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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 August 2016 (04.08.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/122658 A1

(51) International Patent Classification:
G06F 9/54 (2006.01) *G06F 9/46* (2006.01)

(21) International Application Number:
PCT/US2015/013963

(22) International Filing Date:
30 January 2015 (30.01.2015)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP** [US/US]; 11445 Compaq Center Drive West, Houston, TX 77070 (US).

(72) Inventors: **HANSON, Thomas W.**; 3404 E. Harmony Rd., Fort Collins, Colorado 80528-9544 (US). **YORK, Justin**; 11445 Compaq Center Drive W., Houston, Texas 77070 (US). **THOMAS, Eric**; 3404 E. Harmony Rd., Fort Collins, Colorado 80528-9544 (US).

(74) Agents: **SHOOKMAN, Jeb A.** et al.; Hewlett Packard Enterprise, 3404 E. Harmony Road, Mail Stop 79, Fort Collins, CO 80528 (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, BN, 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, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, 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, ST, 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, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to the identity of the inventor (Rule 4.17(i))*

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

Published:

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

(54) Title: REQUEST PROCESSING

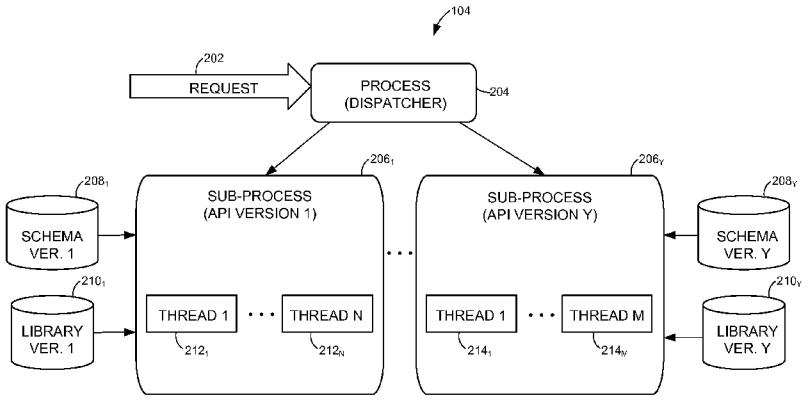

FIG. 2

(57) Abstract: One example of a system receives a request. The request includes an indication of an application programming interface version. The system determines whether a sub-process has been initialized for the application programming interface version. If needed, the system initializes a sub-process including preloading resources for the sub-process for the application programming interface version. The system executes the request using the sub-process for the application programming interface version by creating at least one thread of the sub-process which utilizes the preloaded resources.

WO 2016/122658 A1

REQUEST PROCESSING

Background

[0001] Datacenter systems support the management of devices interconnected via a computer network. The managed devices include management interfaces that allow the managed devices to be configured by a management platform. To allow a management interface and the management platform to properly exchange data, each implements a common Application Programming Interface (API).

Brief Description of the Drawings

[0002] Figure 1 is a block diagram illustrating one example of a system.

[0003] Figure 2 illustrates one example of request processing of the system of Figure 1.

[0004] Figure 3 is a block diagram illustrating one example of a processing system for implementing a management platform.

[0005] Figure 4 is a flow diagram illustrating one example of a method for request processing.

Detailed Description

[0006] In the following detailed description, reference is made to the accompanying drawings which form a part hereof, and in which is shown by way

of illustration specific examples in which the disclosure may be practiced. It is to be understood that other examples may be utilized and structural or logical changes may be made without departing from the scope of the present disclosure. The following detailed description, therefore, is not to be taken in a limiting sense, and the scope of the present disclosure is defined by the appended claims. It is to be understood that features of the various examples described herein may be combined, in part or whole, with each other, unless specifically noted otherwise.

[0007] To allow a management interface of a managed device and a software application of a management platform to properly exchange data, each implements a common Application Programming Interface (API). Over time, the API may evolve to new versions as new data and formats are developed and/or old data is deprecated. Management interfaces may or may not be updated to support new API versions and if they are updated, there may be variable time lags across vendors, model numbers, or particular instances of a device. Therefore, the software application determines which version of the API a particular management interface is using and adapts to communicate using the same API version. For the software application to be able to use the same API version of a particular management interface, the management platform contains or has access to an implementation of each version of the API to be supported.

[0008] Where the software application supports large numbers of discrete management interfaces, the application may use multiple parallel threads of execution to concurrently support multiple different API versions. The application may also support a very high throughput in terms of transactions per second, which requires very quick access to the code that implements the various API versions.

[0009] Examples of this software application contain all or part of the implementation of a particular version of an API in a dynamically loadable library and may also contain a set of associated schemas. The software application determines the version of the API being used by a management interface with which the application is to communicate, identifies the corresponding dynamic

library and schema to load, loads the dynamic library and schema, and then executes some or all of the contained code. The disclosed technique utilizes a tiered architecture of processes and threads leveraging the specific attributes of each to optimize the access time to the dynamic library code and schema and to optimize the resources consumed by the software application, the dynamic library code, and the schemas.

