc.) associated with a path of each test resource 122 addressed in test page 160. In an embodiment, resource loader and analyzer 130 may iterate over the path of each test resource 122 to load external dependencies 152 that may be referenced by the path of the test resource 122. In an example scenario, a test resource 122 may need external resources 132 (e.g., files, images, scripts, etc.) for the test resource 122 to execute properly. Such external resources 132 may be located at a server independent from testing engine 110 or may even be located on a computing device on which testing engine 110 is operating. By iteratively loading external resources 132 (or other dependencies 152) needed by a test

resource 122 into memory in advance, lesser time is required to execute a test because such loading of external resources 132 need not occur in real-time when the test is run.

[0030] In an embodiment, when resource loader and analyzer 130 determines that a test resource path is not resolvable (or accessible), resource loader and analyzer 130 and marker module 140 maintain the placement of the test resource 122, along with the unresolvable path, to enable testing engine 110 to load resources associated with the test resource path at runtime.

[0031] In an embodiment, resource loader and analyzer 130 analyzes each loaded test resource 122 to identify dynamically added dependencies 152. Dynamically added dependencies 152 may be scripts or any other files that may be needed by a test resource 122 at runtime. In an embodiment, resource loader and analyzer 130 may identify dynamically added dependencies 152 by evaluating scripts using a script runtime engine (e.g., JavaScript runtime engine) and parsing the test resource 122 to identify dependency hints. Such dependency hints may include, for example, resource requirement calls.

[0032] In an embodiment, resource loader and analyzer 130 adds each new identified dependency 152 either after the top level parent resource 122 (i.e., mimicking a browser's runtime HTML load order), or before the top level parent resource 122 (i.e., mimicking the dynamic resolution load order). Marker module 140 also provides each new dependency 152 with a reference to a parent marker 142 that is associated with the top level parent resource 122.

[0033] In an embodiment, resource loader and analyzer 130 continues to identify and add each new dependency 152 to a top level parent resource 122, until no new resources 122 are identified. In this way, by iteratively loading external resources 132 (or other dependencies) needed by a test resource 122 in advance of testing, lesser time is required to execute the test because such loading of external resources 132 need not occur in real-time when the test is run.

[0034] As noted above, marker module 140 is configured to place markers 142 identifying the location of each extracted test resource 132 within HTML defining test page 160. A marker 142 can be any form of identifier or data structure that is used by marker module 140 to identify and maintain the location of each extracted test resource 122. In an embodiment, in-lining module 150 replaces each marker 142 with extracted

resources 132 and dependencies 152 that reference their respective marker 142 to generate in-lined test page 162.

[0035] In an embodiment, any unresolved markers 142 (or markers associated with unresolved paths) are left unchanged so that testing engine 110 may load resources 122 associated with those unresolved paths at runtime. Testing engine 110 then performs a script test using in-lined test page 162. Because test resources 122 (e.g., scripts) and their dependencies 152 (e.g., external files) are retrieved prior to testing the test page 160 and directly included within HTML defining in-lined test page 162, embodiments need not expend server and browser resources to load test resources 122 and their dependencies 152 into memory at test run time. In this way, embodiments reduce load and execute time of a script test.

[0036] FIG. 2A is a flowchart illustrating method 200. Method 200 is an exemplary overall operation of testing engine 110, according to an embodiment.

[0037] Method 200 begins with testing engine 110 reading test page 160 (step 202). As an example, such a test page 160 may be a HTML file or any other file. In an embodiment, a test page 160 includes one or more test resources 122. Test resources 122 may include script code (e.g., JavaScript).

[0038] Test resource extractor 120 extract script tags 124 associated with test resources 122 in test page 160 (step 204). As discussed above, such script tags 124 may be embedded within HTML defining test page 160. In an embodiment, test resource extractor 120 parses test page 160 to identify and extract script tags 124 and associated scripts 126.

