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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
6 February 2014 (06.02.2014)

WIPO | PCT

(10) International Publication Number
WO 2014/019093 A1

(51) International Patent Classification:
G06F 9/44 (2006.01)

(21) International Application Number:
PCT/CA2013/050599

(22) International Filing Date:
1 August 2013 (01.08.2013)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/678,395 1 August 2012 (01.08.2012) US

(71) Applicant: **SHERPA TECHNOLOGIES INC.** [CA/CA]; 527 Marie-C-Daveluy, Boisbriand, Québec J7G 3G7 (CA).

(72) Inventor: **MÉNARD, Éric-Pierre**; 527 Marie-C-Daveluy, Boisbriand, Québec J7G 3G7 (CA).

(74) Agent: **ROBIC, LLP**; 1001, Square-Victoria Bloc E - 8th floor, Montréal, Québec H2Z 2B7 (CA).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

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

— *with amended claims and statement (Art. 19(1))*

(54) Title: SYSTEM AND METHOD FOR MANAGING VERSIONS OF PROGRAM ASSETS

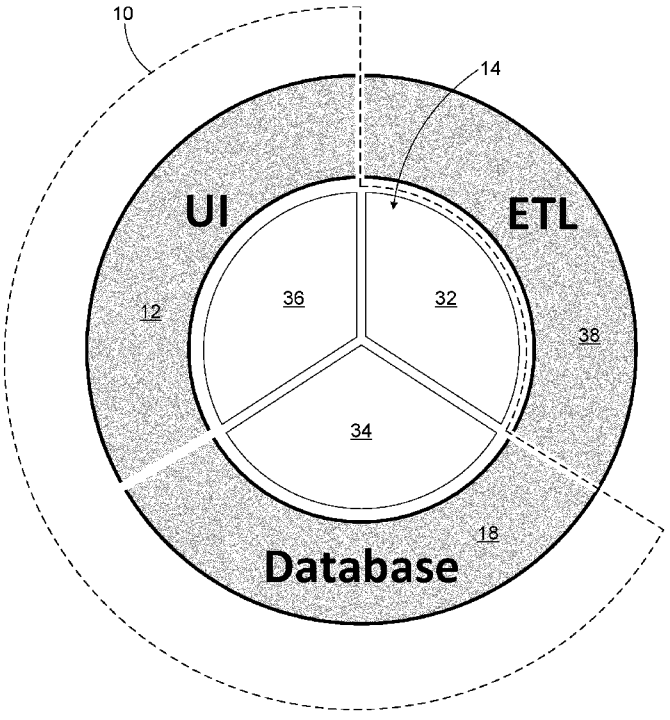


FIG. 6

(57) Abstract: A method and system for managing versions of program assets of a library is disclosed, to be used for example with IBM *Infosphere Datastage*™. Each program asset has source code which is protected. A selection of one or more program asset to be exported into the utility application is selected. Instructions for building the source code of each program asset is extracted from the library and into a digest. A database stores each digest as a new instance of the digest in a data storage and associates thereto a new version identifier representing a new version of the corresponding program asset. A checked-in status is further associated to each new instance of digest, to indicate that the digest is stored in the utility application.

WO 2014/019093 A1

SYSTEM AND METHOD FOR MANAGING VERSIONS OF PROGRAM ASSETS

Field of the invention:

The present invention relates to a version control system and method. More particularly, the present invention relates to a version control method for controlling versions of protected source code and to a system for performing the same.

Background of the invention:

Source control, also known as revision control or version control, is an important practice of software development. It allows for the management of changes to documents and programs, by registering the source code at each change, and also provides developers a variety of functionalities, including the reservation of files by means of a check-in, check-out procedure and can also handle conflicts between simultaneous changes of the same program ("merging").

Release management, in software development, automates and/or allows better control of the deployment and maintenance of all the different versions of programs through the evolutionary phases, such as development, testing and production environments

Extract-Transform-Load (ETL) is a field of information technology that handles the transportation and integration of data. ETL programs make possible the transmission of data between various computer systems such as sending billing information to an application responsible of invoicing, from a product sold using a customer relationship management application (CRM). ETL programs are also heavily used in loading data warehouses and when replacing outdated computer systems by new technology that requires preserving relevant data accumulated throughout the years in the older system.

IBM Infosphere Datastage™ (also referred to herein as "Datastage™") is a component of the IBM Information Server™ suite of applications, and is recognized worldwide as a leader in the field of ETL. The latter is widely distributed throughout North America, Europe and Asia.

Version control and release management practices are widely spread in the IT community. There are to date more than two dozen unique solutions, with as many offered under free licence as paid proprietary licenses. IBM Rational ClearCase™, CVS™, Subversion™, Microsoft Team Foundation Server™ and Git™ are among the best known.

Despite the multitude of applications available, no software known to the Applicant is adapted to integrate programs such as those created by DataStage™, due to the complexity and uniqueness of its architecture. While modern programming is mostly text-based and usually consisting of several independent text files, each of which can be accessed and saved individually (Java™, PHP, C/C++, etc.), DataStage™ on the other hand is a graphical tool (see FIG. 1). Template modules representing functions are dragged to the design screen from a palette and are linked together to be finally customized for specific needs. Behind the scenes, the actual code is separated into design files, executable binaries and metadata stored in a database. All those artifacts compose a single program. Those components are write-protected by Datastage™ so as to prevent direct access. In such an environment, modifications to programs must be done via an application layer of Datastage™.

It is however possible to manually extract a summary of each program composition into either an XML format file or a file format proprietary to DataStage™, called "DSX". This summary can then be used by Datastage™ to recreate a program in its original form. This is the most common practice today for managing Datastage™ programs. Users export each component either individually or as a bundle into a

processing summary file. This file is then uploaded into a source management program. When an archived version of a program is required in a Datastage™ project, the appropriate file is extracted from the source management program and then manually imported into the project. This is a tedious task which, since it requires manual manipulations, increases the risk of errors.

Shown in FIG. 2A and 2B are two flow charts illustrating the manual versioning steps required, namely FIG. 2A exemplifies the exporting of a program from a Datastage™ project, and FIG. 2B exemplifies the importing of a program into a target Datastage™ project (i.e. recreating the program in Datastage™). FIG. 3 illustrates the data flow between the Datastage™ environments using a conventional source control application.

DataStage™ does provide some level of automation for extracting and importing of programs. DataStage™ provides an implementation of certain key controls by various DOS or UNIX commands, and gives access via an application program interface (API) that allows C / C++ programmers to access a limited number of methods of the program.

With release 8.5 of the IBM Information Server™ suite, features were added to the DataStage™ application, allowing the check-in and check-out of source code into two source control applications: IBM Rational ClearCase™ and Concurrent Versions System™ (CVS), directly from the graphical user interface (GUI) of DataStage™. However, this feature does not serve as a release management application as it does not allow for example the deployment of packages or bundles of programs, from the release management application itself.

IBM has recently developed an application suite called Jazz Rational Team Concert™ or Jazz RTC™ (http://jazz.net) whose mission is to enable closer collaboration between the various units of a development team such as business analysts,

architects, developers and other manager types. Jazz RTC™ contains several modules, including one for managing source control and release management. However, this application has been designed for common text-based programming, as for previously stated solutions, and is therefore not readily integrated with DataStage™.

As ETL programming is a particular niche of information technology and as software source control and release management applications are designed to handle the integration of a wide range of applications, no custom module fitted for a single program such as DataStage™ is known to the applicant.

Hence, in light of the aforementioned, there is a need for an improved system which, by virtue of its design and components, would be able to overcome some of the above-discussed prior art concerns.

Summary of the invention:

The object of the present invention is to provide a solution which better integrates write-protected and/or complex programs, such as DataStage™, in a suite of release management and source control, and is thus an improvement over other related version control or release management systems and/or related methods known in the prior art.

In accordance with the present invention, the above mentioned object is achieved, as will be easily understood, by a version control system and method such as the one briefly described herein and such as the one exemplified in the accompanying drawings.

In accordance with an aspect of the invention, there is provided a method for managing versions of program assets of a library, each of said program assets having source code which is protected, the method being executable by a single utility application having an integration module which is embedded in a processor, the method comprising the steps of:

i) receiving a selection of one or more program asset to be exported into the utility application for storage;

ii) extracting from the library and into a digest, for each of the one or more program asset selected, instructions for building the source code of the corresponding program asset, by means of the integration module;

iii) storing, by means of the integration module, each digest as a new instance of the digest in a data storage;

iv) associating in the data storage, by means of the integration module, a new version identifier to each new instance of digest, the new version identifier representing a new version of the corresponding program asset; and

v) in the data storage, associating a checked-in status to each new instance of digest stored at step (iii), by means of the integration module, to indicate that each of said new instance of digest is stored in the utility application.

In a particular embodiment of the above-mentioned aspect, the data storage comprises a plurality of said digests, each digest comprising instructions to rebuild a corresponding program asset in the library, the method further comprising:

vi) receiving, via a user interface, a selection of one or more of said program assets to be imported into the library and the corresponding version information;

vii) retrieving an instance of the digest from the data storage for each of said one or more program asset to be imported, by means of the integration module, being associated to the version information received at step (vi); and

viii) for each digest retrieved at step (vii), executing the instructions to rebuild the corresponding program asset, by means of the integration module, in order to import a new version of the corresponding program asset into the library.

ix) in the data storage, replacing a checked-in status associated each instance of the digest retrieved at step (vii) with a checked-out status, by means of the integration module, to indicate that the corresponding one or more program asset is currently being updated.

In another particular embodiment of the above-mentioned aspect, the data storage comprises a plurality of said digests, each digest comprising instructions to rebuild a corresponding program asset in the library, the data storage storing multiple instances of at least one of the digests, each instance corresponding to a version of the corresponding program asset, the method further comprising:

- receiving a selection of two or more digest instances of the data storage and corresponding version identifier, to be compared;
- retrieving from the data storage the instances of the digest corresponding to the selection received;
- by means of the integration module, comparing the content of the digest instance, to generate comparison information; and
- returning the comparison information on a user interface component.

In accordance with another aspect of the present invention, there is provided a system for managing versions of program assets of a library, each of said program assets having source code which is protected, the system comprising:

- a user interface for receiving a selection of one or more program asset to be exported into a utility application for editing;
- an integration module embedded in a processor which is in communication with the user interface, the integration module comprising an exportation module for extracting from the library into a digest, for each of the one or more

program asset selected, instructions for building the source code of the corresponding program asset; and

- a data storage, in communication with the integration module, for storing each digest as a new instance of the digest, and for associating a new version identifier to each new instance of digest, the new version identifier representing a new version of the corresponding program asset, and for further associating a checked-in status to each new instance of digest stored to indicate that each of said new instance of digest is stored in the utility application.

In accordance with another aspect of the present invention, there is provided a storage medium for managing versions of program assets of a library, each of said program assets having source code which is protected, the storage medium being processor-readable and non-transitory, the storage medium comprising instructions for execution by a processor, via a single utility application, to:

i) receive a selection of one or more program asset to be exported into the utility application for storage;

ii) extract from the library and into a digest, for each of the one or more program asset selected, instructions for building the source code of the corresponding program asset, by means of the integration module;

iii) store, by means of the integration module, each digest as a new instance of the digest in a data storage;

iv) associate in the data storage, by means of the integration module, a new version identifier to each new instance of digest, the new version identifier representing a new version of the corresponding program asset; and

v) associated, in the data storage, a checked-in status to each new instance of digest stored at (iii), by means of the integration module, to indicate that each of said new instance of digest is stored in the utility application.

Program asset export ("check-in" to the version control system)

In accordance with another aspect of the invention, there is provided a method for exporting a program asset from an extract-transform-load (ETL) library storing a plurality of said program assets, each program asset being protected in the ETL library, the method comprising steps of:

a) receiving, via a user interface, a command for exporting said program asset;

b) exporting, by means of an integration module, the program asset from the ETL library into a digest, the digest comprising instructions for rebuilding the program asset in the ETL library;

c) storing, by means of the integration module, a new instance of the digest in the data storage;

d) associating in the data storage, by means of the integration module, a new version to said new instance of the digest; and

e) by means of the integration module, setting a checked-in status to the new instance of the digest in the data storage.

In a particular embodiment of the above-mentioned aspect, step (d) of the method includes:

- querying the data storage to locate an instance of the digest being associated to a latest version of the digest; and
- if no instance of the digest is located in the data storage, said new version is a first version, and otherwise, said new version is obtained by incrementing an originating version associated to the digest.

In accordance with particular embodiments of the present invention, rules are defined in the integration modules which increment the version based on allowed increases. For example, when a version to check-in is the highest, major updates increment the first digit (1.0 to 2.0), while minor updates update the second digit (3.3 to 3.4). When checking-in an intermediate version, a major update upgrades the second digit (4.1.2 to 4.2.0) and minor updates increment the third digit (5.3.4 to 5.3.5). A fourth level of change could be implemented on customer request for specific needs.

In a particular embodiment of the above-mentioned aspect, instances of digests are organized in a tree defining branches, each branch for a given digest representing a subset of versions of the corresponding program asset. In this particular embodiment, the method further includes prior to step (d), receiving branch information identifying a selected branch in the data storage to which the new instance of the digest is to be associated to, and said new version of step (d) is assigned based on said selected branch.

In accordance with another aspect of the present invention, there is provided a method for exporting one or more program asset from an ETL library storing a plurality of said program assets, each program asset being protected in the ETL library, the method comprising steps of:

a) receiving, via a user interface, a command for exporting said one or more program asset;

b) exporting, by means of an integration module, the one or more program asset from the ETL library into respective one or more digest, each digest comprising instructions for rebuilding the corresponding program asset in the ETL library;

c) storing, by means of the integration module, a new instance of each of the one or more digest in the data storage;

d) associating in the data storage, by means of the integration module, a new version to each of said new instance; and

e) by means of the integration module, setting a checked-in status to the new instance of each of the one or more digest in the data storage.

In accordance with another aspect of the present invention, there is provided a version control system for an ETL library adapted to store a plurality of protected program assets, each of the protected program assets being exportable in the format of a digest of instructions for rebuilding the corresponding program asset, the version control system comprising:

- a user interface for exchanging information with a user;
- a data storage for storing instances of said digests of program assets and corresponding version information; and
- an integration module being in communication with the user interface, with the storage module and with the ETL library, in order to generate a digest from said ETL library upon receiving a corresponding command from the user interface, to generate corresponding version information and to store said digest and version information in the data storage.

Program asset import ("check-out" from the version control system)

In accordance with another aspect of the present invention, there is provided a method for importing a versioned program asset into an ETL library from a data storage, said program asset being buildable in the ETL library from a corresponding digest of instructions, one or more instance of said digest being stored in the data storage, each instance being associated to a version of the digest, the method comprising steps of:

a) receiving, via a user interface, a command for importing said program asset;
b) receiving, via the user interface, version information of the program asset to be imported;
c) retrieving an instance of the digest from the data storage, by means of an integration module, in accordance with the version information received at step (b);
d) by means of the integration module, setting a checked-out status to the instance of the digest in the data storage; and
e) executing the instructions of said instance of the digest (to build the corresponding program), by means of the integration module, in order to import the corresponding program asset in the ETL library.

In a particular embodiment of the above-mentioned aspect, the method further comprises after step (c), validating whether said instance of digest retrieved at step (c), has a checked-out status, and only if the program asset does not have a checked-out status, proceeding to the following steps of the method.

In a particular embodiment of the above-mentioned aspect, instances of digests are organized in a tree defining branches, each branch for a given digest representing a subset of versions of the corresponding program asset. In this particular embodiment, the version information received at step (b) further includes branch information, and the retrieving of step (c) takes into account the branch information.

