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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
25 June 2015 (25.06.2015)

WIPO | PCT

(10) International Publication Number
WO 2015/092519 A1

(51) International Patent Classification:
G06F 9/44 (2006.01) *G06F 17/30* (2006.01)
G06Q 10/00 (2012.01)

(21) International Application Number:
PCT/IB2014/002787

(22) International Filing Date:
16 December 2014 (16.12.2014)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
3944/MUM/2013 16 December 2013 (16.12.2013) IN

(71) Applicant: **KRONOSIS HYPER TECHNOLOGIES PRIVATE LIMITED** [IN/IN]; 207 Unique Industrial Estate, Off Veer Savarkar Road, Near Sidhivinayak Temple, Prabhadevi, 400025 Mumbai (IN).

(72) Inventors: **VAIDYA, Anuraag Ravi**; 251/102 Ambika Ravi, Prof U U Bhat Marg, Matunga (e), 400019 Mumbai (IN). **SHAH, Abhi Jayant**; 344 Dharamshi Bhuvan, 4th floor, Chandavarkar Cross Road, Matunga (e), 400019 Mumbai (IN).

(74) Agents: **RAE, Konpal** et al.; Lakshmikumaran & Sridharan, B-6/10 Safdarjung Enclave, New Delhi 110 029 (IN).

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

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

Declarations under Rule 4.17:

— *of inventorship (Rule 4.17(iv))*

Published:

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

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

(54) Title: SYSTEMS AND METHODS FOR DEVELOPING APPLICATION PROGRAMS

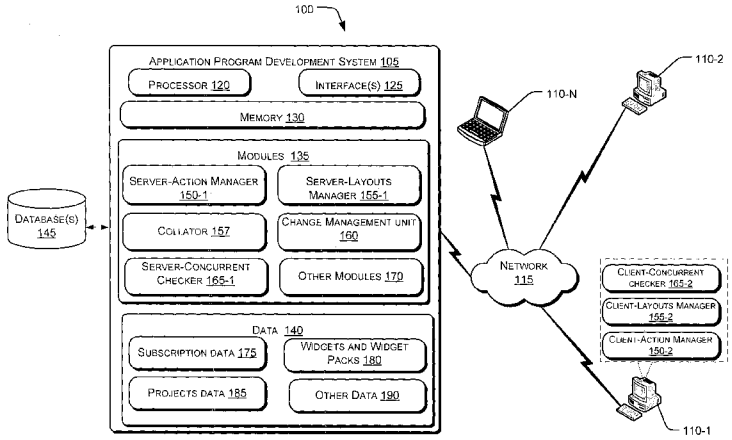

Fig. 1

(57) Abstract: The present subject matter relates to systems (105) and methods for developing application programs. In an example, an application development system (105) may include a widgets and widget packs unit (216) and a collator (157). The widgets and widget packs unit (216) may receive user inputs for configuring one or more widgets to develop an application program. A widget (330) may include a pre-developed code snippet to generate a part of an application source code of the application program. Further, the collator (157) may initiate execution of the one or more widgets to provide a corresponding source code, based on user inputs and the pre-developed code snippet. The source code generated by the widgets may be collated, before the application program is deployed. Furthermore, the source code corresponding to the one or more widgets may be appended in a corresponding section of the application program.

WO 2015/092519 A1

SYSTEMS AND METHODS FOR DEVELOPING APPLICATION PROGRAMS

TECHNICAL FIELD

[0001] The present subject matter relates to application programs and, more particularly but not exclusively, to systems and methods for developing application programs.

BACKGROUND

[0002] In recent times there has been a rapid increase in the use of computing devices, which in turn has encouraged the use of application programs configured to run on the computing devices. An application program may be understood as a computer program that performs an intended task. Examples of the application program include banking applications, database applications, such as employee record maintenance applications, and shopping portals.

[0003] Generally, implementation of an application program involves multiple stages, such as planning, development, and deployment and maintenance. The planning stage includes requirement analysis to gather information pertaining to the end result, i.e., the intended task. Subsequently, a team of developers works on the requirements captured in the planning stage to develop the application program. The development of an application program may include multiple steps, for example, designing of user interface, structuring of associated databases, development of underlying logic of the application program, and compilation of code corresponding to the application program. Typically, all the steps in the development stage are performed manually by a developer.

BRIEF DESCRIPTION OF THE FIGURES

[0004] The detailed description is described with reference to the accompanying figures. In the figures, the left-most digit(s) of a reference number identifies the figure in which the reference number first appears. The same numbers are used throughout the figures to reference like features and components. Some embodiments of system or methods in accordance with embodiments of the present subject matter are now described, by way of example, and with reference to the accompanying figures, in which:

[0005] Fig. 1 illustrates a network environment for developing an application program, in accordance with an embodiment of the present subject matter;

[0006] Fig. 2 illustrates various components of an application program development system, in accordance with an embodiment of the present subject matter.

[0007] Fig. 3a-3d illustrates structure of various components of an application program and the application program development system, in accordance with an embodiment of the present subject matter;

[0008] Fig. 4 illustrates a structure of a widget unit of the application program development system, in accordance with an embodiment of the present subject matter;

[0009] Fig. 5 illustrates components of a concurrent checker of the application program development system, according to an embodiment of the present subject matter;

[0010] Fig. 6 illustrates components of a collator of the application program development system, according to an embodiment of the present subject matter;

[0011] Fig. 7 illustrates a method for developing an application program, according to an embodiment of the present subject matter; and

[0012] Fig. 8 illustrates a method for creating the application program, according to an embodiment of the present subject matter.

[0013] It should be appreciated by those skilled in the art that any block diagrams herein represent conceptual views of illustrative systems embodying the principles of the present subject matter. Similarly, it will be appreciated that any flow charts, flow diagrams, state transition diagrams, pseudo codes, and the like, represent various processes which may be substantially represented in computer readable medium and so executed by a computer or processor, whether or not such computer or processor is explicitly shown.

DETAILED DESCRIPTION

[0014] Systems and methods for development of application programs are described herein. An application program may be understood as a set of instructions, which when executed perform an intended task. The application programs may be developed for a variety of purposes, for example, banking purposes, such Internet banking, running an automated teller machines (ATM) machines, and updating user account; gaming and entertainment, corporate finance management, enterprise resource planning, supply chain management, learning management, Customer relationship management, customer portals, big data (big

data solutions) management, and for shopping portals, business intelligence, and analytics applications.

[0015] Generally, development of an application program requires substantial manual handling, which not only makes the development process a time consuming task but also makes the application program prone to human errors. Such errors may expose the application program to various risks, such as security vulnerabilities, performance impacts, and high maintenance needs. For example, a minor glitch in the application program may result in inefficient utilization of computing resources, such as processor and memory, thereby impacting the performance the application program and also the associated computing device. For instance, an anomaly in the application program may overload a processor with tasks, which in turn may impact performance of processor. Further, in case of a complex application program, the development may take up to several months, which in turn may prove to be cost intensive besides being time consuming.

[0016] According to an implementation of the present subject matter, systems and methods for development of application programs are described herein. In an implementation, the application program may be considered to include multiple sections. A section may include a program corresponding to a given functionality, which may be realized through one or more widgets. For the purpose of explanation, a program structure may be considered to be analogous to a folder, which may hold widgets and may keep a record of inter-connections between widgets. Further, a widget corresponds to a pre-developed code snippet that generates a source code, upon deployment, the source code being task specific. On execution of the source code a task, such as manipulation, representation, storage, retrieval, and obfuscation of information, is performed. The pre-developed code snippet may be configurable, for example, the pre-developed code snippets may use user inputs and inputs received from other widgets to generate the source code. Additionally, the widgets may also be provided by way of a widget pack, which is a collection of a plurality of widgets that perform a task, such as a login task. Further, the generated source code may vary based on application requirement and user inputs.

[0017] Further, in addition to widgets, the development of the application program may involve development of, various other components, such as databases, and layout of the sections and the databases, may also be defined. For developing the application program, a

user, such as a developer, may provide various inputs to perform an action. Each action may have an action ID associated with it and based on the action ID a development unit corresponding to the action may be selected. As mentioned earlier, development of an application program involves development of multiple features or components, and one or more of such components may be developed using a development unit. Examples of the development unit includes, but are not limited to, a project setting units, a structures unit, a databases unit, a widgets and widget packs unit, a sections unit, a fields unit, and a layouts unit. In an example, the multiple development units may be provided in an action manager, which may be configured to analyze a received action and provide the received action to one of the development units, based on the action ID. For example, in case the action pertains to selection of a widget, the action may be provided to the widgets and the widget packs unit for further analysis. In another example, in case the action pertains to creation of a database, the action may be provided to the database unit for further analysis.

[0018] On receiving the action, the corresponding development unit may check for validity of the action, based on validation rules. The validation rules may include checks for, for example, inclusion of special characters and syntax related errors. The validation rules may check for issues, such as hacking and basic errors. In an implementation, a dual validation is performed, once at user end, i.e., client side validation, and if validated at the client end, the action is again validated at server end. Further, the action may be performed, if the action is validated at both the client and the server side.

