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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
25 September 2008 (25.09.2008)

PCT

(10) International Publication Number
WO 2008/115644 A1

(51) International Patent Classification:
G06F 15/16 (2006.01)

(21) International Application Number:
PCT/US2008/054319

(22) International Filing Date:
19 February 2008 (19.02.2008)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
11/725,258 19 March 2007 (19.03.2007) US

(71) Applicant *(for all designated States except US)*: **MICROSOFT CORPORATION** [US/US]; One Microsoft Way, Redmond, Washington 98052-6399 (US).

(72) Inventor: **MARIUS, Gabriel**; One Microsoft Way, Redmond, Washington 98052-6399 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, 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, KM, 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, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, 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, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MT, NL, NO, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:
— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*
— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:
— *with international search report*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

(54) **Title:** USING COLLABORATIVE DEVELOPMENT INFORMATION IN A TEAM ENVIRONMENT

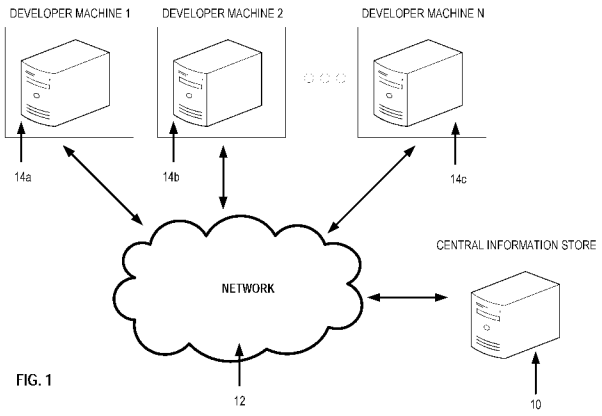

FIG. 1

(57) **Abstract:** Various technologies and techniques are disclosed that provide and interact with a collaborative development information store in a team software development environment. A submission service updates an active meta-model of an application in a central information store that is used by multiple users. A notification service operating in a particular software development environment receives notice that changes have been made to the active meta-model. Information received from the notification service is then used to update a display in the particular software development environment. On the database server, a reception service is provided that receives active meta-model information of the application being developed by the multiple users as the information changes. A storage service is provided to store the received active meta-model information in a specific relational database structure that is operable to allow artifacts to be added without alteration to the specific relational database structure.

WO 2008/115644 A1

USING COLLABORATIVE DEVELOPMENT INFORMATION IN A TEAM ENVIRONMENT

BACKGROUND

[001] Software applications must be transformed from source code into machine instructions in order for the application to execute. This transformation process is called "compiling". During compilation, the source code is first turned into a language-agnostic and machine-agnostic set of instructions, sometimes referred to as "intermediate code". The intermediate code is then turned into machine instructions specific to the particular computer platform on which the particular application will run. Compilation typically occurs on a single computer, and when completed, the intermediate code generated by the compilation process is discarded. Thus, any useful information that could be obtained from this intermediate code later in the development process is lost.

[002] In many cases, software applications are developed in team-based environments. These teams are comprised of team members playing several roles that support the overall project goals. Each team member typically runs the software development application on their own local computer. When a particular team member compiles the program, runs a code analyzer to analyze performance, performs debugging, or other various development-related tasks, the resulting details from these processes are typically stored on that team member's local computer. While that team member may submit the source code to a source code control server that the entire team can access, the various system generated artifacts resulting from the development process that led up to the version being checked in to the server are typically either lost or are not easily distributable. This means that other team members do not get whatever benefit may be gained by accessing those results. For example, as a developer team member is working on a given application, the developer may use a code profiler to analyze the performance of the application. In doing so, the developer gains specific knowledge about the applications performance. The artifacts and knowledge gained by creating the artifacts from profiling the applications are local to the developer and are not easily shared.

[003] Furthermore, in the course of executing their assigned roles, team members often create other artifacts related to the project in addition to the source code itself, such as models, diagrams, work items, etc. Just like with the system generated artifacts, in many cases, these user-created artifacts are also stored on a particular team member's computer only, are lost, or are not easily distributable to the other team members. Thus, large amounts of valuable development data and artifacts related to a particular software development project being developed in a team environment are either dispersed across various team member computers and thus inaccessible by the entire team, or they are lost forever.

SUMMARY

[004] Various technologies and techniques are disclosed that provide and interact with a collaborative development information store in a team software development environment. A submission service updates an active meta-model of an application in a central information store that is used by multiple users. A notification service operating in a particular software development environment receives notice that changes have been made to the active meta-model. Information received from the notification service is then used to update a display in the particular software development environment being used by a particular user. As one non-limiting example, the information can be displayed as the particular user is typing code to note something relevant to that particular code that has occurred as a result of an action of another team member.

[005] On the database server, a reception service is provided that receives active meta-model information of the application being developed by the multiple users as the information changes. A storage service is provided to store the received active meta-model information in a specific relational database structure that is operable to allow artifacts to be added without alteration to the specific relational database structure.

[006] This Summary was provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the

claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter.

BRIEF DESCRIPTION OF THE DRAWINGS

[007] Figure 1 is a diagrammatic view of a computer system using a collaborative development information store.

[008] Figure 2 is a diagrammatic view of submission and notification application of one implementation of the system of Figure 1.

[009] Figure 3 is a diagrammatic view of meta-model storage application of one implementation operating on the system of Figure 1.

[010] Figure 4 is a process flow diagram for one implementation illustrating the high level stages of the system of Figure 1.

[011] Figure 5 is a process flow diagram for one implementation illustrating the stages involved in interacting with an active meta-model of an application in a central information store.

[012] Figure 6 is a diagram illustrating element-specific fields contained in the central information store in one implementation of the system of Figure 1.

[013] Figure 7 is a diagram illustrating an exemplary element definition that has been defined using the element-specific fields shown in Figure 6.

[014] Figure 8 is a diagram illustrating connection-specific fields contained in the central information store in one implementation of the system of Figure 1.

[015] Figure 9 is a diagram illustrating an exemplary connection definition that has been defined using the connection-specific fields shown in Figure 8.

[016] Figure 10 is a diagram illustrating a special property that can be used with a connection definition in one implementation of the system of Figure 1.

[017] Figure 11 is a diagrammatic view of a computer system of one implementation of the system of Figure 1.

DETAILED DESCRIPTION

[018] For the purposes of promoting an understanding of the principles of the invention, reference will now be made to the embodiments illustrated in the drawings and specific language will be used to describe the same. It will nevertheless be understood that no limitation of the scope is thereby intended. Any

alterations and further modifications in the described embodiments, and any further applications of the principles as described herein are contemplated as would normally occur to one skilled in the art.

[019] The system may be described in the general context as a software development application. One or more of the techniques described herein can be implemented as features within a software development program such as MICROSOFT® VISUAL STUDIO®, or from any other type of program or service that participates in the software development process in a team environment.

[020] In one implementation, a system is provided that uses a centralized database or other information store to store real-time, active meta-model information about a particular software application being developed by a team of developers and/or other users. The term "active meta-model" as used herein refers to system-generated artifacts and user-generated artifacts that reflect a current state of the application, the structure of the application, and a history of how the application has evolved. System-generated artifacts include application structure, intermediate code, annotations, and other analysis data that the system generates. User-generated artifacts include work items, project plans, diagrams, annotations explaining certain code, and so on that could potentially be associated with the underlying application structure. The centralized database is then accessed by the software development applications being used by the team of developers to provide the team with access to the active meta-model information. By having access to this information, developers are provided with real-time, relevant information about the active state of the application under development by a team of other people. As one non-limiting example, this information can include notifying developer A that a code profiling process that was run by developer B has just marked a certain portion of code as "slow" that developer A is about to call from his code. In one implementation, this is accomplished by associating the performance characteristics of a given code module, type, method etc. with the meta-representation. Thus, as developer B adds a call to the method in question, the associated performance data can be found which was added by A and action can be taken.

[021] As shown in Figure 1, an exemplary computer system to use for implementing one or more parts of the system includes one or more development machines, such as central information store 10, network 12, and development machines 14a, 14b, and 14c. Development machines are computers that are used by software developers for the purpose of writing software applications. Central information store 12 includes one or more computers that store data and allow other network resources to access that data. In one implementation, central information store 12 stores active meta-model information of an application being developed by multiple users in a team environment. In one implementation, a local central information store 12 can alternatively or additionally be stored on one or more of development machines 14a, 14b, and 14c, such as to allow participation in a more peer-to-peer level sharing on larger projects and/or for implementations that do not use a central store. Network 12 is used to provide communication between development machines 14a, 14b, 14c, and central information store 10. Network 12 can be implemented as a local area network, wide area network, over the Internet, using a wired or wireless connection, and/or in other such variations as would occur to one of ordinary skill in the computer software art.

