



- (51) **International Patent Classification:**
G06F 9/445 (2006.01) *G06F 9/44* (2006.01)
- (21) **International Application Number:**
PCT/US2014/010691
- (22) **International Filing Date:**
8 January 2014 (08.01.2014)
- (25) **Filing Language:** English
- (26) **Publication Language:** English
- (71) **Applicant:** HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P. [US/US]; 11445 Compaq Center Drive W., Houston, Texas 77070 (US).
- (72) **Inventors:** KHARBANDA, Deepti; Sy. No. 192, Whitefield Road, Mahadevapura Post, Karnataka, Bangalore 560 048 (IN). RANGANATHAIAH, Mahesh Tavarekere; Sy. No. 192, Whitefield Road, Mahadevapura Post, Karnataka, Bangalore 560 048 (IN). PADLIA, Abhay; Sy.

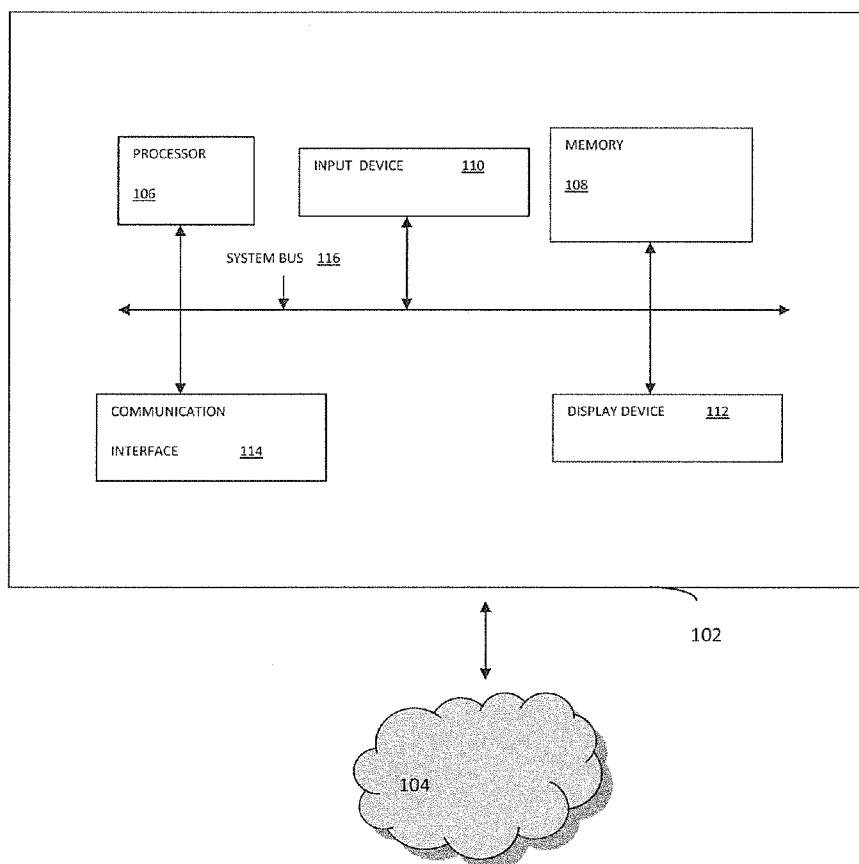
No. 192, Whitefield Road, Mahadevapura Post, Karnataka, Bangalore 560 048 (IN).

- (74) **Agents:** WEBB, Steven, L. et al.; Hewlett-Packard Company, Intellectual Property Administration, 3404 E. Harmony Road, Mail Stop 35, Fort Collins, Colorado 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, LT, 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.

[Continued on next page]

- (54) **Title:** DYNAMICALLY APPLYING A SOFTWARE PATCH TO A COMPUTER PROGRAM



- (57) **Abstract:** Provided is a method of dynamically applying a software patch to a computer program. A current version of a computer program is determined. Upon determination, compared to the current version, increasingly higher versions of the software patch are sequentially applied to the computer program, wherein the higher versions of the software patch are provided in form of a package.