Package Creation and Deployment

In accordance with another aspect of the present invention, there is provided a method for importing a package of versioned program assets into an ETL library from a data storage, each of said program asset being buildable in the ETL library from a corresponding digest of instructions, one or more instance of said digest being stored in the data storage, each instance being associated to a version of the digest, the method comprising steps of:

a) receiving, via a user interface, a command for importing a new package;
b) receiving, via the user interface, the program assets to be imported via the new package and corresponding version information;
c) generating the new package in the data storage, by means of the integration module;
d) retrieving from the data storage, instances of the digests corresponding to the program assets to be imported, by means of the integration module, in accordance with the version information received at step (b);
e) associating in the data storage, the instances retrieved at step (d) with the new package;

f) by means of the integration module, setting a deployment status to the new package in the data storage; and

g) executing the instructions of each of said instances associated to the new package, by means of the integration module, in order to import the corresponding program assets in the ETL library.

In a particular embodiment of the above-mentioned aspect, the one or more instance of the digest are grouped by branches in the data storage, each branch corresponding to a subset of versions of the digest. In this particular embodiment the version information received at step (b) further includes branch information, and the retrieving of step (c) is takes into account the branch information.

Version Comparison

In accordance with another aspect of the present invention, there is provided a method for comparing versions of a given program asset in an ETL library, the given program asset being protected and buildable from a digest of instructions stored in a data storage, the data storage storing multiple instances of the digest, each instance corresponding to a version of the given program asset, the method comprising steps of:

a) receiving, via a user interface, instructions to compare two versions of said given program asset of the ETL library;

b) retrieving from the data storage, by means of an integration module, two instances of the digest corresponding to said two versions of said given program asset;

c) by means of an integration module, generating comparison information, by pairing matching components of the two instances; and

d) returning, by means of the integration module, the comparison information on the user interface.

Terminology

In the context of the present invention, a "program asset" (also referred to herein as an "asset" or "component") may be a DS job (Datastage™ program), a routine, a data connection, and/or any other unitary component that may be exported from the ELT library (example: Datastage™) and versioned independently.

In the context of the present invention, each of said "integration module", "ETL library" and "data storage" is located on a server or a plurality of server(s). It is to be understood that two or more of said "integration module", "ETL library" and "database" may share one or more same server(s).

An "ETL library", in the context of the present invention, refers to an ETL system such as the Datastage™ tool, for example, including the program assets it defines for a given project within a particular development environment (development, testing, production, etc.). In the context of Datastage™, program assets are each defined by a plurality of "artifacts" which may include source code, an object, an instruction, a graphical component, etc. in the form of a file, table, a pointer or reference, or portion thereof for example, which read-protected and write-protected.

A "digest" (also referred to herein as "summary"), in the context of the present invention, may be a file or group of files and/or the like, comprising a set of instructions to build an instance of the corresponding program asset in the ETL library. Thus, with said digest, an instance of the program asset is built in a format which can be independently stored by a user (i.e. a developer).

In the context of the present invention, the expressions "source control", "revision control", "version control", "release management", "source management program", "source control application", "source program", and/or the like, as well as compound terms thereof, are used interchangeably.

Other aspects of the invention

In accordance with another aspect of the invention, there is provided a method for exporting a program component from a library of program components, the library storing artefacts, each program component being defined by a plurality of said artefacts, the method comprising steps of:

a) extracting from the library, a digest of the artifacts being associated to the program component to be exported, the digest comprising instructions for rebuilding the program component in the library;
b) storing the digest in a data storage; and
c) associating version data to said digest in the data storage, said version data being indicative of a new version of the program component.

In a particular embodiment of the present invention, the steps of the above-method are performed by means of an integration module being in communication with the library, the data storage and the user interface.

In accordance with another aspect of the invention, there is provided a version control method for a library of protected program components, each program component being convertible into a digest comprising instructions for building the corresponding program component, the method comprising steps of:

a) generating a digest of one of said program components, the digest comprising instructions for rebuilding the program component in the library;
b) storing the digest in a data storage; and
c) associating version data to said digest in the data storage, said version data being indicative of a new version of the program component.

In accordance with another aspect of the invention, there is provided a version control system for controlling versions of a program component of a library of said program

components, each program component being protected in the library and being further convertible into a digest comprising instructions for building the corresponding program component, said version control system comprising:

a) a user interface for exchanging data with a user;

b) a data storage for storing version data related to said program component of the library;

c) an integration module being in communication with the user interface for receiving a user command to generate a new version of one of said program components, the integration module being in communication with the library of program components for extracting therefrom an instance of a digest corresponding to said program component and for associating thereto a new version, the integration module being further in communication with the data storage for storing therein said instance of the digest and the new version.

In accordance with yet another aspect of the invention, there is provided a version control system for controlling versions of program components of a library of said program components. Each program component is either protected in the library or defined by a plurality of artifacts accessible by the library. Each program component is further convertible into a digest of instructions for rebuilding the corresponding program component in the library. The version control system comprises:

a) a user interface for exchanging data with a user;

b) a data storage for storing instances of digests corresponding to the program components, and for storing version data related each instance of said digest, each instance of said digest representing a version of said program component of the library;

c) an integration module being in communication with the user interface for receiving a user command and with the data storage in order to interact with the data storage, based on the user command.

In accordance with another embodiment of the present invention, there is provided a computer readable storage medium having stored thereon, data and instructions for performing one or more of the above-mentioned methods.

The objects, advantages and features of the present invention will become more apparent upon reading of the following non-restrictive description of preferred embodiments thereof, given for the purpose of exemplification only, with reference to the accompanying drawings.

Brief description of the drawings:

FIG. 1 is a screen shot of graphical components defining a program in the Datastage environment, in accordance with the prior art.

FIG. 2A is a flow chart showing the manual steps carried out in exporting a Datastage™ program, in accordance with the prior art.

FIG. 2B is a flow chart showing the manual steps carried out in importing a program into a Datastage™ project, in accordance with the prior art.

FIG. 3 is a bloc diagram illustrating a data flow between the Datastage™ environments and a source control application, in accordance with the prior art.

FIG. 4 is a schematic diagram showing a three-tier architecture of a version control system, namely, a user interface, a coordinating module (or "logical layer") and database, in accordance with an embodiment of the present invention.

FIG. 5 is a schematic diagram showing a Linux-Apache-MySQL-PHP (LAMP) configuration of the user interface shown in FIG. 4.

FIG. 6 is a schematic diagram representing an ETL axis, a user interface axis and a database axis of the version control system shown in FIG. 4.

FIG. 7 is a hierarchical class diagram showing classes and subclasses of the ETL axis represented in FIG. 6.

FIG. 8 is a hierarchical class diagram showing classes and subclasses of the database axis represented in FIG. 6.

FIG. 9 is a data model showing the tables of the database represented in FIG. 6.

FIG. 10 is a sequence diagram of steps performed by the version control system, for checking-in a component, according to an embodiment of the present invention.

FIG. 11 is a sequence diagram of steps performed by the version control system, for checking-out a component, according to an embodiment of the present invention.

FIG. 12 is a sequence diagram of steps performed by the version control system, for creating and deploying a package, according to an embodiment of the present invention.

FIG. 13 is a sequence diagram of steps performed by the version control system, for comparing versions of a component, according to an embodiment of the present invention.

FIG. 14 is a bloc diagram of a system in accordance with an embodiment of the present invention.

Detailed description of preferred embodiments of the invention:

In the following description, the same numerical references refer to similar elements. The embodiments mentioned and/or configurations and architecture shown in the figures or described in the present description are embodiments of the present invention only, given for exemplification purposes only.

Broadly described, the present invention according to a preferred embodiment thereof, as exemplified in the accompanying drawings, is a version control system for a *IBM Infosphere Datastage*™ framework.

As better illustrated in FIG. 4, the version control system **10**, in accordance with an embodiment of the present invention, is designed following a three-tier architecture, namely comprising: a user interface **12** (also referred to herein as "UI"), a logical layer **14** (also referred to herein as the "integration module") and a data storage **16** provided by a database **18**.

Three-Tier Architecture: 1.User Interface

Model

In accordance with the present embodiment, the user interface model is very similar to a LAMP platform (Linux-Apache-MySQL-PHP) for use in conjunction with web browsers located on client terminal **20**. A LAMP configuration is exemplified in FIG. 5. The source program interface resides on a Unix server **22**. An Apache HTTP server **24** acts as a bridge between the source program **14** and user requests. The user interface code **26** is written in PHP and the data specific to the interface such as user accounts, images and configurations are stored in a MySQL database **28**.

Site Plan

The user interface comprises four (4) main windows, presenting functionalities which may be summarized as follows:

1. Login window:
    a. To create a user account
    b. To retrieve a lost password
    c. To access the program after successful login
2. Version Control Management window:
    a. For creating, maintaining and accessing versions of DataStage™ programs.
    b. To consult the history and metadata of a program
    c. To create reports on programs
3. Release Management window:
    a. To Create and maintain packages of program versions
    b. To deploy packages in environments
    c. To consult release history
    d. To create reports on releases
4. Administration window:
    a. To manage users, roles and responsabilities
    b. To create and maintain branches and foundation components to versions and releases.
    c. To configure connection settings for DataStage™ servers and environments

The Unix server **22** designated to host the user interface is preferably provided by client users. The Apache HTTP Server, the MySQL database and PHP development framework are licensed under open source and are freely available.

Three-Tier Architecture: 2. Logical Layer

Model

The pie chart shown in FIG 6, illustrates three main class segments **32**, **34**, **36** of the version control system **10** of the present embodiment.

Programmed in object-oriented C++, the logical layer **14** contains classes and methods **32** interacting with DataStage™ (i.e. ETL) **38**. The logical layer **14** further comprises classes and methods **34** interacting with the database **18** containing versioned source code and other artefacts. The logical layer **14** further comprises classes and methods **36** interacting with the user interface **12**. Compiled into a library, the logical layer **14** may be source code protected to avoid accessibility to customers.

ETL Axis

The ETL Axis or "class segment" **32** contains classes interacting with the DataStage™ software and/or with other ETL tools. The classes and subclasses of the ETL axis **32**, namely for DataStage™, will now be described with reference to FIG. 7.

Class Details

Abstract ETL class (3200). The embodiment described herein is intended to target IBM Infosphere DataStage™ programs as well as other ETL suites (for example Informatica™ **3220** or SSIS™ **3222**). For this reason, an abstract class ETL **3200** is defined above the DataStage class **3202**.

DataStage class (3202). This class **3202** inherits from the abstract ETL class **3200** to instantiate an object of type DataStage™. It does not directly interface with DataStage™. To do this, each object will instantiate four objects: a DSAPI class **3204** to access methods for the API methods offered by DataStage™, a DSTools class

3206 to export and import ETL programs and components, a DSXmeta object **3208** to query the DataStage™ database and finally, and a DSCompare class **3210** to analyze and compare different versions of an ETL program.

DSAPI class (3204). The DSAPI class **3204** allows access to methods made available by the DataStage™ API. The API is offered by DataStage™ to allow access to certain internal methods of the application. It allows among other things to list projects and programs. It also allows controls over the execution of programs. Embodiments of the present invention are intended to further enable the management of program executions, for example, via methods provided by the Datastage API in order to launch the execution of Datastage™ programs.

DSTools class (3206). DataStage™ provides ways to extract and create or replace programs by means of DOS or UNIX commands under either Windows or Unix. This class **3206** contains the methods required to automate these function calls.

DSXmeta class (3208). The DSXmeta class **3208** queries the DataStage™ database directly. It can extract the list of ETL programs of an object and other useful data. Embodiments of the present invention are intended to lock programs for editing, thus acting as a "check-out" feature, preventing changes in applications without having first reserved a version of a program in the integration module.

DSCompare class (3210). The data files extracted from DataStage™ for versioning do not represent the source code data but rather a list of instructions to build an instance of a program. This can be likened to a Lego block montage and its set of instructions. Commonly, software versioning would keep a copy of the actual finished product. Because of current DataStage™ constraints, only the instructions can be versioned. DataStage™ protects direct access to source code and provides only a summary of the program in a proprietary format called dsx or in the form of XML. The instructions contained in a summary are complex and contain not only the business

rules, but since ETL program is graphical, the summary also contains all data relating to the positioning, size and alignment of each object and links. Comparison of two evolutions (or versions) of a DataStage™ program is rarely useful and provides virtually no information of interest. A DataStage™ "program" is also referred to as a Datastage™ "job", and corresponds to an "asset" or "component" in the context of the present description. This class **3210** provides methods for analyzing summary files and translate the results into quantity of objects each in turn containing instances of other child objects of different classes with specific properties. Once analyzed, two summaries could then be compared by isolating and comparing each sub-component programs. Different levels of comparison may be provided, in according with embodiments of the present invention, ranging from surface analysis (where only the presence and names of modules and children are compared) to in-depth analysis, where the positioning and alignment of components are also considered.

DSJob class (3212). When analyzing a program summary, an object of this class **3212** represents an ETL program. The latter may consist of objects of the Module class **3214** and Thread class **3216**.

Module class (3214). This class **3214** represents a processing block in a DataStage™ program. It can be passive if it only reads or writes data from files or databases or active if it applies transformations to the data. Business rules application, sorting, filters and data aggregation are some of the operations performed by a module. Each module contains objects of the Attribute class.

Attribute class (3216). An object of the Attribute class defines an attribute of a record that is subject to any kind of transformation.

Thread class (3218). A thread connects two modules together and incidentally allows data flow. Each thread contains one input port and one output port. Each port is

connected to a module. This class is used to record data transmitted between each module of a program.

Database Axis

The classes in the database segment **34** allow interactions with the database **18** where versions of components and other artefacts are stored. The classes and subclasses of the Database axis **34**, namely for the Oracle™ database, will now be described with reference to FIG. 8.

Abstract Database class (3400). Although Oracle is the solution of choice for most DataStage™ users, some customers might be using DB2 or some other database product, such as DB2 **3410**, MySQL **3412** and/or the like. Thus, an abstract class exists above the Oracle class to allow integration of different databases. The database class provides data storage and retrieval.

Abstract Oracle (3402). This class inherits from the Database class and allows the storage and retrieval of source code under an Oracle database. It is not designed to instantiate objects but to allow the creation of objects of child classes for specific versions of Oracle.

Oracle11g class (3404). This class **3404** inherits from the abstract class Oracle **3402** and allows interaction with the Oracle Database 11g.

Oracle10g class (3406). This class **3406** inherits from the abstract class Oracle **3402** and allows interaction with the Oracle Database 10g.

Oracle9i class (3408). This class **3408** inherits from the abstract class Oracle **3402** and allows interaction with the Oracle 9i database.

UI Segment

This class segment **36** interacts with the UI **12**. It interprets requests from the presentation layer and returns results. At this stage of development, only one class is included in this segment.

UI class. A class of interaction with the user interface named UI will receive user requests, process these requests by calling methods of and ETL object and methods of a Database object.

Main Methods Overview

The main methods found under the UI class will be described further below, with reference to the flowcharts shown in FIG. 10 to 13

Three-Tier Architecture: 3. Database Layer

Model

The database **18**, better shown in FIG. 9, is a relational database and contains data related to version control **1810** and release management **1820**. The database **18** cooperates with the UI database **28** (see FIG. 5) which includes administration data **1850**, as illustrated in FIG. 9 . Each table in the data model is detailed below with a summary and description of each column, in according with the present embodiment.

It is to be understood that the database **18**, may include the administration data 1850 and/or the UI database **28**, in accordance with alternative embodiments of the present invention.

Asset table (i.e. component). An Asset table **1802**, having columns represented in TABLE 1 below, contains a list of each entity having at least one versioned instance.

An asset may be a DSjob, a routine, a data connection, etc. In other words, any component that can be exported from Datastage™ as a unit.

- A component must have at least one version.
- A component can have multiple versions

No	Name	Description	Type	pk	fk	Unique
1	Asset_Id	Unique identifier	NUMBER	X		X
2	Name	Entity Name	VARCHAR2(255)			X
3	Type	Asset Type	VARCHAR2(50)			
4	Status	Usage status	VARCHAR2(30)			

TABLE 1

Version table. A Version table **1804** is represented in TABLE 2 below. Each version of an entity is a frozen image of a component code at specific point in time.

