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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
13 January 2022 (13.01.2022)

WIPO | PCT

(10) International Publication Number
WO 2022/010491 A1

(51) International Patent Classification:
G06F 9/44 (2018.01)

(21) International Application Number:
PCT/US2020/041614

(22) International Filing Date:
10 July 2020 (10.07.2020)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P.** [US/US]; 10300 Energy Drive, Spring, Texas 77389 (US).

(72) Inventors: **COUTINHO MORAES, Mauricio**; Av. Ipiranga, 6681, Bld. 45C, Predios 5-6, 90619-900 Porto Alegre (BR). **MERTZ, Jhonny Marcos Acordi**; Av. Ipiranga, 6681, Bld. 45C, Predios 5-6, 90619-900 Porto Alegre (BR). **CALVETTI, Patrick Ucker**; Av. Ipiranga, 6681, Bld. 45C, Predios 5-6, 90619-900 Porto Alegre (BR).

(74) Agent: **WOODWORTH, Jeffrey C.** et al.; HP Inc., 3390 E. Harmony Road, Mail Stop 35, Fort Collins, Colorado 80528-9544 (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, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, IT, JO, JP, KE, KG, KH, KN, KP, KR, KW, 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, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, 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))*

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

(54) Title: APPLICATION VERSION SWITCHING

(57) Abstract: An example system includes an application engine to: execute an application. The example system includes a version history engine to: access a version history of the application. The example system includes an application modification engine to modify the application engine according to the version history, to cause the application engine to: determine an indication of a version of the application to execute as indicated by the version history; and execute the version of the application specified by the indication.

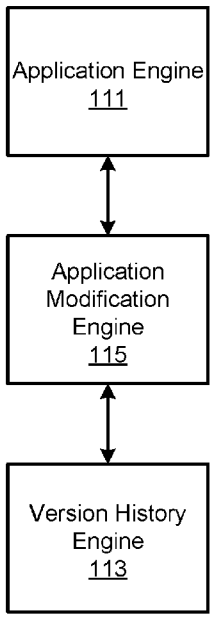

Figure 1

WO 2022/010491 A1

APPLICATION VERSION SWITCHING

BACKGROUND

[0001] Feature switching is sometimes performed using toggling to toggle features on and off, for example at websites and/or other types of applications. For example, particular subsets of code may be enabled or disabled to toggle the features on and off.

BRIEF DESCRIPTION OF THE DRAWINGS

[0002] Reference will now be made, by way of example only, to the accompanying drawings in which:

[0003] Figure 1 is a block diagram of an example system to implement application version switching.

[0004] Figure 2 is a block diagram of another example system to implement application version switching.

[0005] Figure 3 is a flow diagram of an example method to implement application version switching.

[0006] Figure 4 is a block diagram of an example computer-readable medium including instructions that causes a processor to implement application version switching.

[0007] Figure 5 is an example of a first version of an application.

[0008] Figure 6 is an example of a second version of the application of Figure 5.

[0009] Figure 7 is an example of an application engine to execute different versions of an application.

[0010] Figure 8 is another example of an application engine to execute different versions of an application.

[0011] Figure 9 is an example sequence diagram showing components of the system of Figure 2 implementing application version switching.

DETAILED DESCRIPTION

[0012] Feature switching is sometimes performed using toggling to toggle features on and off, for example at websites and/or other types of applications. For example, particular subsets of code may be enabled or disabled to toggle the features on and off.

[0013] Feature switching may be implemented manually to modify code to include feature toggles and/or version toggles, which is generally inefficient, and is often done on a feature by feature basis.

[0014] Hence, provided herein is a system which modifies an application engine as different versions of an application are produced. The different versions may be switched and/or toggled to execute the different versions of the application as controlled, for example, by toggle engines (which for example, may determine which version of the application to be executed based on configuration data and/or context data, such as a geographic location from which a request for the application was received, and the like). The different versions of the application may be determined via a version history (e.g. as stored at a version history database, and the like) and the application engine may be modified to include a command that retrieves an indication of which version of the application to execute (e.g. such as a version number, and the like). The application engine may be modified to include commands to call the version of the application specified by the indication, and/or the application engine may be modified to include respective code segments of versions of the application, as indicated by the version history, the respective code segments executed based on the indication. However, application engine may be modified in any suitable manner to execute the different versions of the application as indicated by the indication.

[0015] Hence, an aspect of the present specification is directed to a system comprising: an application engine to: execute an application; a version history engine to: access a version history of the application; and an application modification engine to modify the application engine according to the version history, to cause the application engine to: determine an indication of a version

of the application to execute as indicated by the version history; and execute the version of the application specified by the indication.

[0016] Another aspect of the present specification is directed to a method comprising: receiving, at a computing device, an execution command to execute an application having a plurality of versions identified by respective indications; determining, at the computing device, an indication of a version of the application to execute based on: context data of the execution command; and version configuration data determined using the context data; and executing, at the computing device, an application engine automatically modified, using a version history of the application, to: execute the version of the application specified by the indication.

[0017] Another aspect of the present specification is directed to a non-transitory computer-readable medium comprising instructions that, when executed by a processor, cause the processor to: execute an application module; execute a version history module to: access a version history of the application; and execute an application modification module to modify the application module according to the version history to cause the processor, when executing the application module to: determine an indication of a version of the application to execute as indicated by the version history.

[0018] Figure 1 is a block diagram of an example system 100 to implement application version switching. The system 100 includes an application engine 111, a version history engine 113 and an application modification engine 115. Communication between components and/or engines described herein is shown in the figures of the present specification as arrows therebetween.

[0019] As used herein, the term “engine” refers to hardware (e.g., a processor, such as a central processing unit (CPU) an integrated circuit or other circuitry) or a combination of hardware and software (e.g., programming such as machine- or processor-executable instructions, commands, or code such as firmware, a device driver, programming, object code, etc. as stored on hardware). Hardware includes a hardware element with no software elements such as an application specific integrated circuit (ASIC), a Field Programmable Gate Array (FPGA), etc. A combination of hardware and software includes

software hosted at hardware (e.g., a software module that is stored at a processor-readable memory such as random access memory (RAM), a hard-disk or solid-state drive, resistive memory, or optical media such as a digital versatile disc (DVD), and/or implemented or interpreted by a processor), or hardware and software hosted at hardware.

[0020] For example, the application engine 111 may comprise hardware or a combination of software and hardware for implementing functionality to: implement an application, such as a website, and the like, however the application engine 111 may be to implement any suitable application. The application engine 111 may comprise a portion of a server and/or computing device which hosts the application engine 111. However, the application engine 111 may comprise hardware or a combination of software and hardware of any suitable server and/or computing device and/or more than one suitable server and/or computing device.

[0021] In particular, as will be described in more detail below, the application engine 111 may comprise code segments to implement the application, the code segments comprising portions and/or segments of code and/or source code and/or executable instructions executed when the application engine 111 is implemented, and the like.

[0022] Initially, the application engine 111 is to execute one version of an application, for example an initial version of the application. However, developers may generally update the application to newer versions, for example to add features and/or fix issues (e.g. bugs), and the like.