(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, 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))*

-1-

DYNAMICALLY APPLYING A SOFTWARE PATCH TO A COMPUTER PROGRAM

BACKGROUND

[001] Computer programs installed in a computer system are typically updated by applying a software patch (or software patches). A software patch upon installation may provide protection against security threats, fix bugs, or generally improve the usability or performance of a computer program. Since a vulnerable computer program could cause a loss of valuable data, confidential information, downtime etc., software patches play an important role in keeping a computer program (and computer system) safe and up-to-date.

-2-

BRIEF DESCRIPTION OF THE DRAWINGS

[002] For a better understanding of the solution, embodiments will now be described, purely by way of example, with reference to the accompanying drawings, in which:

[003] FIG. 1 illustrates a system for dynamically applying a software patch to a computer program, according to an example.

[004] FIG. 2 illustrates a method of dynamically applying a software patch to a computer program, according to an example.

[005] FIG. 3 illustrates a system for dynamically applying a software patch to a computer program, according to an example.

-3-

DETAILED DESCRIPTION OF THE INVENTION

[006] Software developers often release a series of security and feature patches to address various requirements or bugs found in the operating environment of a customer's computer systems. Most of these patch update mechanisms replace a defective binary on a target system and require a reboot or a restart of the affected computer application to bring the fix into effect. This brings the system up-to-date to the latest version of the target binary. These mechanisms are known as static patching. Thus, in static patching, a software update typically requires stopping and restarting an application. However, there are many scenarios where computer systems cannot afford to halt service. Dynamic or online patching allows a computer program of a running system to be dynamically updated with new code and data with-out requiring a reboot or downtime. Dynamic patching permits programs to be updated while they run.

[007] Present online patching solutions provide dynamic patches which limit the patching scope to the current patch version. These patches update a target binary from a previous patch level to the current patch level. For example, a target system (presently at patch level 0) may directly be updated to the current patch level (for example, patch level 3). However, there may be a scenario where a target system (presently at patch level 0) may need to be updated to a higher patch level (for example, patch 1) before the latest online patch can be applied (for example, patch 2) to it. When dealing with a complex system setup and multiple online patches, this scenario can pose a big challenge to an administrator to bring the system up-to-date with the latest dynamic patch fix version, without opting for a reboot or application restart.

[008] Proposed solution provides a method of dynamically applying a software patch to a computer program. Proposed solution packages multi-versions of software patches into a package for delivering an online patching fix. A current

-4-

version of a computer program in a computer system is determined and, upon such determination, increasingly higher versions of the software patch are sequentially applied to the computer program until the current version. Proposed solution provides a dynamically loadable module which carries only the changes made in each patch level from its previous. The module is dynamically loaded into the memory of the computer system thereby providing the changed function for the patching infrastructure to access. On successful patching, the changed function in the newly loaded module is accessed thereon instead of its older version in the memory. Thus, the patches are applied in such a way that an administrator (or another user) can apply the patches from any lower patch level that supports dynamic patching. Further, the administrator can also roll back a patch to a desired previous patch level.

[009] FIG. 1 illustrates a system 100 for dynamically applying a software patch to a computer program, according to an example. System includes computer system 102 and computer network 104.

[0010] Computer system 102 may include, by way of example, a computer server, desktop computer, notebook computer, tablet computer, mobile phone, personal digital assistant (PDA), or the like. Computer system 102 may be a computer server, desktop computer, notebook computer, tablet computer, mobile phone, personal digital assistant (PDA), or the like. Computer system 202 may include processor 106, memory 108, input device 110, display device 112, and a communication interface 114. The components of the computer system 102 may be coupled together through system bus 116.

[0011] Processor 106 may include any type of processor, microprocessor, or processing logic that interprets and executes instructions.

-5-

[0012] Memory 108 may include a random access memory (RAM) or another type of dynamic storage device that may store information and instructions non-transitorily for execution by processor 106. For example, memory 108 can be SDRAM (Synchronous DRAM), DDR (Double Data Rate SDRAM), Rambus DRAM (RDRAM), Rambus RAM, etc. or storage memory media, such as, a floppy disk, a hard disk, a CD-ROM, a DVD, a pen drive, etc. In an example, memory 108 includes a computer program (or computer programs). The computer program may include an operating system, application software (machine-executable instructions), library and the like. Further, a computer program installed on computer system may be a first version of the program (i.e. patch level 0) or a subsequent higher version of the program (such as patch level 1, 2, 3, and so and so forth).