[0039] Resource loader and analyzer 130 loads external resources 132 associated with a path of each test resource 122 (step 206). In an example scenario, a test resource 122 may need external resources 132 (e.g., files) for the test resource 122 to execute properly. Such external resources 132 may be located at a server independent from testing engine 110 or may even be located on a computing device on which testing engine 110 is operating. Such external resources 132 may be loaded iteratively by resource loader and analyzer 130.

[0040] Resource loader and analyzer 130 also checks if there are any external test resource paths that cannot be resolved (step 208). When resource loader and analyzer

130 determines that a test resource path is not resolvable (step 208), resource loader and analyzer 130 and marker module 140 maintain the placement of the test resource 122, along with the un-resolvable path (step 210) and method 200 proceeds to step 212.

[0041] Returning to step 208, if resource loader and analyzer 130 determines that all external test resource paths have been resolved (step 208), resource loader and analyzer 130 analyzes each loaded test resource 122 to identify dynamically added dependencies 152 (step 212). As an example, dynamically added dependencies 152 may be scripts 126 or any other files that may be needed by a test resource 122 at runtime.

[0042] Resource loader and analyzer 130 checks if new dependencies 152 are identified in step 212 (step 214). If resource loader and analyzer 130 identifies new dependencies 152 (step 214), method 200 proceeds to step 206, where resource loader and analyzer 130 loads external resources 132 associated with a path of each test resource 122. If resource loader and analyzer 130 identifies no new dependencies (step 214), in-lining module 150 replaces each marker 142 with resources 122 and dependencies 152 that reference their respective marker 142 to generate in-lined test page 162 (step 216).

[0043] FIG. 2B is a flowchart illustrating step 214 of method 200 in greater detail, according to an embodiment. In an embodiment, resource loader and analyzer 130 identifies dynamically added dependencies 152 by evaluating scripts 126 using a script runtime engine 134 (e.g., JavaScript runtime) and parsing the test resource 122 (e.g., script) to identify dependency 'hints' (step 220). Such hints may include, for example, resource requirement calls.

[0044] Resource loader and analyzer 130 adds each new identified dependency 152 either after the top level parent resource 122 (i.e., mimicking a browser's runtime HTML load order), or before the parent resource 122 (i.e., mimicking the dynamic resolution load order) (step 222). Marker module 140 also provides each new dependency 152 with a reference to a parent marker 142 that is associated with the top level parent resource 122 (step 224).

[0045] In this way, by iteratively loading external resources 132 (or other dependencies) needed by a test resource 122 in advance of testing the test page 160, lesser time is required to execute the test because such loading of external resources 132 need not occur in real-time when the test is run.

[0046] Example Computer Embodiment

[0047] In an embodiment, the system and components of the embodiments described herein are implemented using one or more computers, such as example computer 302 shown in FIG. 3. For example, testing engine 110 can be implemented using computer(s) 302.

[0048] Computer 302 can be any commercially available and well known computer capable of performing the functions described herein.

[0049] Computer 302 includes one or more processors (also called central processing units, or CPUs), such as a processor 306. Processor 306 is connected to a communication infrastructure 304.

[0050] Computer 302 also includes a main or primary memory 308, such as random access memory (RAM). Primary memory 308 has stored therein control logic 368A (computer software), and data.

[0051] Computer 302 also includes one or more secondary storage devices 310. Secondary storage devices 310 include, for example, a hard disk drive 312 and/or a removable storage device or drive 314, as well as other types of storage devices, such as memory cards and memory sticks. Removable storage drive 314 represents a floppy disk drive, a magnetic tape drive, a compact disk drive, an optical storage device, tape backup, etc.

[0052] Removable storage drive 314 interacts with a removable storage unit 316. Removable storage unit 316 includes a computer useable or readable storage medium 364A having stored therein computer software 368B (control logic) and/or data. Removable storage unit 316 represents a floppy disk, magnetic tape, compact disk, DVD, optical storage disk, or any other computer data storage device. Removable storage drive 314 reads from and/or writes to removable storage unit 316 in a well known manner.

[0053] Computer 302 also includes input/output/display devices 366, such as monitors, keyboards, pointing devices, Bluetooth devices, etc.