[0023] As such, the system 100 further comprises a version history engine 113 to access a version history (not depicted) of the application, for example at a memory or a database storing versions of the application. Such a memory and/or database may comprise an enterprise memory and/or an enterprise database (e.g. of an enterprise providing the application and to which developers who work for the enterprise may submit versions of the application) and/or a public memory and/or public database (e.g. to which anyone may submit versions of the application, for example in an open source environment).

Regardless, the version history engine 113 may be to determine when a different version of the application has been stored at, and/or uploaded to, the memory and/or database. As such, the version history engine 113 may be to periodically query a version history memory and/or database for version history updates, and/or a version history memory and/or database (and/or a computing device thereof) may be to notify the application modification engine 113 of version history updates when another version of the application is stored at, and/or uploaded to, the version history memory and/or database.

[0024] Put another way, the version history engine 113 may be to: access the version history of the application as the version history is updated with newer versions of the application.

[0025] The system 100 further comprises the application modification engine 115 to modify the application engine 111 according to the version history. For example, the version history engine 113 may determine when another version of the application has been added to the memory and/or database and alert the application modification engine 115 thereto. However, the functionality of the version history engine 113 may be combined with the application modification engine 115.

[0026] The application modification engine 115 is generally to modify the application engine 111 according to the version history to cause the application engine 111 to: determine an indication of a version of the application to execute as indicated by the version history; and execute the version of the application specified by the indication.

[0027] For example, the application modification engine 115 may modify the application engine 111 to cause the application engine 111 to determine the indication of the version of the application to execute by: modifying the application engine 111 (e.g. modify and/or add to code segments of the application engine 111) to include a command to retrieve the indication. For example, the indication may be determined by run-time components and/or engines at a time of execution of the application engine 111 (e.g. at a run-time of the application), described in more detail below; the command may be used to

retrieve the indication (e.g. from the run-time components) to assist the application engine 111 in determining which version of the application to execute at run-time. In particular, the application modification engine 115 may insert, into the application engine, prior to code segments to execute the version of the application, a command to retrieve the indication when executing the application.

[0028] Furthermore, the application modification engine 115 may modify the application engine 111 (e.g. modify and/or add to code segments of the application engine 111) to cause the application engine 111 to execute the version of the application specified by the indication by: modifying the application engine 111 to include commands to call the version of the application specified by the indication. In some of these examples, the application modification engine 115 may be further to: generate a plurality of application files and/or functions according to the version history, the plurality of application files and/or functions identified by respective distinct names associated with respective indications; and modify the application engine 111 according to the version history by incorporating the respective distinct names into the application engine 111 such that the application engine 111 calls an application file and/or function according to the indication associated with a respective distinct name thereof. In a particular example, the application files and/or functions may be named according to a version thereof (e.g. “App_V1”, “App_V2”, for two versions of an application) and incorporated into the application engine 111 and/or stored at a memory accessible to the application engine 111. In response to the application engine 111 determining a version of the application to execute, the application engine 111 may call the determined version of the application by calling the associated named application file and/or function; for example, when a “Version 2” of the application is to be executed, the application engine 111 may call the application file and/or function that represents “Version 2” with the associated name. However, any suitable include commands and/or scheme and/or method to call a particular version of an application specified by an indication is within the scope of the present specification.

[0029] In yet further examples, the application modification engine 115 may be to modify the application engine 111 according to the version history, to cause the application engine 111 to execute the version of the application specified by the indication by: modifying the application engine 111 to include respective code segments of versions of the application, as indicated by the version history, the respective code segments executed based on the indication.

[0030] For example, the application modification engine 115 may be to incorporate code segments associated with the versions of the application into the application engine 111, the code segments identified in the application engine 111, by respective indications associated with the versions of the application, such that the code segments identified by the indication are executed by the application engine 111 and the code segments identified by remaining respective indications are refrained from being executed. In particular, the application modification engine 115 may insert, into the application engine 111, "IF/ELSE" sets of commands and/or code segments and the like, which specify that "IF" the indication indicates a first version of the application, then the application engine 111 is to execute code segments of the first version of the application, which are also incorporated into the application engine 111 (e.g. and refrain from executing code segments of other versions of the application); however, (e.g. "ELSE") if the indication indicates a second version of the application, then the application engine 111 is to execute code segments of the second version if the application (e.g. and refrain from executing code segments of other versions of the application), which are also incorporated into the application engine 111. When third, fourth, fifth, etc. versions of the application are available, associated code segments for executing such versions are also incorporated into the application engine 111.

[0031] While two examples for modifying the application engine 111 to execute different versions of an application are provided in the present specification, any suitable process and/or scheme and/or method for modifying the application engine 111 to execute different versions of an application is within the scope of the present specification. In general, the application modification engine 115 modifies the application engine 111 in any suitable manner according to the

version history of the application as updated (e.g. in response to the version history of the being updated with newer versions of the application, as determined by the version history engine 113).

[0032] Attention is next directed to Figure 2 which is a block diagram of another example system 200 to implement application version switching. The system 200 is substantially similar to the system 100, with like components having like numbers, but in a "200" series rather than a "100" series. For example, the system 200 may include an application engine 211, a version history engine 213 and an application modification engine 215 which are respectively substantially similar to the application engine 111, the version history engine 113 and the application modification engine 115.

[0033] In contrast to the system 100, the system 200 further includes a version history memory and/or database 217 (interchangeably referred to hereafter as the database 217) storing two example versions 219-1, 219-2 of an application (e.g. "APP_V1", and "APP_V2") to be executed by the application engine 211 and which is accessible by the version history engine 213 and the application modification engine 215. The versions 219-1, 219-2 are interchangeably referred to hereafter, collectively, as the versions 219 and, generically, as a version 219. While two versions 219 are shown, the database 217 may store any suitable number of versions 219 named in any suitable manner; a number of the versions 219 may change (e.g. increase) as versions 219 are added to the database 217; however, the number of the versions 219 may decrease, when versions 219 are deleted from the database 217.

[0034] For example, the version history engine 213 may access the database 217 to determine when new versions 219 of the application are added, and notify the application modification engine 215 accordingly, and the application modification engine 215 may modify the application engine 211 accordingly. For example, as depicted, the application engine 211 comprises code segments 221 which the application modification engine 215 may modify and/or to which the application modification engine 215 may add. It is understood that the version history engine 213 may access the database 217 to determine when versions

219 of the application are deleted, and notify the application modification engine 215 accordingly, and the application modification engine 215 may modify the code segments 221 and/or the application engine 211 accordingly.

[0035] As depicted, the system 200 further comprises run-time components, in particular a toggle engine 231, a context builder engine 233, and a version determination engine 235 which, at a run-time of the application engine 211, may determine which version 219 of the application the application engine 211 is to execute. While the run-time components are depicted as being separate, in other examples, functionality of the toggle engine 231 and/or the context builder engine 233 and/or the version determination engine 235 may be combined into one engine and/or more than one engine.

[0036] For example, the application engine 211 may receive, from a communication device 241 an execution command to execute an application having a plurality of versions 219 identified by respective indications for the application represented by the versions 219 (e.g. a webpage and the like). For example, the communication device 241 may comprise a communication device of a user who operates the communication device 241 to request webpages, and the like.

