August 2023

METHOD AND SYSTEM FOR MIGRATING A CODE FROM A FIRST FORMAT TO A SECOND FORMAT

RAGHURAM VISWANADHA
Visa

JUSTIN GEORGE
Visa

SURESH PULIKARA
Visa

SUSHEEL VADLAMUDI
Visa

HARISH RAGHAVENDRA
Visa

See next page for additional authors

Inventor(s)
RAGHURAM VISWANADHA, JUSTIN GEORGE, SURESH PULIKARA, SUSHEEL VADLAMUDI, HARISH RAGHAVENDRA, GIRI KESAVAN, PRANAY VYAS, SOWMYA MOTHEKANI, MANISH KAUSHIK, and PARESH GOSWAMI

METHOD AND SYSTEM FOR MIGRATING A CODE FROM A FIRST FORMAT TO A SECOND FORMAT

VISA

INVENTORS:

RAGHURAM VISWANADHA

JUSTIN GEORGE

SURESH PULIKARA

SUSHEEL VADLAMUDI

HARISH RAGHAVENDRA

GIRI KESAVAN

PRANAY VYAS

SOWMYA MOTHEKANI

MANISH KAUSHIK

PARESH GOSWAMI

TECHNICAL FIELD

[0001] The present subject matter relates to code migration and, more particularly, to a method and system for migrating codes from a first format to a second format.

BACKGROUND

[0002] Code migration is the process of transforming of a programming code from one environment to another. More specifically, the code migration may refer to moving a code from one version of a language to another version, transforming from a source/current language to a target language, or moving from one operating system to another operating system. The most often used code migration is a simple movement of a code from one version of a language to a newer, syntactically different version. aIn many cases, the old version of code would actually work, but new and improved routines or modularization can be improved in the code to fit the nature of the newer version which leads to more efficiency in execution. As such, code migrations are often used to improve performance of electronic devices. Moreover, it is important to update outdated or inefficient codes when converting software languages because doing so will lead to increased execution efficiency.

[0003] Conventionally, code migration process is tedious, as it contains a large number of steps that is performed manually. For example, most data sources have source and destination tables which are recreated in regression environment manually. Moreover, migrating codes from one version to another is not straight forward. The existing methods include using regular expressions to match undesired patterns and change them to desired patterns. However, such implementations are erroneous and not reliable. In an example, if PySpark has to be converted from version 2.0 to PySpark 3.0, then existing techniques include converting the code from version 2.0 to 2.2, 2.2 to 2.3, 2.3 to 2.4 then finally 2.4 to 3.0. Such methods are tedious and contains several steps that are done manually. Moreover, manual updates are inherently error prone and time consuming.

[0004] Moreover, sample queries/scripts are run before and after the maintenance which are triggered manually. Further, developers manually capture the logs, application ID's, run times, set parameters etc., to be documented in a Wiki. However, these checks might include output validation which is missed sometimes. In another example, when leveraging Hive CLI (Common Line Interface), developers use Linux environment variables to pass arguments for Hive Query

Language (HQL). However, moving to Visa Data Platform (VDP), Hive CLI is not supported and all Linux environment variables should be parsed as "--hiveconf" or "--hivevar". As such, developers manually make these changes by editing the HQL's or Shell scripts. As Hive CLI is deprecated in VDP, any custom functions that need to be created on Hive should deploy the related Jar's on HiveServer2 and on to Hadoop Distributed File System (HDFS). Once the jars are uploaded, the permissions for usage should be given to the respective developers for creating functions in a schema. Currently, this is a manual effort to copy to each HiveServer2, HDFS and there is no versioning or tracking/audit. Moreover, as new datasets are being created or requested by developers, they are not accessible to them unless the Ranger permissions are appended to TRE cluster. Permissions may be managed manually and as such, the turnaround time is long considering different scenarios.

[0005] After migration, problems arise when migration relies on Ambari Alerts or service checks to determine the state of service. These tests only cover a few scenarios and aren't comprehensive to identify corner cases (like set parameters/ output validations, etc.). Moreover, these are not audited and recorded at centralized location for validating before or after maintenance across different teams.