[0010] Figure 1 is a block diagram illustrating one example of a system 100. System 100 includes a management platform 102, a network 114, and a plurality of managed devices 118_1-118_X, where "X" is any suitable number of managed devices. Management platform 102 is communicatively coupled to each managed device 118_1-118_X through a communication path 112, network 114, and a communication path 116. Network 114 may be an Ethernet network, a fibre channel network, the Internet, another suitable network, or combination thereof.

[0011] Each managed device 118_1-118_X includes a management interface 120_1-120_X, respectively. Each management interface 120_1-120_X implements an API version 122_1-122_Y, respectively, where "Y" is any suitable number of API versions. More than one managed device may implement the same version of the API. Managed devices 118_1-118_X may include any system or device that is individually configurable, such as computer systems, storage systems, network devices (e.g., network switches), and/or individual boards or components within a system or device (e.g., a network adapter).

[0012] Management platform 102 includes at least one management software 106 and request processing 104. Request processing 104 is communicatively coupled to management software 106 through a communication path 108. Management software 106 includes a software application for communicating with and configuring each managed device 118_1-118_X via a respective management interface 120_1-120_X. Management software 106 sends processing requests for a managed device to request processing 104 including an indication of the API version of the managed device. Request processing 104 executes the requests by using a sub-process and threads of the sub-process corresponding to the API version of the managed device as will be described

below with reference to Figure 2. Once processing of a request is complete, request processing 104 returns a response to the request to management software 106.

[0013] Management platform 102 supports the multiple API versions 122_1-122_Y of management interfaces 120_1-120_X while providing high throughput. Request processing 104 uses separate resources such as code libraries and data files for each API version which are loaded once upon a first request for a particular API version. Keeping the libraries and data files separate from the common core of request processing significantly reduces the size of the request processing executable file and eliminates the need for updates to request processing to support new API versions. Updates are still used, however, to provide new libraries and schemas for new API versions.

[0014] In one specific example for an Adapter Management Interface Library (A-MIL), a management interface is on an interface card such as a network adapter. A-MIL implements a continuously running service that translates human readable Extensible Markup Language (XML) data into binary data formatted in ASN.1 and vice versa. In A-MIL, there are two sets of resources that are specific to the API version associated with an incoming request including the dynamic library containing the code and the schema files that define the XML interface. Obtaining access to the dynamic library and schema files causes a delay. The impact of the delay, however, is mitigated by leveraging the characteristics of processes and threads as described herein.

[0015] Figure 2 illustrates one example of request processing 104 of system 100 of Figure 1. An executing application runs in a context that contains a set of resources including virtual memory, file descriptors, and Input/Output (I/O) connections. The specific set of resources varies based on the operating system. This context is typically referred to as a “process” or “thread.” Generally, a process is an independent context while a thread shares many of the resources of a parent process with other threads under the same parent but executes independently of other threads. The specifics of process versus thread are operating system dependent. For example, Linux considers each to be a task with a particular set of attributes. These attributes may be selected in

many different combinations to produce a task that is neither process nor thread as those terms are commonly used.

[0016] For purposes of this disclosure, a process is considered to be completely independent of other processes and a thread is considered to share, at a minimum, virtual memory, I/O connections, and files with other threads in the same process. Any Linux or other operating system task with these attributes is considered to be equivalent to the processes and threads disclosed herein.

[0017] Request processing 104 uses a tiered architecture including a top level process as indicated at 204, a variable sized set of sub-processes 206_1-206_Y, and a variable number of threads within each sub-process such as 212_1-212_N. The top level process at 204 is a dispatcher which receives incoming requests 202, maintains the set of sub-processes 206_1-206_Y, and directs requests 202 to the appropriate sub-process 206_1-206_Y. In this example, the set of sub-processes includes sub-processes 206_1-206_Y, where "Y" is equal to the number of different API versions to be supported. Each sub-process 206_1-206_Y is used to execute requests for a particular API version. For example, sub-process 206_1 executes requests for API version 1 and sub-process 206_Y executes requests for API version Y. Each sub-process 206_1-206_Y preloads a schema 208_1-208_Y and a library 210_1-210_Y, respectively, for the API version supported by the sub-process. Each sub-process 206_1-206_Y creates threads to execute requests for the API version supported by the sub-process. For example, sub-process 206_1 creates threads 212_1-212_N, where "N" is any suitable number of threads for concurrently executing requests for sub-process 206_1. Sub-process 206_Y creates threads 214_1-214_M, where "M" is any suitable number of threads for concurrently executing requests for sub-process 206_Y.

[0018] In operation, upon receipt of an incoming request 202, dispatcher 204 examines the request to determine the API version with which the request complies and, if the API version is supported, selects the sub-process that implements that version, and forwards the request to that sub-process. If the API version has not yet been encountered, dispatcher 204 creates a new sub-process, adds the new sub-process to the set of available sub-processes, and forwards the request to the new sub-process. Each incoming request is

associated with an I/O connection (e.g., a socket file descriptor in Linux), that is used to send the response back to the source of the request. This I/O connection is forwarded to the sub-process along with the request.