[0037] The application engine 211 may call and/or communicate with the toggle engine 231 to determine an indication of a version 219 of the application to be implemented for example using the version determination engine 235, as described hereafter.

[0038] The toggle engine 231, the context builder engine 233 and the version determination engine 235 may communicate to determine which version 219 of the application is to be executed, generate an indication thereof, and provide the indication to the application engine 211, which executes the associated version 219, and provides a response to the communication device 241.

[0039] For example, the toggle engine 231, the context builder engine 233 and the version determination engine 235 may determine which version 219 of the application is to be executed based on: context data of the execution command; and version configuration data determined using the context data. For example,

as depicted, the toggle engine 231 may determine the context data of the execution command by communicating with the context builder engine 233 which may determine, for example, a geographic location from which the execution command originated (e.g. using a network address in header data and/or metadata of the execution command) and/or other context data, including, but not limited to, whether a header and/or metadata of the execution command includes data which indicates any particular context such as a request to implement a particular version 219 of the application, whether or not a mobile version 219 of the application is being requested (e.g. or a desktop version) and/or any other suitable context data. The context builder engine 233 may hence be to receive the context data and/or header data and/or metadata and determine context thereof, including, but not limited to, a geographic location associated with a network address, and the like.

[0040] Such context data may be communicated, by the toggle engine 231, to the version determination engine 235 which may communicate with a configuration database 251 storing version configuration data 253. The version configuration data may associate context data to versions 219 of the applications. For example, a first geographic location (e.g. as indicated by the context data) may be associated with the first version 219-1 of the application, and a second geographic location (e.g. as indicated by the context data) may be associated with the second version 219-2 of the application; hence, in this example, when the communication device 241 is located in the first geographic location, the application engine 211 may be forced to implement the first version 219-1 of the application, and/or when the communication device 241 is located in the second geographic location, the application engine 211 may be forced to implement the second version 219-2 of the application. For example, amongst other possibilities, different locations and/or jurisdictions may enforce different requirements for applications, such as websites, and the versions 219 may reflect such requirements, and the requirements may be indicated by the version configuration data 253.

[0041] The version configuration data 253 may be generated and/or updated by an administrator of the system 200 and/or the version configuration data 253

may be generated and/or updated automatically and/or electronically by any suitable component of the system 200, for example when new versions 219 of the application are stored at the database 217; hence, while not depicted, the configuration database 251 may be in communication with other components of the system 200 which may update the version configuration data 253 (e.g. such as the version history engine 213) as versions 219 of the application are stored and/or changed at the database 217.

[0042] The version determination engine 235 may use the context data to search and/or query the configuration database 251 for configuration data which may enable the version determination engine 235 to determine an indication of the version 219 of the application to execute based on the context data, which is returned to the application engine 211 via the toggle engine 231. Communication between the run-time components of the system 200 are described in more detail below with respect to Figure 3 and Figure 9.

[0043] Referring to Figure 3, a flowchart of an example method 300 to implement application version switching is depicted. In order to assist in the explanation of method 300, it will be assumed that method 300 may be performed with the system 200, and at least partially by a computing device implementing the system 200 and/or a processor thereof. Indeed, the method 300 may be one way in which the system 200 may be configured. Furthermore, the following discussion of method 300 may lead to a further understanding of the system 200, and its various components. Furthermore, it is to be emphasized, that method 300 may not be performed in the exact sequence as shown, and various blocks may be performed in parallel rather than in sequence, or in a different sequence altogether. Furthermore, it is to be emphasized that the method 300 may alternatively be performed with the system 100, and at least partially by a computing device implementing the system 100 and/or a processor thereof.

[0044] Beginning at a block 301, a computing device receives an execution command to execute an application having a plurality of versions 219 identified by respective indications. For example, as described above, a computing device

executing the application engine 211 receives an execution command from the communication device 241.

[0045] At a block 303, the computing device determines an indication of a version 219 of the application to execute based on: context data of the execution command (as described above); and version configuration data determined and/or queried using the context data.

[0046] The toggle engine 231 and/or the context builder engine 233 is hence understood to have access to, for example, header data and/or metadata of the execution command in order to determine the context data. As such, the application engine 211 may pass the execution command and/or the header data and/or the metadata thereof to the toggle engine 231, which may pass the execution command and/or the header data and/or the metadata thereof to the context builder engine 233; and/or the execution command and/or the header data and/or the metadata thereof may be at least temporarily stored in a memory accessible to suitable components of the system 200. The context builder engine 233 may then return the context data to the toggle engine 231, which passes the context data to the version determination engine 235.

[0047] For example, the context data is used to query the configuration database 251 (e.g. via the version determination engine 235) to determine which version 219 of the application to execute based on the version configuration data 253. Hence, a set of the version configuration data 253 which indicates which version 219 of the application to execute may be selected based on the context data. In examples where the context data includes collisions between versions 219 of the application to be executed (e.g. a geographic location of the execution command may indicate that the first version 219-1 of the application is to be executed, but the execution command may request the second version 219-2), the version configuration data 253 may indicate which context data is to be given priority (e.g. geography over version requests) and/or the version determination engine 235 (and/or another component of the system 200) may resolve such collisions based on configured rules (not depicted) in any suitable manner.

[0048] As described previously, the method 300 may include the computing device determining the context data of the execution command by communicating with the context builder engine 233.

[0049] Similarly, as described previously, the method 300 may include the computing device: communicating (e.g. via the version determination engine 235) the context data to the configuration database 251 storing the version configuration data 253; and receiving, from the configuration database 251 (e.g. via the version determination engine 235), the indication of the version 219 of the application to execute. For example, the indication may comprise a version number of the application to execute, however the indication may comprise any suitable indicator of a version 219 of the application to execute.

[0050] Hence, in general, the computing device may determine, at the block 303, the indication of the version 219 of the application to execute by executing the version determination engine 235.

[0051] At a block 305, the computing device executes the application engine 211 which, as described herein, has been automatically and/or electronically modified, using a version history of the application, to: execute the version 219 of the application specified by the indication.

[0052] For example, the toggle engine 231 may communicate the indication to the application engine 211, which executes the associated version 219 accordingly, and/or the indication may be stored at a memory accessible to the toggle engine 231 and the application engine 211) and which may be retrieved by the application engine 211 to determine the version 219 of the application to be executed.

[0053] Figure 4 is a block diagram of an example device 400 that includes a computer-readable medium 401 and a processor 402. The computer-readable medium 401 includes instructions that, when implemented by the processor 402, cause the processor 402 to implement application version switching.

[0054] The computer-readable medium 401 may be a non-transitory computer-readable medium, such as a volatile computer-readable medium (e.g., volatile

RAM, a processor cache, a processor register, etc.), a non-volatile computer-readable medium (e.g., a magnetic storage device, an optical storage device, a paper storage device, flash memory, read-only memory, non-volatile RAM, etc.), and/or the like.

[0055] The processor 402 may be a general-purpose processor or special purpose logic, such as a microprocessor (e.g., a central processing unit, a graphics processing unit, etc.), a digital signal processor, a microcontroller, an ASIC, an FPGA, a PAL (programmable array logic), a PLA (programmable logic array), a PLD (programmable logic device), etc. The computer-readable medium 401 or the processor 402 may be distributed among a plurality of computer-readable media or a plurality of processors.