[0054] Computer 302 further includes a communication or network interface 318. Network interface 318 enables computer 302 to communicate with remote devices. For example, network interface 318 allows computer 302 to communicate over communication networks or mediums 364B (representing a form of a computer useable

or readable medium), such as LANs, WANs, the Internet, etc. Network interface 318 may interface with remote sites or networks via wired or wireless connections.

[0055] Control logic 368C may be transmitted to and from computer 302 via communication medium 364B.

[0056] Any tangible apparatus or article of manufacture comprising a computer useable or readable medium having control logic (software) stored therein is referred to herein as a computer program product or program storage device. This includes, but is not limited to, computer 302, main memory 308, secondary storage devices 310 and removable storage unit 316. Such computer program products, having control logic stored therein that, when executed by one or more data processing devices, cause such data processing devices to operate as described herein, represent the embodiments.

[0057] Embodiments can work with software, hardware, and/or operating system implementations other than those described herein. Any software, hardware, and operating system implementations suitable for performing the functions described herein can be used. Embodiments are applicable to both a client and to a server or a combination of both.

[0058] The Summary and Abstract sections may set forth one or more but not all exemplary embodiments as contemplated by the inventor(s), and thus, are not intended to limit the present embodiments and the appended claims in any way.

[0059] The present embodiments have been described above with the aid of functional building blocks illustrating the implementation of specified functions and relationships thereof. The boundaries of these functional building blocks have been arbitrarily defined herein for the convenience of the description. Alternate boundaries can be defined so long as the specified functions and relationships thereof are appropriately performed.

[0060] The foregoing description of the specific embodiments will so fully reveal their general nature that others can, by applying knowledge within the skill of the art, readily modify and/or adapt for various applications such specific embodiments, without undue experimentation, without departing from their general concept. Therefore, such adaptations and modifications are intended to be within the meaning and range of equivalents of the disclosed embodiments, based on the teaching and guidance presented herein. It is to be understood that the phraseology or terminology herein is for the

purpose of description and not of limitation, such that the terminology or phraseology of the present specification is to be interpreted by the skilled artisan in light of the teachings and guidance.

[0061] The breadth and scope of the embodiments should not be limited by any of the above-described exemplary embodiments, but should be defined only in accordance with the following claims and their equivalents.

WHAT IS CLAIMED IS:

1. A computer implemented method to in-line script dependencies, the method comprising:

extracting test resources (122) addressed in language defining a test web page (160);

placing markers (142) identifying a location of each extracted test resource (122) within the language defining the test page (160);

iteratively loading external resources (132) associated with a path of each test resource (122);

analyzing each test resource (122) to identify one or more dynamically added dependencies (152); and

replacing each marker (142) with external resources (132) and dependencies (152) that reference their respective marker (142) to generate updated language defining an updated test page (160, 162),

wherein each step is performed using one or more processors (110).

2. The method of claim 1, further comprising:

adding each identified dependency (152) after or before a top level parent resource (122);

performing the analyzing and the adding until no new dependencies (152) are identified; and

providing each new dependency (152) with a reference to a parent marker (142) associated with the top level parent resource (122).

3. The method of claim 1 or 2, wherein the iteratively loading comprises maintaining the placement of each test resource (122) and its path, when the path cannot be resolved to load external resources (132).

4. The method of any of claims 1-3, wherein the analyzing comprises:

evaluating a script (126) using a script runtime engine (134); and

parsing the script (126) to identify references to resource dependencies (152).

5. A computer based system to in-line script dependencies, comprising:

one or more processors (306);

a test resource extractor (120) configured to extract test resources (122) addressed in language defining a test web page (160);

a marker module (140) configured to maintain markers (142) identifying a location of each test resource (122) within the language defining the test page (160);

a resource loader and analyzer (130) configured to iteratively load external resources (132) associated with a path of each test resource (122) and configured to analyze each test resource (122) to identify one or more dynamically added dependencies (152); and

an in-lining module (150) configured to replace each marker (142) with resources (132) and dependencies (152) that reference their respective marker (142) to generate updated language defining an updated test page (160, 162),

wherein the test resource extractor (120), the marker module (140), the resource loader and analyzer (130) and the in-lining module (150) are implemented on the one or more processors (306).