[022] Turning now to Figure 2 with continued reference to Figure 1, a submission and notification application 200 operating on computing device 600 is illustrated. Submission and notification application 200 is one of the application programs that reside on computing device 600 (of Figure 11). In one implementation, submission and notification application 200 is located on one or more of developer workstations 14A, 14B, or 14C (from Figure 1). However, it will be understood that submission and notification application 200 can alternatively or additionally be embodied as computer-executable instructions on one or more computers and/or in different variations than shown on Figure 11. Alternatively or additionally, one or more parts of submission and notification application 200 can be part of system memory 604 (of Figure 11), on other computers and/or applications 615 (of Figure 11), or other such variations as would occur to one in the computer software art. Submission and notification application 200 includes program logic 204, which is responsible for carrying out some or all of the techniques described herein.

Program logic 204 includes logic for providing a submission service that is operable to submit at least a portion of an active meta-model (e.g. intermediate code, annotations, user-generated artifacts, etc.) of an application to a central information store for use by a plurality of users 206; logic for providing a notification service that is operable to receive a notification relating to a change in the active meta-model (e.g. by a process, by a user, etc.) 208; logic for enabling notification service to be further operable to display meta-model information to the user 210; logic for enabling notification service to be integrated into a development environment such that meta-model information is displayed therein 212; and other logic for operating the application 220. In one implementation, program logic 204 is operable to be called programmatically from another program, such as using a single call to a procedure in program logic 204.

[023] Turning now to Figure 3 with continued reference to Figure 1, a meta-model storage application 240 operating on computing device 600 (of Figure 11) is illustrated. Meta-model storage application 240 is one of the application programs that reside on computing device 600. However, it will be understood that meta-model storage application 240 can alternatively or additionally be embodied as computer-executable instructions on one or more computers and/or in different variations than shown on Figure 11. Alternatively or additionally, one or more parts of meta-model storage application 240 can be part of system memory 604 (of Figure 11), on other computers and/or applications 615 (of Figure 11), or other such variations as would occur to one in the computer software art.

[024] Meta-model storage application 240 includes program logic 244, which is responsible for carrying out some or all of the techniques described herein. Program logic 244 includes logic for providing a reception service to receive active meta-model information of an application being developed by a plurality of users 246; logic for providing a storage service that is operable to store the received active meta-model information in a specific relational database structure that is operable to allow artifacts to be added without alteration to the specific relational database structure 248; logic for receiving a request to retrieve active meta-model information from a notification service that is operable to notify one or more client

devices of changes to the active meta-model 250; logic for enabling the specific relational database structure to store a name identifying the specific element and data describing the element 252; logic for enabling the specific relational database structure to store a name identifying the specific connection, data describing the specific connection, the source of the specific connection, and the destination of the specific connection 254; logic for enabling the specific relational database structure to store a field containing relevancy information for a specific connection 256; and other logic for operating the application 260. In one implementation, program logic 244 is operable to be called programmatically from another program, such as using a single call to a procedure in program logic 244.

[025] Turning now to Figures 4-5 with continued reference to Figure 1, the stages for implementing one or more implementations of the system of Figure 1 are described in further detail. Figure 4 is a process flow diagram illustrating the high-level stages for one implementation of the system of Figure 1. In one form, the process of Figure 4 is at least partially implemented in the operating logic of computing device 600 (of Figure 11). While the steps identified in Figure 4 are described in a certain order, it will be appreciated that these steps can occur in any order, and/or simultaneously with each other, or not at all. The process begins at start point 270 with providing a framework for dynamic definition of elements and relevancy that are used to express system-generated and user-generated artifacts under development by multiple users in a team environment (stage 272). The system-generated and user-generated artifacts are stored as an active meta-model in a central information store as each team member creates and/or changes the artifacts (stage 274). The software development application used by each team member periodically interacts with the central information store to identify relevant updates that have been made to the active meta-model (stage 276). The updated information is used appropriately, such to notify a particular team member of a problem, of information that would be helpful to something they are currently doing, etc. (stage 278). The process ends at end point 280.

[026] Figure 5 is a process flow diagram for one implementation illustrating the stages involved in interacting with an active meta-model of an application in a

central information store. In one form, the process of Figure 5 is at least partially implemented in the operating logic of computing device 600 (of Figure 11). While the steps identified in Figure 5 are described in a certain order, it will be appreciated that these steps can occur in any order, and/or simultaneously with each other, or not at all. The process begins at start point 290 with communicating with a submission service, such as a component of the software development application, to update an active meta-model of an application in a central information store (stage 292). In one implementation, these updates are sent by each respective team member workstation as the team member further develops a particular software application. At any given moment, one or more of the respective team member workstations can receive notifications from a notification service of changes to the active meta-model (stage 294). These changes could have been inspired by work done by the particular team member himself, or by a different team member. The notifications can be configurable beyond just changes, and can be for varying reasons indicate a particular event has happened with the underlying information. For example, the logic that is used to identify the corresponding notifications and people can be limited and/or enhanced based upon policy. As an example of this, a developer working on a section of code a year ago may not receive an update notification of a specification change that just happened, but a developer that worked on the source code two weeks ago would, based on a system-wide setting to indicate who should be notified and when.

[027] The notification service is used to update a display in the software development application used by a respective team member with appropriate information describing the change (stage 296). In one implementation, the software development application intercepts and analyzes the changes to the active-meta model and then determines when and how to notify the user. In another implementation, the software development application receives the notices and just displays them without filtering and/or interpreting them. Some non-limiting examples of the type of information that can be displayed to users include policy violation alerts, dependency alerts, performance alerts, modification alerts, etc. (stage 296). A policy violation alert can indicate that the user has made a change

that impacted the active-meta model in a way that violates a policy set for the system. A dependency alert can inform the user that some code they are changing now depends on something that is no longer available, that has been marked as slow, that is currently being edited by another user, and so on. A performance alert can inform the user that that same code was marked as slow performing by another user who worked on it before. These are just a few non-limiting examples of the types of information that the system can issue upon analyzing the active meta-model contained in the central information store. The process ends at end point 298.

[028] Turning now to Figures 6-10, several diagrams are used to illustrate an exemplary database structure that can be used to implement the active meta-model stored in the central information store and used in the team development environment. Figure 6 is a diagram illustrating element-specific fields 350 contained in the central information store in one implementation of the system of Figure 1. In one implementation, an abstract storage definition is necessary since programs and annotations vary widely. However, the overall structure of the schema described herein for one implementation is non-limiting, and various other database schemas could also be used for implementing some or all of the techniques discussed herein. In one implementation, elements are generic items which serve as nouns in the central information store. A few non-limiting examples of elements include methods, types, assemblies, metadata attributes, etc. Individual elements can have any number of properties as needed in their general definition. Each tool, vendor, and/or system can define elements dynamically and provide data as needed. Each element contains a name field 352 as well as the data field 354. The name field 352 is a string that describes the element in the system. The data field 354 is a binary packet that stores details about the element.

[029] Figure 7 is a diagram 370 that illustrates an example of an element that is being used to define an assembly in one implementation. The program structure is part of the general system provided information. The assembly is not referenced by the data store as an element, but tools from vendors and/or systems can annotate these assembly elements as needed. This in turn enables developers that work on

lower level parts of this particular assembly to get the associated information. Without the shared element in the system, the association between tools working from the assembly on local desktops would be difficult to correlate to the smaller artifacts and/or work happening to the assembly. Suppose for example that developer X works on a test case for method 1 that is part of a class that is part of a namespace that is part of assembly Z which is marked as not needing test coverage at all. The system is now able to notify developer X that the test case being written is non-essential, though the user defining the requirement on assembly Z had no specific knowledge of what was happening on developer X's desktop.

[030] Turning now to Figure 8, a diagram is shown that illustrates connection-specific fields 400 contained in the central information store in one implementation of the system of Figure 1. In one implementation, connections are generic items which serve as verbs in the central information store. A few non-limiting examples of connections include depends, contains, authored, etc. Connections are used to relate two or more elements. Each connection contains a source field 402 and a destination field 404. Figure 9 is a diagram 450 that illustrates an example of a connection that is being used to define a "depends" connection type. The central information store relies on the expression of relationships between defined types. These relationships serve as dynamic working links between nouns and provide an additional dimension of relation. For example, suppose you have an Assembly A as an element, and a Class Z as an element. Both can have annotations from tools associated with them once the system reflects these in the shared store. However, a Connection of type "contains" would further express that Assembly A contains Class Z and thus allows associative lookups to take place where without the connection, this association may not be possible. In one implementation, dynamic relationships are themselves dynamically defined.

[031] Figure 10 is a diagram 500 illustrating a special relevancy property that can be used with a connection definition in one implementation of the system of Figure 1. In one implementation, relevancy is a specialized property associated with every connection. In general terms, relevancy expresses how important a relationship between two elements is in relation to relationships between elements. A specific

relevancy value for a relationship between two elements is assigned to the connection between the elements. Relevancy units can be defined by the individual vendor noting the connection and/or can be defined by the system. As the changes are made to the central information store, the overall use of the data and its effectiveness is reflected by the store. Relevancy provides a mechanism to enable weighting of verbs between artifacts. In this way, the system is able to use various forms of relationships differently. In one implementation, relevancy is an important part of the system as associations between items can number in the hundreds of thousands. In other implementations, relevancy is less important and/or is not even used.