[0056] The computer-readable medium 401 includes modules. As used herein, a "module" (in some examples referred to as a "software module") is a set of instructions that when implemented or interpreted by a processor or stored at a processor-readable medium realizes a component or performs a method.

[0057] As depicted, the computer-readable medium 401 includes various modules which correspond to functionality of the engines of the system 200.

[0058] In particular the computer-readable medium 401 includes an application module 411, which, when processed by the processor 402, may provide the processor 402 with functionality similar to the application engine 211. For example, the application module 411 may cause the processor 402 to execute an application.

[0059] As depicted, the computer-readable medium 401 includes a version history module 413, which, when processed by the processor 402, may provide the processor 402 with functionality similar to the version history engine 213. For example, the version history module 413 may cause the processor 402 to access a version history of the application. For example, as depicted, the computer readable medium 401 further stores the version history database 217, however the database 217 may be stored in a different location and/or memory.

[0060] As depicted, the computer-readable medium 401 includes an application

modification module 415, which, when processed by the processor 402, may provide the processor 402 with functionality similar to the application modification engine 215. For example, the application modification module 415 may cause the processor 402 to modify the application module 411 according to the version history to cause processor 402, when executing the application module 411, to: determine an indication of a version 219 of the application to execute as indicated by the version history.

[0061] In some examples, the version history module 413 may be to further cause the processor 402 to: access the version history of the application as the version history is updated with newer versions 219 of the application, as described above. In some of these examples, the application modification module 415 may be to further cause the processor 402 to: modify the application module 411 according to the version history as updated, as also described above.

[0062] In some examples, the application modification module 415 may be to further cause the processor 402 to: generate a plurality of application files according to the version history, the plurality of application files identified by respective distinct names associated with respective indications; and modify the application module 411 according to the version history by incorporating the respective distinct names into the application module 411 such that the processor 402, when executing the application module 411, calls an application file according to the indication associated with a respective distinct name thereof.

[0063] In some examples, the application modification module 415 may be to further cause the processor 402 to modify the application module 411 according to the version history to cause processor, when executing the application module 411, to determine the indication of the version 219 of the application to execute as indicated by the version history by: inserting, into the application module 411, prior to code segments to execute the version 219 of the application, a command to retrieve the indication when executing the application.

[0064] In other examples, the application modification module 415 may be to further cause the processor 402 to modify the application module 411 according to the version history by: incorporating code segments associated with the versions 219 of the application into the application module 411, the code segments identified in the application module 411, by respective indications associated with the versions 219 of the application, such that the code segments identified by the indication are executed by processor 402 when executing the application module 411 and the code segments identified by remaining respective indications are refrained from being executed.

[0065] As depicted, the computer-readable medium 401 includes a toggle module 431, a context builder module 433, and a version determination module 435, which, when processed by the processor 402, may respectively provide the processor 402 with functionality similar to the toggle engine 231, the context builder engine 233, and the version determination engine 235. As depicted, the computer-readable medium 401 further stores the configuration database 251. However, the modules 431, 433, 435 and/or the configuration database 251 may be stored at a different computing device from the modules 411, 413, 415 and/or the database 217; for example the modules 411, 413, 415 and/or the database 217 may be stored at the device 400 which implements version updating and application execution components of the system 200, and the modules 431, 433, 435 and/or the configuration database 251 may be stored at another computing device which implements run-time components of the system 200.

[0066] Attention is next directed to Figure 5 and Figure 6 which respectively depict code segments of two versions 219-1, 219-2 of an application and in particular a function “handleRequest”. In particular, Figure 5 and Figure 6 depict a code snippets in Javascript-like code. Various functions, and other parts of the code snippets, are on various lines indicated by line numbering 01, 02, 03, etc. As depicted on line 01 of the versions 219, the function “handleRequest” may be annotated with the annotation “@VersionToggle” having a toggle name of “RequestHandle”, which may be defined in a Javascript-like class skeleton (not depicted). As the versions 219 have a same class and a same name, they are

understood to be different versions of a same application which the system 200 (and the like) may toggle between.

[0067] The two versions 219 of the application of Figure 5 and Figure 6 generally performs an authorization of a user based on a string “user” received, along with an integer “input”, for example in an execution command, such a request to process the application engine 111, the application engine 211 and/or the application module 411, as received from the communication device 241, and the like.

[0068] The two versions 219 of the application of Figure 5 and Figure 6 performs an authorization of the “user” for example by determining whether the “user” has access to a given function “businessLogic” (e.g. on line 07), for example as indicated in a database, and the like. Such a determination is made via a function “!hasAccess(user)”).

[0069] With regards to the version 219-1, if the “user” does not have access to the given function “businessLogic”, an error code “403” is returned; however, if the “user” has access, the version 219-1 determines whether the “input” is valid via a function “!isValid(input)”. If the “input” is not valid, an error code “400” is returned. If the “input” is valid, the function “business logic(user, input)” is implemented and a code “200” is returned.

[0070] In contrast, the version 219-2 provides similar functionality but differs from the version 219-1 as next described. In particular, in the version 219-2, if the “user” does not have access to the given function “businessLogic”, an error code “403” is returned as well as a message “unauthorized”; however, if the “user” has access, the version 219-2 determines whether the “input” is valid via a function “!isInputValid(input)”, which is different from the function “!isValid(input)” of the version 219-1, and which may be more accurate and/or less prone to error than the function “!isValid(input)”. If the “input” is not valid, an error code “400” is returned as well as a message “invalid”. If the “input” is valid, the function “business logic(user, input)” is implemented and a code “200” is returned.

[0071] Hence, the two versions 219 provide similar functionality, but the version

219-2 provides error messages (e.g. “unauthorized” and “invalid”) as well as determines whether the “input” is valid via a function that is different from a function of the version 219-1.

[0072] In some examples, prior to the version 219-2 being added to the database 217, the application engine 211 may initially comprise the version 219-1 (but without the annotation on line 01 and/or a different annotation). Put another way, the code segments 221 of the application engine 211 may initially comprise the code segments of Figure 5.

[0073] However, in response to the version 219-2 being added to the database 217, as determined by the version history engine 213, the application modification engine 215 may modify the application engine 211 and/or the code segments 221 thereof as depicted in Figure 7 or Figure 8 to implement a particular version 219 of an application, depending on an indication.

[0074] For example, in Figure 7, the application engine 211, and/or the code segments 221 thereof, has been modified to remove specific code segments of either of the versions 219 of Figure 5 and Figure 6, other than the function name “handleRequest” and the fields for receiving “user” and “input, and replaced with a command “getVersionToToggle ()” which retrieves, for example from a memory and/or the toggle engine 231, the indication of which version 219 of the application to execute. For example, the command “getVersionToToggle ()” may pass the execution command and/or header data and/or metadata thereof to the toggle engine 231 as described herein. As such, while no data is depicted as input to the function getVersionToToggle (), the execution command and/or header data and/or metadata thereof may be used as input to the function getVersionToToggle().