[0019] In an example, while creating an application program, the user may initially create one or more database(s) corresponding to the application program using the database unit. The database may be based on relational database management systems (RDMS) or non- relational database management systems. Further, the database may be, for example, MySQL®, SQL®, NoSQL, Oracle®, or Hadoop®. The databases may be working one same or different RDMS. For example, a first database may be in MySQL®, a second database may be in SQL®, and a third database may be in NoSQL®. Further, a structure of the databases may be created or modified using the structures unit; and various fields that may be included in tables of the databases may be created or modified using the fields unit. Additionally, the user may create or modify the sections of the application program using the sections unit. Upon creating sections, the user may select one or more widgets from a plurality of widgets, based on application requirements. For example, in case records are to be added, deleted, or

modified in a database, a corresponding widget may be added to that section. In another example, the widgets may be used for handle emailing, scheduling tasks, injecting custom native code, creating forms, importing/exporting documents, manipulation files, and managing servers. Thus, instead of having to type-in code for such actions, the user may select the desired widget and corresponding code may be added to the application program, upon deployment. Similarly, instead of having to create the database from the scratch, a skeleton may be provided by way of the databases unit and the user may create the databases with minimal inputs. Additionally, the widgets may have standalone architecture and may work without a compiler, i.e., the underlying code of the widgets may be free from errors.

[0020] Further, to reduce the number of errors and subsequent issues, the actions may be checked in parallel. In other words, even before the application is collated, the actions may be checked for errors and conflicts concurrently, to ensure that there are minimum changes at the final stage. For example, once the program for a section is complete, the section may be checked for probable errors and conflicts, based on error and caution rules. The error and caution rules provide rule for comprehensively checking the conflicts among various widgets, errors pertaining to the databases, layouts, structures and the sections. Thus, certain errors or cautions that may have not been picked by the validation rules are identified during concurrent checking.

[0021] Upon developing, which may include creating and/or modification of the application program, the application program may be collated. The collation may be performed before the application program is deployed on a target device. Further, the collation may include generation of a source code corresponding to the one or more selected widgets and appending of the source code in corresponding section of the application program. The source code may be generated based on the target programming language selected in the project options. If the selected programming language is C#.NET, then the source code can be generated for C# .NET programming platform. Accordingly, the source code may be generated in the selected programming language, and the generated code can be executed in accordance with the target programming language's specification.

[0022] Thus, the present subject matter provides for development of application programs with minimal human intervention, thereby reducing the chances of human errors and reducing the time, both human and computational, required for application development

process. Additionally, the use of multiple checkers and validators provide for reduction in errors. Further, the reduction in such errors may provide better performance of the application program and in turn the computing device hosting the application program. In addition to reduction in errors, the present subject matter reduces the time required to develop an application program with the use of widgets. As mentioned earlier, the widgets are pre-developed collections of source code that can be attached to the application program being developed. Thus, the widgets form a part of application source code. Further, the widgets are customizable and induce a smooth functioning of the desired tasks and functionalities in the application program. Thus, a developer may not have to write a code for a given task and may append the required code using widgets, which may be modified based on user inputs.

[0023] It should be noted that the description and figures merely illustrate the principles of the present subject matter. It will thus be appreciated that those skilled in the art will be able to devise various arrangements that, although not explicitly described or shown herein, embody the principles of the present subject matter and are included within its spirit and scope.

[0024] It will also be appreciated by those skilled in the art that the words during, while, and when as used herein are not exact terms that mean an action takes place instantly upon an initiating action but that there may be some small but reasonable delay, such as a propagation delay, between the initial action and the reaction that is initiated by the initial action. Additionally, the words "connected" and "coupled" are used throughout for clarity of the description and can include either a direct connection or an indirect connection.

[0025] The manner in which the systems and the methods of development of an application program may be implemented has been explained in details with respect to the Figures 1 to 8. While aspects of described systems and methods for development of an application program can be implemented in any number of different computing systems and transmission environments, the embodiments are described in the context of the following system(s).

[0026] Fig. 1 illustrates a network environment 100 implementing an application program development system 105, according to an embodiment of the present subject matter. The application program development (APD) system 105 may assist a plurality of user, such as developer, in developing an application program. The users may access the APD system

105 through their respective user devices 110, such as a user device 110-1, a user device 110-2,.. and a user device 110-n. The user devices 110 may access the APD system 105 through a network 115.

[0027] The network 115 may be a wireless or a wired network, or a combination thereof. The network 115 can be implemented as one of the different types of networks, such as intranet, local area network (LAN), wide area network (WAN), the internet, and such. The network 115 may either be a dedicated network or a shared network, which represents an association of the different types of networks that use a variety of protocols, for example, Hypertext Transfer Protocol (HTTP), HTTP Secure (HTTPS), Transmission Control Protocol/Internet Protocol (TCP/IP), Virtual network, and cloud networks, etc., to communicate with each other.

[0028] Further, the user devices 110 and the APD system 105 may be implemented as a computing device, such as desktop computers, hand-held devices, laptops or other portable computers, tablet computers, and the like. The APD system 105 may include a processor 120, interfaces 125, memory 130, modules 135, and data 140.

[0029] The processor 120 may be implemented as one or more microprocessors, microcomputers, microcontrollers, digital signal processors, central processing units, logic circuitries, and/or any devices that manipulate signals based on operational instructions. Among other capabilities, the processor(s) is configured to fetch and execute computer-readable instructions stored in the memory.

[0030] The functions of the various elements shown in the figure, including any functional blocks labeled as "processor(s)", may be provided through the use of dedicated hardware as well as hardware capable of executing software in association with appropriate software. When provided by a processor, the functions may be provided by a single dedicated processor, by a single shared processor, or by a plurality of individual processors, some of which may be shared. Moreover, explicit use of the term "processor" should not be construed to refer exclusively to hardware capable of executing software, and may implicitly include, without limitation, digital signal processor (DSP) hardware, network processor, application specific integrated circuit (ASIC), field programmable gate array (FPGA), read only memory (ROM) for storing software, random access memory (RAM), non-volatile storage. Other hardware, conventional and/or custom, may also be included.

[0031] The interface(s) may include a variety of software and hardware interfaces that allow the APD system 105 to interact with user devices 110, and other computing devices, such as web servers and external repositories, such as database 145.

[0032] The memory 130 may be coupled to the processor 120 and may include any computer-readable medium known in the art including, for example, volatile memory (e.g., RAM), and/or non-volatile memory (e.g., EPROM, flash memory, etc.)

[0033] The modules 135 include routines, programs, objects, components, data structures, and the like, which perform particular tasks or implement particular abstract data types. The modules 135 further include modules that supplement applications on APD system, for example, modules of an operating system.

[0034] Further, the modules 135 can be implemented in hardware, instructions executed by a processing unit, or by a combination thereof. The processing unit can comprise a computer, a processor, such as the processor 120, a state machine, a logic array or any other suitable devices capable of processing instructions. The processing unit can be a general-purpose processor which executes instructions to cause the general-purpose processor to perform the tasks or, the processing unit can be dedicated to perform the functions.

[0035] In another aspect of the present subject matter, the modules 135 may be machine-readable instructions (software) which, when executed by a processor/processing unit, perform any of the described functionalities. The machine-readable instructions may be stored on an electronic memory device, hard disk, optical disk or other machine-readable storage medium or non-transitory medium. In one implementation, the machine-readable instructions can be also be downloaded to the storage medium via a network connection. The data serves, amongst other things, as a repository for storing data that may be fetched, processed, received, or generated by one or more of the modules.

[0036] The module(s) 135 includes, for example, a server- action manager 150-1, a server-layout manager 155-1, a collator 157, a change management unit 160, a server-concurrent checker 165-1, and other module(s) 170. The other module(s) include programs that supplement applications or functions performed by an infrastructure support optimization system, such as the APD system 105. The data 140 serves, amongst other things, as a repository for storing data obtained and processed by one or more module(s) 135. The data

140 includes, for example, subscription data 175, widgets and widget packs 180, projects data 185, and other data 190. The other data 190 includes data generated as a result of the execution of one or more modules in the other module(s) 170.

[0037] In an implementation, a user may corresponding user device, say user device 110, may access the APD system 105 to develop an application program. It will be understood that the development of an application program includes both creation and modification of the application program. Thus, the user may access the APD system 105 to create a new application program or may modify an already created application program, which may be stored in the projects data 185. Further, the application program may include codes pertaining to various sections, each section including codes to achieve desired functionalities, codes for desired layouts, and codes for associating the application program to one or more databases, such as the database 145. To develop the application program, the user may perform an action, which may be provided to the action manager 150. In an example, the action manager 150 may include a server side component 150-1 and a client side component 150-2. For the purpose of explanation, the server side component may be referred to as server-action manager 150-1 and the client side component may be referred the client-action manager 150-2. The client-action manager 150-2 may be provided on the user device 110. Further, the server-action manager 150-1 and the client-action manager 150-2 may be collectively referred to as the action manger 150.