- A version belongs to a single asset.
- A version can be reserved (checked-out) by a single user.
- A version must be associated with a user on creation.

No	Name	Description	Type	pk	fk	Unique
1	Version_Id	Unique identifier	NUMBER	X		X
2	Asset_Id	Asset identifier	VARCHAR2(255)		X	
3	Version	Version identifier	VARCHAR2(50)			
4	CheckOutStatus	Job reservation status	VARCHAR2(50)			
5	CheckOutUser_Id	Owner of reservation	NUMBER		X	

No	Name	Description	Type	pk	fk	Unique
6	Code	Actual DataStage™ program extraction file	BLOB			
7	Code_Format	Type of file (DSX or XML)	VARCHAR2(50)			
8	CreatedBy	Creation user	NUMBER		X	
9	BaseVersion_Id	Original version to which changes were made	NUMBER		X	

TABLE 2

Table BranchVersion (version branch). A BranchVersion table **1822** is represented in TABLE 3 below and corresponds to an intersection table between versions and branches.

- A version must belong to one or more branches.
- A branch-version can be associated with any one or more packages.
- A branch may be composed of multiple versions of different components.
- Each component can be associated with a branch by only one of its versions.

No	Name	Description	Type	pk	fk	Unique
1	BranchVersion_Id	Unique identifier	NUMBER	X		X
2	Branch_Id	Branch identifier	NUMBER		X	
3	Version_Id	Version identifier	NUMBER		X	

TABLE 3

Table PackageBranchVersion (version of a set of deployment). A PackageBranchVersion table **1824** is represented in TABLE 4 below and corresponds to an intersection table between branch-versions and packages.

No	Name	Description	Type	pk	fk	Unique
1	BranchVersion_Id	BranchVersion identifier	NUMBER		X	
2	Package_Id	Package identifier	NUMBER		X	
3	Operation_Type	Operation type (insertion, update, deletion)	VARCHAR2(30)			

TABLE 4

Table Package (Set of deployment). A Package table **1826** is represented in TABLE 5 below and identifies a group of asset versions to be deployed in a branch as a bundle.

- A package contains a single version of an asset.
- A package contains versions from a single branch.
- A package can be deployed in a single branch
- A package must contain at least one entry in the package status table.
- A package may contain multiple entries in the package status table.

No	Name	Description	Type	pk	fk	Unique
1	Package_Id	Unique Identifier	NUMBER	X		X
2	Branch_Id	Branch identifier (where deployed)	NUMBER		X	
3	Name	Package name	VARCHAR2(50)			
4	Description	Contextual description	VARCHAR2(255)			

TABLE 5

Table PackageStatus (Status of deployment). A PackageStatus **1828** table is represented in TABLE 5 below. Records in this table keep a history of the changes in the status of a package.

- A package status belongs to only one package.
- A package status must refer to a single user.

No	Name	Description	Type	pk	fk	Unique
1	PackageStatus_Id	Unique identifier	NUMBER	X		X
2	Package_Id	Package identifier	NUMBER		X	
3	Status	Deployment status (New, Pending Authorization, Deployed, Cancelled)	VARCHAR2(50)			
4	Created_By	User who updated the status	NUMBER		X	
5	Created_Dt	Record creation date	TIMESTAMP			

TABLE 6

Table Branch (Branch). A Branch table **1830** is represented in TABLE 7 below. A branch is an instance of a project phase: (i.e. development, unit testing, production, etc.)

- A branch must belong to a tree.
- A branch can only belong to one tree.
- A package may have been deployed on a branch.
- A branch must belong to a single development phase.

No	Name	Description	Type	pk	fk	Unique
1	Branch_Id	Unique identifier	NUMBER	X		X
2	Tree_Id	Tree identifier	NUMBER		X	
3	Phase_Id	Phase identifier	NUMBER		X	

No	Name	Description	Type	pk	fk	Unique
4	Version	Evolution number of the branch in relation to other branches of a parent tree (1.0.0, 2.1.5,etc.)	VARCHAR2(10)			
5	ReadOnlyStatus	Read only status identifying a dead branch.	VARCHAR2(30)			

TABLE 7

Tree table (Project). A Tree table **1832** is represented in TABLE 8 below and corresponds to an ETL project which groups common tasks.

- A project must have at least one branch.
- A project can have multiple branches.

No	Name	Description	Type	pk	fk	Unique
1	Tree_Id	Unique identifier	NUMBER	X		X
2	Name	Project Name	VARCHAR2(50)			
3	Status	Project Usage status	VARCHAR2(30)			

TABLE 8

Phase table (Development Phase). A Phase table **1834** is represented in TABLE 9 below and corresponds to a step in the development cycle.

- A phase can be represented by any one or more branches.
- A phase must belong to a single development environment.
- A phase may be referred to as the source of a phase promotion in zero, one or more phases of promotions.

- A phase may be referred to as the target of a phase promotion in zero, one or more phases of promotions.

No	Name	Description	Type	pk	fk	Unique
1	Phase_Id	Unique identifier	NUMBER	X		X
2	Env_Id	Environment identifier	NUMBER		X	
3	Name	Phase name	VARCHAR2(30)			
4	Description	Phase description	VARCHAR2(255)			

TABLE 9

PhasePromotion Table (Promotion Phase). A PhasePromotion table **1836** is represented in TABLE 10 below and identifies which phase jumps are allowed when promoting packages from branches (i.e. development to testing, testing to production).

No	Name	Description	Type	pk	fk	Unique
1	Promotion_Id	Unique identifier	NUMBER	X		X
2	PhaseSrc_Id	Source phase identifier	NUMBER		X	
3	PhaseTrgt_Id	Target phase identifier	NUMBER		X	

TABLE 10

Table Environment (Development Environment). An Environment table **1838** is represented in TABLE 11 below and corresponds to a server instance in DataStage™ (for example, development or production).

- An environment has one or more phases of development.

No	Name	Description	Type	pk	fk	Unique
1	Environment_Id	Unique identifier	NUMBER	X		X
2	Domain	Server domain name	VARCHAR2(255)			
3	Host	Server host name	VARCHAR2(255)			
4	Port	Port number for connexion to the server	NUMBER			

TABLE 11

User table (User). A User table **1852** is represented in TABLE 12 below and identifies user accounts.

- A user can be the creator of zero, one or more versions.
- A user can be associated to a checked-out version
- A user can be associated to a package status update

No	Name	Description	Type	pk	fk	Unique
1	User_Id	Unique identifier	NUMBER	X		X
2	FirstName	User last name	VARCHAR2(50)			
3	LastName	User last name	VARCHAR2(50)			
4	ActiveStatus	User status	VARCHAR2(30)			

TABLE 12

UserRole table (User Role). A UserRole table **1854** is represented in TABLE 13 below and corresponds to an intersection table connecting a user to roles and roles to users.

- A user must occupy at least one role, but can occupy several.
- A role can be associated with any one or more users.

No	Name	Description	Type	pk	fk	Unique
1	User_Id	User identifier	NUMBER		X	
2	Role_Id	Role identifier	NUMBER		X	

TABLE 13

Role Table (Role). A Role table **1856** is represented in TABLE 14 below. Each role can restrict tasks common to several users of the same type.

No	Name	Description	Type	Pk	Fk	Unique
1	Role_Id	Unique identifier	NUMBER	X		X
2	Name	Role Name	VARCHAR2(50)			
3	Description	Role Description	VARCHAR2(255)			
4	ActiveStatus	Role Status	VARCHAR2(30)			

TABLE 14

RolePermission table (Permission by role). A RolePermission table **1858** is represented in TABLE 15 below and corresponds to an intersection table connecting a role to permissions and a permission to roles.

- A role can have zero, one or more permissions.
- Permission may be associated with any one or more roles.

No	Name	Description	Type	pk	fk	Unique
1	Role_Id	Unique identifier	NUMBER		X	
2	Permission_Id	Permission identifier	NUMBER		X	

TABLE 15

Permission table. A Permission table **1860** is represented in TABLE 16 below. Each permission provides access to task or the visibility to certain views.

No	Name	Description	Type	pk	fk	Unique
1	Permission_Id	Unique identifier	NUMBER	X		X
2	Name	Permission name	VARCHAR2(50)			
3	Description	Description	VARCHAR2(255)			
4	ActiveStatus	Usage status	VARCHAR2(30)			
5	Type	Permission type (view, action)	VARCHAR2(30)			

TABLE 16

Main Functional Features

FIG. 10 to 13 illustrate the interactions between the three (3) afore-mentioned tiers, for each of the main functions performed by the version control system, in accordance an embodiment of the present embodiment. The main functions illustrated are:

- Checking-In of a DataStage™ component (see FIG. 10);
- Checking-Out of a DataStage™ component (see FIG. 11);
- Creation and Deployment of a package (see FIG. 12); and
- Component Version Comparison (see FIG. 13).

FIG. 14 shows the components of the system **10**. As previously mentioned, the system **10** comprises a user interface **12**, an integration module **14** and a data storage **16**. The integration module **14** is embedded in a processor **13** and is comprised within a utility application for performing the steps of the methods described herein.

Referring to FIG. 10, there is shown a sequence diagram of steps performed by the version control system, for checking-in a component, according to an embodiment of the present invention.

Namely, a method **2000** for exporting a program asset from Datastage™ (i.e. ETL library) **38** is exemplified. The Datastage™ library **38** stores a plurality of said program assets, each program asset being protected in the Datastage™ library **38**. The method **2000** comprises steps of:

a) receiving at **2034**, via a user interface **12**, a command for exporting said program asset;

b) exporting at **2048**, by means of an integration module **14**, the program asset from Datastage™ **38** into a digest, the digest comprising instructions for rebuilding the program asset in Datastage™ **38**;

c) storing at **2050**, by means of the integration module **14**, a new instance of the digest in the database **18**;

d) associating at **2050** in the database **18**, by means of the integration module **14**, a new version to said new instance of the digest by:

- querying the database **18** to locate an instance of the digest being associated to a latest version of the digest; and
- if no instance of the digest is located in the database **18**, said new version is a first version, and otherwise, said new version is obtained by incrementing an originating version associated to the digest; and

e) by means of the integration module **14**, setting at **2050** a checked-in status to the new instance of the digest in the database **18**.

Instances of digests are organized in a tree defining branches. Each branch for a given digest represents a subset of versions of the corresponding program asset.

Thus, the method **2000** further includes prior to step (d):

- receiving at **2026** branch information identifying a selected branch in the database **18** to which the new instance of the digest is to be associated to, and said new version of step (d) is assigned in association with said selected branch.

In FIG. 10, steps **2012**, **2014**, **2016** and table **1852** relate to user authentication; steps **2018**, **2020** and table **1812** relate to accessing a screen on the user interface **12**; steps **2022**, **2024**, **2026**, **2028** and table **1830** relate to a branch selection; steps **2030**, **2032**, **2034**, **2036**, **2038**, **2040** and table **1802** relate to the selection of asset(s) to check-into the system **10**; steps **2042**, **2044**, **2046**, **2048**, **2050**, **2052** and tables **1814** and **1822** relate to the extraction from the program assets to complete the exporting of the program asset(s).

It is to be understood that multiple program assets may be exported at once. It is to be understood that a plurality of digests may be stored in a single file corresponding to the multiple program assets, so long as each digest (i.e. each program asset) is associated to its own version information. Alternatively, each digest is stored in a separate file.

Thus, with reference to FIG. 14, the integration module **14** comprises an exportation module **3010** having an exportation communication port **3012** for communicating with the user interface **12**.

Referring now to FIG. 11, there is shown a sequence diagram of steps performed by the version control system, for checking-out a component, according to an embodiment of the present invention.

Namely, a method **2200** for importing a versioned program asset into Datastage™ (i.e. ETL library) **38** from database **18** is exemplified. The program asset is buildable in Datastage™ **38** from a corresponding digest of instructions, one or more instance

of said digest being stored in the database **18**, each instance being associated to a version of the digest. The method **2200** comprises steps of:

a) receiving at **2226**, via a user interface **12**, a command for importing said program asset;

b) receiving at **2234**, via the user interface **12**, version information of the program asset to be imported;

c) retrieving at **2242** an instance of the digest from the database **18**, by means of the integration module **14**, corresponding to the version information received at step (b);

- validating at **2242** whether said instance of digest retrieved, has a checked-out status, and only if the program asset does not have a checked-out status, proceeding to the following steps of the method **2200**:

d) at **2244**, by means of the integration module **14**, setting a checked-out status to the instance of the digest in the database **18**; and

e) executing at **2246** the instructions of said instance of the digest, by means of the integration module **14**, in order to import the corresponding program asset in the Datastage™ library **38**.

In FIG. 11, steps **2212**, **2214**, **2216** and table **1852** relate to user authentication; steps **2218**, **2220** and table **1812** relate to accessing a screen on the user interface **12** for prompting the check-out process; steps **2222**, **2224**, **2226**, **2228** and table **1802** relate to the selection of asset(s) to check-out from the system **10**; steps **2230**, **2232**, **2234**, **2036**, and table **1830** relate to a branch selection; steps **2238**, **2240**, **2242**, **2246**, **2244**, **2248** and table **1814** relate to the rebuilding of the program assets to complete the importation into Datastage™.

It is to be understood that a single instance of digest may have either a checked-in status or a checked-out status at any given time. Indeed the checked-in and checked-out status are mutually exclusive.

Instances of digests are organized in a tree defining version branches, each version branch for a given digest representing a subset of versions of the corresponding program asset. Thus, the version information received at step (b) (**2234**) further includes branch information, and the retrieving of step (c) takes into account the branch information.

Thus, with reference to FIG. 14, the integration module **14** comprises further comprises an importation module **3020** comprising an importation input port **3022** for receiving the selection of program asset(s) to be imported into the library and the corresponding version information; a collector **3024** for retrieving an instance of the digest from the data storage for each the program asset(s) to be imported; a builder **3026** for executing, for each digest retrieved at step (vii), the instructions to rebuild the corresponding program asset; and a flagging component **3028** for replacing the checked-in status of each digest retrieved with the checked-out status.

Referring now to FIG. 12, there is shown a sequence diagram of steps performed by the version control system **10** (see FIG. 6), for creating and deploying a package in Datastage™, i.e. an ETL library **38** (see FIG. 6), according to an embodiment of the present invention. The creation and deploying of a package is useful for example, in order to promote a group of versioned program assets from a development environment to a production environment.

Thus, a method **2400** for importing a package of versioned program assets into Datastage™ **38** from a database **18** is exemplified in FIG. 12. Each of said program asset is buildable in the Datastage™ **38** from a corresponding digest of instructions. One or more instance of the digest is stored in the database **18**, each instance being associated to a version of the digest. The method **2400** comprises steps of:

a) receiving at **2422**, via a user interface **12**, a command for importing a new package;

b) receiving at **2438**, via the user interface **12**, the program assets to be imported via the new package and corresponding version information at **2430**;

c) generating at **2428** the new package in the database **18**, by means of an integration module **14**;

d) retrieving at **2444** from the database **18**, instances of the digests corresponding to the program assets to be imported, by means of the integration module **14**, in accordance with the version information received at step (b) (**2438**);

e) associating at **2456** in the database **18**, the instances retrieved at step (d) with the new package;

f) at **2456**, by means of the integration module **14**, setting a deployment status to the new package in the database **18**; and

g) executing at **2458,** the instructions of each of said instances associated to the new package, by means of the integration module, in order to import the corresponding program assets in Datastage™ **38**.