[0075] For example the indication retrieved may be “1” for the version 219-1 of Figure 5, or “2” for the version 219-2 of Figure 6. The variable “versionToToggle” is set to the indication, and an if/else set of commands is used to pass the indication to one of a plurality of application files identified by respective distinct names associated with respective indications. For example, while not depicted, it is understood that the application modification engine 215 has generated an

application file (and/or function) named “handleRequest_V1” comprising the version 219-1 depicted in Figure 5, and has further generated an application file and/or function) named “handleRequest_V2” comprising the version 219-2 depicted in Figure 6. The application files and/or functions “handleRequest_V1”, “handleRequest_V2” may be stored in a memory accessible to the application engine 211 and/or may be components of the application engine 211. The application files and/or functions “handleRequest_V1”, “handleRequest_V2” receive, as input, the strings “user” and “input”.

[0076] With continued reference to Figure 7, in these examples, the if/else set of commands causes the application file and/or function named “handleRequest_V1” to be executed and/or called when the indication is “1”, and the if/else set of commands causes the application file and/or function named “handleRequest_V2” to be executed and/or called when the indication is “2”.

[0077] It is further understood that when a third version 219 of the application is added to the database 217, the application modification engine 215 may modify the application engine 211 to call an application file and/or function named “handleRequest_V3”, and the like, to be executed and/or called when the indication (e.g. versionToToggle) is “3”.

[0078] Attention is next directed to Figure 8 which depicts the application engine 211, and code segments 221 thereof, as modified by the application modification engine 215 in other examples. In Figure 8, the application engine 211 again includes the function “getVersionToToggle()” which returns the indication of the version 219 of the application to implement. However, rather than using application file names, the application modification engine 215 has incorporated the code segments of the versions 219-1, 219-2 as depicted in Figure 5 and Figure 6, into the application engine 211, which are selected using if/else sets of commands. Hence, when versionToToggle=1, the status “403” is returned when the function !hasAccess(user) indicates the “user” does not have access, else when versionToToggle=2, the status “403” is returned with the message “unauthorized” when the function !hasAccess(user) indicates the

"user" does not have access. Similarly, when versionToToggle=1, the function "!isValid(input)" is used to check whether "input is valid, and the status "400" is returned when the function "!isValid(input)" indicates the "input" is not valid; else when versionToToggle=2, the function "!isInputValid(input)" is used to check whether "input is valid, and the status "400" is returned with the message "invalid" when the function "!isInputValid(input)" indicates the "input" is not valid.

[0079] It is further understood that when a third version 219 of the application is added to the database 217, the application modification engine 215 may modify the application engine 211 to include respective code segments thereof which are executed and/or called when the indication (e.g. versionToToggle) is "3".

[0080] In some examples, the application modification engine 215 may modify the application engine 211 and/or the code segments 221 thereof as depicted in Figure 7 and/or Figure 8 depending on a configuration thereof and/or settings by a system administrator. For example, while the example of Figure 7 is simpler than the example of Figure 8, the example of Figure 7 uses more function calls (e.g. functions "handleRequest_V1", "handleRequest_V2" are called which, in turn call functions) than the example of Figure 8 (and hence may use more processing resources to implement at run-time). However, the example of Figure 7 may be simpler to generate then the example of Figure 8. Hence, the application modification engine 215 may be further to determine which of the examples to implement based, for example, on complexity of generation and/or error checking (which may be based on complexity of the original applications (e.g. applications 219) and/or an evaluation of respective processing resources used to implement the respective examples of Figure 7 and Figure 8 (e.g. which may be based on numbers of function calls, and/or any other suitable parameter).

[0081] Attention is next directed to Figure 9 which is an example sequence diagram 900 showing components of the system of Figure 2 implementing application version switching at a run-time of the application engine 211. It is understood in Figure 9 that the application engine 211 has been modified to implement different versions of an application depending on an indication of

version of the application to be determined. Hence, for example, in Figure 9, the application engine 211 may be modified similar to Figure 7 or Figure 8.

[0082] As depicted, the communication device 241 transmits an execution command 901 to the application engine 211 which communicates the execution command 901, and/or header data and/or metadata thereof, to the toggle engine 231 (e.g. which may occur via the application engine 211 implementing the "getVersionToToggle ()" function).

[0083] The toggle engine 231 transmits a request 903 for context data to the context builder engine 233; the request 903 may include the execution command and/or header data and/or metadata thereof. The context builder engine 233 returns context data 905, as described above (e.g. a geographic location from which the execution command was received, for example as indicated by a network address of the execution command and/or header data and/or metadata thereof).

[0084] The toggle engine 231 receives the context data 905 and transmits the context data 905 to the version determination engine 235. The version determination engine 235 may use the context data 905 to read and/or query 907 the configuration database 251 (which may occur via a suitable computing device, which may also be referred to as a "connector" and the like) to receive configuration data 909, which is used, by the version determination engine 235 to determine an indication 911 of which version 219 of the application is to be executed.

[0085] The version determination engine 235 returns the indication 911 to the toggle engine 231, which returns the indication 911 to the application engine 211. The application engine 211 executes 913 the version 219 of the application indicated by the indication 911 to return a response 915 to the communication device 241.

[0086] In this manner, the application engine 211 may be automatically and/or electronically adapted to automatically toggle between different versions of an application.

[0087] It should be recognized that features and aspects of the various examples provided above may be combined into further examples that also fall within the scope of the present disclosure.

CLAIMS

1. A system comprising:

an application engine to: execute an application;

a version history engine to: access a version history of the application; and

an application modification engine to modify the application engine according to the version history, to cause the application engine to: determine an indication of a version of the application to execute as indicated by the version history; and execute the version of the application specified by the indication.

2. The system of claim 1, wherein the application modification engine is further to modify the application engine according to the version history, to cause the application engine to determine the indication of the version of the application to execute by: modifying the application engine to include a command to retrieve the indication when executing the application.

3. The system of claim 1, wherein the application modification engine is further to modify the application engine according to the version history to cause the application engine to execute the version of the application specified by the indication by: modifying the application engine to include commands to call the version of the application specified by the indication.

4. The system of claim 1, wherein the application modification engine is further to modify the application engine according to the version history to cause the application engine to execute the version of the application specified by the indication by: modifying the application engine to include respective code segments of versions of the application, as indicated by the version history, the respective code segments executed based on the indication.

5. The system of claim 1, wherein the version history engine is further to access the version history of the application at a memory or a database storing

versions of the application.

6. A method comprising:

receiving, at a computing device, an execution command to execute an application having a plurality of versions identified by respective indications;

determining, at the computing device, an indication of a version of the application to execute based on: context data of the execution command; and version configuration data determined using the context data; and

executing, at the computing device, an application engine automatically modified, using a version history of the application, to: execute the version of the application specified by the indication.

7. The method of claim 6, further comprising:

determining the context data of the execution command by communicating with a context builder engine.

8. The method of claim 6, further comprising:

determining the context data of the execution command by communicating with a context builder engine;

communicating the context data to a configuration database storing the version configuration data; and

receiving, from the configuration database, the indication of the version of the application to execute.

9. The method of claim 6, wherein the determining, at the computing device, the indication of the version of the application occurs by executing a version determination engine.

10. The method of claim 6, further comprising:

storing the indication of the version to execute which is retrieved by the application engine.