[0006] In view of the above, there exists a need for transforming codes from older version to newer version to improve the efficiency and performance of the software application and the electronic device on which the code is executed. Furthermore, it would be advantageous to automate the code migration process to preclude the disadvantages of manual code migration process explained above.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] The accompanying drawings, which are incorporated in and constitute a part of this disclosure, illustrate exemplary embodiments and, together with the description, serve to explain the disclosed principles. In the figures, the left-most digit(s) of a reference number identifies the figure in which the reference number first appears. The same numbers are used throughout the figures to reference like features and components. Some embodiments of device or system and/or

methods in accordance with embodiments of the present subject matter are now described, by way of example only, and with reference to the accompanying figures, in which:

[0008] FIG. 1 illustrates a simplified representation of an environment related to a code migrating system.

[0009] FIG. 2 illustrates a block diagram of the code migration system for implementing embodiments consistent with the present disclosure.

[0010] FIG. 3 depicts a flowchart illustrating a process for transforming a code in a first format to a second format.

[0011] The figures depict embodiments of the disclosure for purposes of illustration only. One skilled in the art will readily recognize from the following description that alternative embodiments of the structures and methods illustrated herein may be employed without departing from the principles of the disclosure described herein.

DESCRIPTION OF THE DISCLOSURE

[0012] In the present document, the word "exemplary" is used herein to mean "serving as an example, instance, or illustration." Any embodiment or implementation of the present subject matter described herein as "exemplary" is not necessarily to be construed as preferred or advantageous over other embodiments.

[0013] While the disclosure is susceptible to various modifications and alternative forms, specific embodiment thereof has been shown by way of example in the drawings and will be described in detail below. It should be understood, however that it is not intended to limit the disclosure to the particular forms disclosed, but on the contrary, the disclosure is to cover all modifications, equivalents, and alternative falling within the spirit and the scope of the disclosure.

[0014] The terms “comprises”, “comprising”, or any other variations thereof, are intended to cover a non-exclusive inclusion, such that a setup, device or method that comprises a list of components

or steps does not include only those components or steps but may include other components or steps not expressly listed or inherent to such setup or device or method. In other words, one or more elements in a device or system or apparatus proceeded by “comprises… a” does not, without more constraints, preclude the existence of other elements or additional elements in the device or system or apparatus.

[0015] The terms "an embodiment", "embodiment", "embodiments", "the embodiment", "the embodiments", "one or more embodiments", "some embodiments", and "one embodiment" mean "one or more (but not all) embodiments of the invention(s)" unless expressly specified otherwise.

[0016] The terms "including", "comprising", “having” and variations thereof mean "including but not limited to", unless expressly specified otherwise.

[0017] The term “code” as used herein refers to a text listing of commands to be compiled or assembled into an executable computer program. Further, the term “migration” as used herein refers to the process of moving an application program from one environment to another. More specifically, the transformation of a code from an older version to a newer version is referred to as migration herein. In an example, PySpark version 3 is faster and more efficient than PySpark 2 and hence, migration of code from PySpark version 2 to version 3 is referred to as code version. For example, a current billing application of a business is updated to a newer version which includes additional features.

[0018] In the following detailed description of the embodiments of the disclosure, reference is made to the accompanying drawings that form a part hereof, and in which are shown by way of illustration specific embodiments in which the disclosure may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the disclosure, and it is to be understood that other embodiments may be utilized and that changes may be made without departing from the scope of the present disclosure. The following description is, therefore, not to be taken in a limiting sense.

[0019] **FIG.1** illustrates a simplified representation of an environment 100, in which at least some example embodiments of the disclosure can be implemented. The environment 100 exemplarily depicts a user 108 controlling a device 110 for developing software applications for a variety of applications. It is noted that the user 108 may use one or more electronic devices, such as a smartphone, a laptop, a desktop, a personal computer, or any spatial computing device to develop codes or perform code migration. For example, the user 108 may work on multiple platforms and/or Operating Systems to develop codes which support various business applications. More specifically, the user 108 manually transforms a code from one version of a language to another version, transforming from a source/current language to a target language, or moving from one operating system to another operating system, etc. As such, the user 108 may retrieve a code in a first format or a first platform from the database 106 and transform to a target platform or target format/second format by manually checking and updating the syntaxes to suit the target format. Some examples of the code may include but not limited to Hive HQL code, spark codes such as PySpark, Java, Scala and the like.

[0020] The transformed code (i.e., target format/ target platform) may be stored in a database 106 via a communication network 102. It is understood that the device 110 may be in operative communication with the communication network 102, such as the Internet, enabled by a network provider, also known as an Internet Service Provider (ISP). The device 110 may connect to the communication network 102 using a wired network, a wireless network, or a combination of wired and wireless networks. Some non-limiting examples of wired networks may include the Ethernet, the Local Area Network (LAN), a fiber-optic network, and the like. Some non-limiting examples of wireless networks may include the Wireless LAN (WLAN), cellular networks, Bluetooth or ZigBee networks, and the like.