In FIG. 12, steps **2412**, **2414**, **2416** and table **1852** relate to user authentication; steps **2418**, **2420** and table **1812** relate to accessing a screen on the user interface **12** for accessing a release management user menu; steps **2422**, **2424**, **2426**, **2428** and table **1826** relate to the creation of a package to be deployed in Datastage™; steps **2430**, **2432**, **2434**, **2436**, and table **1830** relate to a version branch selection; steps **2438**, **2440**, **2442**, **2444**, and table **1822** relate to versions of digests selected to include in the package; steps **2446** and **2448** relate to determining a target branch, namely the target environment in Datastage™ (development, production, test, etc.); steps **2450**, **2452**, **2454**, **2458**, **2456**, **2460** and tables **1826**, **1824** and **1828** relate to the deployment of the package in order to import the corresponding assets into Datastage™.

The one or more instance of the digest are grouped by branches in the database **18**. Each branch corresponds to a subset of versions of the digest. Thus, the version

information received at step (b) (**2442**) further includes branch information, and the retrieving of step (c) (**2428**) takes into account the branch information.

Thus, with reference to FIG. 14, the importation module **3020** further comprises a packaging module **3030** for generating a package and associating the package to import a plurality of the program assets received at the input port **3022**, and for setting a deployed status to the package in the data storage to indicate that the package has updated the associated program assets in the library.

Referring now to FIG. 13, there is shown a sequence diagram of steps performed by the version control system, for comparing versions of a Datastage™ component, according to an embodiment of the present invention.

More particularly, a method **2600** for comparing versions of a given program asset in Datastage™ (i.e. ETL library) **38** is exemplified in FIG. 12. The given program asset is protected and buildable from a digest of instructions stored in a database **18**, which stores multiple instances of the digest, each instance corresponding to a version of the given program asset (i.e. the database **18** stores several versions of a same program asset).

The method **2600** comprises steps of:

a) receiving at **2626** and **2634**, via a user interface **12**, instructions to compare two versions of said given program asset of Datastage™;

b) at **2628**, retrieving from the database **18** two instances of the digest corresponding to said two versions of said given program asset, by means of an integration module **14**;

c) at **2638**, by means of an integration module **14**, generating comparison information, by pairing matching components of the two instances; and

d) returning at **2634**, by means of the integration module **14**, the comparison information on the user interface **12**.

In FIG. 13, steps **2612**, **2614**, **2216** and table **1852** relate to user authentication; steps **2618**, **2620** and table **1812** relate to accessing a screen on the user interface **12** for prompting the comparison process; steps **2622**, **2624**, **2626**, **2628** and table **1814** relate to the selection of versions of asset(s) to be compared; steps **2630**, **2632**, **2634**, **2636**, **2638**, **2640** and table **1814** relate to the comparison of the program assets and the presenting of the resulting comparison information on the user interface **12**.

Thus, with reference to FIG. 14, the integration module **14** further comprises a comparison module **3040** comprising: a comparison input port **3042** for receiving, a selection of the digest instances to be compared and corresponding version identifier; a retriever **3044** for retrieving the instances of the digest corresponding to the selection received; a comparer **3046** for comparing the content of the instances of the digest, to generate associated comparison information; and a comparison output port **3048** to send the comparison information for presentation on the user interface **12**.

It is to be understood that one or more of a series of steps of the methods illustrated in FIG. 10 to 13, may be performed within a same user session, i.e. without requiring a user long-on or even entering separate menu screens for each operation. Indeed, further to performing a check-in, for example, a user may immediately follow-up with a check-out operation, a package deployment operation and/or a comparison operation, or any combination thereof, without requiring to log-on between each operation, as may be easily understood by a person skilled in the art.

The above-described embodiments are considered in all respect only as illustrative and not restrictive, and the present application is intended to cover any adaptations or variations thereof, as apparent to a person skilled in the art. Of course, numerous other modifications could be made to the above-described embodiments without departing from the scope of the invention, as apparent to a person skilled in the art.

Claims:

1. A method for managing versions of program assets of a library, each of said program assets having source code which is protected, the method being executable by a single utility application having an integration module which is embedded in a processor, the method comprising the steps of:
   i) receiving a selection of one or more program asset to be exported into the utility application for storage;
   ii) extracting from the library and into a digest, for each of the one or more program asset selected, instructions for building the source code of the corresponding program asset, by means of the integration module;
   iii) storing, by means of the integration module, each digest as a new instance of the digest in a data storage;
   iv) associating in the data storage, by means of the integration module, a new version identifier to each new instance of digest, the new version identifier representing a new version of the corresponding program asset; and
   v) in the data storage, associating a checked-in status to each new instance of digest stored at step (iii), by means of the integration module, to indicate that each of said new instance of digest is stored in the utility application.

2. A method according to claim 1, wherein step (iv) comprises, for each digest:
   - querying the data storage to locate a prior instance of the digest; and
   - if said prior instance of the digest is located, determining a corresponding previous version identifier and setting said new version identifier associated to the digest, by incrementing the previous version identifier, or otherwise, setting said new version identifier to represent a first instance of the digest.

3. A method according to claim 2, wherein the incrementing of step (iv) is executed in accordance with one or more predefined incrementing rule.

4. A method according to claim 1 or 2, wherein the data storage stores instances of previously stored digests which are organized in a format of a tree having branches, each branch for a given one of the stored digests representing a subset of versions of the corresponding program asset, the method further comprising, prior to step (iv):

- receiving a branch selection to which the new instance of the digest is to be associated with; and
- retrieving branch information identifying the selected branch from the data storage; and

wherein the new version identifier of step (iv) is set based on said branch information.

5. A method according to any one of claims 1 to 4, wherein each digest of step (ii) is provided in a file.

6. A method according to any one of claims 1 to 4, wherein the one or more digest of step (ii) is provided in a same file.

7. A method according to any one of claims 1 to 3, wherein the data storage comprises a plurality of said digests, each digest comprising instructions to rebuild a corresponding program asset in the library, the method further comprising:

vi) receiving, via a user interface, a selection of one or more of said program assets to be imported into the library and the corresponding version information;

vii) retrieving an instance of the digest from the data storage for each of said one or more program asset to be imported, by means of the integration

module, being associated to the version information received at step (vi); and

viii) for each digest retrieved at step (vii), executing the instructions to rebuild the corresponding program asset, by means of the integration module, in order to import a new version of the corresponding program asset into the library.

ix) in the data storage, replacing a checked-in status associated each instance of the digest retrieved at step (vii) with a checked-out status, by means of the integration module, to indicate that the corresponding one or more program asset is currently being updated.

8. A method according to claim 7, further comprising prior to step (viii):

- validating whether said instance of digest retrieved at step (vii), has a checked-out status, and only if the program asset does not have a checked-out status, proceeding to the following steps of the method.

9. A method according to claim 7, wherein instances of digests are organized in the data storage, in a format of a tree having branches, each branch for a given digest representing a subset of versions of the corresponding program asset, wherein the version information received at step (vi) comprises branch information.

10. A method according to any one of claims 7 to 9, wherein the selection received at step (vi) comprises a plurality of said program assets, the method further comprising:

- generating a package to import the selection of program assets;
- after step (vii), associating in the data storage, the instances retrieved at step (vii) with the package; and

- after step (viii), setting a deployed status to the new package in the data storage to indicate that the package has updated the associated program assets in the library.

11. A method according to any one of claims 1 to 3, wherein the data storage comprises a plurality of said digests, each digest comprising instructions to rebuild a corresponding program asset in the library, the data storage storing multiple instances of at least one of the digests, each instance corresponding to a version of the corresponding program asset, the method further comprising:

- receiving a selection of two or more digest instances of the data storage and corresponding version identifier, to be compared;
- retrieving from the data storage the instances of the digest corresponding to the selection received;
- by means of the integration module, comparing the content of the digest instance, to generate comparison information; and
- returning the comparison information on a user interface component.

12. A method according to claim 11, wherein said comparison information is returned as at least one of:

- text comparison of each digest instance to be compared; and
- comparison of program features of the program asset associated to each digest instance to be compared.

13. A system for managing versions of program assets of a library, each of said program assets having source code which is protected, the system comprising:

- a user interface for receiving a selection of one or more program asset to be exported into a utility application for editing;
- an integration module embedded in a processor which is in communication with the user interface, the integration module comprising an exportation module for extracting from the library into a digest, for each of the one or

more program asset selected, instructions for building the source code of the corresponding program asset; and

- a data storage, in communication with the integration module, for storing each digest as a new instance of the digest, and for associating a new version identifier to each new instance of digest, the new version identifier representing a new version of the corresponding program asset, and for further associating a checked-in status to each new instance of digest stored to indicate that each of said new instance of digest is stored in the utility application.

14. A system according to claim 13, wherein the data storage comprises a plurality of said digests, each digest comprising instructions to rebuild a corresponding program asset in the library, wherein the integration module further comprises an importation module comprising:

- an importation input port for receiving, from the user interface, a selection of one or more of said program assets to be imported into the library and the corresponding version information;
- a collector for retrieving an instance of the digest from the data storage for each of said one or more program asset to be imported, being associated to the version information received by the user interface;
- a builder for executing, for each digest retrieved at step (vii), the instructions to rebuild the corresponding program asset, by means of the integration module, in order to import a new version of the corresponding program asset into the library; and
- a flagging component for replacing a checked-in status associated with each instance of the digest retrieved at step (vii) in the data storage with a checked-out status, in order to indicate that the corresponding one or more program asset is currently being updated.

15. A system according to claim 14, wherein the importation module further comprises:

- a packaging module for generating a package and associating said package to import a plurality of the program assets received at the input port, and for setting a deployed status to the package in the data storage to indicate that the package has updated the associated program assets in the library.

16. A system according to any one of claims 13 to 15, wherein the integration module further comprises a comparison module comprising:

- a comparison input port for receiving, from the user interface, a selection of two or more digest instances of the data storage and corresponding version identifier, to be compared;
- an retriever for retrieving from the data storage, the instances of the digest corresponding to the selection received;
- a comparer for comparing the content of the instances of the digest, to generate associated comparison information; and
- a comparison output port to send the comparison information for presentation on the user interface.

17. A storage medium for managing versions of program assets of a library, each of said program assets having source code which is protected, the storage medium being processor-readable and non-transitory, the storage medium comprising instructions for execution by a processor, via a single utility application, to:

i) receive a selection of one or more program asset to be exported into the utility application for storage;

ii) extract from the library and into a digest, for each of the one or more program asset selected, instructions for building the source code of the corresponding program asset, by means of the integration module;

iii) store, by means of the integration module, each digest as a new instance of the digest in a data storage;

iv) associate in the data storage, by means of the integration module, a new version identifier to each new instance of digest, the new version identifier representing a new version of the corresponding program asset; and

v) associated, in the data storage, a checked-in status to each new instance of digest stored at (iii), by means of the integration module, to indicate that each of said new instance of digest is stored in the utility application.

18. A storage medium according to claim 17, wherein the instructions to associate at (iv) comprise instructions to:

- query the data storage to locate a prior instance of the digest; and
- if said prior instance of the digest is located, determine a corresponding previous version identifier and set said new version identifier associated to the digest, by incrementing the previous version identifier, or otherwise, set said new version identifier to represent a first instance of the digest.

19. A storage medium according to claim 18, wherein the instructions to increment are executable in accordance with one or more predefined incrementing rule.

20. A storage medium according to claim 17, wherein the data storage stores instances of previously stored digests which are organized in a format of a tree having branches, each branch for a given one of the stored digests representing a subset of versions of the corresponding program asset, the storage medium further comprising instructions to, prior to the associating at (iv):

- receive a branch selection to which the new instance of the digest is to be associated with; and
- retrieve branch information identifying the selected branch from the data storage; and

wherein the new version identifier of step (iv) is set based on said branch information.

21. A storage medium according to claim 17, wherein the instructions to extract at (ii) comprise instructions to generate each digest in a file.

22. A storage medium according to claim 17, wherein the instructions to extract at (ii) comprise instructions to generate one or more of said digest in a same file.

AMENDED CLAIMS
received by the International Bureau on 20 December 2013 (20.12.2013)

Claims:

1. A method for managing versions of program assets of a library, each of said program assets having source code which is protected, the method being executable by a single utility application having an integration module which is embedded in a processor, the method comprising the steps of:
   i) receiving a selection of one or more program asset to be exported into the utility application for storage;
   ii) extracting from the library and into a digest, for each of the one or more program asset selected, instructions for building the source code of the corresponding program asset, by means of the integration module;
   iii) storing, by means of the integration module, each digest as a new instance of the digest in a data storage;
   iv) associating in the data storage, by means of the integration module, a new version identifier to each new instance of digest, the new version identifier representing a new version of the corresponding program asset; and
   v) in the data storage, associating a checked-in status to each new instance of digest stored at step (iii), by means of the integration module, to indicate that each of said new instance of digest is stored in the utility application.

2. A method according to claim 1, wherein step (iv) comprises, for each digest:
   - querying the data storage to locate a prior instance of the digest; and
   - if said prior instance of the digest is located, determining a corresponding previous version identifier and setting said new version identifier associated to the digest, by incrementing the previous version identifier, or otherwise, setting said new version identifier to represent a first instance of the digest.

3. A method according to claim 2, wherein the incrementing of step (iv) is executed in accordance with one or more predefined incrementing rule.

4. A method according to claim 1 or 2, wherein the data storage stores instances of previously stored digests which are organized in a format of a tree having branches, each branch for a given one of the stored digests representing a subset of versions of the corresponding program asset, the method further comprising, prior to step (iv):

- receiving a branch selection to which the new instance of the digest is to be associated with; and
- retrieving branch information identifying the selected branch from the data storage; and

wherein the new version identifier of step (iv) is set based on said branch information.

5. A method according to any one of claims 1 to 4, wherein each digest of step (ii) is provided in a file.

6. A method according to any one of claims 1 to 4, wherein the one or more digest of step (ii) is provided in a same file.

7. A method according to any one of claims 1 to 3, wherein the data storage comprises a plurality of said digests, each digest comprising instructions to rebuild a corresponding program asset in the library, the method further comprising:

vi) receiving, via a user interface, a selection of one or more of said program assets to be imported into the library and the corresponding version information;

vii) retrieving an instance of the digest from the data storage for each of said one or more program asset to be imported, by means of the integration module, being associated to the version information received at step (vi);

viii) for each digest retrieved at step (vii), executing the instructions to rebuild the corresponding program asset, by means of the integration module, in order to import a new version of the corresponding program asset into the library; and

ix) in the data storage, replacing a checked-in status associated each instance of the digest retrieved at step (vii) with a checked-out status, by means of the integration module, to indicate that the corresponding one or more program asset is currently being updated.

8. A method according to any one of claims 1 to 3, wherein the data storage comprises a plurality of said digests, each digest comprising instructions to rebuild a corresponding program asset in the library, the method further comprising:

vi) receiving, via a user interface, a selection of one or more of said program assets to be imported into the library and the corresponding version information;

vii) retrieving an instance of the digest from the data storage for each of said one or more program asset to be imported, by means of the integration module, being associated to the version information received at step (vi); and

viii) validating whether said instance of digest retrieved at step (vii), has a checked-out status, and if the program asset does not have a checked-out status, proceeding to the steps of:

- for each digest retrieved at step (vii), executing the instructions to rebuild the corresponding program asset, by means of the integration module, in order to import a new version of the corresponding program asset into the library; and

- in the data storage, replacing a checked-in status associated each instance of the digest retrieved at step (vii) with a checked-out status, by means of the integration module, to indicate that the corresponding one or more program asset is currently being updated.

9. A method according to claim 7 or 8, wherein instances of digests are organized in the data storage, in a format of a tree having branches, each branch for a given digest representing a subset of versions of the corresponding program asset, wherein the version information received at step (vi) comprises branch information.

10. A method according to any one of claims 7 to 9, wherein the selection received at step (vi) comprises a plurality of said program assets, the method further comprising:
- generating a package to import the selection of program assets;
- after step (vii), associating in the data storage, the instances retrieved at step (vii) with the package; and
- after step (viii), setting a deployed status to the new package in the data storage to indicate that the package has updated the associated program assets in the library.