11. A non-transitory computer-readable medium comprising instructions that, when executed by a processor, cause the processor to:

execute an application module;

execute a version history module to: access a version history of the application; and

execute an application modification module to modify the application module according to the version history to cause the processor, when executing the application module to: determine an indication of a version of the application to execute as indicated by the version history.

12. The non-transitory computer-readable medium of claim 11, wherein:

the version history module is to further cause the processor to: access the version history of the application as the version history is updated with newer versions of the application; and

the application modification module is to further cause the processor to: modify the application module according to the version history as updated.

13. The non-transitory computer-readable medium of claim 11, wherein the application modification module is to further cause the processor to modify the application module according to the version history to cause the processor, when executing the application module, to determine the indication of the version of the application to execute as indicated by the version history by:

inserting, into the application module, prior to code segments to execute the version of the application, a command to retrieve the indication when executing the application.

14. The non-transitory computer-readable medium of claim 11, wherein the application modification module is to further cause the processor to:

generate a plurality of application files according to the version history, the plurality of application files identified by respective distinct names associated with respective indications; and

modify the application module according to the version history by

incorporating the respective distinct names into the application module such that the processor, when executing the application module, calls an application file according to the indication associated with a respective distinct name thereof.

15. The non-transitory computer-readable medium of claim 11, wherein the application modification module is to further cause the processor to modify the application module according to the version history by:

incorporating code segments associated with the versions of the application into the application module, the code segments identified in the application module, by respective indications associated with the versions of the application, such that the code segments identified by the indication are executed by processor when executing the application module and the code segments identified by remaining respective indications are refrained from being executed.

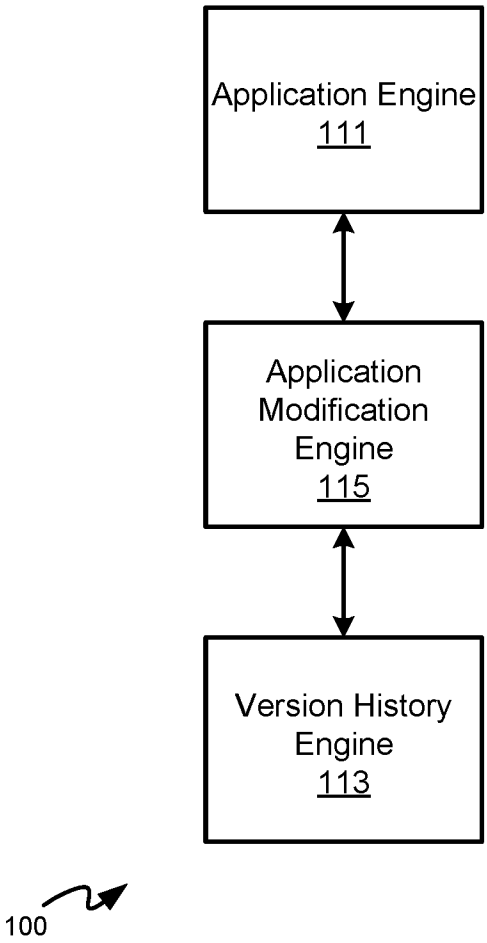
Application Engine
111
Application Modification Engine
115
Version History Engine
113
100

Figure 1

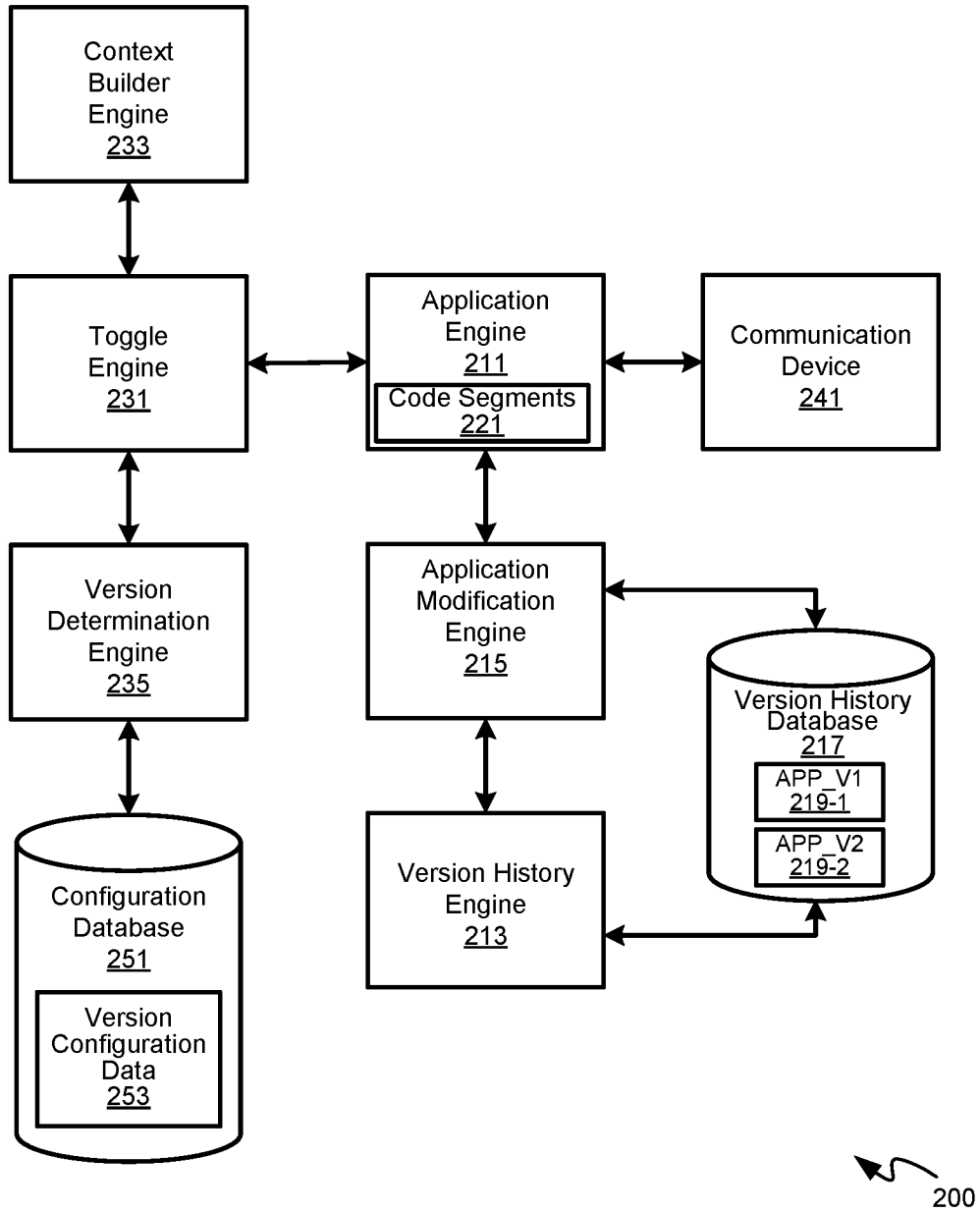
Context Builder Engine 233
Toggle Engine 231
Application Engine 211
Code Segments 221
Communication Device 241
Version Determination Engine 235
Application Modification Engine 215
Version History Database 217
APP_V1 219-1
APP_V2 219-2
Configuration Database 251
Version Configuration Data 253
Version History Engine 213
200