[0013] Input device 110 may include a keyboard, a mouse, a touch-screen, or other input device. Display device 112 may include a liquid crystal display (LCD), a light-emitting diode (LED) display, a plasma display panel, a television, a computer monitor, and the like.

[0014] Communication interface 114 may include any transceiver-like mechanism that enables computing device 102 to communicate with other devices and/or systems via a communication link. Communication interface 114 may be a software program, a hardware, a firmware, or any combination thereof. Communication interface 114 may provide communication through the use of either or both physical and wireless communication links. To provide a few non-limiting examples, communication interface 114 may be an Ethernet card, a modem, an integrated services digital network ("ISDN") card, etc.

[0015] In an implementation, user computer system is communicatively coupled to computer network 104, which may be wired or wireless (for example, a telecom network). Computer network 104 may be a public network such as the

-6-

Internet, or a private network such as an intranet. Computer network 104 may host a computer patch (or computer patches) which may be downloaded from computer network 104 by computer system 102. In an example, various computer patch versions that may be applied to a computer program are bundled together in form of a package which is downloadable from computer network as well. In an example, a computer patch includes a Dynamically Loadable Kernel Module (DLKM). A DLKM module is a modularly-packaged module with the capability to be dynamically loaded into a running kernel. In the kernel configuration context, this means that the DLKM module uses its own master and system files and contains the module wrapper code and additional data structures that provide the dynamic loading and unloading ability. In an example, a DLKM module includes a current version of the fix and meta-data. In an example, the fix includes a current version of the changed functions. In an example, the meta-data includes a list of functions that are being patched in a target computer system, any other data (such as patch level names, etc.) that a patching infrastructure (explained below) may need for carrying out the process, and information regarding the sequence in which the higher versions of a software patch need to be installed on a target system.

[0016] A Dynamically Loadable Kernel Modules (DLKM) Infrastructure may be needed for implementing a DLKM module. A Dynamically Loadable Kernel Modules (DLKM) Infrastructure is an operating system feature that allows “DLKM Enabled” kernel modules to be dynamically loaded into, or unloaded from, the operating system kernel without having to re-link the entire kernel or reboot the system. The DLKM feature not only provides the infrastructure to load kernel modules into a running system, but it also allows a kernel module to be statically linked when rebuilding the kernel. A DLKM infrastructure specifies that a kernel module must be packaged modularly with at least its own master and system files, its own mod.o object file that implements only that module and the mod.o object file must contain the Module Wrapper code. In an example, a Dynamically Loadable Kernel Modules (DLKM) Infrastructure is present on computer system 102.

-7-

[0017] It would be appreciated that the components depicted in FIG. 1 are for the purpose of illustration only and the actual components (including their number) may vary depending on the computing architecture deployed for implementation of the proposed solution.

[0018] FIG. 2 illustrates a method of dynamically applying a software patch to a computer program, according to an example.

[0019] At block 202, a current version of a computer program present (or installed) in a computer system is determined. In an example, a current version of a computer program installed in a computer system is automatically determined by a patching infrastructure (such as a Dynamically Loadable Kernel Modules (DLKM) Infrastructure) which may be present on a computer network (such as the Internet). In this case, the computer system which is hosting a current version of the computer program may be communicatively coupled to said computer network.

[0020] At block 204, upon determination of the current version of a computer program present in a computer system, increasingly higher versions of a software patch are sequentially applied to the computer program. In other words, by way of an illustration, if the current version of a computer program in a computer system, for instance, is at patch level 0 then subsequent higher versions of a software patch (such as patch level 1, 2, 3, etc.) meant for applying onto the computer program (for instance, to bring it up-to-date to the latest version) would be applied in a sequential manner beginning with immediately succeeding patch level i.e., in this case, patch level 1 -> patch level 2 -> patch level 3, and so and so forth. Increasingly higher versions of a software patch may be sequentially applied to the computer program till it gets updated with the latest version of the software patch.