[0038] Further, each component of the action manager 150 may include a plurality of development units as will be explained in detail with reference to description to Fig. 2. Each development unit may assist in developing an attribute, such as database, logic, and layout of the application program. In said example, the received action may be first analyzed at the client-action manager 150-2, and based on the analysis, may be provided to a development unit for validation. For example, in case the action relates to a widget, the action may be provided to a widgets and widget packs unit (shown in Fig. 2). The user may then provide the inputs to configure one or more widgets based on application program requirements. Further, pre-developed code snippets corresponding to the widgets may be stored in the widget and widget packs 180. Similarly, in case the action relates to configuration of layout of an application program, the action may be provided to a layouts unit (shown in Fig. 2). The layouts unit may be interfaced with a client- layouts manager 155-2 to develop the layout of the application program.

[0039] Further, it will be appreciated that similar to the action manager 150, the layouts manager 155 may include the client-layouts manager 155-2 and a server layouts manager 155-1. The client layouts manager 155-2 may assist in developing the layouts at the client side and the server layouts manager 155-1 may assist in developing the layouts at the server side. The APD system 105 provides a user-friendly interface, where users can create, for example, "blocks" or "shapes" on the screen using client layouts manager 155-2. The server layout manager 155-1 may validate whether right data have been passed through the client layouts manager 155-2, and if appropriate, it may be determined whether parameters that define the created layout have been included or not. If validated, the data pertaining to the layouts may be stored in projects data 185, which may include a unit for layouts.

[0040] Thus, to summarize, the action may be provided to corresponding development unit for assisting the user in developing the application program. Further, if a received action is validated at the client-action manager 150-2, the action may be passed to the server-action manager 150-1, where again similar analysis and validation may be performed. The actions may be performed on the application program and may be stored in the projects data 185. For example, the user may provide multiple actions through the client action manager 150-2 and the actions may be auto saved or may be saved based on user input. Upon saving, the actions may be validated by the client-action manager 150-2 and if validated, are provided to the server- action manager 150-1.

[0041] The concurrent checker 165, like the action manager 150, may also have a client side component, a client- concurrent checker 165-2, and the server concurrent checker 165-1. For the sake of brevity, the client action manager 150-2, the client layouts manager 155-2, and the client concurrent checker 165-2 have been illustrated in user device 110-1, however it will be appreciated that the client-action manager 150-2 may be provided in other user devices 110 as well. In addition to validation, the actions may be checked for probable conflicts and errors by the concurrent checker 165, before the collation of the application program, i.e., before the generation of the application source code. The concurrent checker 165 may check for probable conflicts and errors based on errors and caution rules. The client concurrent checker 165-2 may periodically call the server-concurrent checker 165-1 to receive an updated list of errors and cautions, which may have been identified by the server-concurrent checker 165-1 previously. The updated list of the errors and cautions may help the user is avoiding such errors in future actions.

[0042] For instance, the errors may relate to incorrect widget settings. For example, in a file manipulation widget, if "Create a file" is selected, and although the widget asks "which file" or "how to find the file", the value may be left blank by the user; then in such a case the widget may report this to the concurrent checker 165 in the next checking cycle that in the File Manipulation widget, "which file" parameter has been left blank. In another example, the user may select a Fetch Records widget to get records from the database and the user may select a database that has no structures in it. Further, the user may save the changes and move further. In this case, the Fetch Records widget may report to the concurrent checker 165 that the "Which database should be used" is not suitable. In yet another example, some widgets, while configuring, the user may be requested to "select a section" in their configuration options. If the user selects a section, which is different from the section that the widget resides in and saves the widget and later the selected section deleted. The widget may find an invalid section selection for one of its options, which will be reported to the concurrent checker 165.

[0043] Further, the errors may relate to incorrect project settings. For example, incorrect project name, code name, and description; incorrect format for deployment parameters, such as "Target Server Hostname". Furthermore, the errors may relate to incorrect section hierarchy. Such an error may be identified when a section has a duplicate name, or only symbols, etc.

[0044] Further, in order to check for certain compilation errors, a portion or whole of the application program may be collated using the collator 157 anytime during the development process. The collator 157 may collate source codes corresponding to various sections of the application program and upon collation may provide the collated code to a compiler (not shown in the figures) for compiling the application program. In an example, in case only a portion, say, a section, of the application program is to be collated, the collator 157 may collate the source codes pertaining to the selected section. The collation may include, among other things, initiating execution of widgets to generate corresponding source codes, gathering source code generated by various widgets, appending the source codes in a corresponding section to generate a source code of the selected portion in an organized and clean format, as will be explained in detail in subsequent figures. In an example, the selected portion may include all the sections of the application program. The source code of the application program may be referred to as application source code. Finally, the collated

application program may be deployed on a target device using the change management unit 160. Further, the change management unit 160 may store each deployment instance of the application program as a version for future use. The change management unit 160 may provide a list of the deployments for a current project. For instance, if the user has deployed the project ten times using the system 100, the list may ten entries, where each entry may be treated as a different version. Further, the change management unit 160 may allow the user to deploy each version separately. The change management unit 160 will be explained in detail with reference to description of Fig. 2.

[0045] In an implementation, the functionality of the APD system 105 may be provided as a paid service to the users. Further, the users may be provided the option to subscribe to one of the packages. In an example, higher the monetary value paid by the user, wider the range of functionalities may be provided. For instance, based on the package, the number of widgets made available to a user may increase and the number of application programs that a user can develop may increase. The information pertaining to user subscription may be stored in the subscription data 175, which may used by one of the other modules 170 to gather information pertaining to the user. Thus, based on the subscription data 175, a widget and widget packs unit (shown in Figure 2) may provide access to the user to a given set of widgets and/or widget packs and may be allowed to develop a predetermined number of application programs.

[0046] In another implementation, the APD system 105 may include a locale manager (not shown in the figures), which may allow the user to translate the developed application program into multiple languages, and to extend this support over all the sections and widgets. For the purpose, the locale manager may allocate numbers to the messages, i.e., set of codes, written in default-set or primary language, and allows the widgets to use numbers to represent these messages, so that the application program can be programmed to display messages in any supported language selected by the user. The locale manager may support Unicode and may provide for display of messages in any vernacular language.

[0047] In an example, the APD system 105 may provide the user a cell-grid view, where the first row may allow a different language to be selected in columns except the first. For example, A2 can be English, A3 can be Spanish, and so on. The first column may be reserved for string IDs. Then in each column, messages pertaining to or translated in that

column's selected language can be typed or copy pasted. Accordingly, each message is associated with a string ID. Users can use that string ID to represent a message that is available in different languages. The application program that is generated may have all the messages in all the added languages. The messages that appear on the user device may be dependent on a language selected by the user. The widgets also allow changing the language on the user side using the string IDs.

[0048] Fig. 2 schematically illustrates various components of the APD system 105, in accordance with an embodiment of the present subject matter, according to an embodiment of the present subject matter. As illustrated, the user device 110 may be interfaced with the action manager 150, the layouts manager 155, the collator 157, the concurrent checker 165, and the change management unit 160. In an implementation, every action provided by the user may have an action ID associated with it. The action ID may correspond to a development unit to which it relates to. As mentioned earlier, the action manager 150 may include multiple development units, such as a project settings unit 204, a databases unit 206, a structures unit 208, a sections unit 210, a programs unit 212, a layouts unit 214, a widgets and widget packs unit 216, and a fields unit 218. Further, it will be appreciated that although various development units have been illustrated as individual components, the functionalities of one or more units may be combined in a single unit as well.

[0049] In an implementation, based on the action ID, the action manager 150 may identify the corresponding development unit and information pertaining to the action IDs may be stored in the other data 190. Further, each of the development units may validate the received action using the validation rules. The validation rules may include checks for, for example, inclusion of special characters and syntax related errors.

[0050] For the sake of brevity, the foregoing description of the action manager 150 is explained in general without referring to the client and server components; however it will be appreciated the description may be valid for both the client and the server components. Further, in various implementations, only server-action manager 150-1 or only client action manager 150-2 may be provided.

[0051] In case the action ID indicates that the action relates to the project settings unit 204, the action is provided to the project setting unit 204. The project settings unit 204, as mentioned earlier, may validate the action based on corresponding validation rules. For

instance, the validation rules may return an error in case name of a project is not of a predetermined character length or includes a restricted character. In an example, the project setting units 204 may provide details pertaining to all the projects being handled by the user and interface the each project with a corresponding project file. The project file may include various details, such as date of creation, date of last modification, list of various sections, structures, and associated databases.