Figure 2

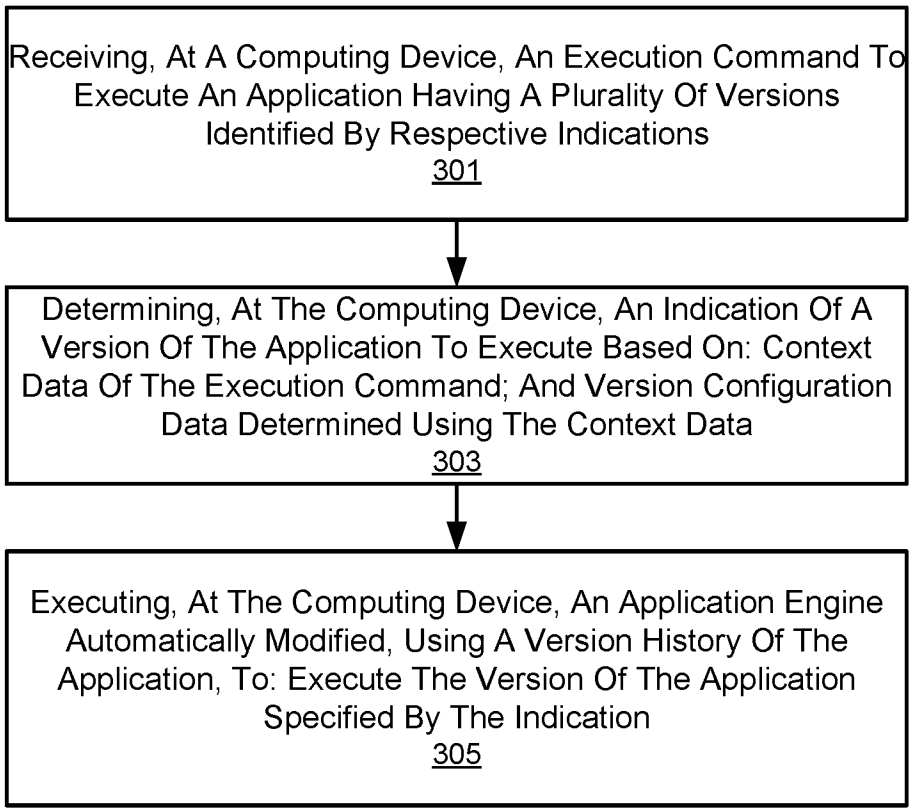
Receiving, At A Computing Device, An Execution Command To Execute An Application Having A Plurality Of Versions Identified By Respective Indications
301
Determining, At The Computing Device, An Indication Of A Version Of The Application To Execute Based On: Context Data Of The Execution Command; And Version Configuration Data Determined Using The Context Data
303
Executing, At The Computing Device, An Application Engine Automatically Modified, Using A Version History Of The Application, To: Execute The Version Of The Application Specified By The Indication
305
300

Figure 3

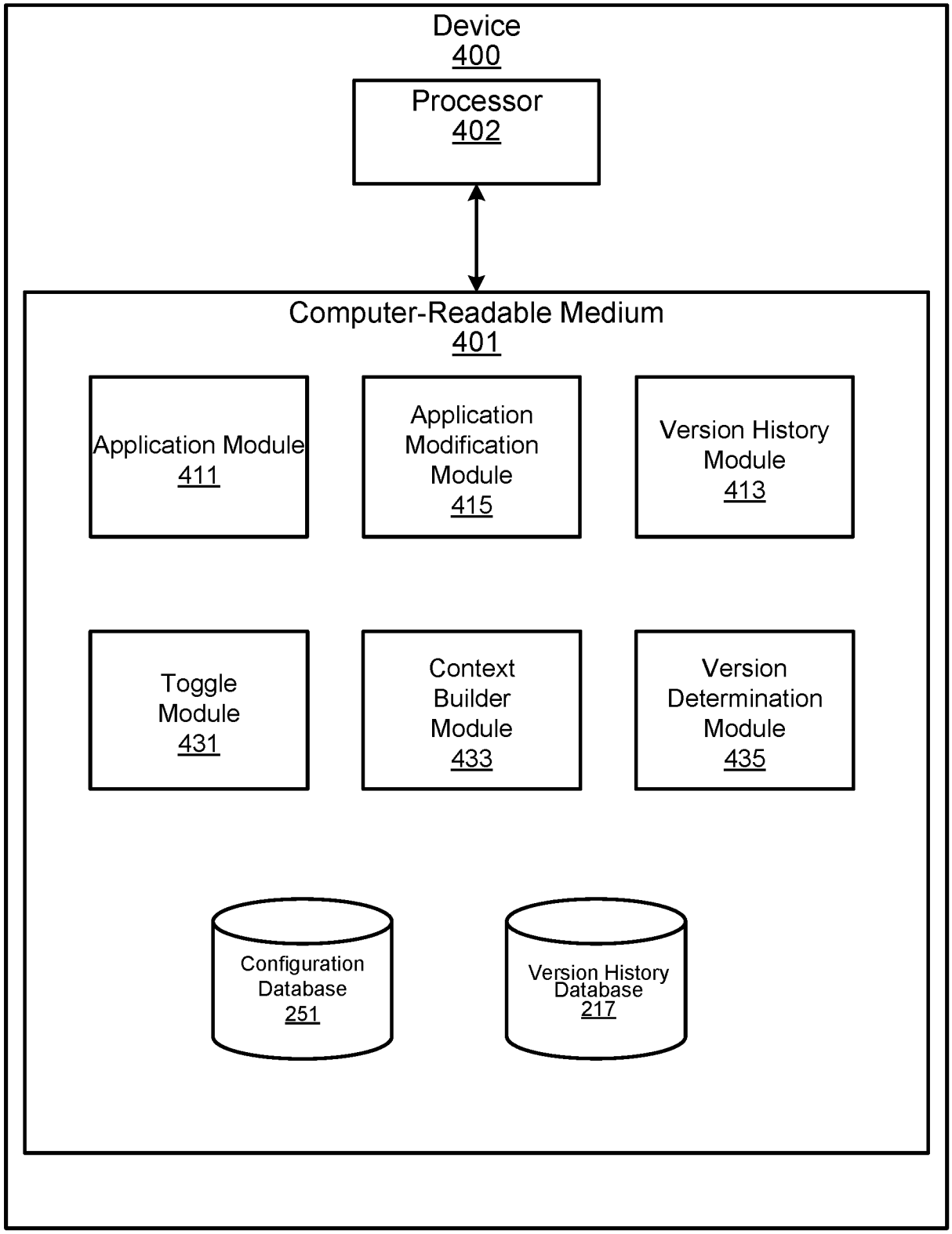
Device
400
Processor
402
Computer-Readable Medium
401
Application Module
411
Application Modification Module
415
Version History Module
413
Toggle Module
431
Context Builder Module
433
Version Determination Module
435
Configuration Database
251
Version History Database
217

Figure 4

219-1

```
01: @VersionToggle ( 'RequestHandle' )
02: function handleRequest ( user , input ) {
03:   if (!hasAccess ( user ))
04:         return { status: 403 }
05:   if (!isValid ( input ))
06:         return { status: 400 }
07:   businessLogic ( user , input )
08:   return { status: 200 }
09: }
```