11. A method according to any one of claims 1 to 3, wherein the data storage comprises a plurality of said digests, each digest comprising instructions to rebuild a corresponding program asset in the library, the data storage storing multiple instances of at least one of the digests, each instance corresponding to a version of the corresponding program asset, the method further comprising:
- receiving a selection of two or more digest instances of the data storage and corresponding version identifier, to be compared;

- retrieving from the data storage the instances of the digest corresponding to the selection received;
- by means of the integration module, comparing the content of the digest instance, to generate comparison information; and
- returning the comparison information on a user interface component.

12. A method according to claim 11, wherein said comparison information is returned as at least one of:
- text comparison of each digest instance to be compared; and
- comparison of program features of the program asset associated to each digest instance to be compared.

13. A system for managing versions of program assets of a library, each of said program assets having source code which is protected, the system comprising:
- a user interface for receiving a selection of one or more program asset to be exported into a utility application for editing;
- an integration module embedded in a processor which is in communication with the user interface, the integration module comprising an exportation module for extracting from the library into a digest, for each of the one or more program asset selected, instructions for building the source code of the corresponding program asset; and
- a data storage, in communication with the integration module, for storing each digest as a new instance of the digest, and for associating a new version identifier to each new instance of digest, the new version identifier representing a new version of the corresponding program asset, and for further associating a checked-in status to each new instance of digest stored to indicate that each of said new instance of digest is stored in the utility application.

14. A system according to claim 13, wherein the data storage comprises a plurality of said digests, each digest comprising instructions to rebuild a corresponding

program asset in the library, wherein the integration module further comprises an importation module comprising:

- an importation input port for receiving, from the user interface, a selection of one or more of said program assets to be imported into the library and the corresponding version information;
- a collector for retrieving an instance of the digest from the data storage for each of said one or more program asset to be imported, being associated to the version information received by the user interface;
- a builder for executing, for each digest retrieved, the instructions to rebuild the corresponding program asset, by means of the integration module, in order to import a new version of the corresponding program asset into the library; and
- a flagging component for replacing a checked-in status associated with each instance of the digest retrieved in the data storage with a checked-out status, in order to indicate that the corresponding one or more program asset is currently being updated.

15. A system according to claim 14, wherein the importation module further comprises:

- a packaging module for generating a package and associating said package to import a plurality of the program assets received at the input port, and for setting a deployed status to the package in the data storage to indicate that the package has updated the associated program assets in the library.

16. A system according to any one of claims 13 to 15, wherein the integration module further comprises a comparison module comprising:

- a comparison input port for receiving, from the user interface, a selection of two or more digest instances of the data storage and corresponding version identifier, to be compared;

- an retriever for retrieving from the data storage, the instances of the digest corresponding to the selection received;
- a comparer for comparing the content of the instances of the digest, to generate associated comparison information; and
- a comparison output port to send the comparison information for presentation on the user interface.

17. A storage medium for managing versions of program assets of a library, each of said program assets having source code which is protected, the storage medium being processor-readable and non-transitory, the storage medium comprising instructions for execution by a processor, via a single utility application, to:

i) receive a selection of one or more program asset to be exported into the utility application for storage;

ii) extract from the library and into a digest, for each of the one or more program asset selected, instructions for building the source code of the corresponding program asset, by means of the integration module;

iii) store, by means of the integration module, each digest as a new instance of the digest in a data storage;

iv) associate in the data storage, by means of the integration module, a new version identifier to each new instance of digest, the new version identifier representing a new version of the corresponding program asset; and

v) associated, in the data storage, a checked-in status to each new instance of digest stored at (iii), by means of the integration module, to indicate that each of said new instance of digest is stored in the utility application.

18. A storage medium according to claim 17, wherein the instructions to associate at (iv) comprise instructions to:

- query the data storage to locate a prior instance of the digest; and

- if said prior instance of the digest is located, determine a corresponding previous version identifier and set said new version identifier associated to the digest, by incrementing the previous version identifier, or otherwise, set said new version identifier to represent a first instance of the digest.

19. A storage medium according to claim 18, wherein the instructions to increment are executable in accordance with one or more predefined incrementing rule.

20. A storage medium according to claim 17, wherein the data storage stores instances of previously stored digests which are organized in a format of a tree having branches, each branch for a given one of the stored digests representing a subset of versions of the corresponding program asset, the storage medium further comprising instructions to, prior to the associating at (iv):

- receive a branch selection to which the new instance of the digest is to be associated with; and
- retrieve branch information identifying the selected branch from the data storage; and

wherein the new version identifier of step (iv) is set based on said branch information.

21. A storage medium according to claim 17, wherein the instructions to extract at (ii) comprise instructions to generate each digest in a file.

22. A storage medium according to claim 17, wherein the instructions to extract at (ii) comprise instructions to generate one or more of said digest in a same file.

STATEMENT UNDER ARTICLE 19 (1)

Amendment Statement

Claim 7, 8, 9 and 14 have been amended to correct clerical errors and to address observations made in the Written Opinion at Box VIII.

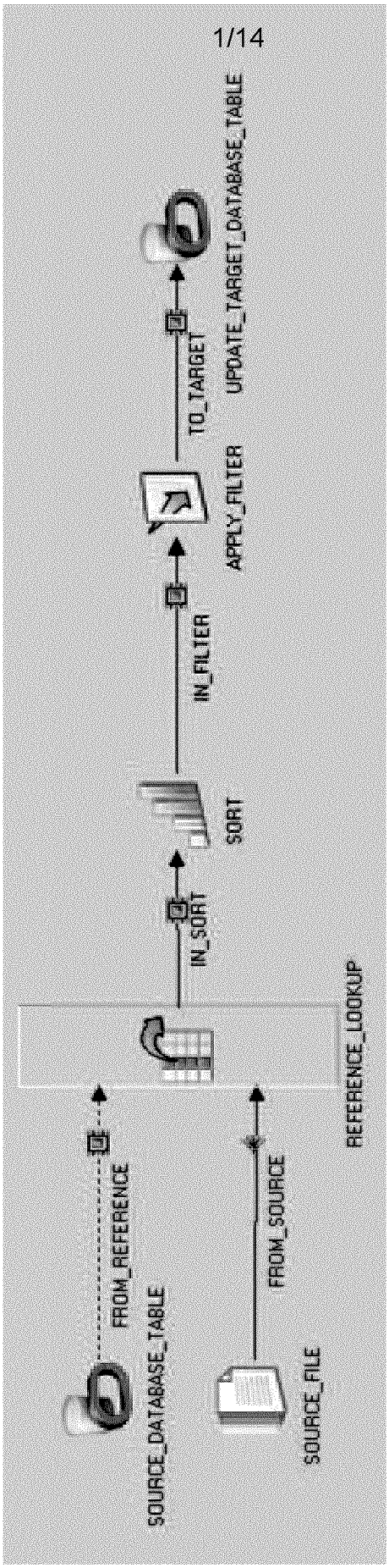


FIG. 1
(PRIOR ART)

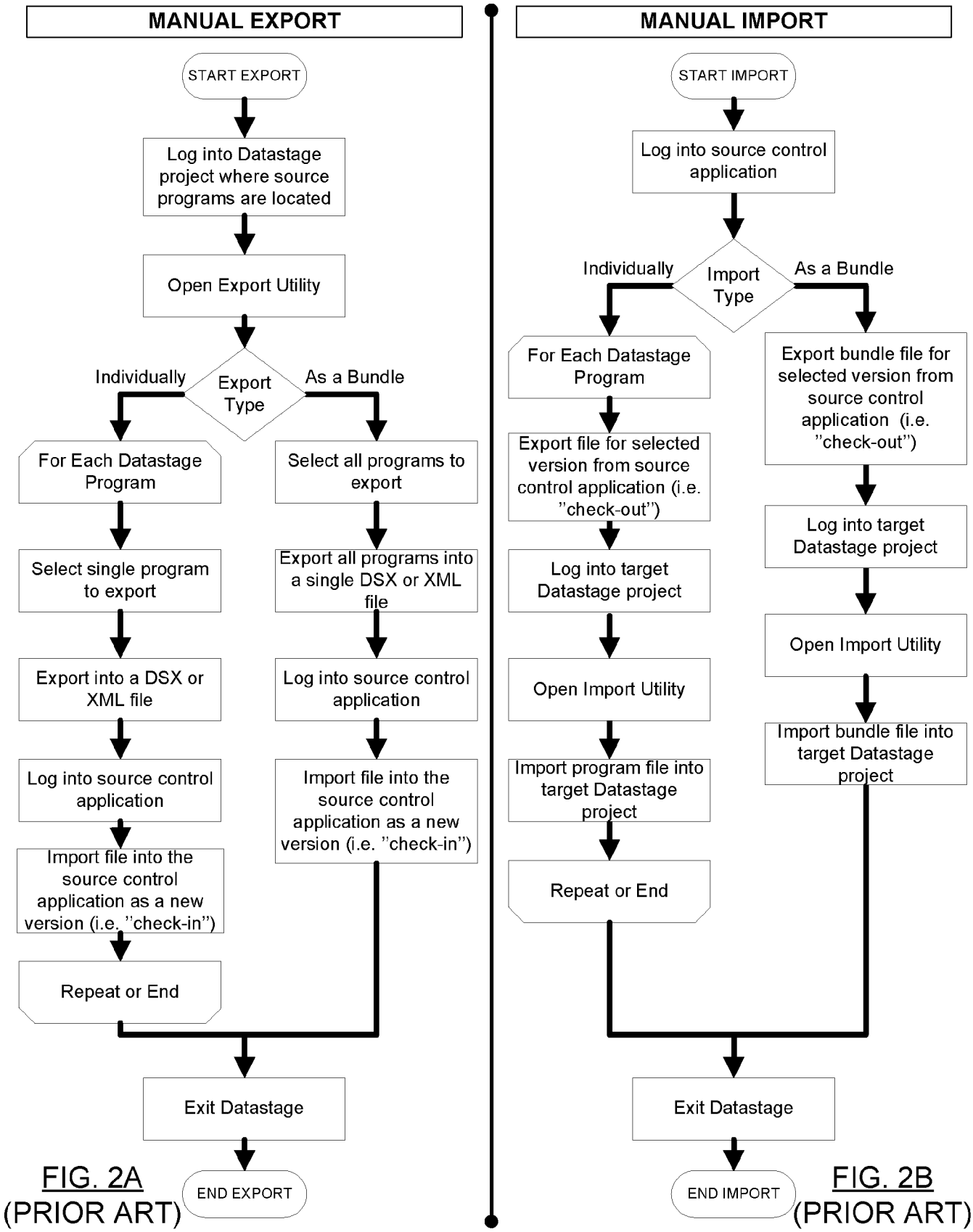


FIG. 2A (PRIOR ART)

FIG. 2B (PRIOR ART)

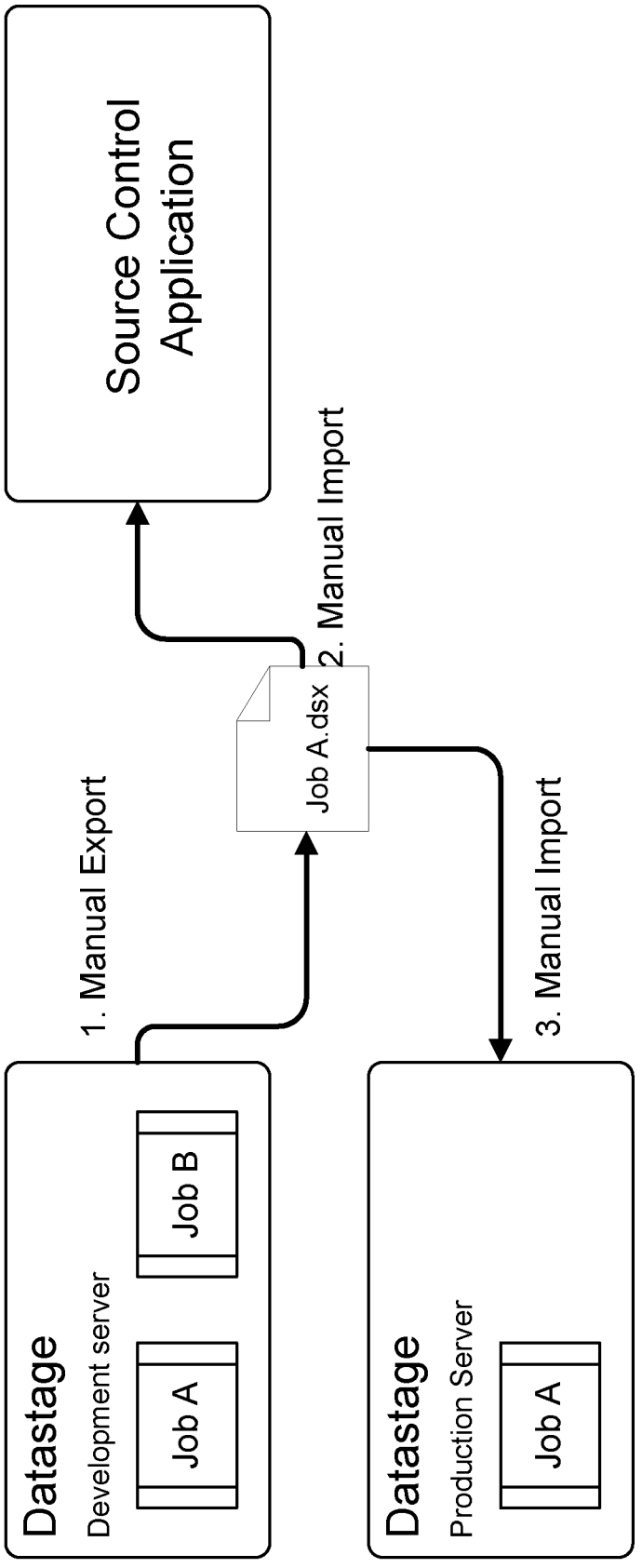


FIG. 3
(PRIOR ART)