[0021] Various embodiments of the present disclosure disclose a code migration system 104 for transforming the code in a first format to a second format. The term ‘first format’ as used herein refers to an older version of the code and the term ‘second format’ as used herein refers to a newer version of the code. In other words, the code migration system 104 automatically transforms the code from an older version to a newer version automatically based on a modified abstract syntax tree. Further, the code migration system 104 facilitates parsing of other environment variables to

incorporate latest changes that are needed for beeline and quick validation. In addition, the code migration system 104 automates code migration by creating a framework to automatically provide services as input parameters and is later used to validate the services and output of these service validations are recorded to understand service state. Moreover, the code migration system 104 automates processes to capture the logs, application ID, output and job counters.

[0022] **FIG. 2** illustrates a block diagram of a code migration system 104 for implementing embodiments consistent with the present disclosure. In an embodiment, the code migration system 104 may be embodied within the device 110 and migration happens locally within the device 110. In another embodiment, the code migration system 104 may be a distributed or a centralized server performing one or more of the operations described herein.

[0023] The code migration system 104 (hereinafter referred to interchangeably as 'system 104') comprises a processor 202, a memory 204, an input/output module 206 and a communication interface 208. It shall be noted that, in some embodiments, the system 104 may include more or fewer components than those depicted herein. The various components of the system 104 may be implemented using hardware, software, firmware or any combinations thereof. Further, the various components of the system 104 may be operably coupled with each other. More specifically, various components of the system 104 may be capable of communicating with each other using communication channel media (such as buses, interconnects, etc.). It is also noted that one or more components of the system 104 may be implemented in a single server or a plurality of servers, which are remotely placed from each other.

[0024] In one embodiment, the processor 202 may be embodied as a multi-core processor, a single core processor, or a combination of one or more multi-core processors and one or more single core processors. For example, the processor 202 may be embodied as one or more of various processing devices, such as a coprocessor, a microprocessor, a controller, a digital signal processor (DSP), a processing circuitry with or without an accompanying DSP, or various other processing devices including, a microcontroller unit (MCU), a hardware accelerator, a special-purpose computer chip, or the like. The processor 202 includes a parser 210, a data source extractor 212 and migration tools 214.

[0025] In one embodiment, the memory 204 is capable of storing machine executable instructions, referred to herein as instructions 205. In an embodiment, the processor 202 is embodied as an executor of software instructions. As such, the processor 202 is capable of executing the instructions 205 stored in the memory 204 to perform one or more operations described herein. The memory 204 can be any type of storage accessible to the processor 202 to perform respective functionalities. For example, the memory 204 may include one or more volatile or non-volatile memories, or a combination thereof. For example, the memory 204 may be embodied as semiconductor memories, such as flash memory, mask ROM, PROM (programmable ROM), EPROM (erasable PROM), RAM (random access memory), etc. and the like.

[0026] In an embodiment, the processor 202 is configured to execute the instructions 205 for : (1) receiving a code in a first format, (2) generating an Abstract Syntax tree (AST) of the code in the first format by parsing, (3) generating a modified AST from the AST, (4) generating a code in a second format from the modified AST.

[0027] In an embodiment, the I/O module 206 may include mechanisms configured to receive inputs from and provide outputs to peripheral devices such as, an operator of the system (i.e., the user 108). To enable reception of inputs and provide outputs to the system 104, the I/O module 206 may include at least one input interface and/or at least one output interface. For example, code in the first format may be provided by the user 108 to the system 104 using the keyboard/mouse and the code in the second format (i.e., target format) may be displayed on the display for the user 108. Examples of the input interface may include, but are not limited to, a keyboard, a mouse, a joystick, a keypad, a touch screen, soft keys, a microphone, and the like. Examples of the output interface may include, but are not limited to, a display such as a light emitting diode display, a thin-film transistor (TFT) display, a liquid crystal display, an active-matrix organic light-emitting diode (AMOLED) display, a microphone, a speaker, a ringer, and the like.