[0019] When a sub-process is created by dispatcher 204, the sub-process is informed which API version to support. As part of the initialization process of the sub-process, the sub-process loads the correct library and schemas, configures memory, and may perform other steps to enable processing of requests for that API version. These other steps may include for example creating a global set of function pointers that address the functions supplied by the dynamic library. At this point, the cost (i.e. time delay) of the initialization of the sub-process has been incurred and requests can be processed without again incurring this cost. Once initialized, a sub-process waits for requests to be forwarded by dispatcher 204 and processes the requests as they arrive. For each request, the sub-process creates a new thread and provides the thread with the request and the associated I/O connection. Once the thread has been created, the sub-process is free to accept and process another request and may create a parallel thread for each subsequent request.

[0020] Each thread, when created, has available to it all of the resources that were initialized by its parent sub-process. This includes the library, function pointers, and schema files as well as the I/O connection to the requestor. The thread makes use of these resources to process the received request and send the response back to the requestor via the associated I/O connection. After processing of a request is complete, the thread exits (i.e. terminates). In this example, no idle threads are maintained at any time, thereby avoiding the overhead of a thread pool. In another example, a fixed or dynamic thread pool may be maintained by a sub-process rather than the sub-process creating threads for each request. There may be as many, or as few, concurrent threads as needed to handle the processing load at any point in time.

[0021] By using request processing 104, management platform 102 provides low and predictable response times, uses minimal resources, and is less complex than some alternative solutions. Specifically, in request processing 104, the library and schema files associated with a particular API version are

loaded once when a request associated with that API version is first received. All subsequent requests for the same API version re-use the preloaded resources. Since no new resources are loaded if the API version has already been encountered, there is little variability between requests other than that associated with the complexity of the request. Management platform 102 is compatible with a variable number of API versions to be supported, while the number of supported API versions operational at one time can vary from one or two versions to ten or more versions.

[0022] Figure 3 is a block diagram illustrating one example of a system 300. System 300 may include at least one computing device and may provide management platform 102 previously described and illustrated with reference to Figures 1 and 2. System 300 includes a processor 302 and a machine-readable storage medium 306. Processor 302 is communicatively coupled to machine-readable storage medium 306 through a communication path 304. Although the following description refers to a single processor and a single machine-readable storage medium, the description may also apply to a system with multiple processors and multiple machine-readable storage mediums. In such examples, the instructions may be distributed (e.g., stored) across multiple machine-readable storage mediums and the instructions may be distributed (e.g., executed by) across multiple processors.

[0023] Processor 302 includes one or more Central Processing Units (CPUs), microprocessors, and/or other suitable hardware devices for retrieval and execution of instructions stored in machine-readable storage medium 306. Processor 302 may fetch, decode, and execute instructions 308 to receive a request, instructions 310 to determine whether a sub-process has been initialized, instructions 312 to initialize a sub-process if a sub-process has not been initialized, and instructions 314 to execute the request. As an alternative or in addition to retrieving and executing instructions, processor 302 may include one or more electronic circuits comprising a number of electronic components for performing the functionality of one or more of the instructions in machine-readable storage medium 306. With respect to the executable instruction representations (e.g., boxes) described and illustrated herein, it

should be understood that part or all of the executable instructions and/or electronic circuits included within one box may, in alternate examples, be included in a different box illustrated in the figures or in a different box not shown.

[0024] Machine-readable storage medium 306 is a non-transitory storage medium and may be any suitable electronic, magnetic, optical, or other physical storage device that stores executable instructions. Thus, machine-readable storage medium 306 may be, for example, Random Access Memory (RAM), an Electrically-Erasable Programmable Read-Only Memory (EEPROM), a storage drive, an optical disc, and the like. Machine-readable storage medium 306 may be disposed within system 300, as illustrated in Figure 3. In this case, the executable instructions may be installed on system 300. Alternatively, machine-readable storage medium 306 may be a portable, external, or remote storage medium that allows system 300 to download the instructions from the portable/external/remote storage medium. In this case, the executable instructions may be part of an installation package.

[0025] Machine-readable storage medium 306 stores instructions to be executed by a processor (e.g., processor 302) including instructions 308, 310, 312, and 314 to process requests as previously described and illustrated with reference to Figures 1 and 2. Processor 302 may execute instructions 308 to receive a request, the request including an indication of an application programming interface version. Processor 302 may execute instructions 310 to determine whether a sub-process has been initialized for the application programming interface version.

[0026] Processor 302 may execute instructions 312 to initialize a sub-process including preloading resources for the sub-process for the application programming interface version in response to determining that a sub-process has not been initialized for the application programming interface version. In one example, the sub-process is initialized by loading a schema and a library for the application programming interface version. A plurality of sub-processes for a plurality of application programming interface versions may be maintained.

[0027] Processor 302 may execute instructions 314 to execute the request using the sub-process for the application programming interface version by creating at least one thread of the sub-process. In one example, the request is executed by the sub-process creating a thread pool to process the received request. The request may be associated with an input/output connection that is forwarded with the request to the sub-process for the application programming interface version.