[032] In one implementation, trust is an additional specialized property given by the system to users, vendors and/or vendor-provided tools. As the system's data evolves, tool and user findings change over time. If tools are found to be inaccurate, the default trust weight associated with a specific vendor, tool or user can be changed. Doing so means the relevancy calculation is altered, weighting the calculation positively or negatively depending on the power of the trust factor. For example, static analysis of code can be used to find the links between test cases and code by examining the calling relationship of the test case to the development method. For this example, Tool X looks at these relationships and adds links between test code and development methods based on static relationships. By default, the system examines these relationships and notifies users of changes to modules connected by these links. If test A calls development method B, and method B changes, then the author of A should be notified. However, as the development cycle progresses, many times the users being notified of the changes may be notified when they do not care. This is because Tool X does not take into account additional factors (such as code complexity, number of calls, etc) that can mean a more or less important connection. Due to all of the negative feedback from users, the administrator or the system can lesson Tool X's Trust rating. As a direct result, when the calculation for which users are to be notified of a change takes place, a connection between two users from Tool X's links will be rated differently for Tool Y, in which case, Tool X's information will be trusted less.

[033] As shown in Figure 11, an exemplary computer system to use for implementing one or more parts of the system includes a computing device, such as computing device 600. In its most basic configuration, computing device 600 typically includes at least one processing unit 602 and memory 604. Depending on the exact configuration and type of computing device, memory 604 may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.) or some combination of the two. This most basic configuration is illustrated in Figure 1 by dashed line 606.

[034] Additionally, device 600 may also have additional features/functionality. For example, device 600 may also include additional storage (removable and/or non-removable) including, but not limited to, magnetic or optical disks or tape. Such additional storage is illustrated in Figure 11 by removable storage 608 and non-removable storage 610. Computer storage media includes volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Memory 604, removable storage 608 and non-removable storage 610 are all examples of computer storage media. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can accessed by device 600. Any such computer storage media may be part of device 600.

[035] Computing device 600 includes one or more communication connections 614 that allow computing device 600 to communicate with other computers/applications 615. Device 600 may also have input device(s) 612 such as keyboard, mouse, pen, voice input device, touch input device, etc. Output device(s) 611 such as a display, speakers, printer, etc. may also be included. These devices are well known in the art and need not be discussed at length here. In one implementation, computing device 600 includes submission and notification application 200 and/or meta-model storage application 240.

[036] Although the subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the specific features or acts described above. Rather, the specific features and acts described above are disclosed as example forms of implementing the claims. All equivalents, changes, and modifications that come within the spirit of the implementations as described herein and/or by the following claims are desired to be protected.

[037] For example, a person of ordinary skill in the computer software art will recognize that the client and/or server arrangements, user interface screen content, and/or data layouts as described in the examples discussed herein could be organized differently on one or more computers to include fewer or additional options or features than as portrayed in the examples.

What is claimed is:

1. A computer-readable medium having computer executable instructions for causing a computer to perform steps comprising:

provide a submission service that is operable to submit at least a portion of an active meta-model of an application to a central information store for use by a plurality of users; and

provide a notification service that is operable to receive a notification relating to a change in the active meta-model.

2. The computer-readable medium of claim 1, wherein the notification service is operable to receive changes that are made by a process.

3. The computer-readable medium of claim 1, wherein the notification service is operable to receive changes that are made by a user operation.

4. The computer-readable medium of claim 1, wherein the active meta-model contains intermediate code.

5. The computer-readable medium of claim 1, wherein the active meta-model contains annotations.

6. The computer-readable medium of claim 1, wherein the active meta-model contains user-generated artifacts.

7. The computer-readable medium of claim 1, wherein the notification service is further operable to display meta-model information to a particular one of the plurality of users.

8. The computer-readable medium of claim 1, wherein the notification service is operable to be integrated into a development environment such that meta-model information is displayed in the development environment.

9. A computer-readable medium having computer-executable instructions for causing a computer to perform the steps comprising:

provide a reception service that is operable to receive active meta-model information of an application being developed by a plurality of users; and

provide a storage service that is operable to store the received active meta-model information in a specific relational database structure that is operable to

allow artifacts to be added without alteration to the specific relational database structure.

10. The computer-readable medium of claim 9, further having computer-executable instructions for causing a computer to perform the step comprising:

receive a request to retrieve active meta-model information from a notification service, the notification service being operable to notify one or more client devices of changes to the active meta-model.

11. The computer-readable medium of claim 9, wherein the specific relational database structure is operable to store a name identifying the specific element and data describing the element.

12. The computer-readable medium of claim 9, wherein the specific relational database structure is operable to store a name identifying the specific connection, data describing the specific connection, the source of the specific connection, and the destination of the specific connection.

13. The computer-readable medium of claim 12, wherein the specific relational database structure is operable to store relevancy information for a specific connection.

14. A method for providing a software development application comprising the steps of:

communicating with a submission service to update an active meta-model of an application in a central information store used by a plurality of users;

receiving a notification from a notification service of changes to the active meta-model; and

using the notification service to update a display in a software development application.

15. The method of claim 14, wherein the submission service is a component of the software development application.

16. The method of claim 14, wherein the notification service provides policy violation alerts.

17. The method of claim 14, wherein the notification service provides dependency alerts.

18. The method of claim 14, wherein the notification service provides performance alerts.

19. The method of claim 14, wherein the notification service provides alerts to indicate application code currently being modified by another user.

20. A computer-readable medium having computer-executable instructions for causing a computer to perform the steps recited in claim 14.

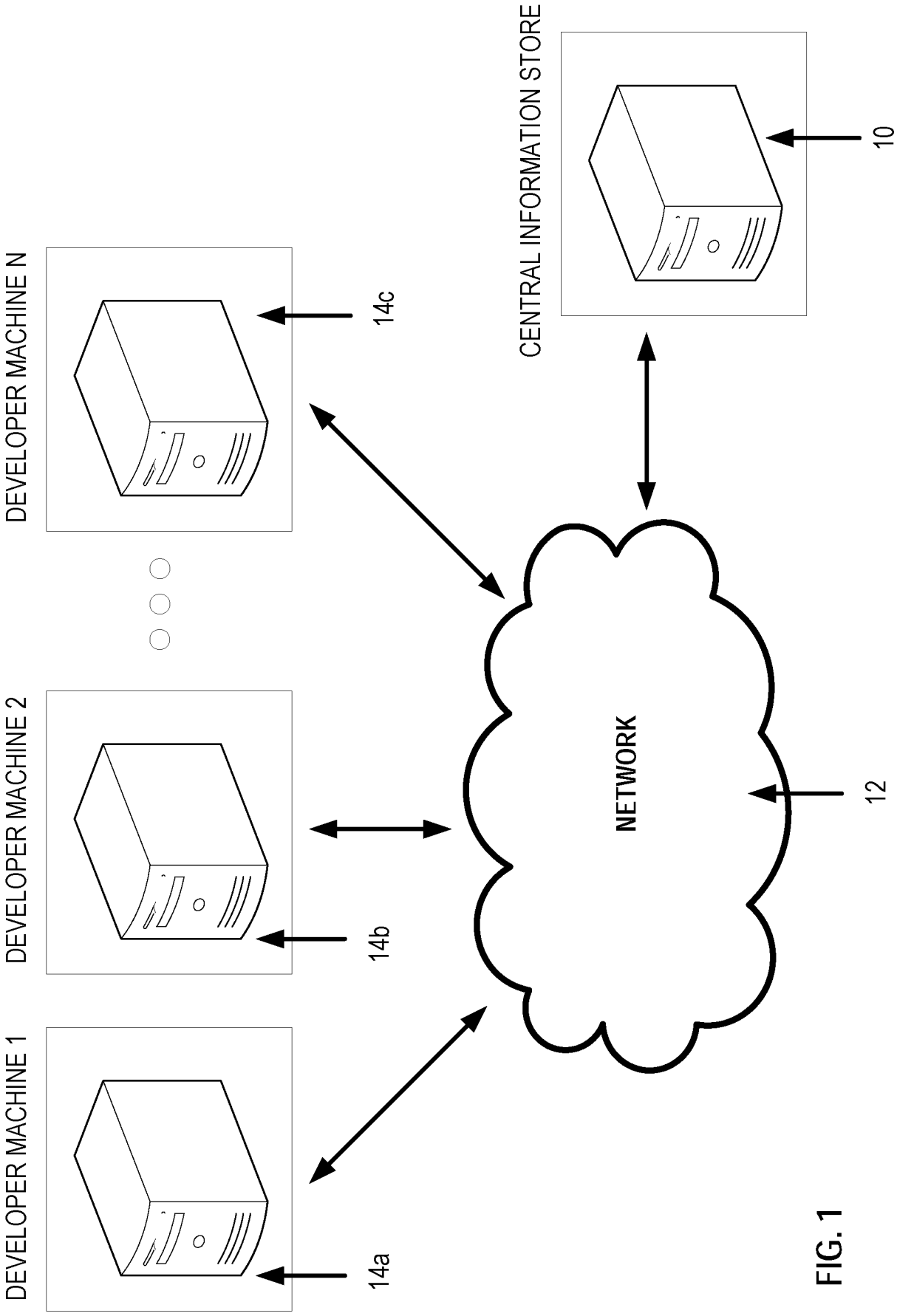
CENTRAL INFORMATION STORE
10
DEVELOPER MACHINE N
14c
DEVELOPER MACHINE 2
14b
DEVELOPER MACHINE 1
14a
NETWORK
12