6. The system of claim 5, wherein the resource loader and analyzer (130) is further configured to:

add each identified dependency (152) after or before a top level parent resource (122); and

provide each new dependency (152) with a reference to a parent marker (142) associated with the top level parent resource (122).

7. The system of claim 5 or 6, wherein the marker module (140) is further configured to maintain the placement of each test resource (122) and its path, when the path cannot be resolved to load external resources.

8. The system of any of claims 5-7, wherein the resource loader and analyzer (130) is further configured to:

evaluate a script (126) using a script runtime engine (134); and

parse the script (126) to identify references to resource dependencies (152).

9. An article of manufacture including a non-transitory computer-readable medium having instructions stored thereon that, when executed by a computing device (302), cause the computing device (302) to perform operations comprising:

extracting test resources (122) addressed in language defining a test web page (160);

placing markers (142) identifying a location of each extracted test resource (122) within the language defining the test page (160);

iteratively loading external resources (132) associated with a path of each test resource (122);

analyzing each test resource (122) to identify one or more dynamically added dependencies (152); and

replacing each marker (142) with external resources (132) and dependencies (152) that reference their respective marker (142) to generate updated language defining an updated test page (160, 162).

10. The article of manufacture of claim 9, further comprising instructions that cause the computing device to perform operations comprising:

adding each identified dependency (152) after or before a top level parent resource (122);

performing the analyzing and the adding until no new dependencies (152) are identified; and

providing each new dependency (152) with a reference to a parent marker (142) associated with the top level parent resource (122).

11. The article of manufacture of claim 9 or 10, wherein the iteratively loading further comprises instructions that cause the computing device to perform operations comprising maintaining the placement of each test resource (122) and its path, when the path cannot be resolved to load external resources (132).

12. The article of manufacture of any of claims 9-11, wherein the analyzing further comprises

instructions that cause the computing device to perform operations comprising:

evaluating a script (126) using a script runtime engine (134); and

parsing the script (126) to identify references to resource dependencies (152).