[0028] Figure 4 is a flow diagram illustrating one example of a method 400 for request processing. At 402, a first request is received, the first request including an indication of a first application programming interface version of a first managed device and an associated input/output connection for the first request. At 404, it is determined whether a first sub-process has been initialized for the first application programming interface version. If a new sub-process is needed, at 406, a first sub-process is initialized including preloading resources for use by the first sub-process for processing requests associated with the first application programming interface version. This initialization is in response to determining that a sub-process has not been initialized for the first application programming interface version. In one example, initializing the first sub-process includes loading a schema and a library for the first application programming interface version. Initializing the first sub-process may also include creating a global set of function pointers that address functions supplied by the library. At 408, the first request and the associated input/output connection for the first request are forwarded to the first sub-process. At 410, the first request is executed using the first sub-process by creating at least one thread of the first sub-process.

[0029] In one example, the method further includes receiving a second request, the second request including an indication of a second application programming interface version of a second managed device and an associated input/output connection for the second request. The method includes determining whether a second sub-process has been initialized for the second application programming interface version. The method includes initializing a second sub-process including preloading resources for use by the second sub-process for processing requests associated with the second application programming

interface version in response to determining that a sub-process has not been initialized for the second application programming interface version. The method includes maintaining the first sub-process, forwarding the second request and the associated input/output connection for the second request to the second sub-process, and executing the second request using the second sub-process by creating at least one thread of the second sub-process.

[0030] The method may further include receiving a third request, the third request including an indication of the first application programming interface version and an associated input/output connection for the third request. The method includes forwarding the third request and the associated input/output connection for the third request to the first sub-process and executing the third request using the first sub-process by creating at least one new thread of the first sub-process or by using at least one existing thread of the first sub-process.

[0031] Although specific examples have been illustrated and described herein, a variety of alternate and/or equivalent implementations may be substituted for the specific examples shown and described without departing from the scope of the present disclosure. This application is intended to cover any adaptations or variations of the specific examples discussed herein. Therefore, it is intended that this disclosure be limited only by the claims and the equivalents thereof.

CLAIMS

1. A system comprising:

a management platform; and

a plurality of managed devices communicatively coupled to the management platform, each managed device comprising a management interface having an application programming interface version to communicate with the management platform,

wherein the management platform is to receive a request, the request including an indication of an application programming interface version of a managed device, determine whether a sub-process has been initialized for the application programming interface version of the managed device, initialize a sub-process including preloading resources for the sub-process for the application programming interface version of the managed device in response to determining that a sub-process has not been initialized for the application programming interface version of the managed device, and execute the request using the sub-process for the application programming interface version of the managed device by creating at least one thread of the sub-process.

2. The system of claim 1, wherein the management platform is to initialize the sub-process by loading a schema and a library for the application programming interface version of the managed device.

3. The system of claim 1, wherein the management platform is to terminate the at least one thread once the request is completed.

4. The system of claim 1, wherein the management platform maintains a thread pool for the sub-process to execute the request.

5. The system of claim 1, wherein the managed device is a network adapter.

6. A machine-readable storage medium encoded with instructions, the instructions executable by a processor of a system to cause the system to:

receive a request, the request including an indication of an application programming interface version;

determine whether a sub-process has been initialized for the application programming interface version;

initialize a sub-process including preloading resources for the sub-process for the application programming interface version in response to determining that a sub-process has not been initialized for the application programming interface version; and

execute the request using the sub-process for the application programming interface version by creating at least one thread of the sub-process.

7. The machine-readable storage medium of claim 6, wherein the sub-process is initialized by loading a schema and a library for the application programming interface version.

8. The machine-readable storage medium of claim 6, wherein the request is executed by the sub-process creating a thread pool to process the received request.

9. The machine-readable storage medium of claim 6, wherein the request is associated with an input/output connection that is forwarded with the request to the sub-process for the application programming interface version.

10. The machine-readable storage medium of claim 6, wherein the instructions are executable by the processor to further cause the system to:

maintain a plurality of sub-processes for a plurality of application programming interface versions.

11. A method comprising:

receiving a first request, the first request including an indication of a first application programming interface version of a first managed device and an associated input/output connection for the first request;

determining whether a first sub-process has been initialized for the first application programming interface version;

initializing a first sub-process including preloading resources for the first sub-process for the first application programming interface version in response to determining that a sub-process has not been initialized for the first application programming interface version;

forwarding the first request and the associated input/output connection for the first request to the first sub-process; and

executing the first request using the first sub-process by creating at least one thread of the first sub-process.

12. The method of claim 11, further comprising:

receiving a second request, the second request including an indication of a second application programming interface version of a second managed device and an associated input/output connection for the second request;

determining whether a second sub-process has been initialized for the second application programming interface version;

initializing a second sub-process including preloading resources for the second sub-process for the second application programming interface version in response to determining that a sub-process has not been initialized for the second application programming interface version;

maintaining the first sub-process;

forwarding the second request and the associated input/output connection for the second request to the second sub-process; and

executing the second request using the second sub-process by creating at least one thread of the second sub-process.

13. The method of claim 11, wherein initializing the first sub-process comprises loading a schema and a library for the first application programming interface version.

14. The method of claim 13, wherein initializing the first sub-process further comprises creating a global set of function pointers that address functions supplied by the library.

15. The method of claim 11, further comprising:

receiving a third request, the third request including an indication of the first application programming interface version and an associated input/output connection for the third request;

forwarding the third request and the associated input/output connection for the third request to the first sub-process; and

executing the third request using the first sub-process by creating at least one new thread of the first sub-process or by using at least one existing thread of the first sub-process.