FIG. 1

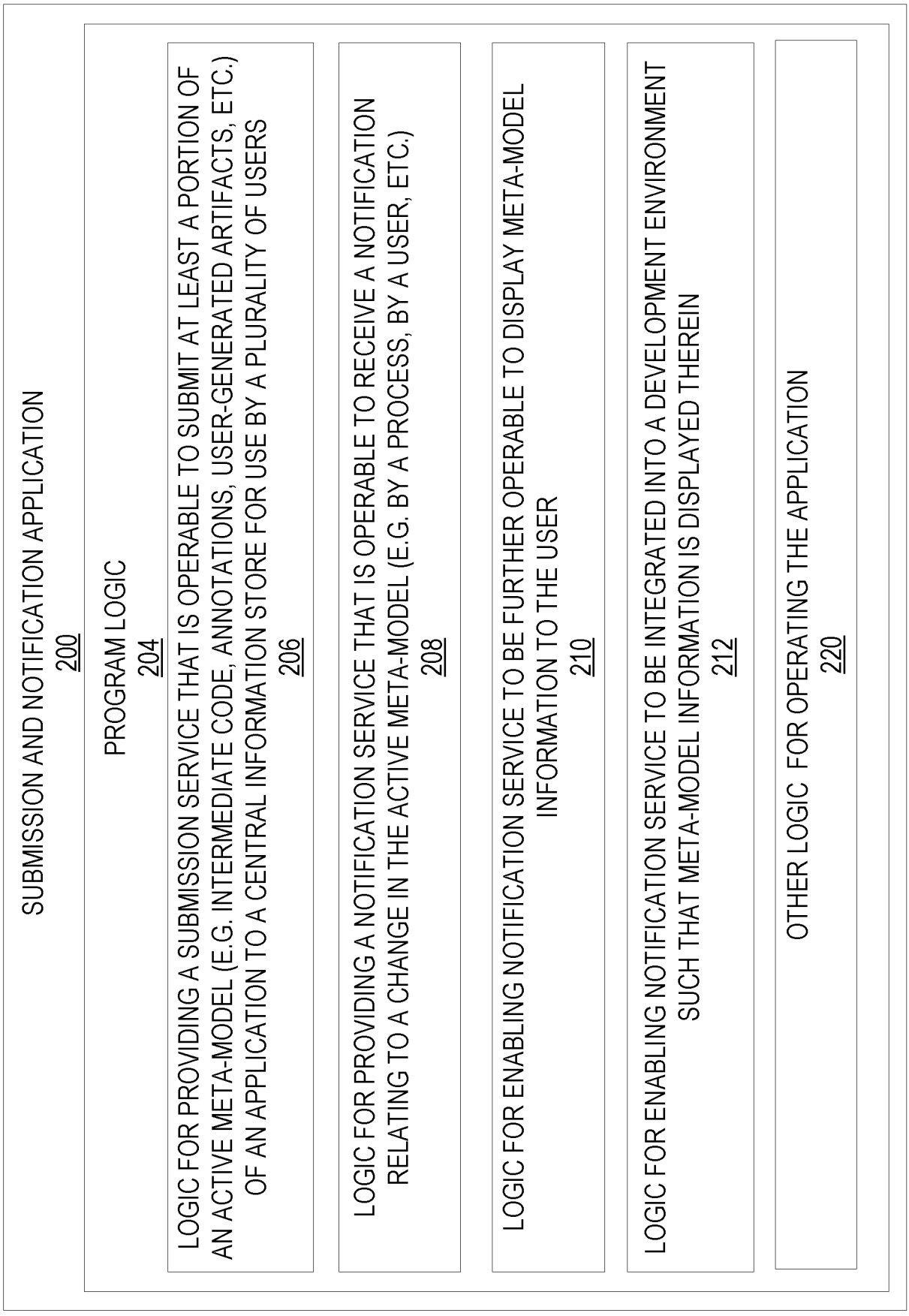
SUBMISSION AND NOTIFICATION APPLICATION
200
PROGRAM LOGIC
204
LOGIC FOR PROVIDING A SUBMISSION SERVICE THAT IS OPERABLE TO SUBMIT AT LEAST A PORTION OF AN ACTIVE META-MODEL (E.G. INTERMEDIATE CODE, ANNOTATIONS, USER-GENERATED ARTIFACTS, ETC.) OF AN APPLICATION TO A CENTRAL INFORMATION STORE FOR USE BY A PLURALITY OF USERS
206
LOGIC FOR PROVIDING A NOTIFICATION SERVICE THAT IS OPERABLE TO RECEIVE A NOTIFICATION RELATING TO A CHANGE IN THE ACTIVE META-MODEL (E.G. BY A PROCESS, BY A USER, ETC.)
208
LOGIC FOR ENABLING NOTIFICATION SERVICE TO BE FURTHER OPERABLE TO DISPLAY META-MODEL INFORMATION TO THE USER
210
LOGIC FOR ENABLING NOTIFICATION SERVICE TO BE INTEGRATED INTO A DEVELOPMENT ENVIRONMENT SUCH THAT META-MODEL INFORMATION IS DISPLAYED THEREIN
212
OTHER LOGIC FOR OPERATING THE APPLICATION
220