[0028] In an embodiment, the communication interface 208 may include mechanisms configured to communicate with other entities in the environment 100. In other words, the communication interface 208 is configured to access the plurality of source code corresponding to a plurality of

users 108. In an example, the plurality of source code may be stored in the database 106 and the communication interface 208 may access the plurality of source code in the first format for migrating to the second format. In yet another example, the plurality of source code may correspond to codes obtained using the I/O module 206, for example, keyboard. As such, the plurality of source code in the first format are received by the communication interface 208 and sent to the system 104 which performs one or more operations described herein to

[0029] The system 104 is depicted to be in operative communication with the database 106. In one embodiment, the database 106 is configured to store a plurality of codes. The plurality of codes can be but not limited to Hive HQL code, spark codes such as PySpark, Java, Scala and the like. Further the database 106 is configured to store the code in the first format and also store the codes that have been migrated from the first format to the second format. Further, the database 106 may include a plurality of syntactic rules which govern transformation of the Abstract Syntax Tree (AST) to the modified AST supporting the second format or newer format of the same code. An example of a syntactic rule may be a change in an existing operator to add on an additional variable to improve the performance of the code. Apart from the plurality of syntactic rules, the database 106 may include a plurality of transformation rules for transforming each AST to a corresponding modified AST. More specifically, these transformation rules define, for example, a transformation rule may define on converting a AST corresponding to a first version (e.g., PySpark 2.0) to a modified AST corresponding to a second version (e.g., Pyspark 2.6). In another example, a transformation rule may define on converting from a first version (e.g., PySpark 2.0) to a latest/newer version (e.g., PySpark 3.0).

[0030] The database 106 may include multiple storage units such as hard disks and/or solid-state disks in a redundant array of inexpensive disks (RAID) configuration. In some embodiments, the database 106 may include a storage area network (SAN) and/or a network attached storage (NAS) system. In one embodiment, the database 106 may correspond to a distributed storage system, wherein individual databases are configured to store custom information, such as, supported code types, AST structures for every version/format, modified AST structures for different formats/version, data type definition supported for each code, etc.

[0031] In some embodiments, the database 106 is integrated within the system 104. For example, the system 104 may include one or more hard disk drives as the database 106. In other embodiments, the database 106 is external to the system 104 and may be accessed by the system 104 using a storage interface (not shown in FIG. 2). The storage interface is any component capable of providing the processor 202 with access to the database 106. The storage interface may include, for example, an Advanced Technology Attachment (ATA) adapter, a Serial ATA (SATA) adapter, a Small Computer System Interface (SCSI) adapter, a RAID controller, a SAN adapter, a network adapter, and/or any component providing the processor 202 with access to the database 106.

[0032] As already explained, the communication interface 208 is configured to receive a code in the first format. Further, the user 108 may provide a conversion input indicating the format/version to which the code has to be converted to. More specifically, the conversion input specifies the second format which refers to the target format of the code. For example, the conversion input may define that PySpark code 2.0 has to be converted to PySpark code 3.0. The communication interface 208 forwards the code in the first format and the conversion input to the processor 202. The modules of the processor 202 in conjunction with the instructions 216 in the memory 204 are configured to process the code in the first format and generate a code in the second format.

[0033] The operations of the processor 202 for processing the code in the first format to generate a code in the second format is explained as process steps with reference to **FIG. 3**.

[0034] **FIG. 3** depicts a flowchart 300 illustrating a process of migrating a code in a first format to a code in a second format. The steps of the flowchart 300 may be performed by modules of the processor 202.

[0035] At 302, a code in the first format is received by a system such as, the code migration system 104. Some examples of the code, include, but not limited to, Hive HQL code, spark codes such as PySpark, Java, Scala and the like. The code is in an older version and needs to be updated to a newer version (i.e., second format) specified by the user 108.

[0036] At 304, a code in the first format is parsed using the parser 210 to generate an Abstract Syntax tree (AST). The term “parser” as used herein refers to a computer program that breaks down text into recognized strings of characters for further analysis. More specifically, a code specific parser may be used to analyze the syntax of the code to generate the AST. In an example, if the code is in PySpark, then Python parser is used to convert the code into the AST. The term “Abstract Syntax tree (AST)” as used herein refers to a tree representation of the abstract syntactic structure of source code written in a formal language. Each node of the tree denotes a construct occurring in the source code. The syntax is said to be "abstract" when it does not represent every detail appearing in the real syntax, but rather just the structural or content-related details. In yet another example, if the code in the first format is in HiveQL, HQL scripts are parsed, and the tables are identified to be replicated to TRE.