-8-

[0021] As mentioned earlier, a software patch may include a Dynamically Loadable Kernel Module (DLKM) that may include a current version a current version of the changed functions and meta-data. For instance, a Dynamically Loadable Kernel Module (DLKM) for a dynamic software patch level 1 is created by encapsulating only the delta of changed or fixed code of patch level 0 and patch level 1 into a module format. In an example, said DLKM module may be named as <patch_1><DLKM_name>. Thus, the dynamic patch level 1 contains <patch_1><DLKM_name> and relevant metadata information.

[0022] Similarly, a DLKM for dynamic software patch level 2 is created by encapsulating only the delta of changed or fixed code of patch level 1 and patch level 2 into a module format. Such DLKM module may be named as <patch_2><DLKM_name>. The dynamic patch level 2 now contains <patch_1><DLKM_name>, <patch_2><DLKM_name> and relevant metadata information.

[0023] In a like manner, further DLKMs for higher software patch levels may be created wherein each of the higher versions of the software patch includes updates or changes specific to each of the higher versions of the software patch along with updates from all preceding versions of the software patch.

[0024] In an implementation, the higher versions of the software patch are bundled together in form of a package, which may be present on a computer network (such as the Internet). In an example, the higher versions of the software patch are sequentially extracted and applied from the package.

[0025] In an implementation, various versions (or levels) of a software patch applied to a computer program may be removed or un-installed in a reverse chronological order of installation. This helps an administrator in removing dynamically installed changes in step by step manner.

[0026] It may be noted that a system reboot or restart of a computer program is not needed while sequentially increasingly higher versions of a software patch are applied to a computer program. Likewise, a reboot or restart of the computer system is not needed while sequentially un-installing the higher versions of a software patch in a reverse chronological order of installation.

[0027] FIG. 3 illustrates an example system 300 for synchronizing a cloud service template with an infrastructure template, according to an example. System 300 includes a processor 302 that executes instructions in a memory 304. The memory 304 includes a patch module 306 to determine a current version of a computer program, and sequentially apply, compared to the current version, increasingly higher versions of the software patch to the computer program, wherein each of the higher versions of the software patch includes updates specific to each of the higher versions of the software patch along with updates from all preceding versions of the software patch.

[0028] Proposed solution provides a host of advantages by packaging a software patch with successive patch levels into a bundle. For instance, it relieves the administrator from the responsibility of sorting out patch interdependencies between different patch levels. Additionally, in the event, the customer systems are at different patch levels, a legacy cumulative patch would limit the target levels on which the patch could be installed. For example, if a customer system is at patch level 1 and a dynamic patch containing cumulative fixes from patch level 0 to patch level 2 is provided, the patch cannot be applied without tweaks or not applied at all. Further, the proposed solution with logically segregated fixes provides granularity to uninstall or rollback patches in reverse order of installation to patch level 0 or restore to a desired previous patch level on target system. An administrator is not forced to remove an entire patch bundle. The method also simplifies the patch

-10-

creation process since a new patch is created with only changes from the previous patch.

[0029] For the purpose of simplicity of explanation, the example method of FIG. 2 is shown as executing serially, however it is to be understood and appreciated that the present and other examples are not limited by the illustrated order. The example systems of FIGS. 1 and 3, and method of FIG. 2 may be implemented in the form of a computer program product including computer-executable instructions, such as program code, which may be run on any suitable computing environment in conjunction with a suitable operating system such as, but not limited to, Microsoft Windows, Linux or UNIX operating system. Embodiments within the scope of the present solution may also include program products comprising non-transitory computer-readable media for carrying or having computer-executable instructions or data structures stored thereon. Such computer-readable media can be any available media that can be accessed by a general purpose or special purpose computer. By way of example, such computer-readable media can comprise RAM, ROM, EPROM, EEPROM, CD-ROM, magnetic disk storage or other storage devices, or any other medium which can be used to carry or store desired program code in the form of computer-executable instructions and which can be accessed by a general purpose or special purpose computer. The computer readable instructions can also be accessed from memory and be executed by a processor.