FIG. 2

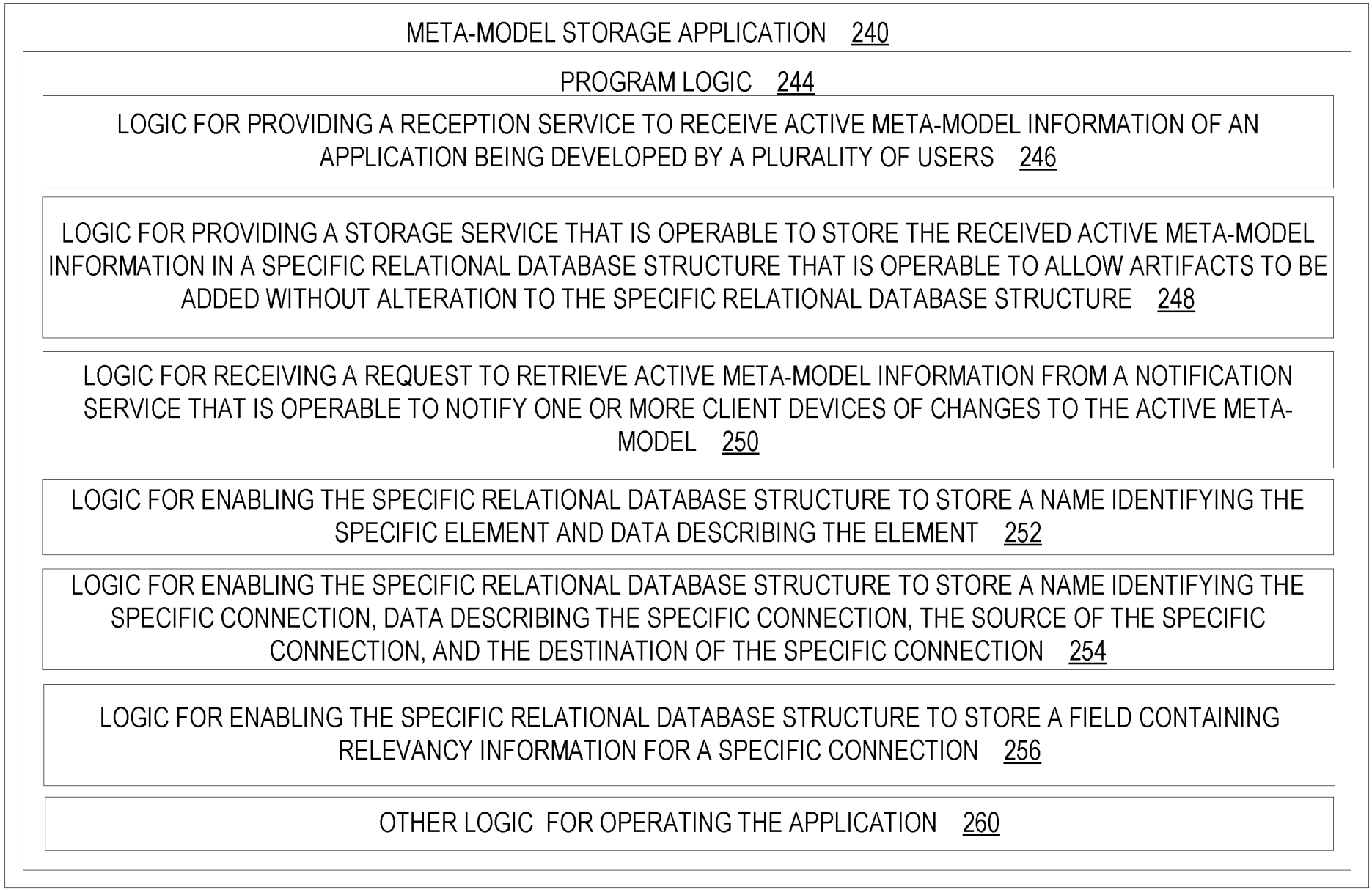
META-MODEL STORAGE APPLICATION 240
PROGRAM LOGIC 244
LOGIC FOR PROVIDING A RECEPTION SERVICE TO RECEIVE ACTIVE META-MODEL INFORMATION OF AN APPLICATION BEING DEVELOPED BY A PLURALITY OF USERS 246
LOGIC FOR PROVIDING A STORAGE SERVICE THAT IS OPERABLE TO STORE THE RECEIVED ACTIVE META-MODEL INFORMATION IN A SPECIFIC RELATIONAL DATABASE STRUCTURE THAT IS OPERABLE TO ALLOW ARTIFACTS TO BE ADDED WITHOUT ALTERATION TO THE SPECIFIC RELATIONAL DATABASE STRUCTURE 248
LOGIC FOR RECEIVING A REQUEST TO RETRIEVE ACTIVE META-MODEL INFORMATION FROM A NOTIFICATION SERVICE THAT IS OPERABLE TO NOTIFY ONE OR MORE CLIENT DEVICES OF CHANGES TO THE ACTIVE META-MODEL 250
LOGIC FOR ENABLING THE SPECIFIC RELATIONAL DATABASE STRUCTURE TO STORE A NAME IDENTIFYING THE SPECIFIC ELEMENT AND DATA DESCRIBING THE ELEMENT 252
LOGIC FOR ENABLING THE SPECIFIC RELATIONAL DATABASE STRUCTURE TO STORE A NAME IDENTIFYING THE SPECIFIC CONNECTION, DATA DESCRIBING THE SPECIFIC CONNECTION, THE SOURCE OF THE SPECIFIC CONNECTION, AND THE DESTINATION OF THE SPECIFIC CONNECTION 254
LOGIC FOR ENABLING THE SPECIFIC RELATIONAL DATABASE STRUCTURE TO STORE A FIELD CONTAINING RELEVANCY INFORMATION FOR A SPECIFIC CONNECTION 256
OTHER LOGIC FOR OPERATING THE APPLICATION 260

FIG. 3

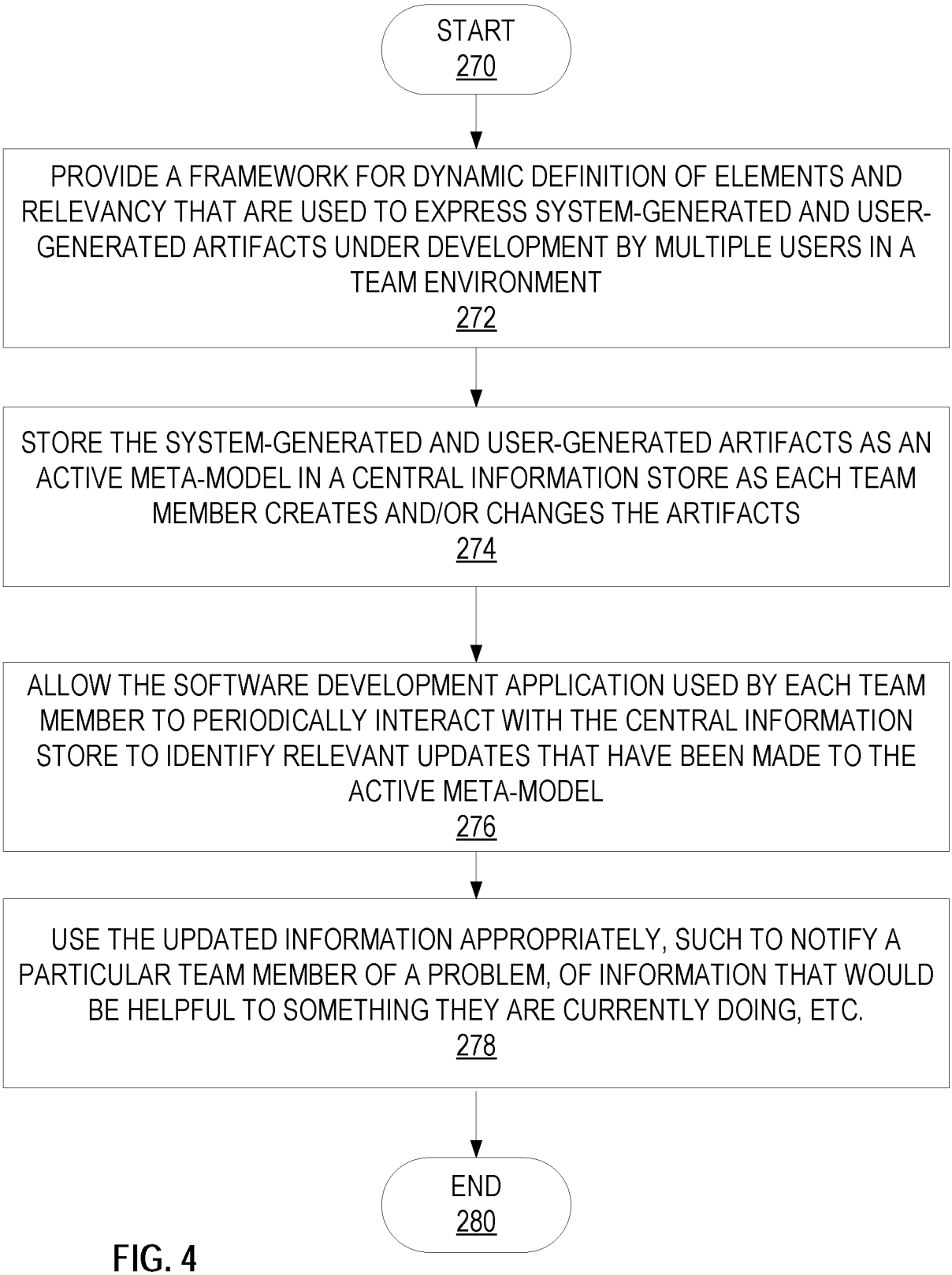
START
270
PROVIDE A FRAMEWORK FOR DYNAMIC DEFINITION OF ELEMENTS AND RELEVANCY THAT ARE USED TO EXPRESS SYSTEM-GENERATED AND USER-GENERATED ARTIFACTS UNDER DEVELOPMENT BY MULTIPLE USERS IN A TEAM ENVIRONMENT
272
STORE THE SYSTEM-GENERATED AND USER-GENERATED ARTIFACTS AS AN ACTIVE META-MODEL IN A CENTRAL INFORMATION STORE AS EACH TEAM MEMBER CREATES AND/OR CHANGES THE ARTIFACTS
274
ALLOW THE SOFTWARE DEVELOPMENT APPLICATION USED BY EACH TEAM MEMBER TO PERIODICALLY INTERACT WITH THE CENTRAL INFORMATION STORE TO IDENTIFY RELEVANT UPDATES THAT HAVE BEEN MADE TO THE ACTIVE META-MODEL
276
USE THE UPDATED INFORMATION APPROPRIATELY, SUCH TO NOTIFY A PARTICULAR TEAM MEMBER OF A PROBLEM, OF INFORMATION THAT WOULD BE HELPFUL TO SOMETHING THEY ARE CURRENTLY DOING, ETC.
278
END
280