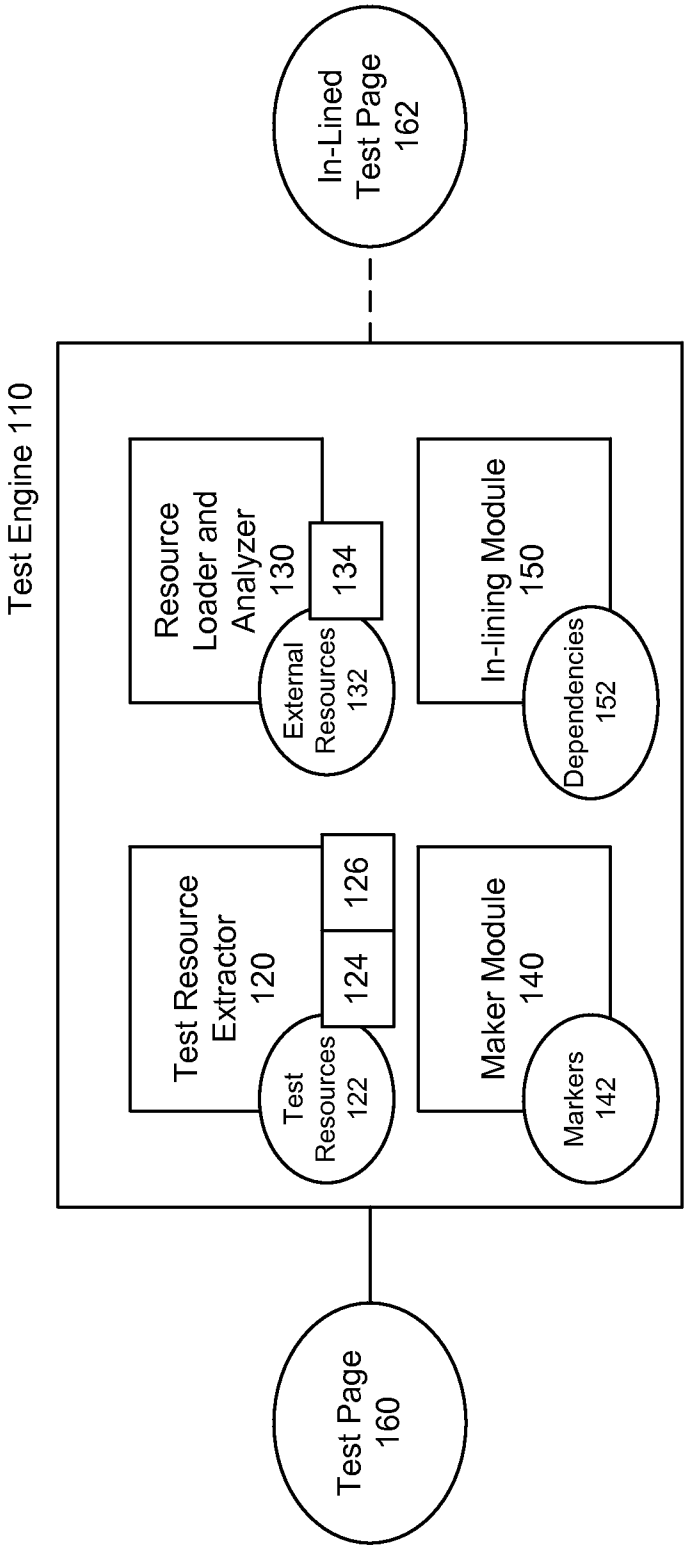
In-Lined Test Page 162
Test Engine 110
Resource Loader and Analyzer 130
134
External Resources 132
In-lining Module 150
Dependencies 152
Test Resource Extractor 120
126
124
Test Resources 122
Maker Module 140
Markers 142
Test Page 160