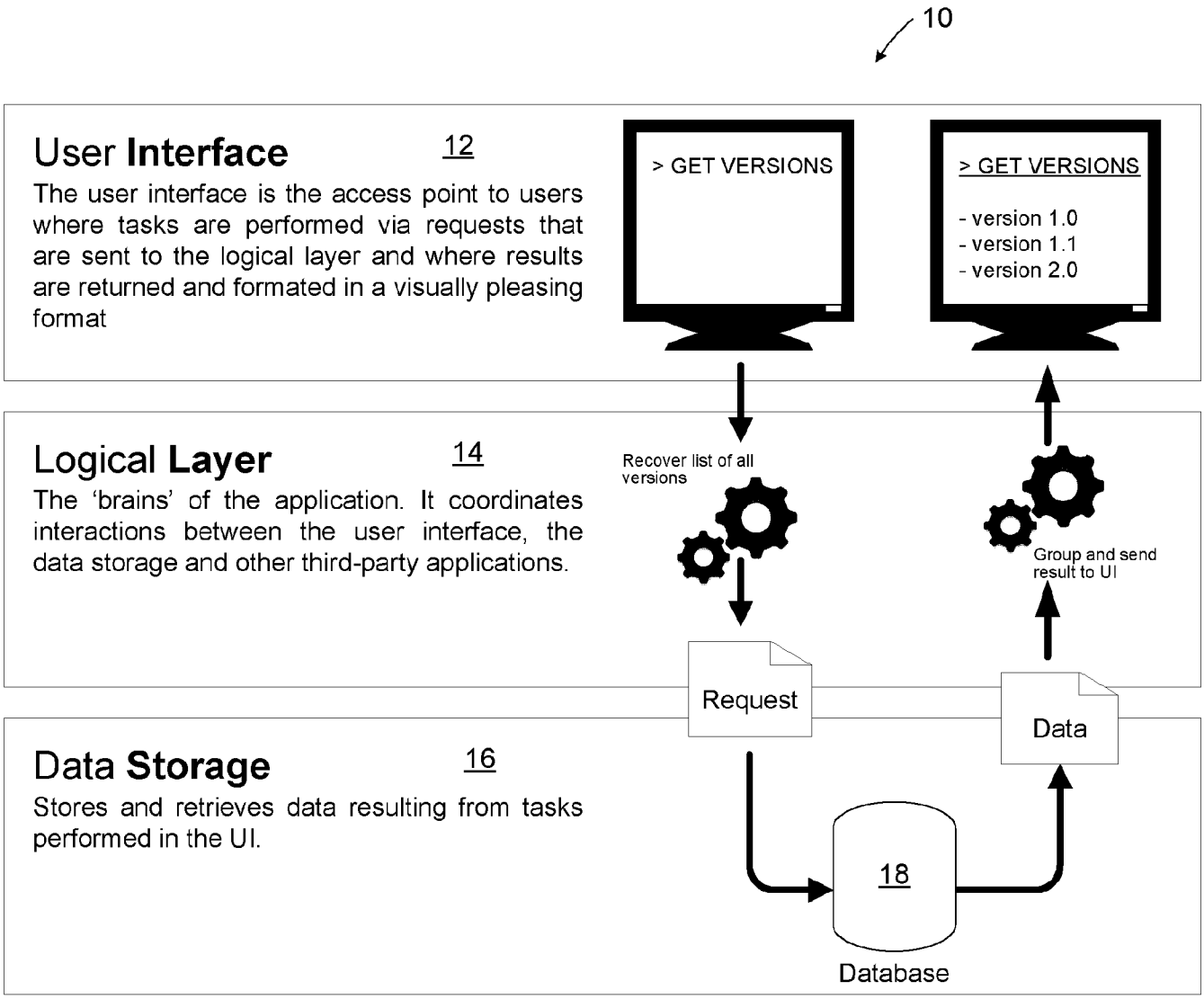

10
User Interface
12
The user interface is the access point to users where tasks are performed via requests that are sent to the logical layer and where results are returned and formated in a visually pleasing format
> GET VERSIONS
> GET VERSIONS
- version 1.0
- version 1.1
- version 2.0
Logical Layer
14
The 'brains' of the application. It coordinates interactions between the user interface, the data storage and other third-party applications.
Recover list of all versions
Group and send result to UI
Request
Data
Data Storage
16
Stores and retrieves data resulting from tasks performed in the UI.
18
Database

FIG. 4

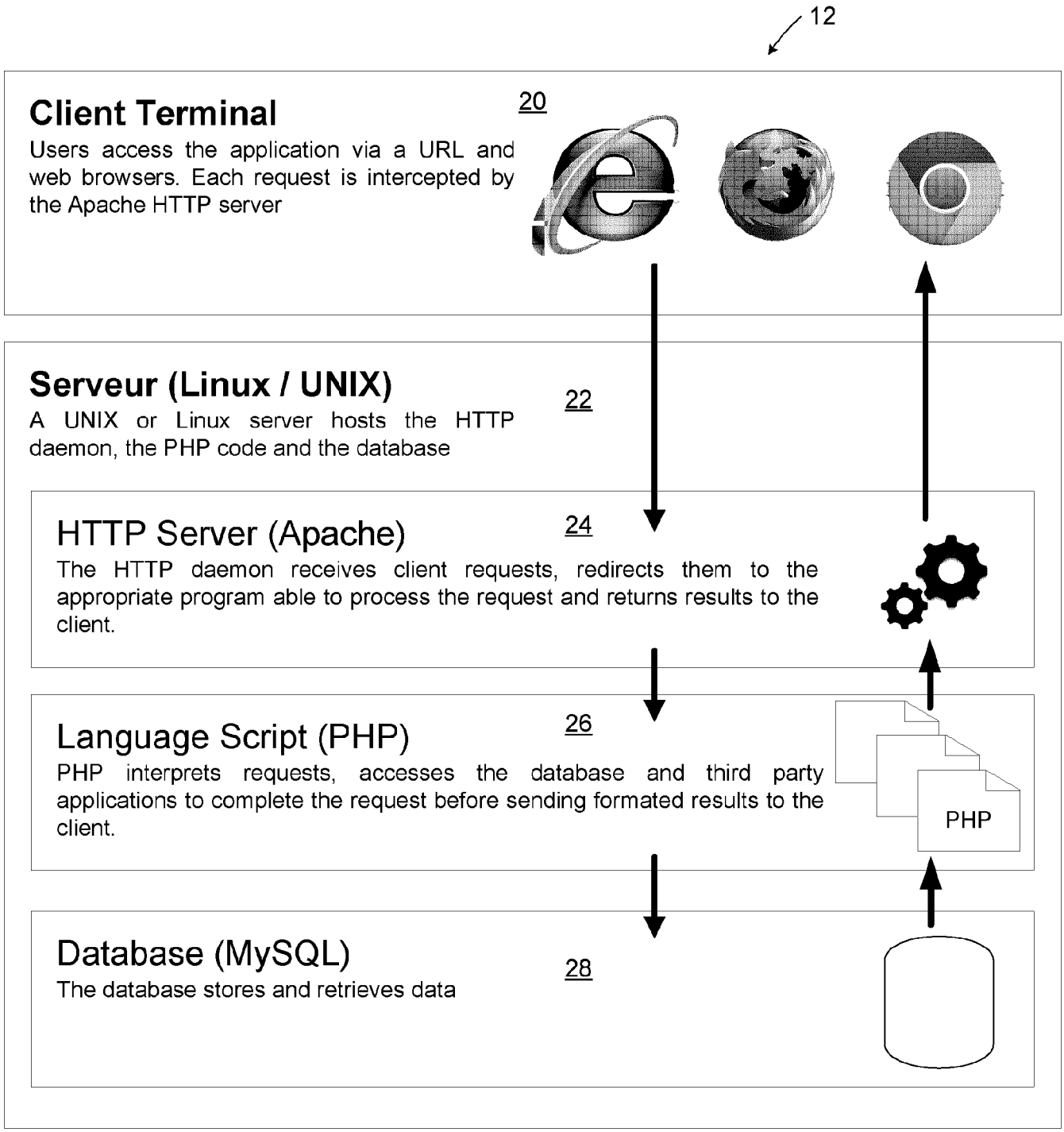

12
Client Terminal
Users access the application via a URL and web browsers. Each request is intercepted by the Apache HTTP server
20
Serveur (Linux / UNIX)
A UNIX or Linux server hosts the HTTP daemon, the PHP code and the database
22
HTTP Server (Apache)
24
The HTTP daemon receives client requests, redirects them to the appropriate program able to process the request and returns results to the client.
Language Script (PHP)
26
PHP interprets requests, accesses the database and third party applications to complete the request before sending formated results to the client.
PHP
Database (MySQL)
28
The database stores and retrieves data

FIG. 5

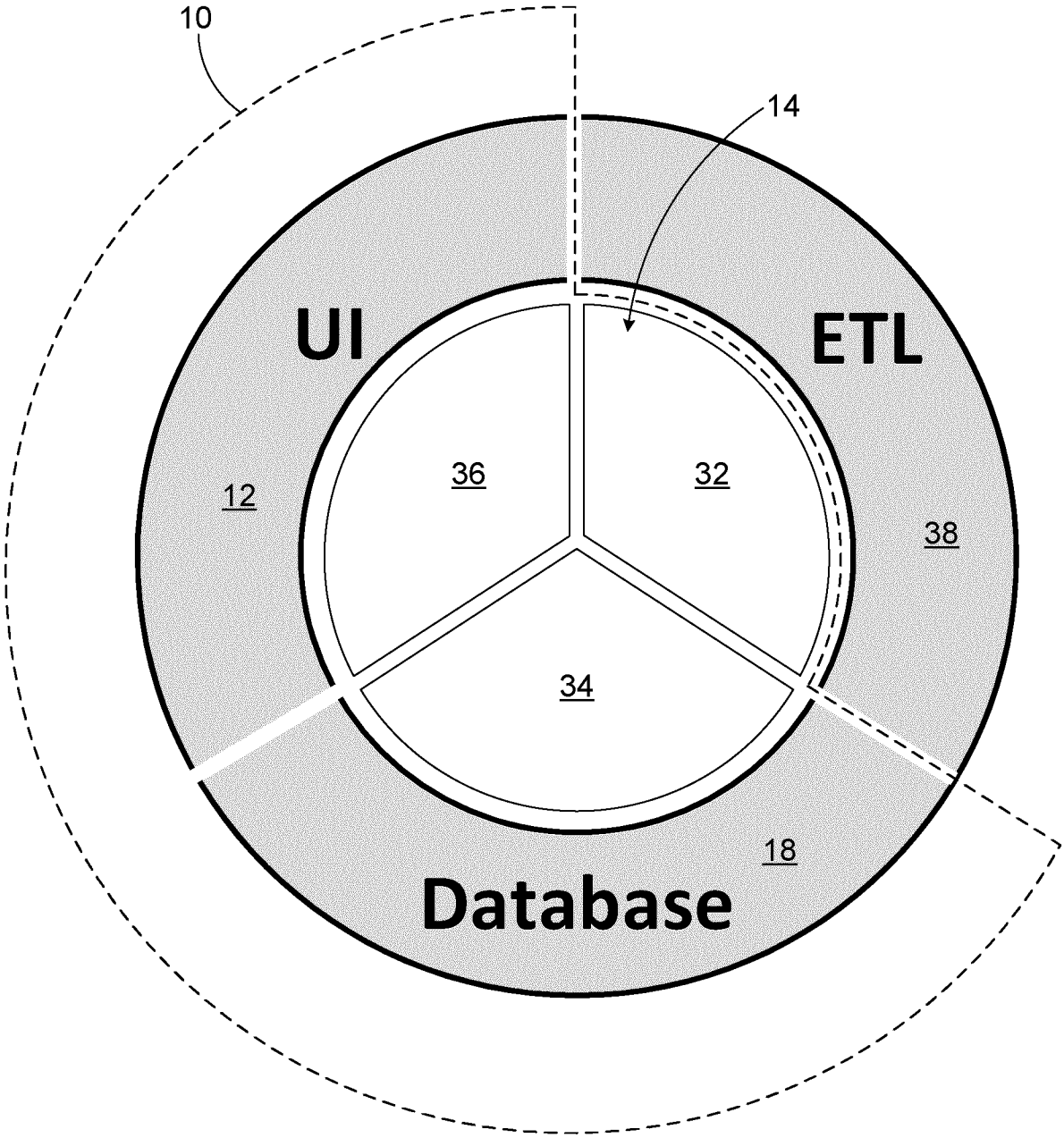


FIG. 6

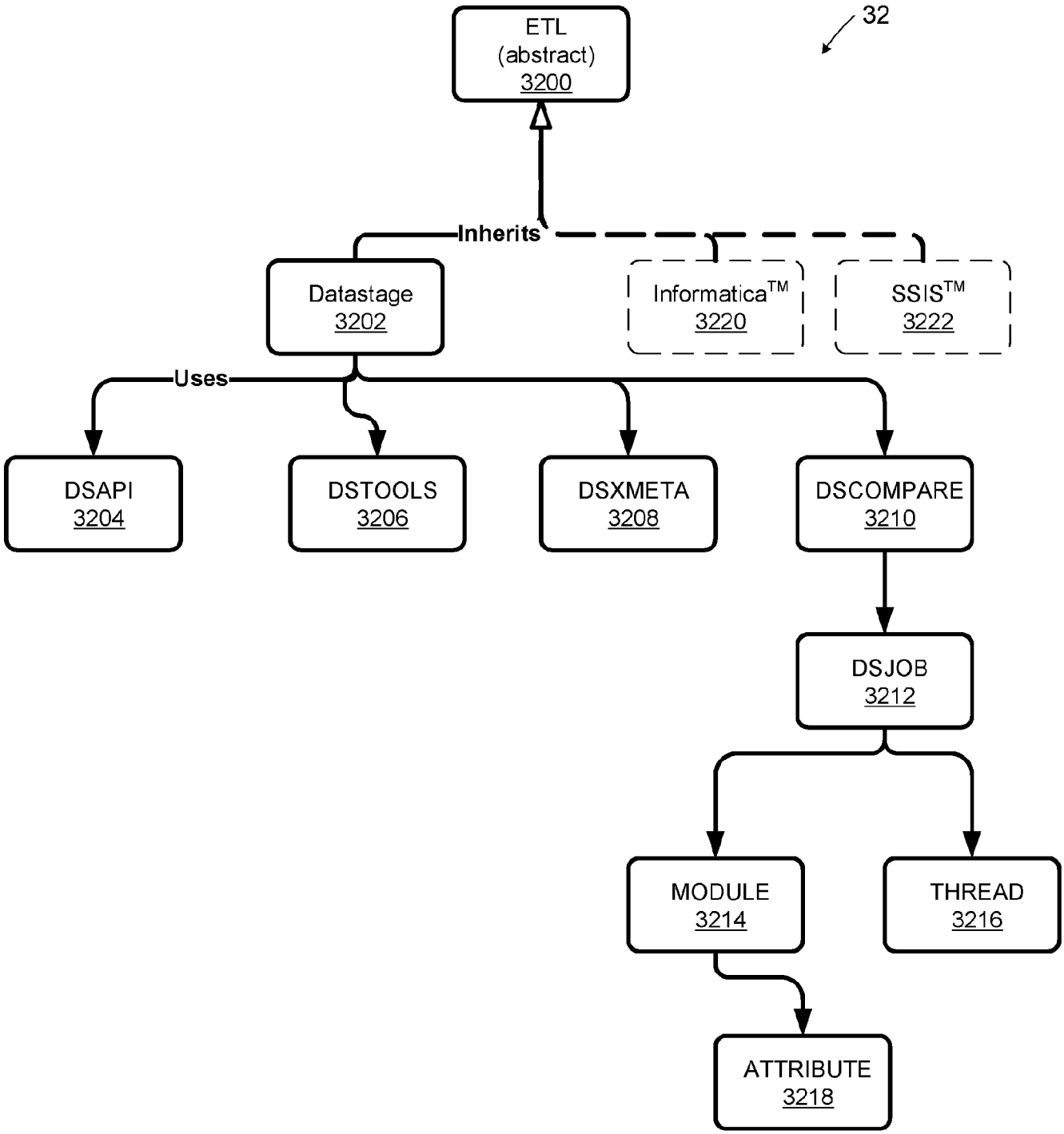

32
ETL
(abstract)
3200
Inherits
Datastage
3202
Informatica™
3220
SSIS™
3222
Uses
DSAPI
3204
DSTOOLS
3206
DSXMETA
3208
DSCOMPARE
3210
DSJOB
3212
MODULE
3214
THREAD
3216
ATTRIBUTE
3218