FIG. 4

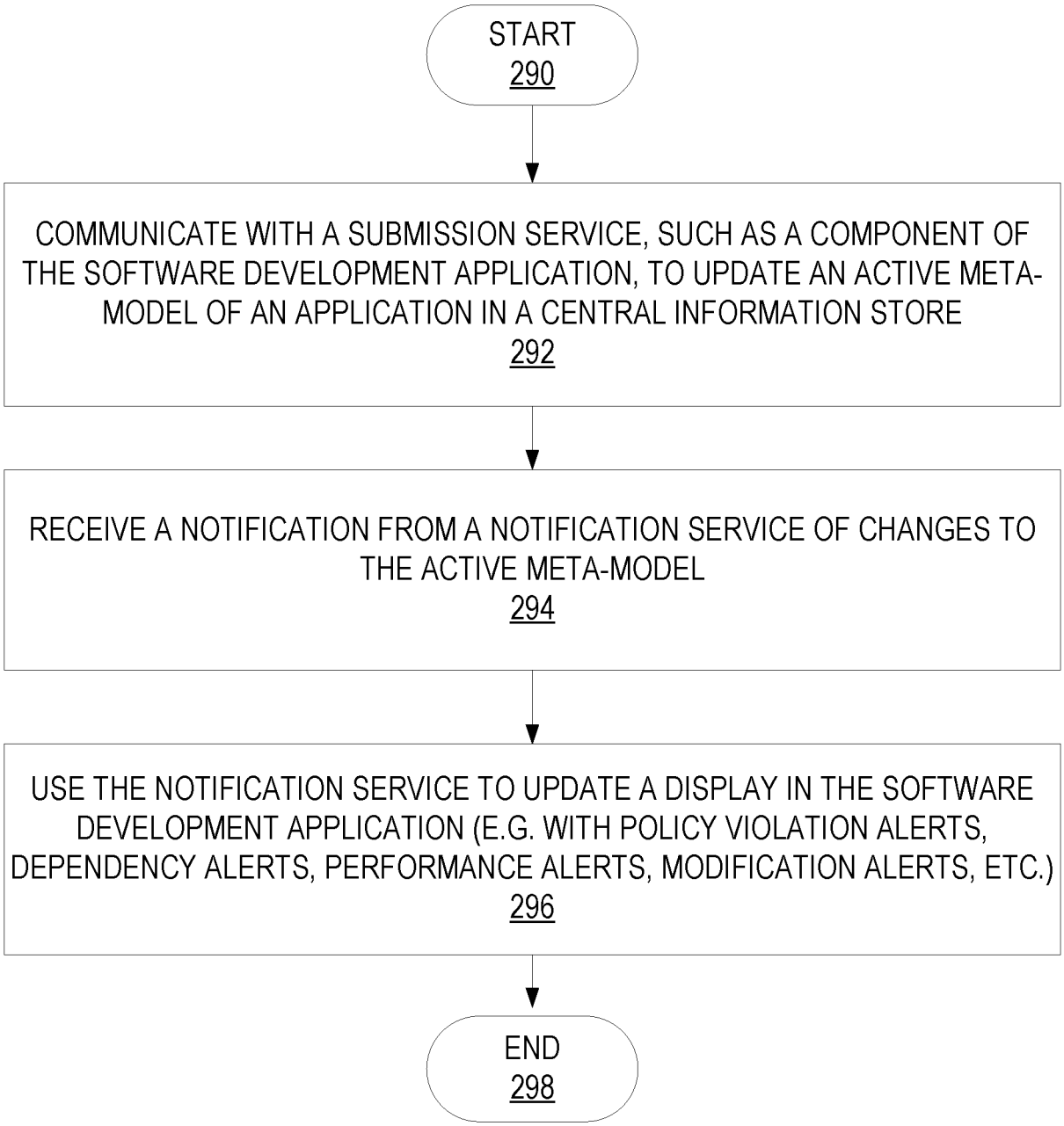
START
290
COMMUNICATE WITH A SUBMISSION SERVICE, SUCH AS A COMPONENT OF THE SOFTWARE DEVELOPMENT APPLICATION, TO UPDATE AN ACTIVE META-MODEL OF AN APPLICATION IN A CENTRAL INFORMATION STORE
292
RECEIVE A NOTIFICATION FROM A NOTIFICATION SERVICE OF CHANGES TO THE ACTIVE META-MODEL
294
USE THE NOTIFICATION SERVICE TO UPDATE A DISPLAY IN THE SOFTWARE DEVELOPMENT APPLICATION (E.G. WITH POLICY VIOLATION ALERTS, DEPENDENCY ALERTS, PERFORMANCE ALERTS, MODIFICATION ALERTS, ETC.)
296
END
298

FIG. 5

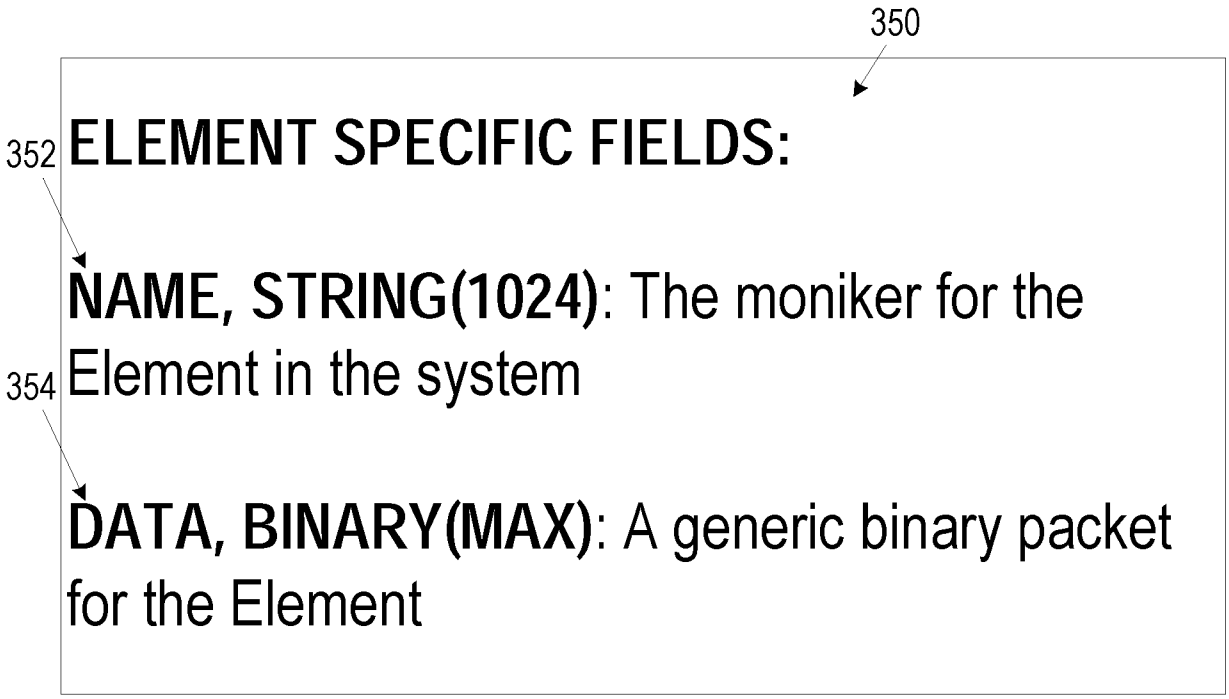
350
352
ELEMENT SPECIFIC FIELDS:
NAME, STRING(1024): The moniker for the Element in the system
354
DATA, BINARY(MAX): A generic binary packet for the Element

FIG. 6

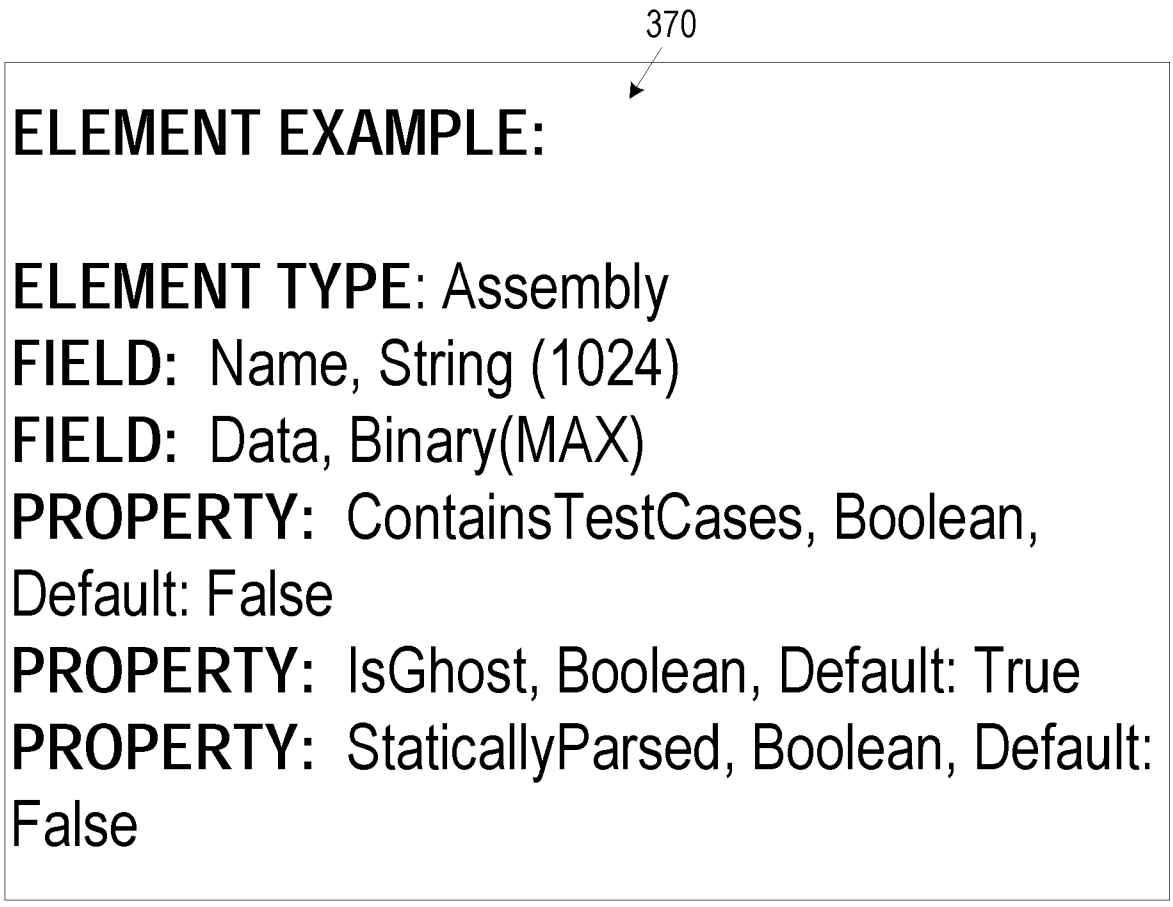
370
ELEMENT EXAMPLE:
ELEMENT TYPE: Assembly
FIELD: Name, String (1024)
FIELD: Data, Binary(MAX)
PROPERTY: ContainsTestCases, Boolean, Default: False
PROPERTY: IsGhost, Boolean, Default: True
PROPERTY: StaticallyParsed, Boolean, Default: False