[0052] An example pseudo code for creation of a project is provided below:

```
{
        Name:<project_name>,
        Desc:<project_description>,
        Codename:<project_codename>,
        Settings:
        {
                PrimaryLocalLanguage: <language_codename>,
                SoftwarePlatform: <PHP / ASP.NET, C#.NET, VB.NET, Python, Ruby, etc>,
                DeployAs: <installer / compressed archive / directftp, directSSH, directp2p, remote desktop etc>
        },
        Defaults:
        {
                DefaultDBProvider: <default_database_provider>
                (e.g. MySQL, Oracle, SQL Server, PostGRESQL etc),
                DefaultDBName: <default_db_name>,
        (More may be included over time)
        }
```

[0053] In case the action ID indicates that the action relates to the databases, the action is provided to the databases unit 206. The databases unit 206, like the project settings unit 204, may validate the action based on corresponding validation rules. Further, the action may relate to database attributes illustrated in Fig. 3a. As illustrated, the databases unit 206 may assist the user in actions relating to the database attributes, such as a database name 305-1, structures 305-2, a platform 305-3, and database flag(s) 305-4. In an example, when the action relates to creation of a database, the user may provide inputs pertaining to each of the database attributes, such as name of the database to be created and underlying platform, such as ORACLE® and SQL®. The provision of choosing the platform of the database provides for platform independent creation of the application programs, i.e., the user may not be restricted to select only a particular platform for the database. Additionally, in case back-up database is created, the back-up database may be of a different platform as compared to the main

database. Thus, the present matter provides the flexibility of choosing the platform of the databases to be created. Further, the user may also provide actions to modify the name or the platform of the databases already created.

[0054] In addition, the user may also provide inputs pertaining to the database flags 305-4, which may include special instructions for advanced users. For example, a database '#ND' informs a deployer 224 to not deploy the database while initial deployment. This may be useful in case the user wants to deploy the database himself. Further, a database flag '#NC' informs the collator 157 to not create the databases at all. In another example, a database flag '#CP="123456"' informs the deployer 224 to change the database password to 123456 after deployment.

[0055] Further, as it will be understood, the database may include multiple tables, which are provided by way of the structures 305-2. The structures may be understood as set of data containers that can hold information in different columns. Each column may become a table's field upon deployment and each structure may become a database table on deployment. The database tables are used for several purposes, such as storing customer lists, phone numbers, user-access permissions, error logs, and in many situations, customer-provided data. Accordingly, a database may be understood as a set that holds various structures.

[0056] An example of a pseudo code in JSON format for creation of a database is provided below:

```
{
        name: <database_name>,
        structures: [
                0: <structure_name>,
                1:<structure_name>,
                continued to n:
        ]
}
```

[0057] Further, the actions pertaining to the structures 305-2 may be processed using the structures unit 208. The structures unit 208 may validate and process the actions pertaining to the structure attributes illustrated in Fig. 3b. As illustrated, the structure 305-2 may include a structure name 310-1, field(s) 310-2, structure flag(s) 310-3, and structure element(s) 310-4. The structure name 310-1 and structure flags 310-3 may be considered to

be similar to database name 305-1 and database flags 310-4. The structure elements 310-4 may include, for example, keys 315-1 and attributes 315-2. The keys 315-1 may aid in defining keys, such as primary keys and foreign keys.

[0058] In an example, attributes refer to information that is requested for a particular database, structure, field, section, or widget etc. In an example attributes for databases may be:

name=Main_database;
structures: Employees, Users, Clients.
In another example, attributes for structures may be:
Name: Employees;
Fields: => [
Employee_id=(primary key,integer(8)),
Employee_name=(varchar(100)),
Employee_joindate=(date(10)),
Employee_DOB=(date(10)
] etc

[0059] Further, the fields 310-2 indicate the various columns to be added in the structure 305-2. The fields 310-2 may in turn include field attributes, for example, field name 320-1, field type 320-2, field size 320-3, field default 320-4, field order 320-5, and field flags 320-6. As the label indicates, field name 320-1 relates to name of the field name, such as employee name and salary, field type 320-2 indicates the data type, such as strings, numeric, and alphanumeric, to be populated in a field, the field size 320-3 indicates the data size of data to be populated, such as character length, fields default 320-4 indicate the default value that is to be added, in case no value is provided by the user, field order 320-5 indicates the order in which various fields should be added in a structure, and field flags 320-6 are similar to structure flags 310-3 and indicate special instructions that may be provided for advance users.

[0060] As indicated in fig. 2, the structures unit 208 is interfaced with both the databases unit 206 and the fields unit 218. Thus, the actions pertaining to fields may be provided to the fields unit 218 for further processing and analysis. The databases unit 206, the structures unit 208, and the fields unit 218 simplify the process of database creation, by using the entered information and translating it into physical databases and tables, and data-columns. The structures and fields may be used by the widgets and widget packs unit 216 to link and filter information.

[0061] An example of a pseudo code in JSON format for creation of a structure is provided below:

```
{
name: <Structure_name>,
fields: [
        0: {
Name: <field_name>,
Type:<field_type>,
Size: <field_size>,
Default:        <field_tags>,
Order:<field_order>,
Flags: <field_flags>

                },
                continued to n:
        ]
}
```

[0062] An example of a pseudo code in JSON format for addition of a structure to a database is provided below:

```
{
        database_name: <database_name >,
        structure_name : <structure_name>
}
```

[0063] Where the databases unit 206, the structures unit 208, and the fields unit 218 aid in development of a database associated with an application program, the sections unit 210, the programs unit 212, and the widgets and widget packs unit 216 assist in developing the various sections of the application program. As mentioned earlier, based on the action ID, the action is provided to corresponding development unit for further analysis and validation. Accordingly, in case, an action relates to a section, the action may be provided to the sections unit 210. Sections may be understood as sets of different parts of an application program that are converted into individual pages, upon deployment. Further, a section may include programs that hold a desired functionality, which may be brought through the widgets. Thus, a program may be considered to be a collection of pieces of codes that are present in a single source-code file.

[0064] Development of a section may include defining of various section attributes, for example, a section name 325-1, a section description 325-2, a section layout 325-3, a parent section 325-4, a program 325-5 corresponding to the section, and section flags 325-6. The sections name 325-1 indicates a name of a section. The section description 325-2

indicates a description of a section provided by a user. The user may provide such descriptions, for example, to avoid confusion among similar sections. The parent section 325-4 indicates a section to which a current section may refer to. Program 325-5 may be automatically created upon creation of sections. However, the programs 325-5 may be created manually as well.

[0065] The section flags 325-6 may be understood to be similar to flags defined above. The section layout 325-3 may indicate a layout to be developed for a section of the application program. For the purpose, the sections unit 210 may couple to the layouts unit 214, which in turn may be coupled to the layouts manager 155. Thus, an action pertaining to creation or modification of a section may be provided to the sections unit 210, which may process and validate the action. Accordingly, the user may only need to provide information pertaining to one or more section attributes and a section may be created and/or modified accordingly. Further, the layouts may be accessed by the collator 157 to apply the designs to sections. For example, the layouts manager 155 may be used to create an interface for a "Create Employee" section. As will be explained in detail subsequently, the collator 157 ensures that the layout attached to a section is included in the source code as well. This inclusion may be understood to be an insertion of client-side code into the section. The client-side code renders the same layout that the user selected, while designing the application program to the section that is loaded while run-time.

[0066] As mentioned earlier, each section may include a program 325-5, which in turn may include one or more widgets 330 as illustrated in Fig. 3d. Further, the widgets 330 may be developed using the widget and widget packs unit 216. As illustrated in Fig. 2, the sections unit 210 may be interfaced with the programs unit 212, which in turn may be coupled to the widgets and widget packs unit 216. The widgets and widget packs unit 216, hereinafter referred to as widgets unit 216. Further, the widgets unit 216 may provide a user with a list of widget and widget packs, where each widget 330 corresponds to a pre-developed code snippet to achieve a desired functionality. Examples of the widgets 330 include widget for multiple database connections, query generators, add records to database, delete records from a database, fetch records from a database, modify records from a database, accept user input through fields, session management, file manipulation, file input/output, string manipulation, array manipulation, data manipulation, logical manipulation, mathematics, communications, information exchangers, graphical user

interface (GUI) messaging, data exchangers, import/export, uniform resource locator (URL) flow, charts and graphs, conditions and loops, big-data widgets, NoSQL database widgets, and API connectors. For example, the query generator widget may be used for generating insert, select, update, delete, truncate, and alter queries; the file manipulation generator widget may be used to create, delete, move files and folders, check permissions, get list of files and folders; the string manipulations widget may be used to declare string variables, store or read data from the variables, perform string specific operations, such as find and replace, find indexes, and substrings; and the big-data widgets may aid in storing data using big-data storage techniques.

[0067] Further, the widgets 330 may be provided by way of a widget pack. The widget pack may include multiple widgets 330 to achieve a desired functionality. Examples of widget packs include, but are not limited to, login pack, registration pack, user permission pack, add, edit, delete, display data pack, search and filter pack, business reports pack, data logging and error reporting pack, scheduled database backups pack, newsletter subscription pack, interactive calendar pack, hints and tooltips pack, client-side localization pack. For instance, the login pack widget pack may be used for creating a login mechanism, which may involve multiple sections and widgets. The login pack may include database access widgets, fetch records widgets, array manipulation widgets, and GUI messaging widgets, and login-success widgets, login-failure widgets, forgot-password widgets, and login form widgets. The widgets may be spread across various sections and which may be linked together at the time of collation of the application program. Similarly, the business reports pack may be used for creating financial and project management reports. Thus, in case of functionality involving multiple widgets, a user may select a widget pack corresponding to that functionality thereby saving on time and efforts.