Figure 5

219-2

```
01: @VersionToggle ( 'RequestHandle' )
02: function handleRequest ( user , input ) {
03:   if (!hasAccess ( user ))
04:         return { message: "unauthorized" , status: 403 }
05:   if (!isInputValid ( input ))
06:         return { message: "invalid" , status: 400 }
07:   businessLogic ( user , input )
09:   return { status: 200 }
09: }
```

Figure 6

211

```
01: function handleRequest ( user , input ) {
02:   let versionToToggle = getVersionToToggle ()
03:   if ( versionToToggle == 1 )
04:         return handleRequest_V1 ( user , input )
05:   else
06         return handleRequest_V2 ( user , input )
07 }
```

221

Figure 7

211

```
01 function handleRequest ( user , input ) {
02    let versionToToggle = getVersionToToggle ()
03    if (! hasAccess ( user ))
04          return ( versionToToggle == 1 ?
05                      { status: 403 } :
06                      { message: "unauthorized" , status: 403 })
07    if (( versionToToggle == 1 && ! isValid ( input )) ||
08             ( versionToToggle != 1 && ! isInputValid ( input )))
09          return ( versionToToggle == 1 ?
10                      { status: 400 } :
11                      { message: "invalid" , status: 400 })
12    businessLogic ( user , input )
13    return ( versionToToggle == 1 ) ?
14    { status: 200 } :
15    { status : 200 }))
```

221

Figure 8

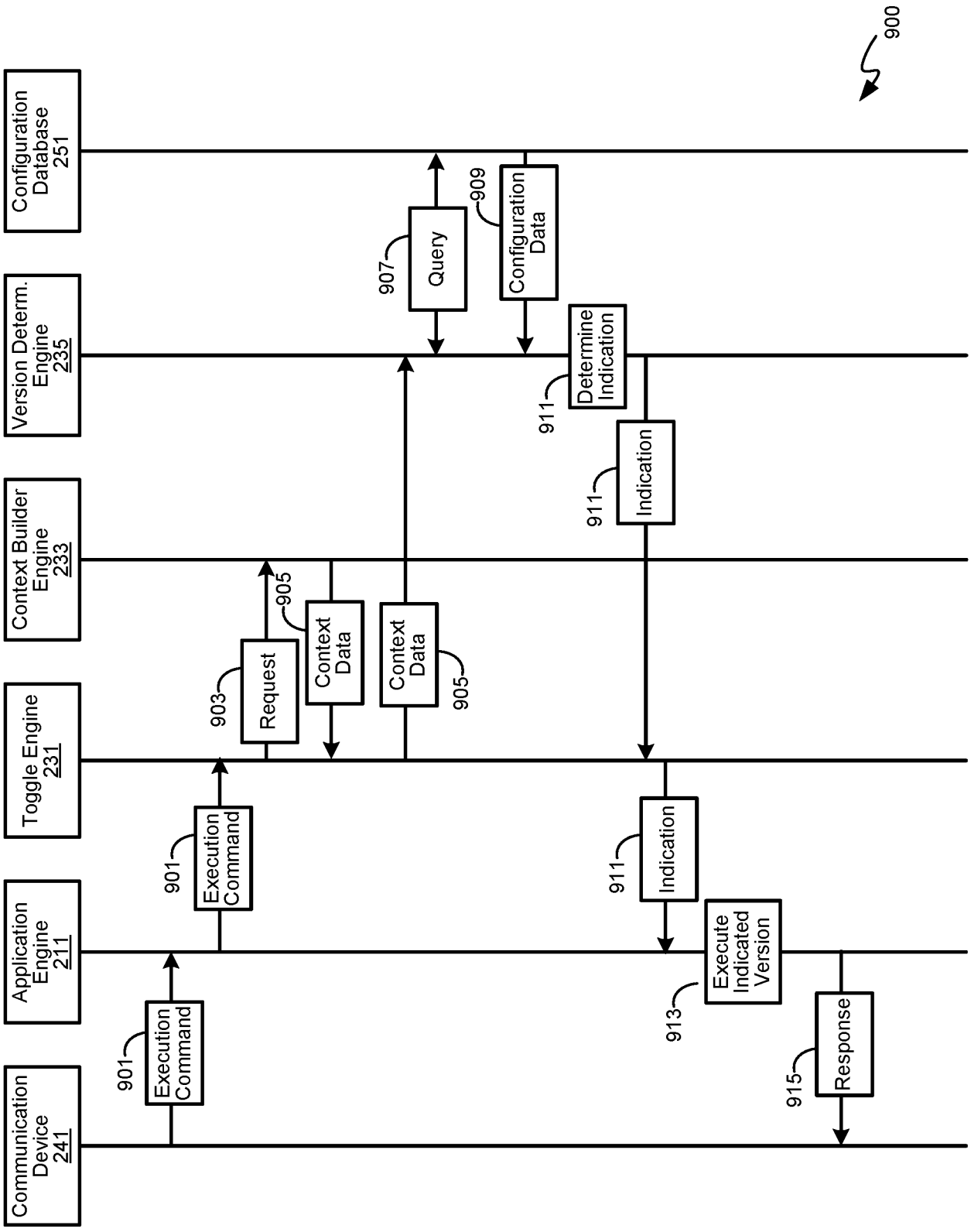
900
Configuration Database 251
Version Determ. Engine 235
Context Builder Engine 233
Toggle Engine 231
Application Engine 211
Communication Device 241
Execution Command
901
Query
907
Configuration Data
909
Request
903
Context Data
905
Context Data
905
Determine Indication
911
Indication
911
Execution Command
901
Indication
911
Execute Indicated Version
913
Response
915

Figure 9

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US 2020/041614

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/44 *(2018.01)*

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 15/00-15/16, G06F 09/00-09/445, G06F 21/00

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)

EAPATIS, ESPACENET, PAJ, RUPTO, USPTO, PatSearch

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	WO 2005/024666 A2 (MICROSOFT CORP) 17.03.2005	1-15
A	US 2008/0209390 A1 (MICROSOFT CORP) 28.08.2008	1-15
A	US 2012/0036440 A1 (OPENPEAK INC.) 09.02.2012	1-15
A	US 2016/0371172 A1 (ADOBE SYSTEMS INC) 22.12.2016	1-15
A	US 2017/0323089 A1 (ENTERPRISEWEB LLC) 09.11.2017	1-15
A	WO 2004/040399 A2 (COMMERCE ONE OPERATIONS INC) 13.05.2004	1-15

[X] 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

"D" document cited by the applicant in the international application

"E" earlier document 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
11 March 2021 (11.03.2021)	25 March 2021 (25.03.2021)
Name and mailing address of the ISA/RU: Federal Institute of Industrial Property, Berezhkovskaya nab., 30-1, Moscow, G-59, GSP-3, Russia, 125993 Facsimile No: (8-495) 531-63-18, (8-499) 243-33-37	Authorized officer Z. Nabieva Telephone No. (499) 240-25-91

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US 2020/041614

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	WO 2009/082821 A1 (GTN ENTERTAINMENT LTD) 09.07.2009	1-15