FIG. 7

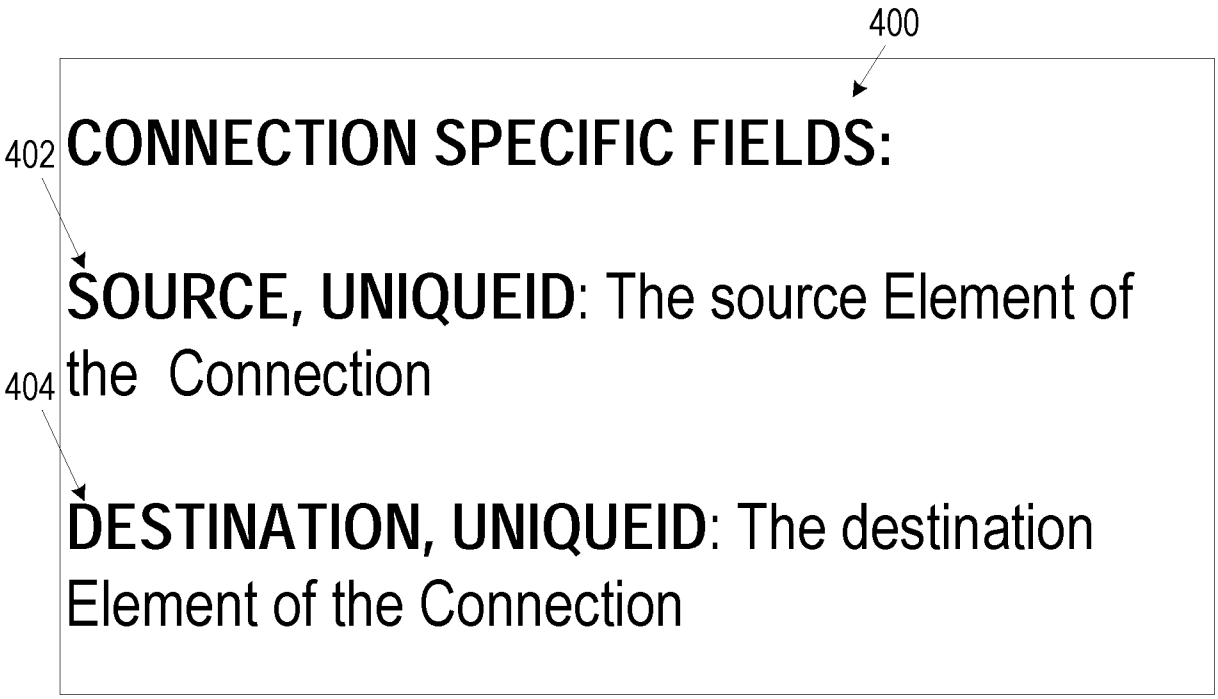
400
402
CONNECTION SPECIFIC FIELDS:
SOURCE, UNIQUEID: The source Element of the Connection
404
DESTINATION, UNIQUEID: The destination Element of the Connection

FIG. 8

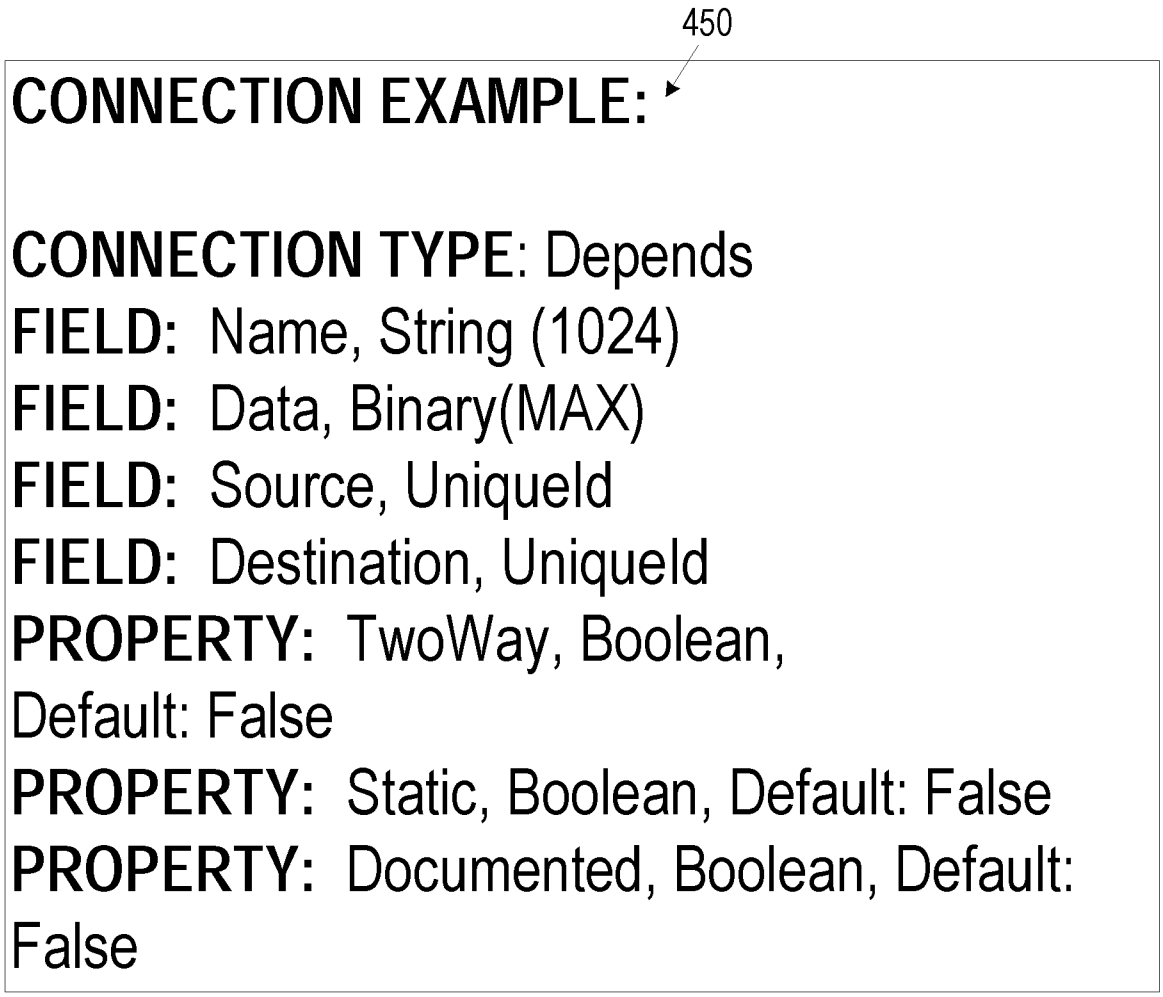
450
CONNECTION EXAMPLE:
CONNECTION TYPE: Depends
FIELD: Name, String (1024)
FIELD: Data, Binary(MAX)
FIELD: Source, UniqueId
FIELD: Destination, UniqueId
PROPERTY: TwoWay, Boolean, Default: False
PROPERTY: Static, Boolean, Default: False
PROPERTY: Documented, Boolean, Default: False