FIG. 7

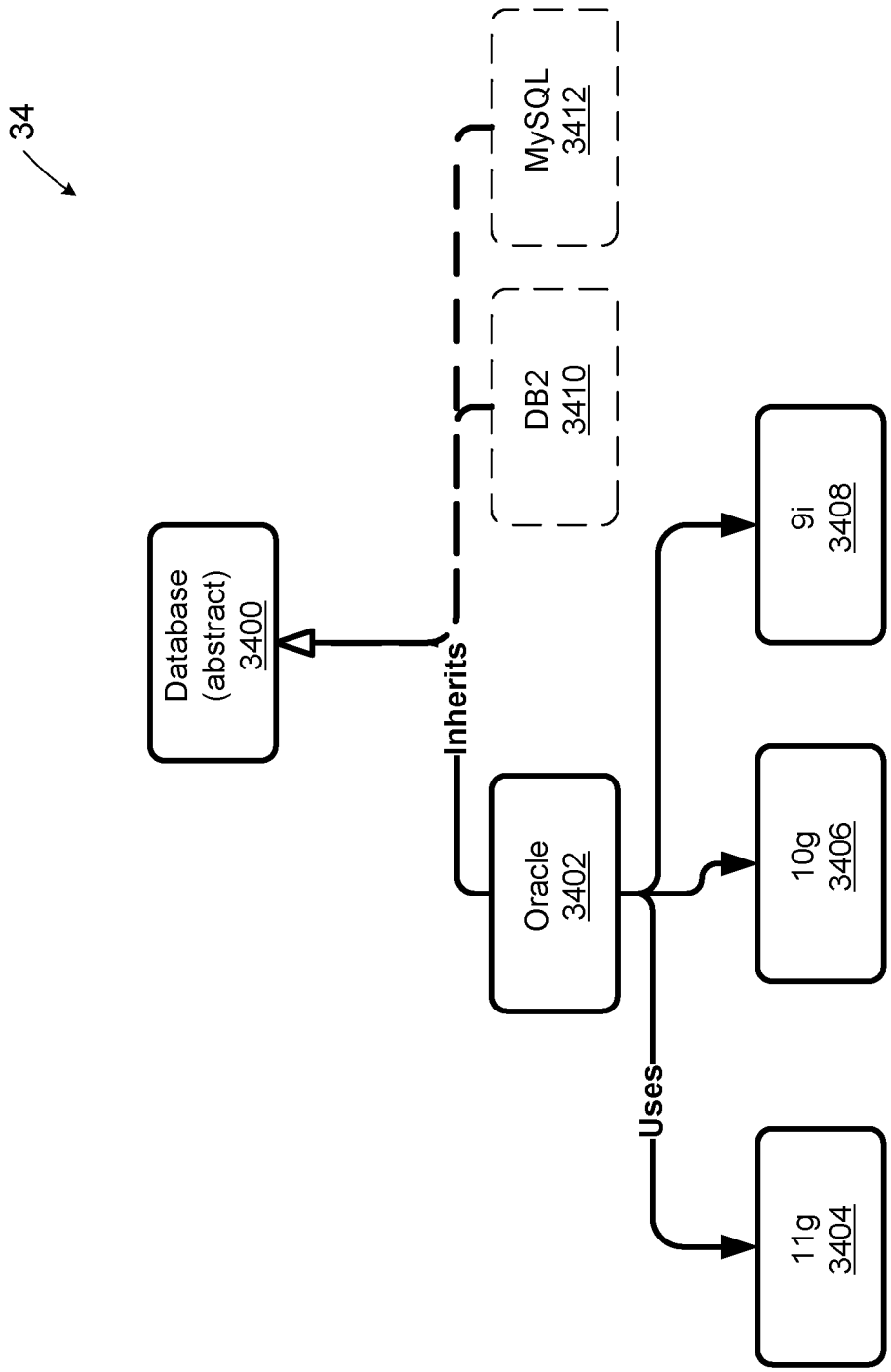
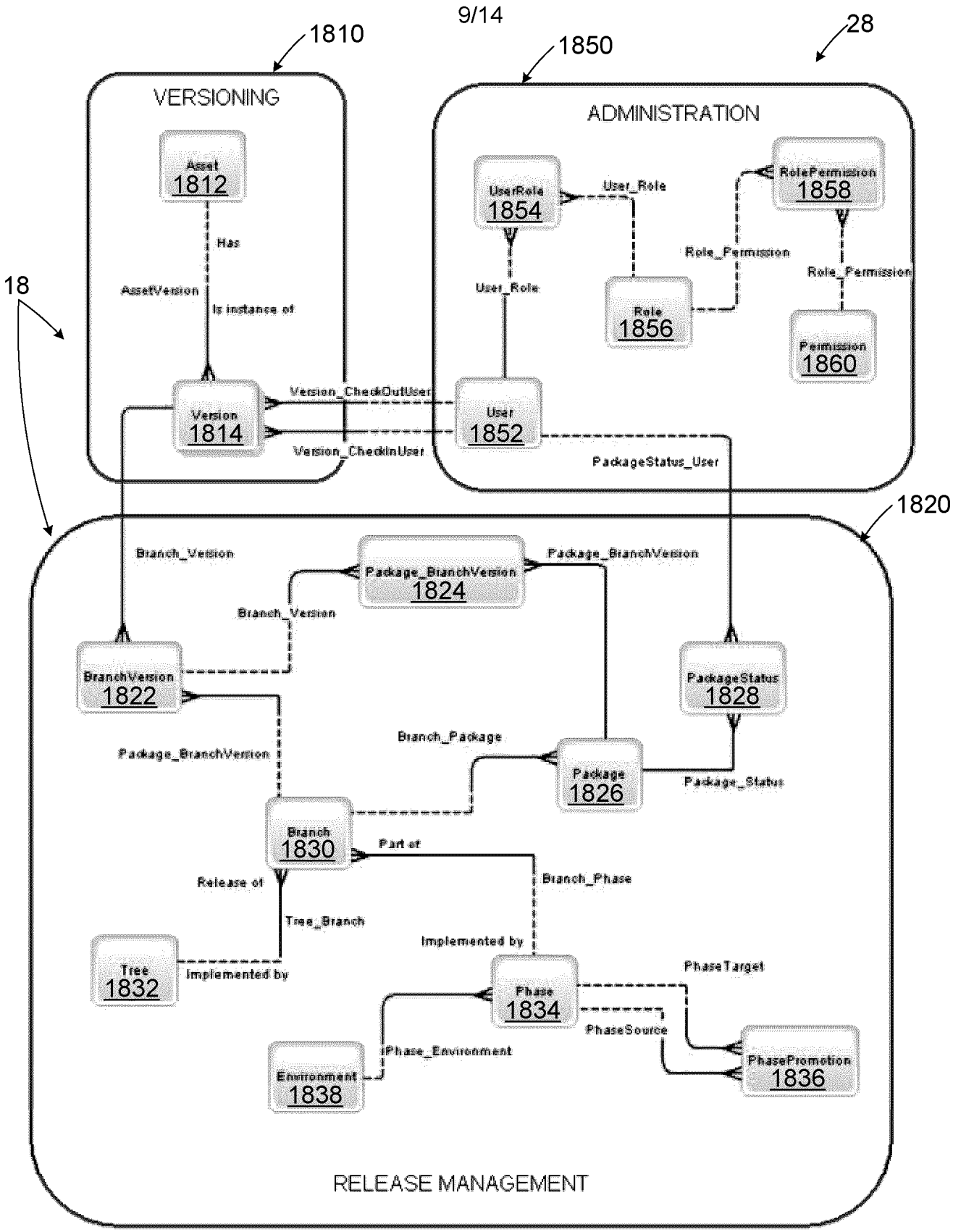

34
Database (abstract) 3400
Inherits
DB2 3410
MySQL 3412
Oracle 3402
9i 3408
10g 3406
11g 3404
Uses

FIG. 8



Conceptual Database Model

FIG. 9

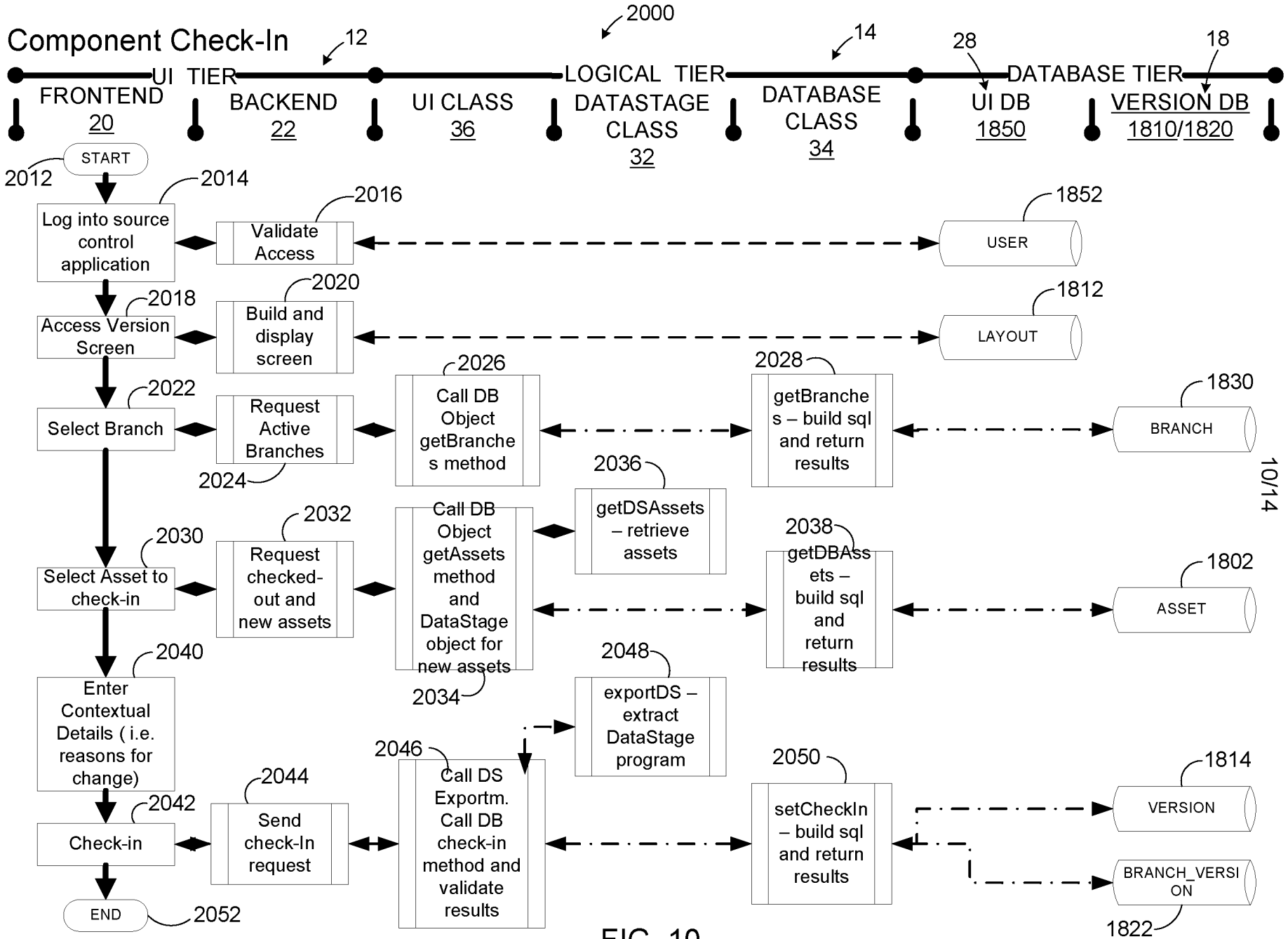

Component Check-In
2000
12
14
28
18
UI TIER
LOGICAL TIER
DATABASE TIER
FRONTEND 20
BACKEND 22
UI CLASS 36
DATASTAGE CLASS 32
DATABASE CLASS 34
UI DB 1850
VERSION DB 1810/1820
START
2012
Log into source control application
2014
Validate Access
2016
USER
1852
Access Version Screen
2018
Build and display screen
2020
LAYOUT
1812
Select Branch
2022
Request Active Branches
2024
Call DB Object getBranches method
2026
getBranches – build sql and return results
2028
BRANCH
1830
Select Asset to check-in
2030
Request checked-out and new assets
2032
Call DB Object getAssets method and DataStage object for new assets
2034
getDSAssets – retrieve assets
2036
getDBAssets – build sql and return results
2038
ASSET
1802
Enter Contextual Details (i.e. reasons for change)
2040
exportDS – extract DataStage program
2048
Check-in
2042
Send check-In request
2044
Call DS Exportm. Call DB check-in method and validate results
2046
setCheckIn – build sql and return results
2050
VERSION
1814
BRANCH_VERSION
1822
END
2052

FIG. 10

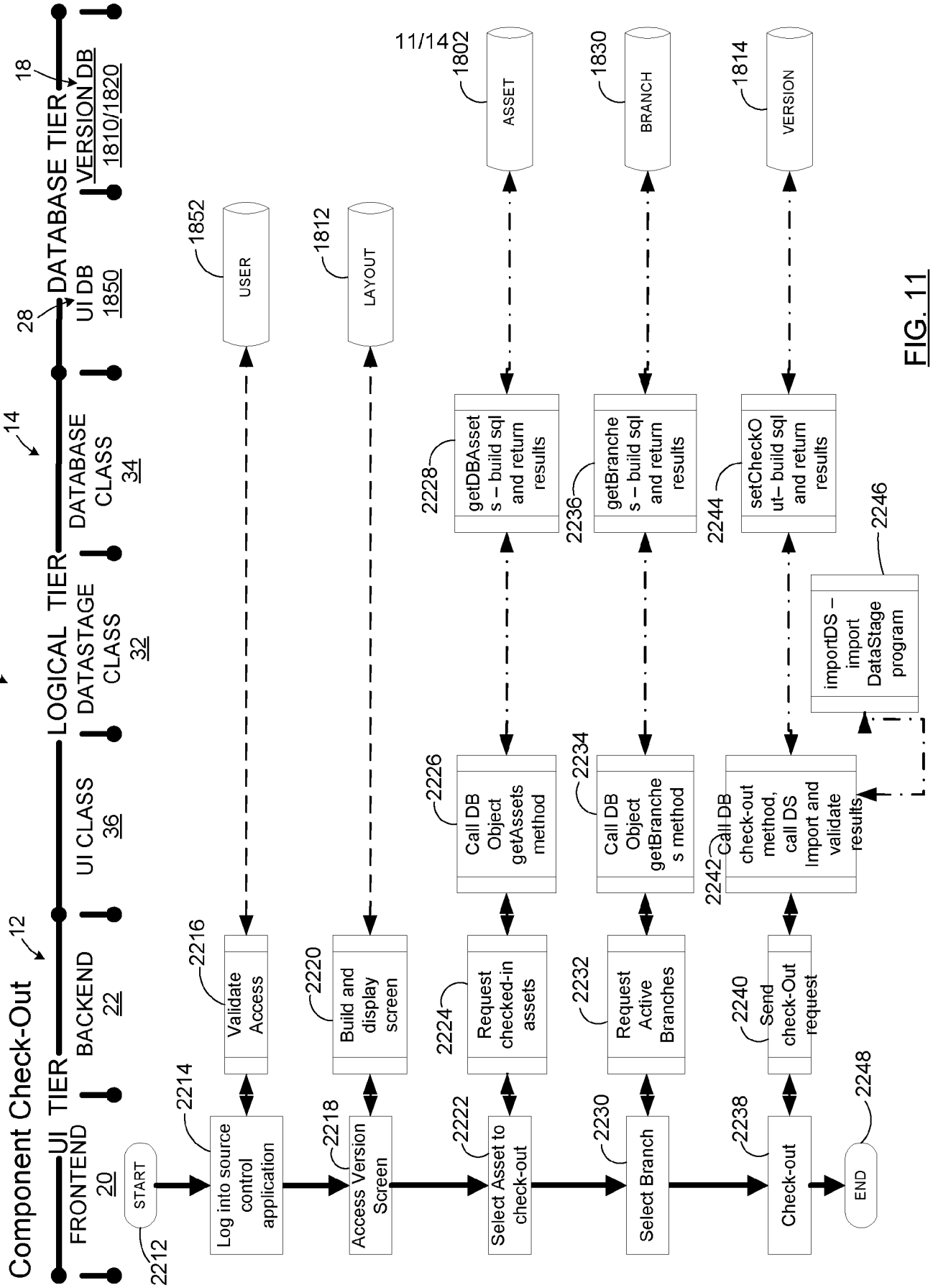

2200
Component Check-Out
12
UI TIER
FRONTEND 20
BACKEND 22
UI CLASS 36
LOGICAL TIER
14
DATASTAGE CLASS 32
DATABASE CLASS 34
28
DATABASE TIER
18
UI DB 1850
VERSION DB 1810/1820
11/14
1852 USER
1812 LAYOUT
1802 ASSET
1830 BRANCH
1814 VERSION
START 2212
2214 Log into source control application
2216 Validate Access
2218 Access Version Screen
2220 Build and display screen
2222 Select Asset to check-out
2224 Request checked-in assets
2226 Call DB Object getAssets method
2228 getDBAssets – build sql and return results
2230 Select Branch
2232 Request Active Branches
2234 Call DB Object getBranches method
2236 getBranches – build sql and return results
2238 Check-out
2240 Send check-Out request
2242 Call DB check-out method, call DS Import and validate results
2244 setCheckOut– build sql and return results
2246 importDS – import DataStage program
END 2248