FIG. 1

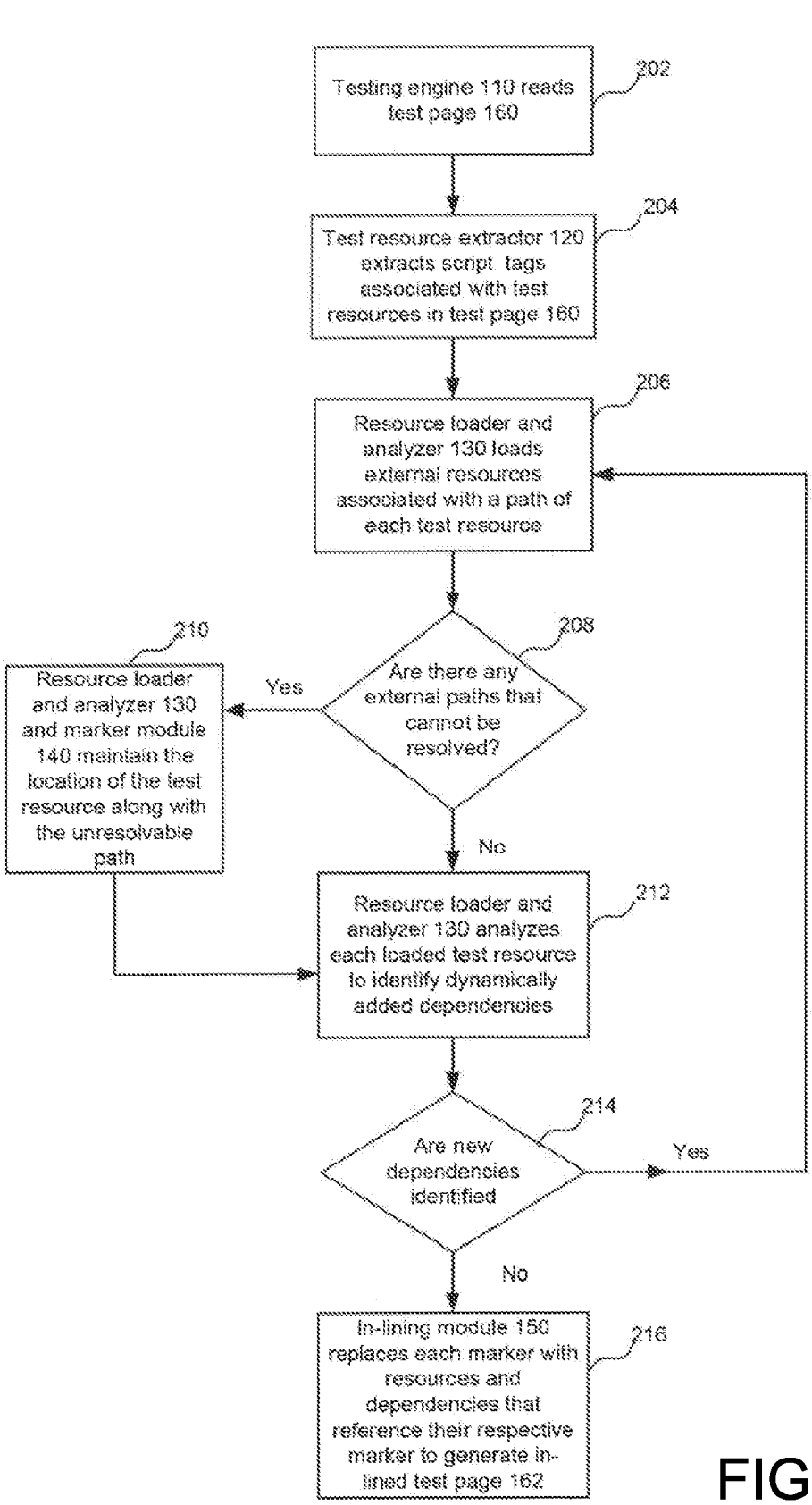
200
Testing engine 110 reads test page 160
202
Test resource extractor 120 extracts script tags associated with test resources in test page 160
204
Resource loader and analyzer 130 loads external resources associated with a path of each test resource
206
Are there any external paths that cannot be resolved?
208
Yes
Resource loader and analyzer 130 and marker module 140 maintain the location of the test resource along with the unresolvable path
210
No
Resource loader and analyzer 130 analyzes each loaded test resource to identify dynamically added dependencies
212
Are new dependencies identified
214
Yes
No
In-lining module 150 replaces each marker with resources and dependencies that reference their respective marker to generate in-lined test page 162
216