[0030] For the sake of clarity, the term "module", as used in this document, may mean to include a software component, a hardware component or a combination thereof. A module may include, by way of example, components, such as software components, processes, tasks, co-routines, functions, attributes, procedures, drivers, firmware, data, databases, data structures, Application Specific Integrated Circuits (ASIC) and other computing devices. The module may

-11-

reside on a volatile or non-volatile storage medium and configured to interact with a processor of a computer system.

[0031] It would be appreciated that the system components depicted in the illustrated figures are for the purpose of illustration only and the actual components may vary depending on the computing system and architecture deployed for implementation of the present solution. The various components described above may be hosted on a single computing system or multiple computer systems, including servers, connected together through suitable means.

[0032] It should be noted that the above-described embodiment of the present solution is for the purpose of illustration only. Although the solution has been described in conjunction with a specific embodiment thereof, numerous modifications are possible without materially departing from the teachings and advantages of the subject matter described herein. Other substitutions, modifications and changes may be made without departing from the spirit of the present solution.

-12-

CLAIMS:

1. A method of dynamically applying a software patch to a computer program,
5 comprising:

determining a current version of the computer program on a computer system; and

sequentially applying, compared to the current version, increasingly higher versions of the software patch to the computer program, wherein the higher
10 versions of the software patch are provided in form of a package.

2. The method of claim 1, wherein a system reboot or restart of the computer program is not needed while sequentially applying increasingly higher versions of the software patch to the computer program.

3. The method of claim 1, wherein each of the higher versions of the software patch includes updates specific to each of the higher versions of the software patch along with updates from all preceding versions of the software patch.

4. The method of claim 1, further comprising applying a latest version of the software patch to the computer program.

5. The method of claim 1, further comprising sequentially extracting the higher versions of the software patch from the package.

6. The method of claim 1, further comprising sequentially uninstalling the increasingly higher versions of the software patch in reverse chronological order to desired level.

-13-

7. The method of claim 1, wherein the package includes meta-data information describing installation sequence of the higher versions of the software patch.

8. The method of claim 1, further comprising packaging each of the higher versions of the software patch into a module containing updates pertaining to a particular version along with updates from all preceding versions of the software patch.

9. A system, comprising:

a computer system, wherein the computer system includes a version of a computer program; and

a patch module to determine the version of the computer program and sequentially installing, compared to the current version, increasingly higher versions of a software patch, wherein a system reboot or restart of the computer program is not needed while sequentially applying increasingly higher versions of the software patch to the computer program.

10. The system of claim 9, wherein each of the higher versions of the software patch includes a module containing updates specific to each of the higher versions of the software patch along with updates from all preceding versions of the software patch.

11. The system of claim 10, wherein the module is a Dynamically Loadable Kernel Module (DLKM).

12. The system of claim 9, wherein the higher versions of the software patch are provided in form of an online package.

13. The system of claim 9, wherein the software patch is downloaded from the internet.

-14-

14. The system of claim 9, wherein the software patch is a new or an updated binary file.

15. A non-transitory processor readable medium, the non-transitory processor
5 readable medium comprising machine executable instructions, the machine
executable instructions when executed by a processor causes the processor to:
determine a current version of a computer program; and
sequentially apply, compared to the current version, increasingly higher
versions of the software patch to the computer program, wherein each of the higher
10 versions of the software patch includes updates specific to each of the higher
versions of the software patch along with updates from all preceding versions of
the software patch.