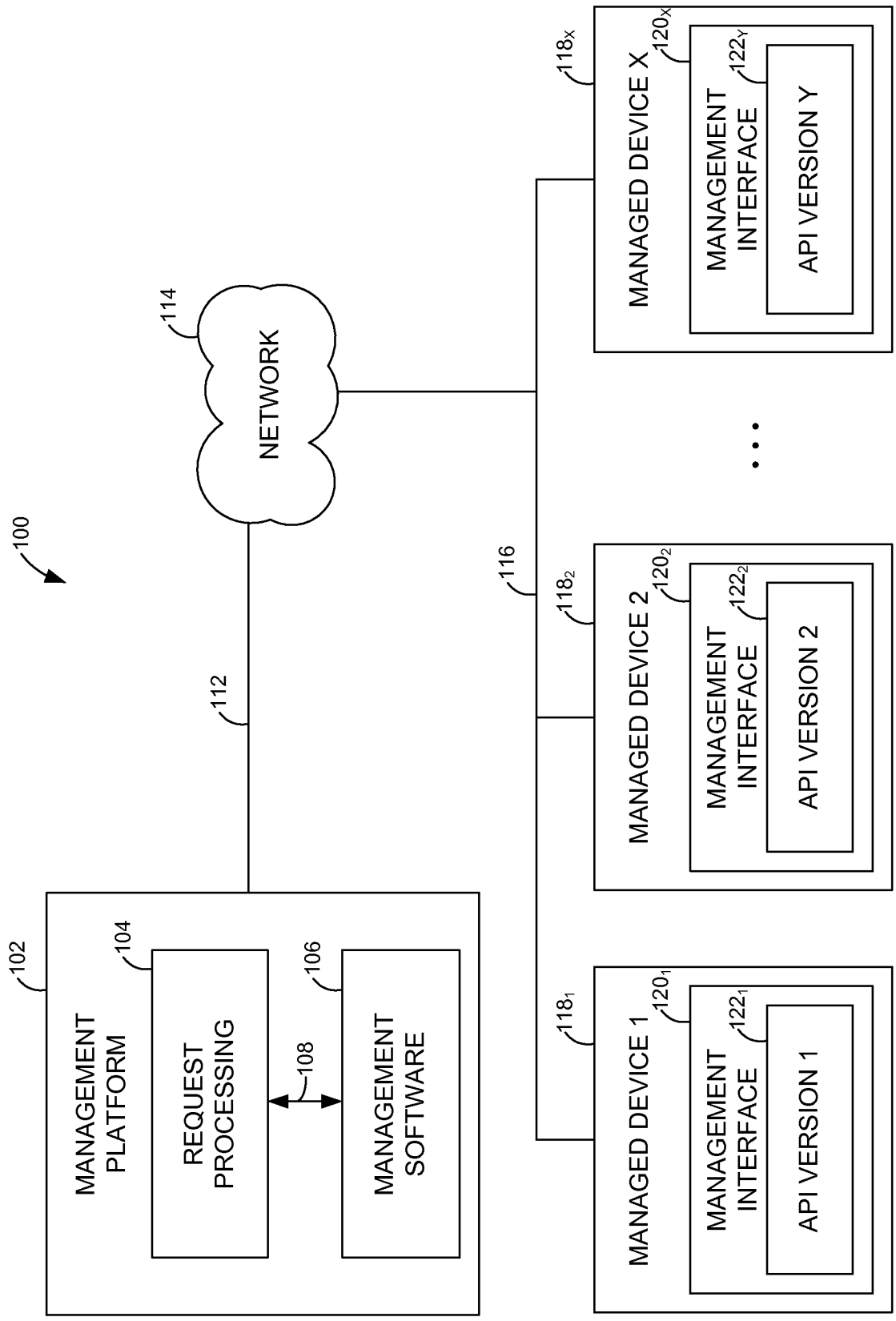
100
102
MANAGEMENT PLATFORM
104
REQUEST PROCESSING
106
MANAGEMENT SOFTWARE
108
112
114
NETWORK
116
118_1
MANAGED DEVICE 1
120_1
MANAGEMENT INTERFACE
122_1
API VERSION 1
118_2
MANAGED DEVICE 2
120_2
MANAGEMENT INTERFACE
122_2
API VERSION 2
118_X
MANAGED DEVICE X
120_X
MANAGEMENT INTERFACE
122_Y
API VERSION Y