[0068] Furthermore, a user may configure the widgets 330, using the widgets unit 216, based on application program requirements. For example, consider that a code for generating a form is to be added in the application program. In such a case, the user may select a form generation widget and add it to a required section of the application program. Then the user may also add another widget for picking up data from the form and placing it in the corresponding database. Upon adding the widgets, the widgets may be configured, for instance, the user may provide inputs pertaining to layout of the form, fields in the form, the database on which the data is to be placed and the like. Considering the example of big-data

widget, the user may provide details, such as number of data channels, filtering options for each data channel, and server IP addresses. Further, the two widgets may be interlinked to achieve the desired functionality. Thus, it will be appreciated that the widgets provide flexibility of changing a code, based on user inputs, without user having to manually type in the entire code.

[0069] In an implementation, an action pertaining to a widget may be provided to the widgets unit 216. Further, for configuring a widget, widget attributes, such as a localization information 335-1, an internal name 335-2, client side configuration attributes 335-3, and server side configuration attributes 335-4 may be defined by a user, such as an application developer, based on user preferences. As mentioned above, attributes refer to information that is requested for a particular database, structure, field, section, or widget, etc. The internal name 335-2 may be a name of widget provided by a user for widget and may be different from a pre-stored widget name. The client side configuration attributes 335-3 may include option name 340-1, element type 340-2, group name 340-3, localization information 340-4, option name 340-5, and option value 340-6. Further, server side configuration attributes 335-4 may include option name 345-1 and option type 345-2.

[0070] Based on the above-mentioned description, it can be gathered that the action manager 150 provides for validation and analysis of received actions to develop the application program. Further, as mentioned before, a dual validation may be performed, i.e., once at the client side, or to say the user device 100, and once at the server side, or to say the APD system 105. Once both the validations are positive, the action being tested may be executed and may be added to the project file. However, in case, the client-action manager 150-2 ascertains that the received action is not valid, the action may be dropped and an error may be logged. Further, in case, an action is validated by the client-action manager 150-2; however the action is not validated by the server- action manager 150-1, the action may not be executed and may be dropped. Additionally, the client-action manager 150-2 may temporarily store the actions prior to sending the actions for server- side validation.

[0071] Further, since the validators at the client-action manager will not permit and invalid action to be added, there is a very low probability that an action will reach the server validators without being properly validated and approved by the client-side validators.

Therefore, the client side may assume that its validators are reliable, and it will allow the user to perform actions without bothering him to save them after every few steps.

[0072] Further, the action manager 150 also provides for reversal of an action, i.e., undoing an action, based on action identifiers. Such a provision lets the user to modify the steps that were taken while creating the application program. Such steps may relate to, for example, sections, structures, fields, databases, and widgets. For deleting actions, the actions manager 150 may prompt the user to provide inputs that aid in determining the action identifiers corresponding to such actions. For instance, the user may provide a delete operation for deleting a section or a layout. Accordingly, based on the inputs, the selected actions may be deleted from the projects file stored in the projects data 185.

[0073] The deletion of actions using the actions managers 150 ensures that unchecked, erroneous, and incomplete codes are not kept in the application program to have zero or minimal syntax error-potential; and therefore is different from simple un-doing of changes on a source-code of an application program. For example, since, all the actions are validated before being saved at the APD system 105, which in turn ensures that only validated actions can affect or make changes to the application program. Therefore, unchecked and erroneous codes may not be generated. Further, the action reversal functionality may be provided at both client and server end only difference being in their coding technique / technicality, such as programming language, etc.

[0074] Thus, the users can use the action reversal functionality of the actions manager 150 to go back to any number of steps or actions that they performed on the application program. Accordingly, the actions manager 150 provides for recording of all actions performed by the user including, but not limited to, creation, editing, and deletion of sections, structures, databases, layouts, widgets, and programs, etc., so that the user, if he wishes, may revert to an action in order to undo certain undesired additions or changes.

[0075] In an implementation, in addition to development of databases and underlying logic, the user is also provided with the option of developing layout of various sections. The same may be achieved using the layouts manager 155. As illustrated in Fig. 2, the layouts manager 155 is coupled to the layouts unit 214, which in turn is coupled to the sections unit 210 and the programs unit 212. The layouts manager 155 allows the users to design, develop, and include various layouts and business reports in the application program. For example, the

layouts manager 155 assist the user in developing reports, graphical user interfaces, and attractive Hyper Text Markup Language (HTML) designs powered with JavaScript. The layouts manager 155 allows the users to define the layout style, layout type and layout settings to allow maximum flexibility. The layouts manager 155 also allows the users to drag and drop various user-interface components, configure functionality and physical appearance of the user-interface components, form fields, data fields, and text fields to pages. The layouts manager 155 accepts and exchange data from/to the widgets.

[0076] The data provided by the user is automatically recognized and based on the available content, the data is fit onto a layout according to the user-specified format. The available content can be an HTML output, a text output, an image source, a link, depending on which widget is sending it. The content goes in specific data holders. Each data holder has a specialized GUI component and has its own configuration just as widgets do, except that they don't need to be executed on the server. Further, the layouts may be dynamic in nature for allowing changes. However, there may also be a static version that can be exported in a digital document format, such as portable document format (PDF), doc, docx, or xls.

[0077] The layouts manager 155 may develop a layout based on inputs provided by the user regarding various layout features, such as documents, scopes, pallets, data holders, and themes. A document may be understood to be a collection of scopes that holds them in one set. A document can be treated as one printable page. Further, a document may have no fixed size and may be stretched without pixel-blurring to any extent depending on the content. For example, the layout manager 155 may implement vector graphics technique to avoid pixel blurring. Additionally, a document may be printed and exported in any standardized size.

[0078] A scope may be understood to be a collection of pallets that holds them in one set. A scope, like the document, may have no defined width, but it may have a defined height according to the requirement. The scope of a document can also be stretched according to size of the document. Further, the scope may be themed using theme manager component of the layouts manager 155. Additionally, the scope may have a priority associated with it, may allow the layout manager 155 to arrange scopes in the order of relevance. In an example, the priority may be provided by the user. However, the priority may also be over-ridden by widgets. For instance, in certain cases, a widget can be used in specific circumstances when a

scope needs to be brought on the top of the page, or a certain kind of sorting of scopes needs to be achieved.

[0079] A pallet may be understood to be a collection of data-holders and fields arranged, for example, horizontally and/or vertically, according the design specification provided by a user. Further, the pallet may not be rasterized. The pallet may be defined in terms of percent values that describe their width and height. Further, the pallet can have width and height values, according to the design specification. This in turn may allow the users to create layouts as per the desired layout style. Further, a pallet can be themed according to styles specified in corresponding parent scope. Similar to scopes, the pallets may have priority defined, but it may be overridden by the surrounding components as per relevance.

[0080] A data holder may be understood to be a data receiver packed with scripts that receive data from the widgets. Examples of data holders include, but are not limited to, buttons, menus, icons, labels, lists, tabs, graphs, charts, and customized GUI components inherited from one another. The user may provide inputs indicating what data is to be provided to which data holder. The data-holders may be generated using client-side script, such as Javascript and the data holders, when executed, may render HTML CSS as an output. The data holders may be used to import the data to the sections, during runtime, by source code of the widgets included in the sections. For instance, data-holders may accept inputs and pass the input to the collated section file.

[0081] The imported data may then be used for performing a required task, for instance, to draw a bar graph. Further, the data holder may hold any data type, based on the design specification provided by the user. Furthermore, the data holder may have fields arranged as per priority, which may allow the layouts manager 155 to arrange fields as per relevance. The data holders may provide for changing settings, for example, width, height, and theme combinations. Further, the user may through the layouts managers 155 using the data holders can set certain permissions, such as AllowResize, AllowArrange, AllowSort, AllowInvert, AllowTranspose, AllowCopy, in order to allow or disallow end users from manipulating the outlook and information stored in the data holders.

[0082] Further, the data holders may have two parameters, one being a data holder program and other being the data. The data holder program parameter may be provided on the

client side and may indicate how the data holder should behave. In an example, the data holders may be code snippets, which may be available at the client-layout manager 155-2 to allow dynamic access to the layouts. The data parameter may be used while executing the data-holder program.