FIG. 9

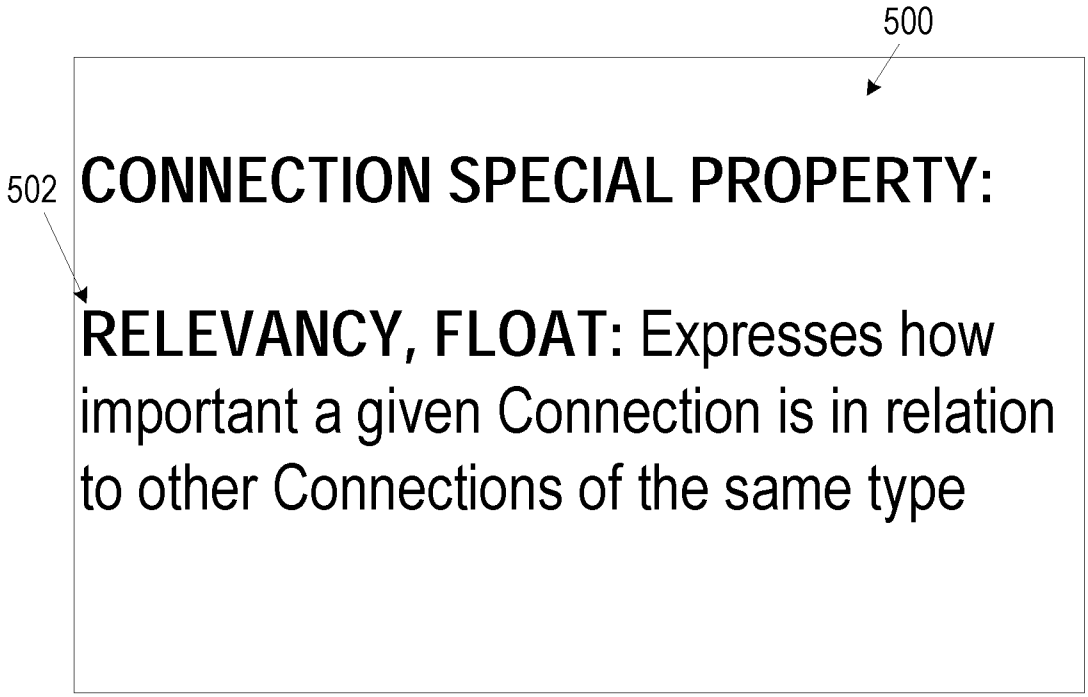
500
502
CONNECTION SPECIAL PROPERTY:
RELEVANCY, FLOAT: Expresses how important a given Connection is in relation to other Connections of the same type

FIG. 10

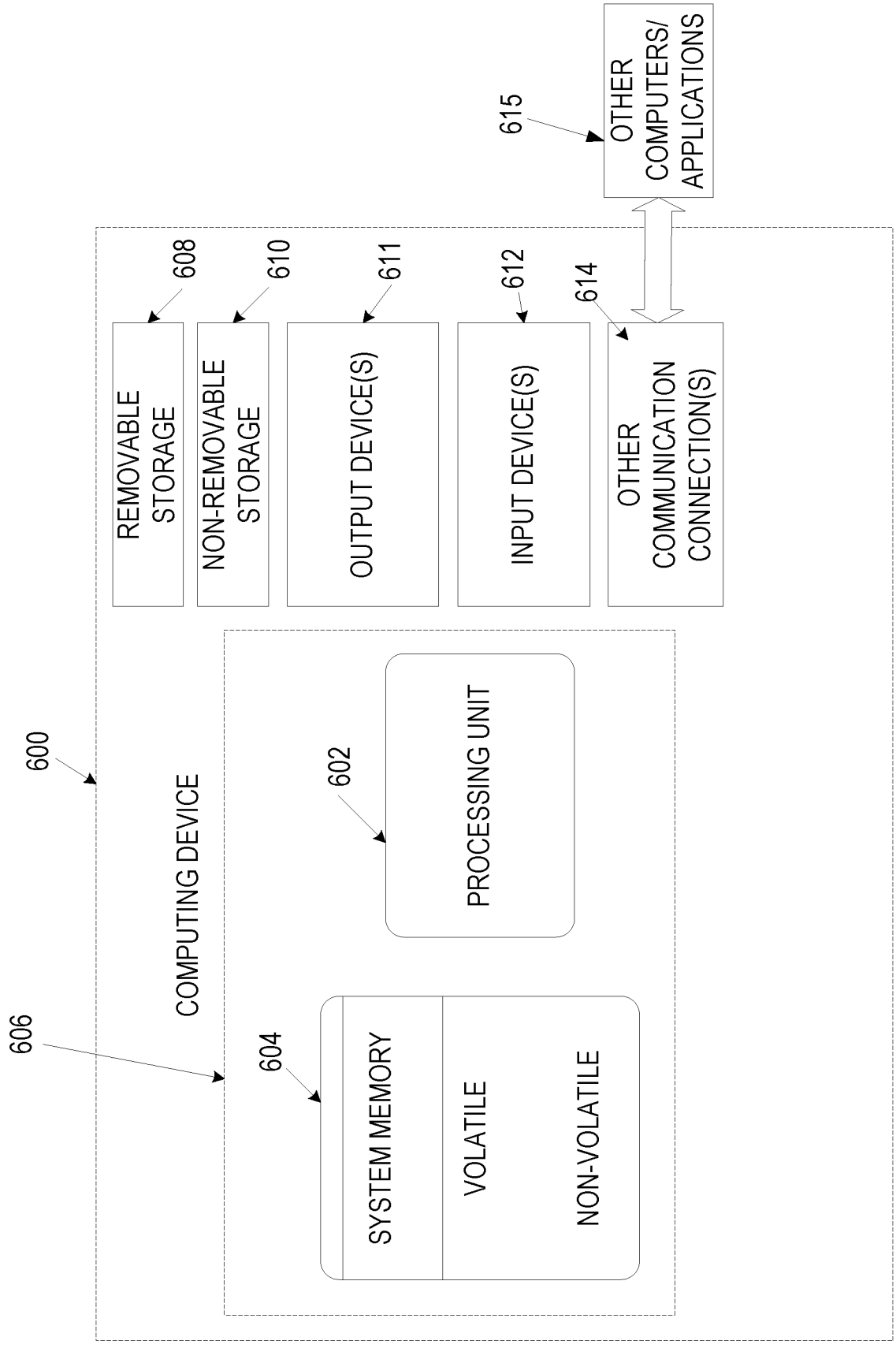
600
COMPUTING DEVICE
606
602
PROCESSING UNIT
604
SYSTEM MEMORY
VOLATILE
NON-VOLATILE
608
REMOVABLE STORAGE
610
NON-REMOVABLE STORAGE
611
OUTPUT DEVICE(S)
612
INPUT DEVICE(S)
614
OTHER COMMUNICATION CONNECTION(S)
615
OTHER COMPUTERS/ APPLICATIONS

FIG. 11

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2008/054319

A. CLASSIFICATION OF SUBJECT MATTER

G06F 15/16(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 8 G06F

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

Korean Utility models and applications for Utility models since 1975

Japanese Utility models and applications for Utility models since 1975

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

eKIPASS (KIPO internal) & Keywords: "software, development, collaborative, sever, information"

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	JP 2000-235496 A (NTT COMMUNICATIONWARE CO.) 29 Aug. 2000 See the abstract; figures 1, 4; paragraphs [0012] - [0024].	1-20
Y	JP 2001-092650 A (HITACHI INFORMATION SYSTEMS LTD.) 06 Apr. 2001 See the abstract; figures 1-3; paragraphs [0013] - [0024].	1-20
A	US 05911073 A (MATTSON, JR. et al.) 08 Jun. 1999 See the abstract; figures 4A-4C; column 9 line 37 - column 10 line 29.	1-20
A	US 05848274 A (HAMBY et al.) 08 Dec. 1998 See the abstract; figures 2A, 2B; column 15 lines 18-42.	1-20

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents:

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

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

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of 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
24 JULY 2008 (24.07.2008)	**24 JULY 2008 (24.07.2008)**

Name and mailing address of the ISA/KR

Korean Intellectual Property Office
Government Complex-Daejeon, 139 Seonsa-ro, Seo-gu, Daejeon 302-701, Republic of Korea

Facsimile No. 82-42-472-7140

Authorized officer

BAE, Kyung Hwan

Telephone No. 82-42-481-5768

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2008/054319

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
JP 2000-235496 A	29.08.2000	NONE	
JP 2001-092650 A	06.04.2001	NONE	
US 05911073 A	08.06.1999	DE 69823153 T2	14.10.2004
		EP 0926592 A2	30.06.1999
		EP 0926592 A3	16.08.2000
		EP 0926592 B1	14.04.2004
		JP 11-232138 A	27.08.1999
US 05848274 A	08.12.1998	AU 1997-19568 A1	16.09.1997
		CA 2248181 A	04.09.1997
		EP 0883844 A1	16.12.1998
		JP 2000-507373 A	13.06.2000
		WO 9732250 A	04.09.1997