FIG. 1

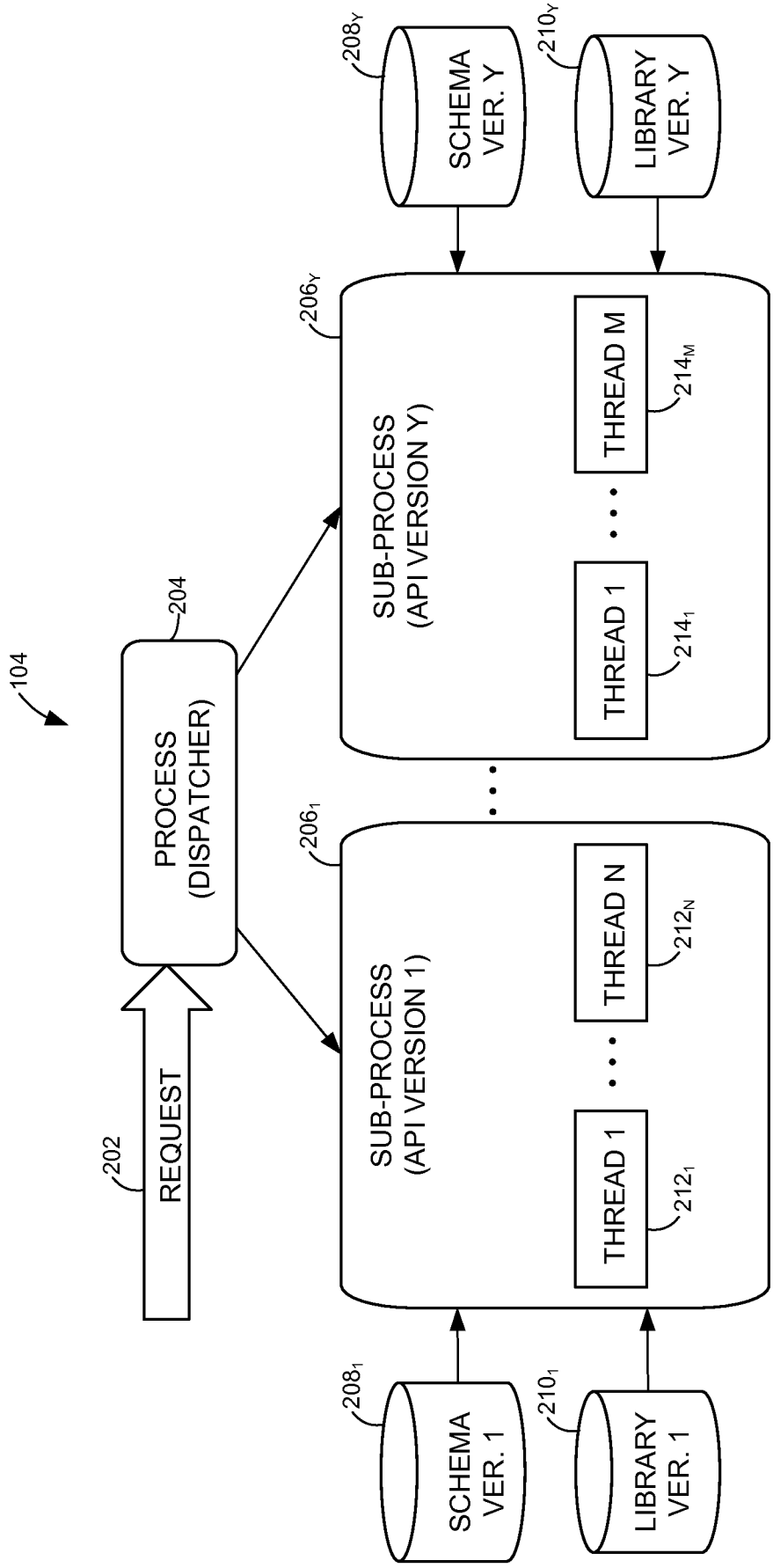
104
PROCESS (DISPATCHER)
204
REQUEST
202
SUB-PROCESS (API VERSION 1)
206$_1$
THREAD 1
212$_1$
THREAD N
212$_N$
SUB-PROCESS (API VERSION Y)
206$_Y$
THREAD 1
214$_1$
THREAD M
214$_M$
SCHEMA VER. 1
208$_1$
LIBRARY VER. 1
210$_1$
SCHEMA VER. Y
208$_Y$
LIBRARY VER. Y
210$_Y$