[0037] In an embodiment, the parser 210 is configured to parse and modify any HQL including Linux environment variables to incorporate latest changes that are needed for beeline and a quick validation to ensure the script can be compiled and executed on beeline before providing the modified HQL for the user 108. This ensures the user 108 is not manually making such changes by editing the HQL's or Shell scripts.

[0038] During parsing, there are several steps involved. In an example, if the source code is in HiveQL, the steps for modification/ migration may include but not limited to (1) data source extraction and migration, (2) application testing framework, (3) Hive variables migration, (4) Hive Jar migration, (5) dataset format migration and the like.

[0039] As part of parsing, the data source extractor 212 is configured to identify one or more tables from the parsed data to be replicated to TRE. More specifically, data sources from SQL/Shell or Python Scripts are identified to expedite creation of tables on Tusker environment and copy relevant data from non-production environments to TRE in an automated fashion. After the identification, the metadata of the tables are captured and created on TRE through a script and subsequently the data is copied from non-production environment to TRE using distcp. In some cases, non-production environments may not have these tables as the application is deployed in production. In such scenarios, production data should be identified, and Secure Data Service (SDS)

service should be leveraged to mask and de-identify the data so it can be copied to TRE environment or TRE CERT environment.

[0040] Further, as Hive CLI is deprecated in VDP, any Custom functions that needs to be created on Hive should deploy the related Jar's on HiveServer2 and on to HDFS and once the jars are uploaded the permissions for usage is given to the respective user for creating functions in a schema. Therefore, to migrate Hive Jar, a script is built to perform jar validation and deployment to all HiveServer2's and HDFS and ensure it is synchronous at all the places. New datasets are being created or requested by users 108 during migration, but they are not accessible for them unless the Ranger permissions are appended on TRE cluster. The processor 202 is configured to automate this process to automatically provide permission for the user 108.

[0041] At 306, a modified AST is generated based on the AST of the code in the first format by the processor 202. More specifically, migration steps of each stage are loaded initially. For example, if the code is in PySpark 2.0 format and has to be migrated to PySpark 3.0, then the migrations steps are initialized for each stage, i.e., migration from PySpark 2.0 to PySpark 2.2, migration from PySpark 2.2 to PySpark 2.4, migration from PySpark 2.4 to PySpark 2.6, migration from PySpark 2.6 to PySpark 2.8 and migration from PySpark 2.8 to PySpark 3.0. Thereafter, each node in the AST is analyzed to identify if a migration is required for migration in each of the above-mentioned steps. If migration is applicable for a node, then the node is migrated to the next version (i.e., the second format). Similarly, all nodes of the AST are analyzed and nodes which require migration alone are transformed to the second format to generate the modified AST. Similarly, all nodes of the AST are analyzed to generate the modified AST.

[0042] At 308, a code in the second format is generated from the modified AST by the migration tools 214 of the processor. More specifically, data stored in Txt format on the AST is migrated to Parquet format using the migration tools 214. The migration tools 214 reads the data in txt format, infers the schema and generates Parquet format files. After receiving the code in the second format, the migration tools 214 also validates the generated code. A framework is created by the migration tools 214 to provide services as input parameters that can be used to validate the services including the corner scenarios. It also records the output of these service validations by date and time to

confirm success and enable point in time travel to understand service state. Another example of validation may include but not limited to running some sample queries/scripts before and after the generation of the code in the second format. These queries are submitted through an automated script which will capture the logs, application ID, output, job counters and the like. The script specifies a pre/post upgrade argument so it can be compared with as part of validation.

[0043] As part of migration, the system 104 automatically runs some sample queries/scripts before and after the maintenance. The system 104 will assist users 108 to submit these queries through an automation script which will capture the logs, application ID, output and job counters. The script will also specify pre upgrade and post upgrade arguments so that it can be recorded and compared with as part of validation.

[0044] Some advantages achieved by the present disclosure are reduced time taken to migrate code, and reducing the effort required to migrate the code. Moreover, automation of such code migration creates a framework to provide services as input parameters that can be used to validate the services including but not limited to corner scenarios which failed previously and record the output of these service validations by date and time to confirm pass/fail and enable point in time travel to understand service state. In addition, the migration tools use automated scripts which will capture the logs, application ID, output and job counters. Further, automation scripts also specify pre/post upgrade arguments so which can be recorded and compared as part of validation. Further, parsing of other environment variables such as, Linux environment variables is enabled, and modified to incorporate latest changes that are needed for beeline and a quick validation. Such, automations ensure the script can be compiled and executed on beeline before providing the modified HQL for the users. Jar migration which is currently performed manually is automated by building a script to perform the jar validation and deployment to all HiveServer2's and HDFS and ensure it is synchronous at all the places. Further, ranger permissions are automated such that users can request access to new datasets that are created. In another example, automated tools are written to read data in txt format, infer the schema and generate Parquet format files. Therefore, the time taken to perform these steps manually is reduced.