1/3

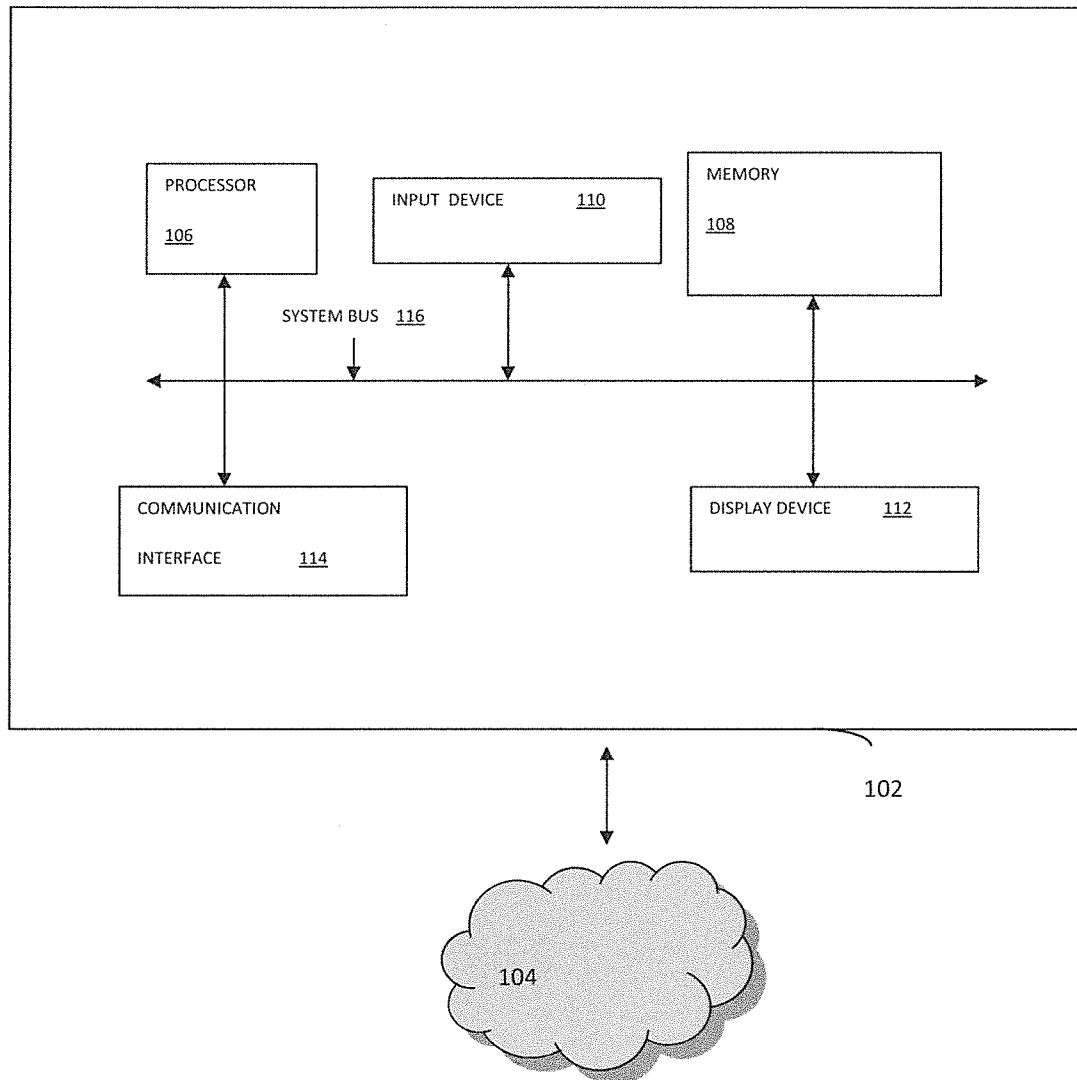


FIG. 1

2/3

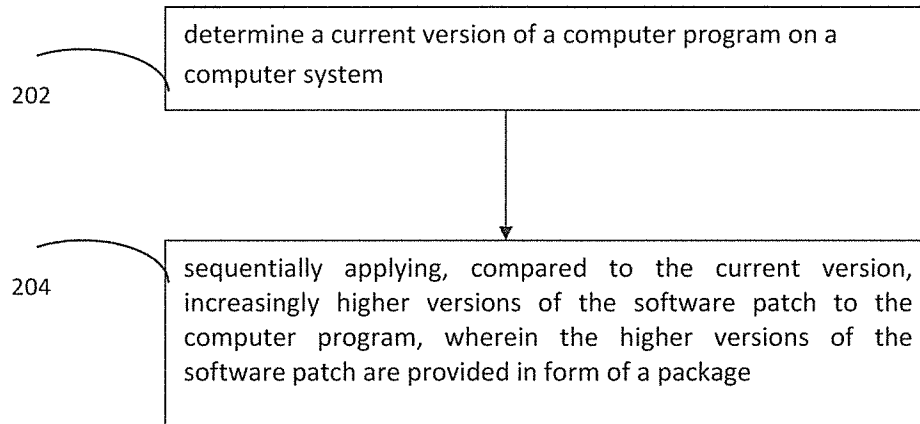


FIG. 2

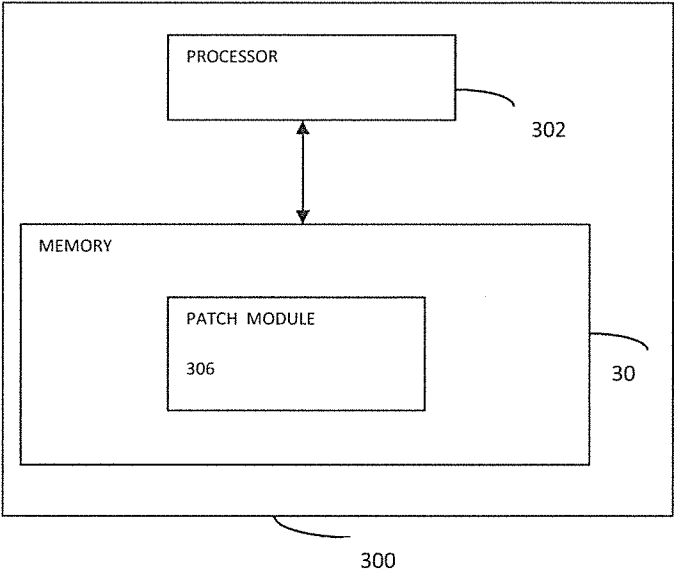


FIG. 3

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2014/010691**A. CLASSIFICATION OF SUBJECT MATTER****G06F 9/445(2006.01)i, G06F 9/44(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 9/445; G06F 9/44; G06F 11/00

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: software, patch, install, package

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 2002-0174422 A1 (KELLEY, M. L. et al.) 21 November 2002 See abstract and paragraphs [0008], [0050], [0061], [0164]-[0168].	1-15
Y	US 2004-0181790 A1 (HERRICK, J. W.) 16 September 2004 See paragraphs [0026]-[0027], [0040], [0043] and figs. 6A and 6B.	1-15
A	US 2006-0265630 A1 (ALBERTI, E. et al.) 23 November 2006 See abstract, paragraphs [0031]-[0037], and fig. 2.	1-15
A	US 2009-0144716 A1 (FELTS, D.) 04 June 2009 See abstract, paragraphs [0011]-[0034], and fig. 1.	1-15
A	US 2008-0209408 A1 (GANESAN, V. et al.) 28 August 2008 See abstract, paragraphs [0040]-[0046], and fig. 3.	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

30 September 2014 (30.09.2014)

Date of mailing of the international search report

30 September 2014 (30.09.2014)

Name and mailing address of the ISA/KR

International Application Division
Korean Intellectual Property Office
189 Cheongsu-ro, Seo-gu, Daejeon Metropolitan City, 302-701,
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/US2014/010691

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2002-0174422 A1	21/11/2002	None	
US 2004-0181790 A1	16/09/2004	WO 2004-081758 A2 WO 2004-081758 A3	23/09/2004 24/03/2005
US 2006-0265630 A1	23/11/2006	US 2011-0126187 A1 US 7937697 B2 US 8495615 B2	26/05/2011 03/05/2011 23/07/2013
US 2009-0144716 A1	04/06/2009	US 2009-0144726 A1 US 2009-0144727 A1 US 2009-0144728 A1 US 8589903 B2 US 8645939 B2	04/06/2009 04/06/2009 04/06/2009 19/11/2013 04/02/2014
US 2008-0209408 A1	28/08/2008	US 8615751 B2	24/12/2013