FIG. 2

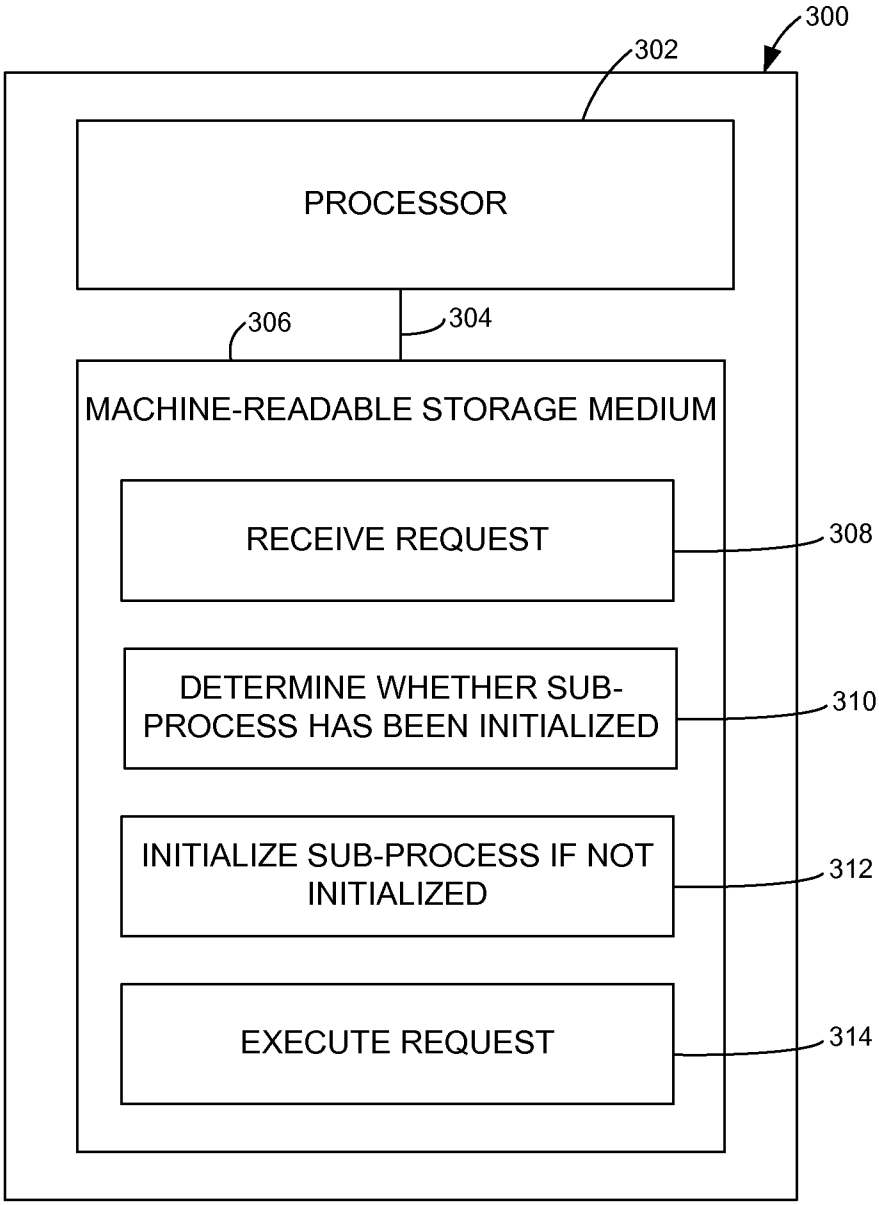
300
302
PROCESSOR
306
304
MACHINE-READABLE STORAGE MEDIUM
RECEIVE REQUEST
308
DETERMINE WHETHER SUB-PROCESS HAS BEEN INITIALIZED
310
INITIALIZE SUB-PROCESS IF NOT INITIALIZED
312
EXECUTE REQUEST
314

FIG. 3

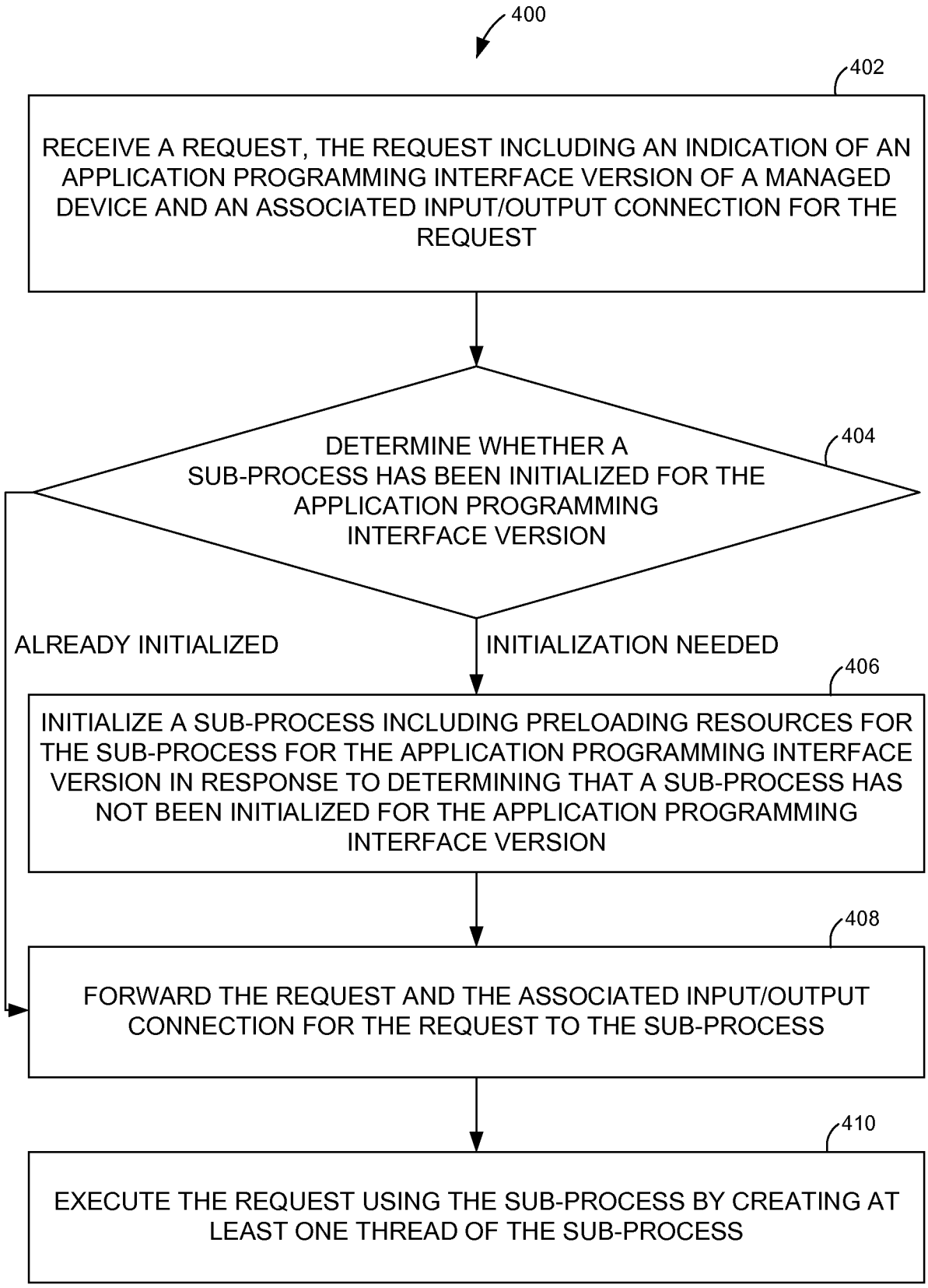
400
402
RECEIVE A REQUEST, THE REQUEST INCLUDING AN INDICATION OF AN APPLICATION PROGRAMMING INTERFACE VERSION OF A MANAGED DEVICE AND AN ASSOCIATED INPUT/OUTPUT CONNECTION FOR THE REQUEST
404
DETERMINE WHETHER A SUB-PROCESS HAS BEEN INITIALIZED FOR THE APPLICATION PROGRAMMING INTERFACE VERSION
ALREADY INITIALIZED
INITIALIZATION NEEDED
406
INITIALIZE A SUB-PROCESS INCLUDING PRELOADING RESOURCES FOR THE SUB-PROCESS FOR THE APPLICATION PROGRAMMING INTERFACE VERSION IN RESPONSE TO DETERMINING THAT A SUB-PROCESS HAS NOT BEEN INITIALIZED FOR THE APPLICATION PROGRAMMING INTERFACE VERSION
408
FORWARD THE REQUEST AND THE ASSOCIATED INPUT/OUTPUT CONNECTION FOR THE REQUEST TO THE SUB-PROCESS
410
EXECUTE THE REQUEST USING THE SUB-PROCESS BY CREATING AT LEAST ONE THREAD OF THE SUB-PROCESS