[0083] There may be various types of data holders, such as tabular data holder, image data holder, textual data holder, and graphical data holder. The tabular data holder may contain another tabular data holder, image data holder, textual data holder, graphical data holder, or actual tabular data. Tabular data holders can be arranged column-wise, sorted, shuffled, inverted, transposed, copied, and pasted. In an example, the tabular data holders form HTML <table> tags, which may be created and changed dynamically. The image data holder may include an image. The image can be from a relative or absolute URL, or direct data encoded using a binary-to-text encoding technique, such as Base64. Image data holders allow the end users to rotate, change the brightness and contrast, scale, crop, and save the image based on the user-specified settings. The textual data holders may include only textual data. The textual data holders may allow change of character casing, font-size, and text decoration operations. Further, graphical data holder may include graphs, charts, and other graphical components.

[0084] In an example, the data holders may be understood to include three segments, an appearance segment, a hierarchy segment, and an event segment. The appearance segment may be used to store user-interface settings, such as sizes and ratios. The user-interface settings may be typically set by the user while designing a layout. The sizes and ratios may be used to determine the sizing of the component in a layout. Further, as the layouts are vector based, the layouts can be scaled to infinite resolutions, except when they have raster based backgrounds.

[0085] The hierarchy segment may be used to inherit traits from a parent data holder. In an example, a data holder may inherit from another data holder. In said example, new data holders may be created from the previously created ones without having to start from scratch, thereby saving on time and resources. For instance, in case a layout of a menu is being generated, the layouts for various windows that open on clicking a tab already provided on the menu, may be created using the concept of inheritance. In another example, consider a bar graph data holder, which may include some basic components, such as labels and buttons. A

bar graph may include labels containing the legend of notations that may inform the users of different aspects of a bar graph using titles and colors. The titles included in a bar graph are typically provided using "Label" data holder, and the buttons typically provided using "Buttons" data holders. Since each data holder has its own events, the inherited data holders events are imported while linking one data holder to another. The events of inherited data holders may be kept intact or may be overridden as needed; which in turn allows the data holders to perform their functions regardless of programming demands received by inherited data holders.

[0086] The event segment may include overridden client-side script events, such as Javascript events. For example, each data holder may offer blank functions, which may be executed by the data holders during when an overridden client-script event is encountered. For instance, there may be a blank function, which may be executed after each click on a certain part of the visible component of a dataholder and the user may change the blank portion, if required. In case the user changes that blank function, the newly mentioned code may be executed during each click instead. This allows all the visible components to be programmed to perform customized tasks. For example, a button can be programmed to refresh the page after the click, or it can be programmed to submit the input specified in a form to a specific section upon click, etc.

[0087] In an example, the data holders may be Javascript objects and may be programmable, i.e., user-specific events may be associated with the objects. The objects may be programmed using Document Object Model (DOM). The DOM may be used to link data holders to widgets in a project. Accordingly, widgets, when collated, may import data to the user device. The data may be imported to user-specified data holders. As mentioned above, the programmable events or the overridden client-side scripts may provide the users an opportunity to configure a data holder to behave in a certain way.

[0088] In another example, the event segment may also include pre-programmed events, which may be reserved for internal use only. The pre-programmed events ensure that a data holder behaves in a desired way. An example of a pre-programmed event of a "Button" data holder is a hover event, which is used to change the appearance of the button upon hovering in order to give the impression of change of state of the button from "active" to "hovered".

[0089] In an example, the pre-programmed counterparts of a data holder may be executed first followed by a corresponding programmable event. Consider a click event on a bar graph data holder. In such case, if a user clicks on a "bar graph" data holder, the pre-programmed click event will be fired first, followed by the programmable click event. The programmable click event can be configured by the user to do a specific task, such as fire an explicit function of a widget, or perform actions mentioned in GUI widgets.

[0090] A theme can be prepared using the theme manager component. The theme manager component provides for selection of backgrounds and borders with the desired colors, which in turn may be applied to the pallets. The theme manager component also provides for creating cascading style sheets and applying them automatically without manual programming.

[0091] Thus, for developing a layout, the user may provide inputs, which may pertain to color shades, textures, borders, relative sizes, ratios, etc., to the layouts unit 214, which in turn may provide the same to the layouts manager 155, which may assist in developing the layout. In an example, priority of one or more of the layout features may be overridden by the user using the client-action manager 150-2 and a widget, such as a GUI widget. This may allow most relevant content to be shown at the top of the page. In an implementation, the layout rendering may be performed by web browser engine of the user device 110 and the actions of sorting and organizing may be performed by the client-layouts manager 155-2. For example, the client-layouts manager 155-2 may sort all the documents in ascending order of priority. Further, each document may be drawn into a loop. For instance, if a page has three documents, then the documents will be drawn one by one, one below another, based on the priority.

[0092] In each document, all the scopes may be sorted in the ascending order of priority. Further, all the scopes in a document may be drawn in a loop. Finally, in each scope, all pallets may be arranged in the ascending order of priority and then each pallet is drawn. Further, the selected themes may be associated with the scopes and the pallets.

[0093] For the sake of clarity, an example of pseudo code in JSON format for developing designs is provided below:

layout_id: <layout ID>,

```
documents:[
        {
            0: {
                Document_id: <Document ID>,
                Priority: < Priority>,
                scopes:[
                    0: {
                        Margin: <margin_top_bottom_right_left>,
                        Padding: <padding_top_bottom_right_left>,
                        Width: <width%>,
                        Height: <height%>,
                        Priority: <Priority number>,
                        pallets:[
                            0:{
                                PalletName: <Pallet_name>
                                Margin:
<margin_top_bottom_right_left>,
                                Padding:
<padding_top_bottom_right_left>,
                                Width: <width%>,
                                Height: <height%>
                            },
                            continued to n:
                        ],
                    },
                    continued to n:
                ],
            }
        }
    themes:[
        0:{
            Name: <theme_name>,
            combinations:[
                0:{
                    Borders:{},
                    Background:{},

                },
                continued to n:
            ]
        },
        continued to n:
    ]
}
```

[0094] Referring back to Fig. 2, in an implementation, all the actions executed by the actions manager 150 are further checked by the concurrent checker 165. The concurrent checker 165 contains a server side component and a client side component for checking for

probable errors and cautions as will be explained in detail with reference to description of Fig. 5.

[0095] Further, anytime during the development process or upon finalizing the application program, the code may be collated and generated using the collator 157; and may be deployed using the change management unit 160. The change management unit 160 allows the users to collate a project several times after making one slot of changes and to retrieve the previously collated projects, if desired. The change management unit 160 may also employ a version control functionality, which may store the application program for a project as one version as against traditional way of storing the project as a set of written codes. Traditionally, version control tools store individual files that may or may not be changed; however, the change management unit 160 holds all actions and the corresponding results that were a part of the project before the project was collated in the project data 185. Since, the source code may be generated after receiving an input from the user to collate the project, the change management unit 160 allows the user to go back and forth between critical changes without tampering the integrity of the application program by changing the source code. Further, the input from the user may indicate to collate the application program with new settings, or use an existing change saved previously.

[0096] For example, if an application program relates to generation of a list including employees in a firm. The user may develop and subsequently deploy the application program on the firm's server. The time of deployment and the date of deployment of the application program may be stored by the change management unit. Upon deployment, consider that certain new features, such as petty cash transactions are also to be included in the application program. In such a case, the user may re-open the application program using the APD system 105, to create the Petty cash functionality, and deploy again. Again, upon making the change, an entry is added to the Change Management along with the date.

[0001] Further, after deployment, the user may realize that by mistake, while creating Petty cash functionality, the settings of the Employees listing functionality have been changed and have caused a negative impact. Now in order to get the stable version back, the user may go back to the change management unit and select previous deployment entry using dates and open the application program as it was on the deployment date selected by the user.

Thus, without changing a single line of code, or causing conflicts everywhere, the user get back old and stable release of the application program.

[0097] As illustrated in Fig. 2, the change management unit 160 is interfaced with a deployer 224, a collator initiator 220, and a collator 157. In an implementation, the collator 157 may be configured to collate and generate source code of a selected portion of the application program or entire application program, based on the user input. Further, the user may provide the input to the collator initiator 220, which may analyze the input to determine whether a portion or the entire application program is to be collated. In an implementation, the user may be allowed to collate the entire application program only when the concurrent checker 165 returns no errors and cautions. At the same time, in case the concurrent checker 165 returns errors and cautions for section A of the application program, the user may still be permitted to collate section B, which may be free from errors. In an example, if a portion of the application program is to be collated, the collator initiator 220 may create a collator thread on the fly. The collator thread would perform the similar function as that of the collator 157 but only on the portions selected by the user. The collator thread collates and subsequently provides for testing of the selected portions on a testbed server. In said example, the collator thread created on the fly, i.e., an auxiliary collator, allows the user to select the sections he wants to collate. In this case, the auxiliary collator copies the project in temporary storage of the server, removes the section that the user did not select, and only keeps the sections that were selected, in the application program. The selected portion is then collated by the collator 157 and subsequently, the deployer 224 deploys the collated code on the testbed server.

[0098] For example, in case the concurrent checker 165 returns no errors or cautions in a pervious checking process, the user may provide a request to collate and test a selected portion of the application program. This allows simultaneous testing to be possible after completion of each section and, the configuration of all the widgets contained within it. Likewise, once the application program is ready, the user may provide a request for final collation to generate the source code of the application program and for subsequent compilations.