[0045] Furthermore, one or more computer-readable storage media may be utilized in implementing embodiments consistent with the present disclosure. A computer-readable storage medium refers to any type of physical memory on which information or data readable by a processor may be stored. Thus, a computer-readable storage medium may store instructions for execution by one or more processors, including instructions for causing the processor(s) to perform steps or stages consistent with the embodiments described herein. The term "computer-readable medium" should be understood to include tangible items and exclude carrier waves and transient signals, i.e., be non-transitory. Examples include Random Access Memory (RAM), Read-Only Memory (ROM), volatile memory, non-volatile memory, hard drives, Compact Disc (CD) ROMs, DVDs, flash drives, disks, and any other known physical storage media.

[0046] The described operations may be implemented as a method, system or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The described operations may be implemented as code maintained in a "non-transitory computer readable medium", where a processor may read and execute the code from the computer readable medium. The processor is at least one of a microprocessor and a processor capable of processing and executing the queries. A non-transitory computer readable medium may include media such as magnetic storage medium (e.g., hard disk drives, floppy disks, tape, etc.), optical storage (CD-ROMs, DVDs, optical disks, etc.), volatile and non-volatile memory devices (e.g., EEPROMs, ROMs, PROMs, RAMs, DRAMs, SRAMs, Flash Memory, firmware, programmable logic, etc.), etc. Further, non-transitory computer-readable media may include all computer-readable media except for a transitory. The code implementing the described operations may further be implemented in hardware logic (e.g., an integrated circuit chip, Programmable Gate Array (PGA), Application Specific Integrated Circuit (ASIC), etc.).

[0047] The illustrated steps are set out to explain the exemplary embodiments shown, and it should be anticipated that ongoing technological development will change the manner in which particular functions are performed. These examples are presented herein for purposes of illustration, and not limitation. Further, the boundaries of the functional building blocks have been arbitrarily defined herein for the convenience of the description. Alternative boundaries can be defined so long as the

specified functions and relationships thereof are appropriately performed. Alternatives (including equivalents, extensions, variations, deviations, etc., of those described herein) will be apparent to persons skilled in the relevant art(s) based on the teachings contained herein. Such alternatives fall within the scope and spirit of the disclosed embodiments. Also, the words "comprising," "having," "containing," and "including," and other similar forms are intended to be equivalent in meaning and be open ended in that an item or items following any one of these words is not meant to be an exhaustive listing of such item or items or meant to be limited to only the listed item or items. It must also be noted that as used herein, the singular forms “a,” “an,” and “the” include plural references unless the context clearly dictates otherwise.

[0048] Furthermore, one or more computer-readable storage media may be utilized in implementing embodiments consistent with the present disclosure. A computer readable storage medium refers to any type of physical memory on which information or data readable by a processor may be stored. Thus, a computer readable storage medium may store instructions for execution by one or more processors, including instructions for causing the processor(s) to perform steps or stages consistent with the embodiments described herein. The term “computer readable medium” should be understood to include tangible items and exclude carrier waves and transient signals, i.e., are non-transitory. Examples include random access memory (RAM), read-only memory (ROM), volatile memory, non-volatile memory, hard drives, CD ROMs, DVDs, flash drives, disks, and any other known physical storage media.

[0049] Finally, the language used in the specification has been principally selected for readability and instructional purposes, and it may not have been selected to delineate or circumscribe the inventive subject matter. Accordingly, the disclosure of the embodiments of the disclosure is intended to be illustrative, but not limiting, of the scope of the disclosure.

[0050] With respect to the use of substantially any plural and/or singular terms herein, those having skill in the art can translate from the plural to the singular and/or from the singular to the plural as is appropriate to the context and/or application. The various singular/plural permutations may be expressly set forth herein for sake of clarity.