FIG. 11

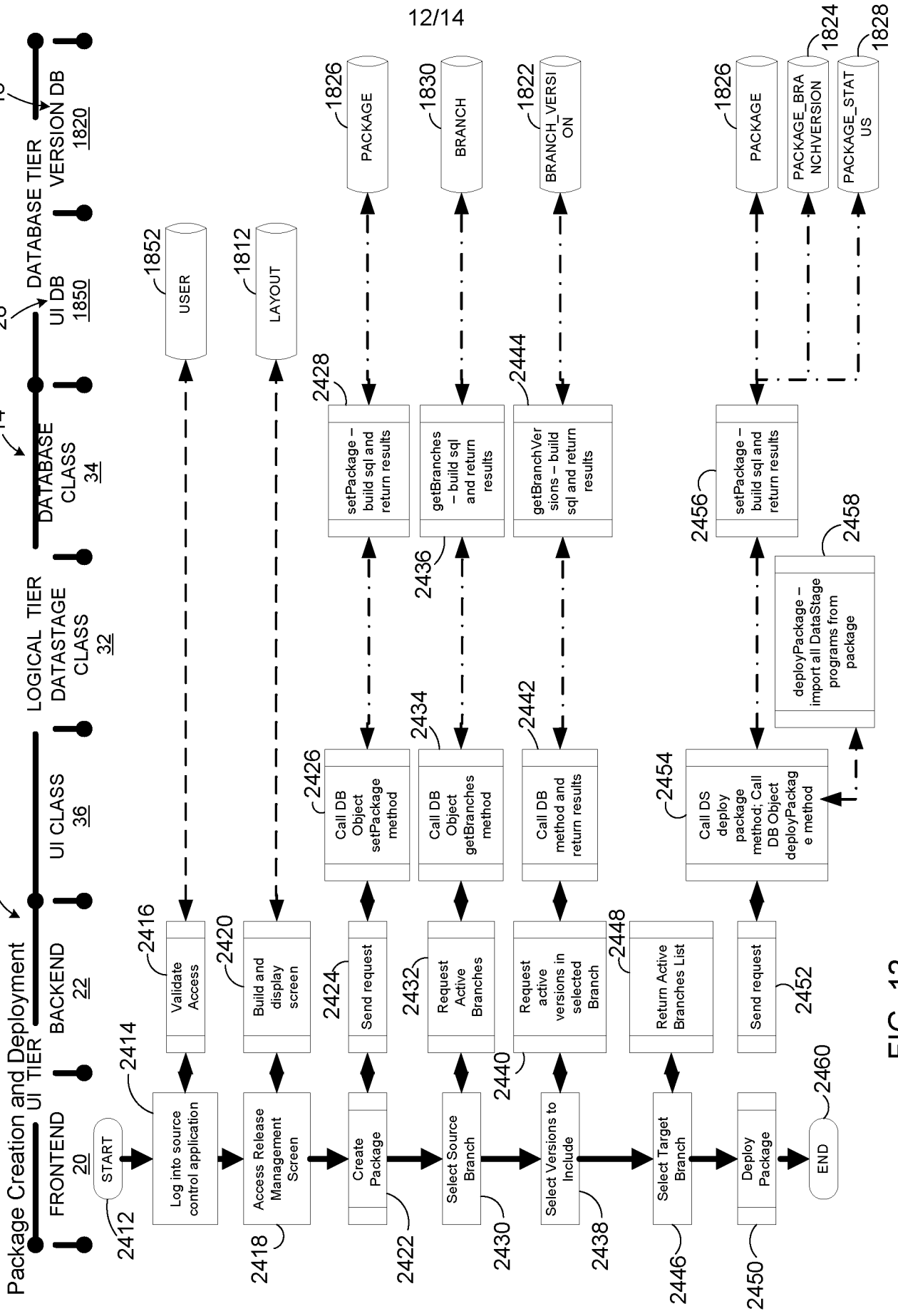

2400
Package Creation and Deployment
UI TIER
FRONTEND 20
BACKEND 22
12
UI CLASS 36
LOGICAL TIER
DATASTAGE CLASS 32
14
DATABASE CLASS 34
28
DATABASE TIER
UI DB 1850
VERSION DB 1820
18
USER
1852
LAYOUT
1812
PACKAGE
1826
BRANCH
1830
BRANCH_VERSION
1822
PACKAGE_BRANCHVERSION
1824
PACKAGE_STATUS
1828
START
2412
Log into source control application
2414
Validate Access
2416
Access Release Management Screen
2418
Build and display screen
2420
Create Package
2422
Send request
2424
Select Source Branch
2430
Request Active Branches
2432
Select Versions to Include
2438
Request active versions in selected Branch
2440
Select Target Branch
2446
Return Active Branches List
2448
Deploy Package
2450
Send request
2452
END
2460
Call DB Object setPackage method
2426
setPackage – build sql and return results
2428
Call DB Object getBranches method
2434
getBranches – build sql and return results
2436
Call DB method and return results
2442
getBranchVersions – build sql and return results
2444
Call DS deploy package method; Call DB Object deployPackage method
2454
setPackage – build sql and return results
2456
deployPackage – import all DataStage programs from package
2458

FIG. 12

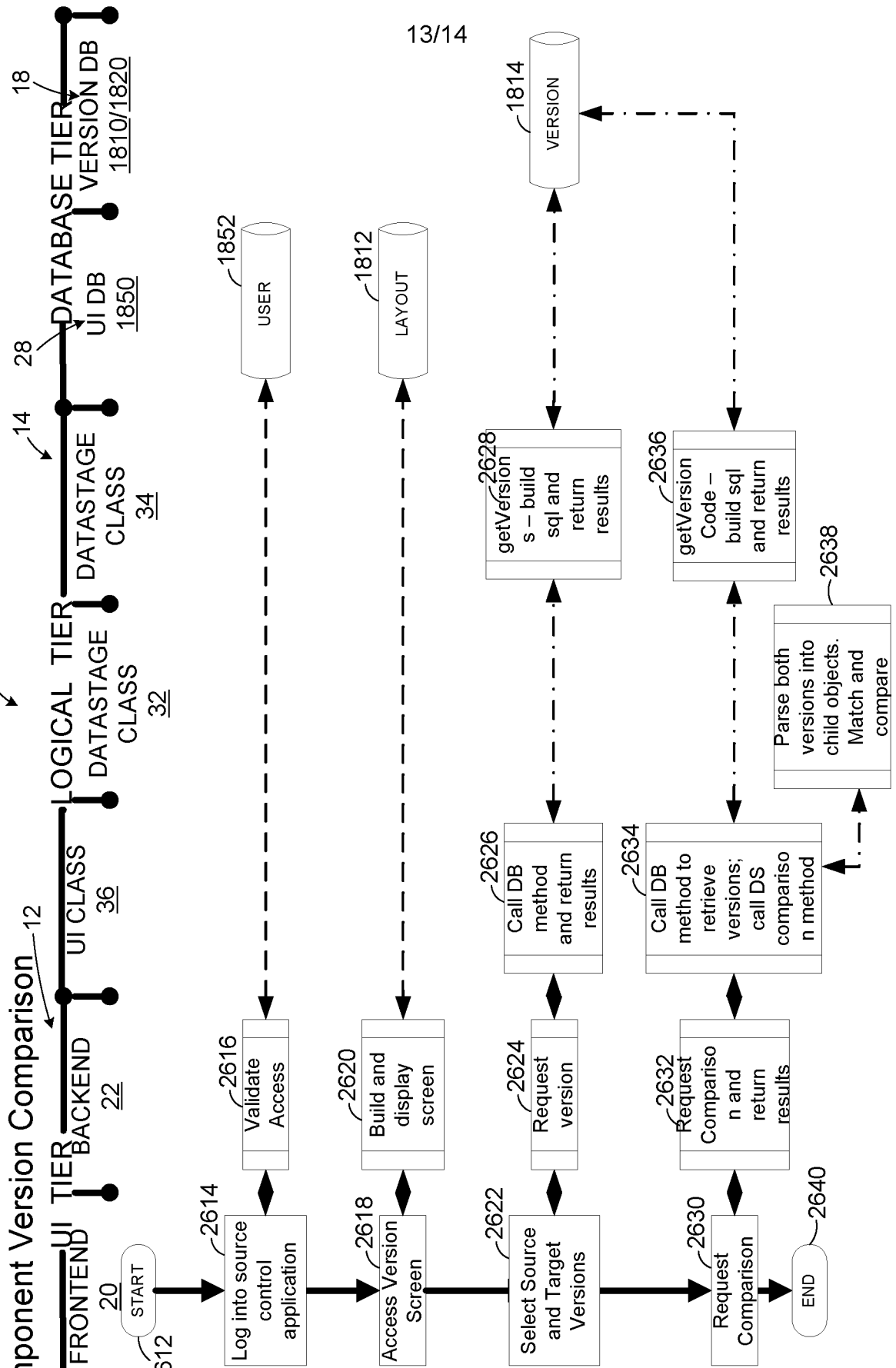

2600
Component Version Comparison
12
FRONTEND UI TIER 20
BACKEND 22
UI CLASS 36
LOGICAL TIER
DATASTAGE CLASS 32
DATASTAGE CLASS 34
14
28
DATABASE TIER
UI DB 1850
VERSION DB 1810/1820
18
USER
1852
LAYOUT
1812
VERSION
1814
START
2612
Log into source control application
2614
Validate Access
2616
Access Version Screen
2618
Build and display screen
2620
Select Source and Target Versions
2622
Request version
2624
Call DB method and return results
2626
getVersion s – build sql and return results
2628
Request Comparison
2630
Request Compariso n and return results
2632
Call DB method to retrieve versions; call DS compariso n method
2634
getVersion Code – build sql and return results
2636
Parse both versions into child objects. Match and compare
2638
END
2640

FIG. 13

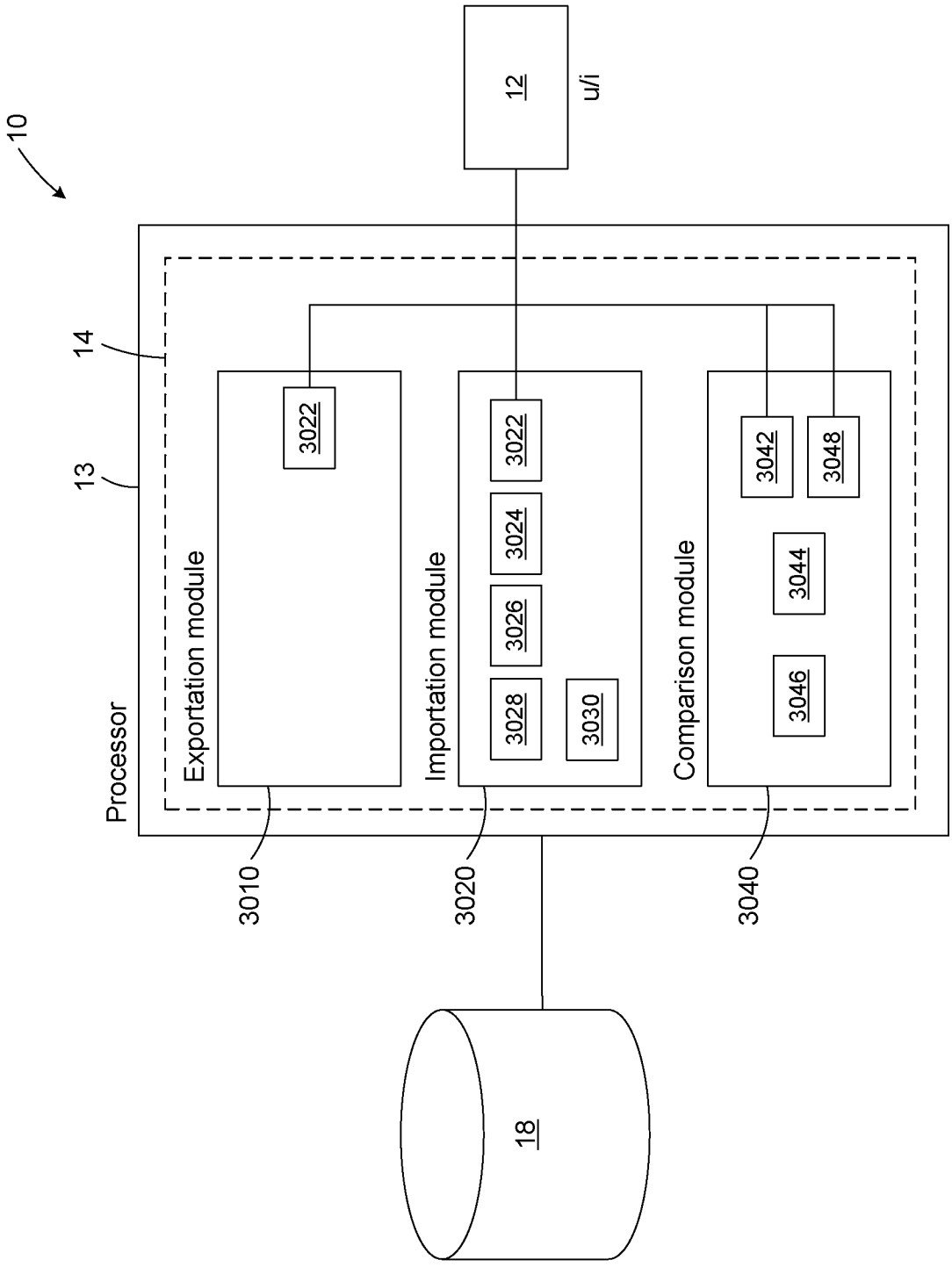

10
12
u/i
14
13
Processor
Exportation module
3022
Importation module
3022
3024
3026
3028
3030
Comparison module
3042
3048
3044
3046
3010
3020
3040
18

FIG. 14

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CA2013/050599

A. CLASSIFICATION OF SUBJECT MATTER
IPC: ***G06F 9/44*** (2006.01)
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)
IPC: ***G06F 9/44*** (2006.01)

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

Electronic database(s) consulted during the international search (name of database(s) and, where practicable, search terms used)
Epoque epodoc and txte; keywords: etl, version, library, protected, digest, extract, transform, datastage, source code, build
totalpatent: source code, digest, library, protected

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US20080082959A1 (Fowler) 03 April 2008 (03-04-2008) *abstract; par. 0085*	1-22
A	US20100293519A1 (Groves et al) 18 November 2010 (18-11-2010) *par. 0034; par. 0067*	1-22

[ ] Further documents are listed in the continuation of Box C.

[X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
23 October 2013 (23-10-2013)	25 October 2013 (25-10-2013)

Name and mailing address of the ISA/CA	Authorized officer
Canadian Intellectual Property Office Place du Portage I, C114 - 1st Floor, Box PCT 50 Victoria Street Gatineau, Quebec K1A 0C9 Facsimile No.: 001-819-953-2476	Howard Sandler (819) 994-0483

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/CA2013/050599

Patent Document Cited in Search Report	Publication Date	Patent Family Member(s)	Publication Date
US20080082959A1	03 April 2008 (03-04-2008)	WO2006043012A1 WO2006043012A9	27 April 2006 (27-04-2006) 27 July 2006 (27-07-2006)
US20100293519A1	18 November 2010 (18-11-2010)	US8413108B2	02 April 2013 (02-04-2013)