[0099] Upon generation of the source code and the compilation, the user may provide a request to the deployer 224 to deploy the application source code, i.e., the application

program on a target device. Although deployer 224 has been illustrated external to the collator 157, it will be appreciated that the deployer 224 may be internal to the collator 157 as well. The details pertaining to the collator 157 and the 224 are provided with reference to description of Fig. 6.

[00100] Fig. 4 illustrates the widget unit 216 of the APD system 105, according to an embodiment of the present subject matter. The widget unit 216 allows for selecting and configuring of the widgets, such as the widget 330. As can be gathered from the description above, the widgets allow manipulations on several factors, such as database connections, encryption techniques, user redirection, browser recognition, maps integration, geo-location recognition, linking with other application programming interfaces (APIs), managing user sessions, files manipulation, interlinking sections, data conversion, signal processing, data mining, data projection, and code injection.

[00101] To develop the widgets, the widget unit 216 may include a client side component, referred to as a client-widget unit 405-1, and a server side component, referred to as a server-widget unit 405-2. In an example, the client-side widget unit 405-1 may include a client side validator 410-1 and may be coupled to a temporary storage 415. As mentioned earlier, upon receiving an action, each development unit may validate the action, based on corresponding validation rules. Accordingly, the client-side validator 410-1 validates the received action to ensure that the action does not include, for example, basic errors, such as syntax errors, and to provide protection against hacking and other malicious activities. Further, the temporary storage 415 may include actions pertaining to the widgets before the action is provided to the server-widget unit 405-2. For instance, the temporary storage 415 may include user inputs for configuring a widget and errors identified by the client-side validator 410-1.

[00102] As mentioned earlier, if an action is validated by the client-side validator 410-1, the action may be provided to the server-widget unit 405-2. Further, the server-widget unit 405-2 may include a server side validator 410-2, error logging unit 420, and a configuration assistance unit 425. The server side validator 410-2, like the client side validator 410-1, validates the received action, based on the validation rules. This ensures dual validation of the received action for better security and efficient utilization of resources. In case validation check returns any validation error, the identified error may be stored in the error logging unit

420, and may be shared with the user device 110. For example, since, the users cannot see the code stored on the server side, but can access the code loaded on their browser; and there are browser plugins available that allow the temporary storage on the user device 110, the data stored at the client side may be manipulated. Therefore, although an error that ideally should have been identified at the client-side, but is not caught during client side validation, due to hacking or manipulation attack, can be detected at the server side.

[00103] However, in case no validation error is identified, the configuration assistance unit 425 may assist the user in configuring the widgets. For example, based on user inputs including configuration details a widget or a widget pack from among a plurality of the widgets and the widget packs may be selected. The configuration details may pertain to various widget attributes, such as name and value. The configuration details may be stored and, when the application program is deployed, a source code of the widget may be generated based the configuration details. For example, in case the user selects an if-then-else widget, the pre-developed code of the widget may already include the logic of if-then-else, and the user may only need to provide the values for the various conditions, like, what value to pick if a parameter has a given value. Further, the configuration assistance unit 425 may regulate a smooth functionality of the computing system by defragmenting client-side temporary storage time-to-time.

[00104] Referring back to figure 4, as illustrated, the server- widget unit 405-2 may also be coupled to the collator 157 and the concurrent checker 165. The concurrent checker 165 may check for probable errors and conflicts pertaining to widgets, based on the error and cautions rules. Referring to the example mentioned above regarding if-then-else widget, in said example, the user may indicate that when a particular condition is met, go to another widget. Thus, a widget may be linked to another widget. In such a case, the concurrent checker 165 may check whether linking the current widget with another widget may result in a conflict and accordingly may return an error to the user. Accordingly, the concurrent checker 165 may continue checking for such errors and conflicts, before the application program is collated, to ensure compilation process is smooth and efficient.

[00105] Further, upon correcting the errors and addressing the cautions, if any, codes of various widgets may be collated using the collator 157. For instance, when a section of the application program is to be collated, the widgets unit 216 may interface with the collator 157

to generate corresponding codes of the selected widgets. Likewise, when the entire application program is to be collated, the widget units 216 may provide information pertaining to selected widgets and corresponding configuration details, and accordingly the collator may generate the source code.

[00106] Fig. 5 illustrates various components of the concurrent checker 165 of the APD system 105, according to an example of the present subject matter. As illustrated, the concurrent checker 165 may include the server-concurrent checker 165-1 and the client-concurrent checker 165-2.

[00107] In an implementation, the server-concurrent checker 165-1 implements checking processes for both errors and cautions. Further, the server-concurrent checker 165-1 may collect results, for example, in an array, which may include the identified errors and cautions. In an example, an algorithmic prediction of a likely fault may trigger a caution, whereas an absolute prediction, through general knowledge and hard-coded rules, may result in errors. Thus, cautions are generated heuristically, i.e., intuitively; while errors are generated using known rules that will surely detect a fault. For example, using a sign in wrong place your equation may trigger a caution; while an error may be triggered when a value is divided by zero.

[00108] Further, a caution may be different from an error as in case an error is identified, the user may not be allowed to proceed to collation, partial or complete, while in case a caution is identified, the user may still be allowed to proceed to collation process. Thus, in case an error is detected, the user may be requested to rectify the error before the application program is collated, while in case of a caution, the user may be allowed to proceed to the collation without to make any change to address the caution.

[00109] The server-concurrent checker 165-1 may include, for example, a project checker 505 and a widgets checker 510. The project checker 505 may loop through each of the components, such as sections, structures, databases, and layouts, of the application program except for widgets. The project checker 505 may loop through each of the components to check for errors and cautions, based on the error and caution rules. In an example, the error and caution rules may include checks for duplication of components, such as sections, structures, databases, or layouts; checks for properties of the components for possible restricted characters that are known to cause errors on server operating systems;

checks for aborted, or left alone components, such as databases left without any structures, or structures without any fields in them.

[00110] Thus, the project checker 505 may check for errors and cautions for each component and may return a collection of errors and cautions identified in various components of a given portion of the application program. The errors and cautions may be stored and may be provided to the client-concurrent checker 165-2, when requested.

[00111] The errors and cautions pertaining to widgets may be obtained by the widgets checker 510. As mentioned before, the widget checker 510 may initiate a widget with the user provided inputs, and upon initiating, the widget may call a validation function to provide probable conflicts and errors. Since every widget may correspond to a code that performs a unique task, the widgets checker 510 may not have specific conditions to match. In other words, based on individual widget requirement, corresponding probable error and caution rules may already be included in the widgets, which may be triggered by the widgets checker 510. Accordingly, widgets checker 510 may rely on individual widgets to provide inputs to the widgets checker 510 about possible errors, and cautions caused due to invalid widget configuration. Thus, new widgets may be added to the APD system 105 or previously added widgets may be modified without performing changes to the concurrent checker 165.

[00112] To check for errors and caution in the widgets, the widgets checker 510 may trigger each widget added in various sections of a selected portion of the application program to perform a check, based on the corresponding errors and cautions rules. The received list of errors and cautions, which is an updated array of errors and cautions, may be stored and provided to the client-concurrent checker 165-2, when requested. In an example, an error may be identified when a conflicts among interlinked widgets is determined. Further, in another example, a caution may be returned, when it is determined that addition of a widget in a section may impact performance of the application program and may make the application program unstable. Additionally, there may be certain errors and cautions that may be determined by the widgets checker independently.

[00113] Further, the client-concurrent checker 165-1 may include a server requester 515 and a timer 520, and may be coupled to a storage unit 525. In an implementation, the server requester 515 may periodically call the server-concurrent checker 165-1, based on a trigger received from the timer 520. The timer 520, after a predetermined time interval, may

provide an input to the server requester 515, which in turn may request the server-concurrent checker 165-1 to provide an updated list of errors and cautions. The updated list of errors and cautions may include a list of all errors and cautions identified so far for a given application program. The list may be stored in the storage unit 525, such as a browser cache of the user device 110. Further, the list may be provided using graphical elements, such as icons or designs in the GUI. Additionally, the various entries in the list may also include details pertaining to the items that resulted in the error or the caution, which in turn may allow the user to keep a watch on the conflicts and may help in reducing the errors.

[00114] From the above mentioned description, it can be gathered that the concurrent checker 165, as against traditional debugger, checks for conflicts and errors among various components of the application program and not the actual code, as the concurrent checker 165 performs the check, before collation and compilation, i.e., before the generation of actual source code. Thus, the concurrent checker 165 also may not require the code to be in execution as in traditional debuggers.

[00115] Further, the concurrent checker 165 may provide for reduction in number of errors. For example, in conventional APD processes, the compilation may provide multiple instances of the similar errors. However, in the present APD process, since the concurrent checker 165 continues to work in the background and checks for errors and conflicts in parallel to development process; therefore the users are informed about the errors in advance. Further, as the user would now have the prior information pertaining to errors, similar errors may be avoided.