FIG. 4

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/013963

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/54(2006.01)i, G06F 9/46(2006.01)i

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 9/54; G06F 15/16; G06F 9/44; G06F 17/30; G06F 9/46; G06F 9/50

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
eKOMPASS(KIPO internal) & Keywords: request, processing, application programming interface

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2011-0087731 A1 (WONG, L. et al.) 14 April 2011 See abstract, paragraphs [0076]-[0084], and fig. 8.	1-15
A	US 2007-0101335 A1 (NAGAMPALLI, N. R. S.S. et al.) 03 May 2007 See abstract, paragraphs [0025]-[0035], and figs. 2 and 3.	1-15
A	US 2010-0131956 A1 (DREPPER, U.) 27 May 2010 See abstract, paragraphs [0032]-[0041], and figs. 3A-3C.	1-15
A	US 2010-0162271 A1 (ARIMILLI, R. K. et al.) 24 June 2010 See abstract, paragraphs [0065]-[0077], and fig. 6.	1-15
A	US 2011-0093870 A1 (SHARMA, R. et al.) 21 April 2011 See abstract, paragraphs [0047]-[0053], and figs. 4 and 5.	1-15

☐ 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
19 October 2015 (19.10.2015)	**19 October 2015 (19.10.2015)**

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea
Facsimile No. +82-42-472-7140

Authorized officer
YU, Jintae
Telephone No. +82-42-481-8530

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
PCT/US2015/013963

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2011-0087731 A1	14/04/2011	CN 102783129 A	14/11/2012
		CN 102783129 B	22/04/2015
		EP 2486722 A1	15/08/2012
		EP 2486722 A4	06/05/2015
		KR 10-1422372 B1	22/07/2014
		KR 10-2012-0068966 A	27/06/2012
		RU 2012116596 A	20/11/2013
		RU 2534953 C2	10/12/2014
		US 2015-215397 A1	30/07/2015
		US 9043401 B2	26/05/2015
		WO 2011-043883 A1	14/04/2011
US 2007-0101335 A1	03/05/2007	US 7979865 B2	12/07/2011
US 2010-0131956 A1	27/05/2010	US 9063783 B2	23/06/2015
US 2010-0162271 A1	24/06/2010	US 8370855 B2	05/02/2013
US 2011-0093870 A1	21/04/2011	CN 102576309 A	11/07/2012
		CN 102576309 B	15/04/2015
		EP 2491489 A1	29/08/2012
		JP 2013-508833 A	07/03/2013
		JP 5669851 B2	18/02/2015
		US 8635632 B2	21/01/2014
		WO 2011-047912 A1	28/04/2011