FIG. 2A

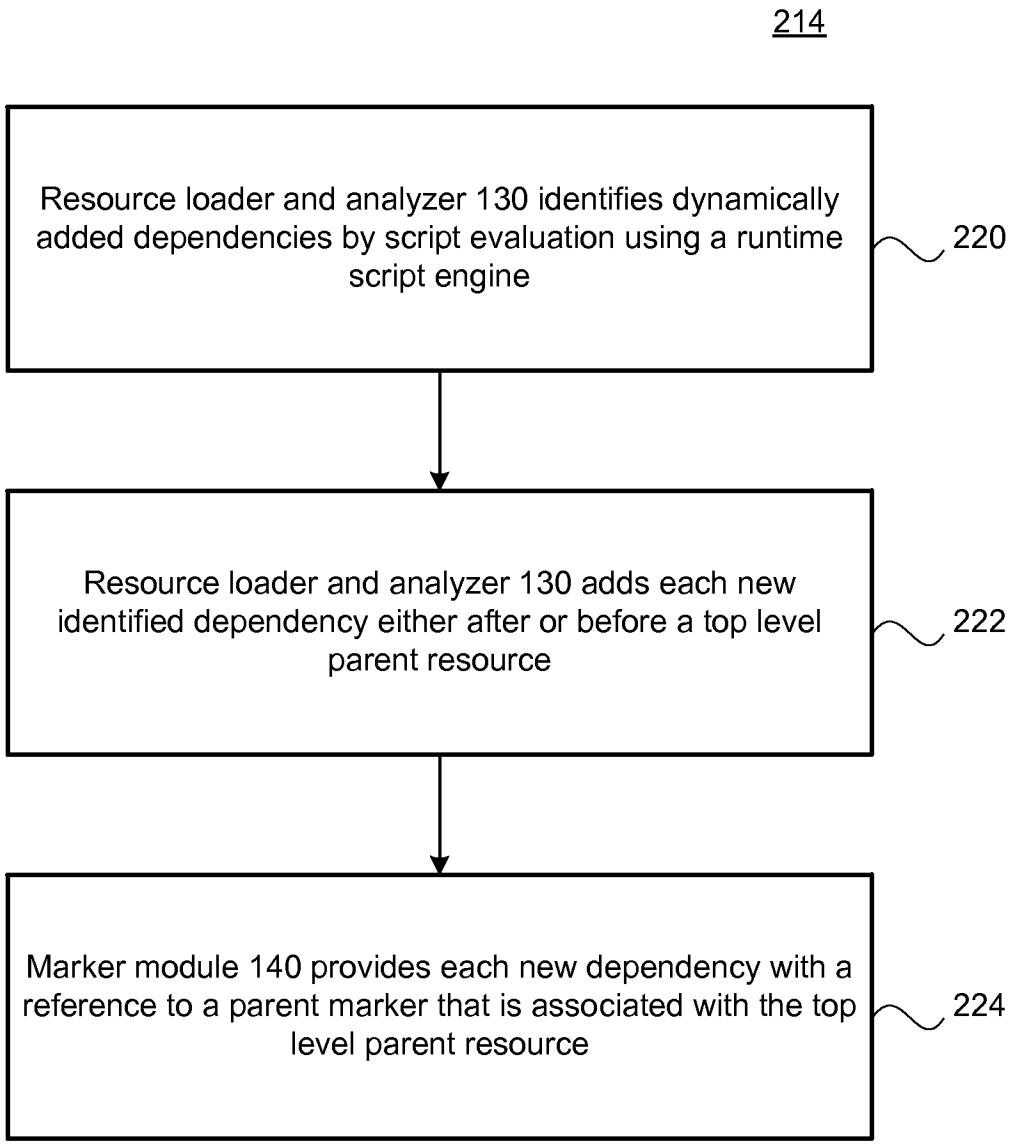
214
Resource loader and analyzer 130 identifies dynamically added dependencies by script evaluation using a runtime script engine
220
Resource loader and analyzer 130 adds each new identified dependency either after or before a top level parent resource
222
Marker module 140 provides each new dependency with a reference to a parent marker that is associated with the top level parent resource
224