METHOD AND SYSTEM FOR MIGRATING A CODE FROM A FIRST FORMAT TO A SECOND FORMAT

Abstract

The present disclosure relates to a migration tool to convert a code in a first format to a code in a second format. The source code received can be from a user (108) or from a database (106). The input code is said to be in the first format. An Abstract Syntax tree (AST) of the code in the first format is generated by a system (104) by parsing the code. A modified AST is generated based on the AST of the code in the first format. A code in the second format is generated from the modified AST.

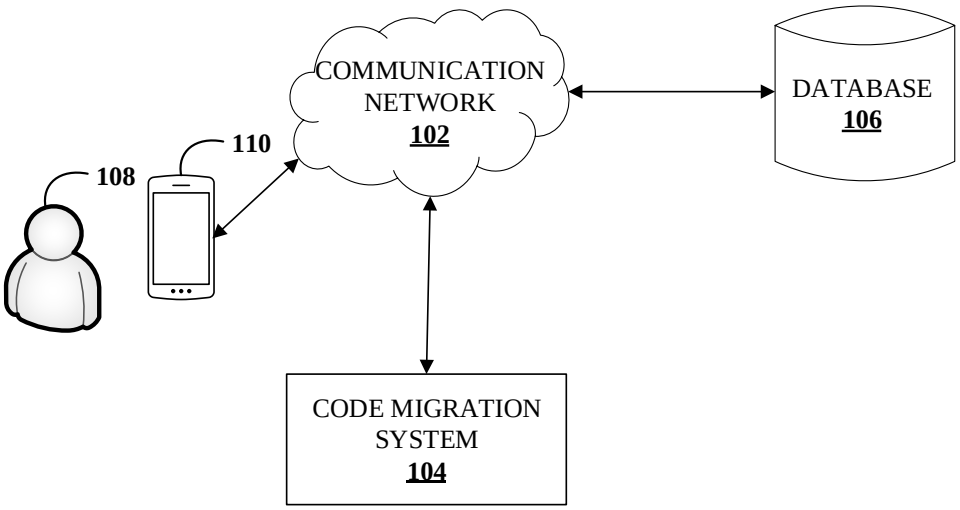
COMMUNICATION NETWORK
102
DATABASE
106
110
108
CODE MIGRATION SYSTEM
104

FIG. 1

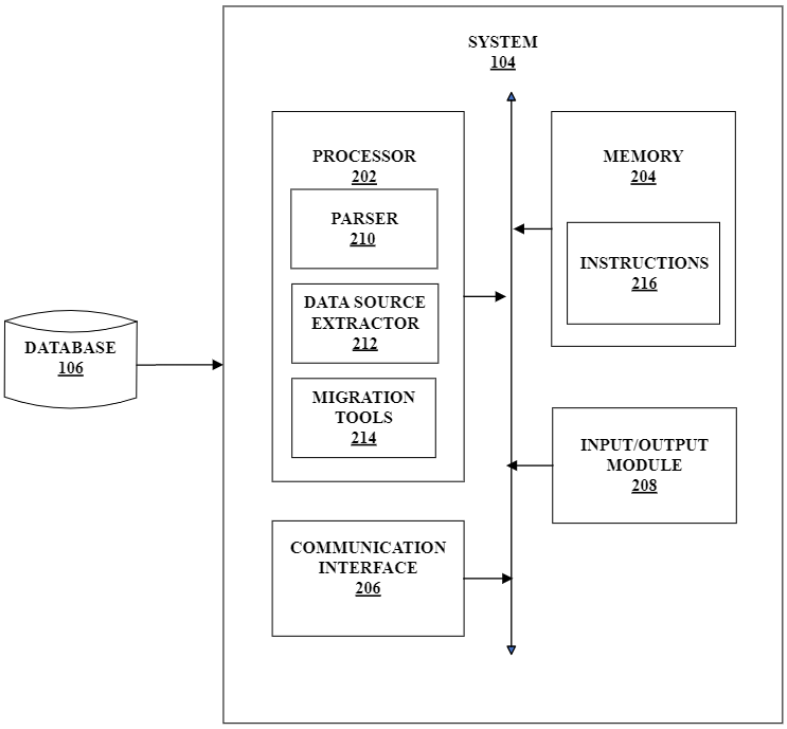
SYSTEM
104
PROCESSOR
202
PARSER
210
DATA SOURCE EXTRACTOR
212
MIGRATION TOOLS
214
MEMORY
204
INSTRUCTIONS
216
INPUT/OUTPUT MODULE
208
COMMUNICATION INTERFACE
206
DATABASE
106

FIG. 2

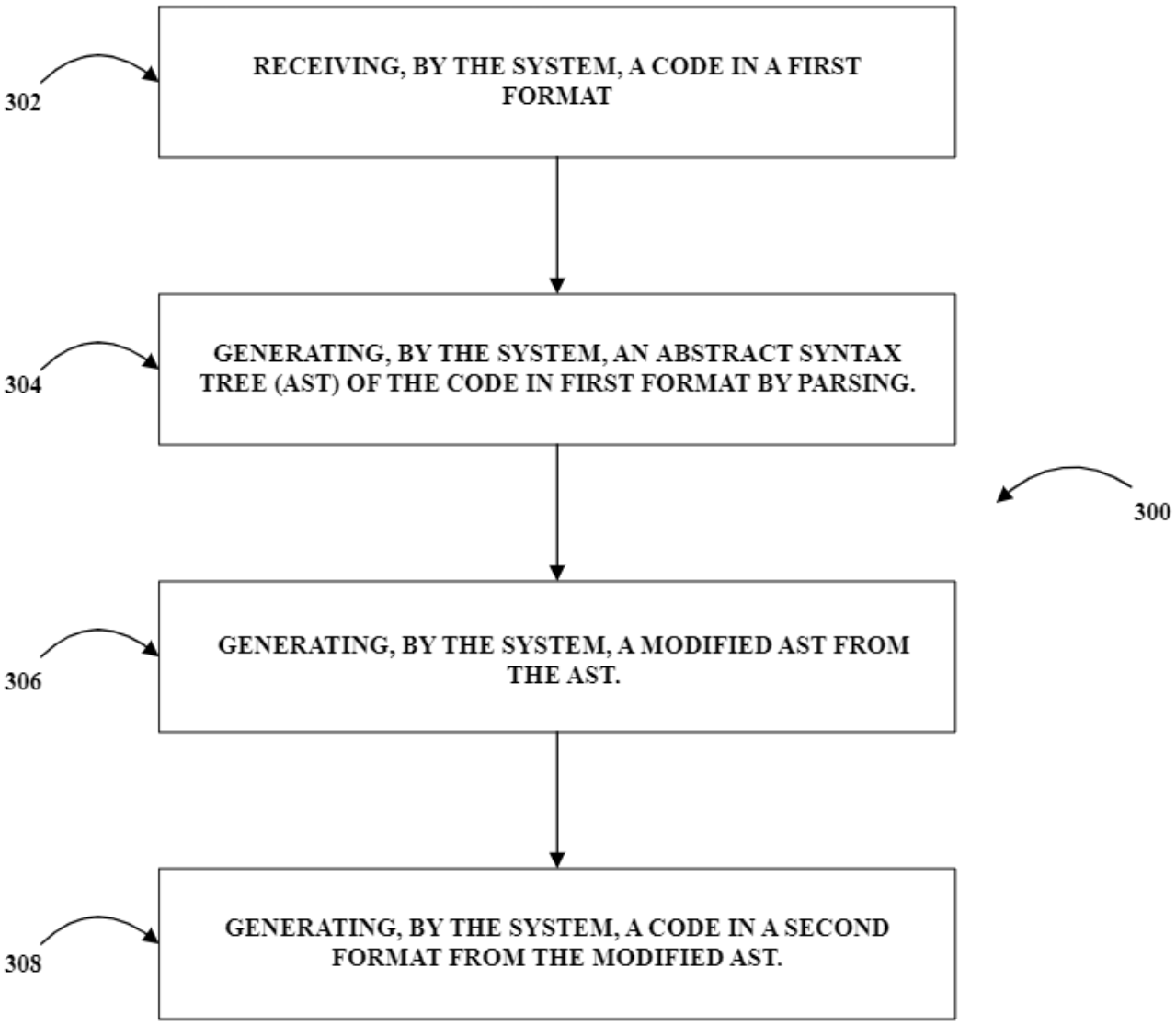
RECEIVING, BY THE SYSTEM, A CODE IN A FIRST FORMAT
302
GENERATING, BY THE SYSTEM, AN ABSTRACT SYNTAX TREE (AST) OF THE CODE IN FIRST FORMAT BY PARSING.
304
300
GENERATING, BY THE SYSTEM, A MODIFIED AST FROM THE AST.
306
GENERATING, BY THE SYSTEM, A CODE IN A SECOND FORMAT FROM THE MODIFIED AST.
308

FIG. 3