FIG. 2B

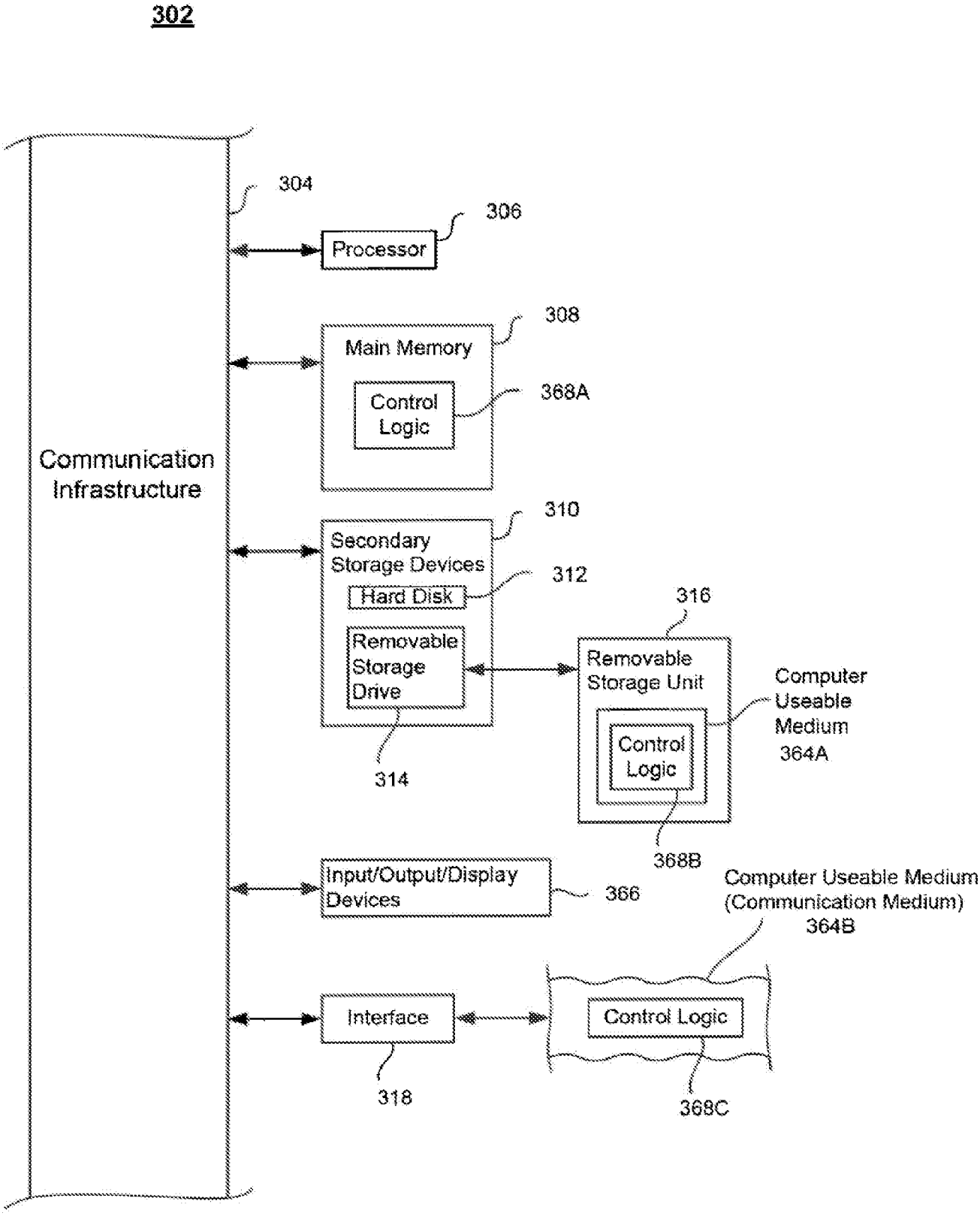
302
304
306
Processor
Communication Infrastructure
308
Main Memory
Control Logic
368A
310
Secondary Storage Devices
312
Hard Disk
316
Removable Storage Drive
Removable Storage Unit
Computer Useable Medium
364A
Control Logic
314
368B
Input/Output/Display Devices
366
Computer Useable Medium (Communication Medium)
364B
Interface
Control Logic
318
368C

FIG. 3

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2012/042127

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F11/36
ADD.

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)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, INSPEC, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X,P	Google developers: "Using ClosureBuilder", , 24 February 2012 (2012-02-24), XP002681918, Retrieved from the Internet: URL:https://developers.google.com/closure/library/docs/closurebuilder [retrieved on 2012-08-17] the whole document	1-12
A	US 2011/067018 A1 (KAWACHIYA KIYOKUNI [JP] ET AL) 17 March 2011 (2011-03-17) abstract; figure 4 paragraph [0022] - paragraph [0025]	1-12

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

☒ See patent family annex.

* 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
20 August 2012	30/08/2012

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Renault, Sophie

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2012/042127

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2011067018 A1	17-03-2011	JP 4806060 B2	02-11-2011
		JP 2011065220 A	31-03-2011
		US 2011067018 A1	17-03-2011