[00116] Fig. 6 illustrates various components of the collator 157 of the APD system 105, according an embodiment of the present subject matter. The collator 157, based on the user inputs and pre-stored code snippets, generates a source code of the application program. In an example, when the collator 157 is initialized by the collator initiator 220, the collator 157 may create separate files according to the sections and store in the projects data 185. The collator 157 may merge the sections, if required. Further, the collator 157 may communicate with the selected widgets to request corresponding source code. Based on the inputs provided by the user regarding configuration of the widget, a corresponding source code is generated by the widget and provided to the collator 157. The source code may be placed into the corresponding files. Further, the collator 157 may also reformat the generated code for

maximum readability. As the widgets generate code that will be valid but may contain no spacing between lines, etc., therefore to make the code readable, the collator 157 may reformat the code without changing the underlying logic of the code. Finally, the collator 157 may generate scripts containing commands that would create the required databases, tables, and fields. These files may be used by the deployer 224 to create databases automatically on the target device.

[00117] In an example, the collator 157 may be interfaced with the deployer 224 and may include a collator handle 605, a collation recurser 610, a project collator 615, a program collator 620, and a widget collator 625. The collator handle 605 may function as a common platform for other development units to communicate to each other. For example, the collator handle 605 may allow the server-side component of widgets and other development units to access shared memory space for easy information exchange. The collator handle 605 may include a collator stack, a collator access list, collator errors, collator flags, and current data. The collator stack may be shared memory for the widgets and the widgets may use collator stack, for example, for storing variable names and common values. The collator access list may be shared list-style memory for the widgets and may be used by the project collator 615, the program collator 620, and the widget collator 625 for references of the source code files. The collator errors may be shared memory for storing errors and cautions encountered during collating process, the collator flags may correspond to expert instructions for expert use, and current data may be reference to the current section being accessed, and references to current program and widget as well.

[00118] The project collator 615 may loop through all the sections and create a directory hierarchy for keeping source code organized. The project collator 615, while looping through sections, may also create empty files in which the source code for that section can be placed. The project collator 615 creates initial entries containing project related data in the collator handle 605. The project related data include all information pertaining to the application program, such as information pertaining to sections, structures, fields, databases, widgets, widget-packs, and localization. An example of pseudo code for the project collator 615 is provided below:

```
/*Project Collator*/
function CollateProject($project_data,&$collator_handle)
{
    //loop through sections
```

```
for($i—0;$i<count($project data['sections']);$i++]
{
    $section_file_path=GetCleanSectionFilePath($project_data['sections'][$i]['name']);
    if($section_file_pathI==' ')
    {
        if(file_exists($section_file_path)= =false) {
            CreateRecursiveDir($section file path);
            CreateEmptyFile($section_file_path);
        //Add entry to the shared space confirming that the file has been made
        AddCollatorStackEntry($section_file_path,$project_data['sections'][$i]['name }
        /*Call Program Collator*/
        CollateProject($project data['sections'][$i]['name'],&$collator handle);
        $sourcecode-ReadData($section file_path);
        $sourcecode=FormatCode($project_data['sections'][$i]['name']
        ); WriteData($section file path,$sourcecode);
        }
    }
}
```

[00119] Further, the project collator 615 may call the program collator 620 for each section in the loop. The program collator 620 may loop through all the widgets included in the program and gather source code of each widget. The source code provided by each widget may be appended into the section file. The path of the section files may be stored in the collator handle 605. Thus, the program collator 620 may assemble the source code of the widgets in the program.

[00120] To collate the source code of widgets in a section, the program collator may call the widget collator 625. The widget collator 625 may ensure that the widget's server side script is existent and is valid. Further, if the server-side script is existent and valid, the widget collator 625 may request the configuration assistance unit 425 to provide widget configuration details. The widget configuration details may be provided by the widget collator 625 to a corresponding widget, which may generate a source code, based on the configuration details provided in a user input.

[00121] The widget collator 625 may call the widget recurser 610 to aid in collation of source code of the widgets, if required. The collation recurser 610 may allow one widget to request another widget to be collated on demand. The collation recurser 610 may be used in cases where one widget needs the source code from another widget. For example, the collation recurser may be used while collating an if-else widget. The if-else widget allows users to add conditional checks and balances. The if-else widget may request the user for a

list of conditions and details pertaining actions that are to be performed if those conditions are met. In certain cases, the user may add other widgets in between multiple conditions. The selection of other widgets within one widget may be done using the widgets unit 216, but the execution of other widgets may be ensured by the collation recurser 610. Thus, the collator recurser 610 may ascertain, for each of the widgets, whether a widget refers to another widget and collate a source code of another widget to the source code of the widget, when the widget refers to another widget.

[00122] In an example, each widget may have a code generator function, which when called by the widget collator 625 may generate the source code based on the configuration details for particular combinations of sections, structures, fields, databases, and layouts. Further, the widget collator 625 may return the generated source code in multiple segments, for example, a GUI segment, server-side segment, and a dependency segment.

[00123] The GUI segment may include client-side scripts that perform specific tasks on the client-side machine, such as on a browser or an application residing on the client machine. The GUI segment may also include names of the files in which GUI Code should be placed in, based inputs received from the widgets.

[00124] The server-side segment includes generated server-side scripts that perform specific tasks on server machine. The server-side segment also includes the names of files that the widget recommends the server-side code should be placed in.

[00125] The dependency segment includes information corresponding to dependency of code generated by a widget. For example, if a widget generates a code that requires a third-party library, or a different section to be loaded beforehand, at the time of execution, the program collator 620 assembles these dependencies based on data included in the dependency segment.

[00126] Further, upon generating the source code for each section and widget, the project collator 615 may trigger the deployer 224 to reformat the source code placed in section files. The deployer 224 may close all the files opened by the program collator 620, and may create an archive including directory-hierarchy and source code files. To generate the archive, the deployer 224 may open each file and may reformat the source code for the

ease of readability. The deployer 224 may provide the path of the archive to the user to make the assembled source code of the application program available to the user for download.

[00127] Once the application source code, i.e., deployment package is generated, the collator 157 may request the user to select retrieval mode. The retrieval mode may indicate a mode in which the user would like to retrieve the application source code. In an example, the retrieval mode indicates that the retrieval is to be done using traditional HTTP download. In another example, the retrieval mode may indicate that the application source code may be uploaded to the user's server directly using File Transfer Protocols (FTP) or secure shell (SSH) protocols. Accordingly, the application source code may be deployed on the target device.

[00128] Finally, once the deployment is complete, the application program may be tested in a test environment may by a testing team and if any errors or irregularities in functionality with respect to what was finalized in the planning stage are found, the user may re-open the project file in the APD system 105 and make the required changes.

[00129] In an example, the inputs provided by the user for a selected widget may also be checked for errors and cautions by the selected widget. Each widget may include corresponding errors and cautions, and may provide the identified errors and cautions to the error logging unit 420. As will be explained in detail in description of Fig. 5, the error logging unit 420 may be interfaced with the server-concurrent checker 165-1 and may in turn provide the errors and cautions for further action.

[00130] Thus, it can be gathered that the various components of the APD system 105 encapsulate different functionalities of development process, thereby providing reductions in errors, efficient utilization of resources, human as well as computing. An example comparison of a typical development process and present APD process is provided below:

Table 1

Development of:	**Time required for manual development:**	**Time required for the development using present APD process**
A form rich application, containing 20 distinct forms, and the code for adding,	~ 100 man hours	~ 8 man hours (Using widgets for GUI, and

editing, and deleting information from that into a database		Databases)
The application mentioned above, with the addition of 10 unique designs, and payment gateway integration, 1 database, and an additional module for managing transaction records	~ 100 + 20 + 40 + 1 = 162 man hours (20 for 2 man hours per design) (40 for payment gateway integration) (1 for transaction management)	~ 8 + 4 + 1 + 0.2 = 13.2 man hours (4 man hours for designs, 1 man hour for Payment Gateway widgets and packs, and 0.2 man hours for transaction records)
The application mentioned above, with the addition of reports for projecting sales (payments), income, and taxes with interactive reports that would contain charts, containing 10 unique reports	~ 162 + (10x15) = 312 man hours (15 man hours per report)	13.2 + (10x2) = 33.2 man hours (2 man hours per report, using reports widgets, widget packs, and the layouts manager)
An ERP software containing 15 different modules, including Customer Relationship Management, Point of sales, Project Management System, Customer Portal	~312 * 15 = 4680 man hours	(33.2 * 15) - 10% = 448.2 man hours (-10% for customizable, self-made widget packs)

[00131] Thus, it can be gathered from table 1 that present subject matter provides for substantial reduction in time and efforts involved in developing an application program.

[00132] Fig. 7 illustrates a method 700 for developing an application program, and Fig. 8 illustrates a method 800 for generating an application program, according to an embodiment of the present subject matter. The order in which the methods is described is not intended to be construed as a limitation, and any number of the described methods blocks can be combined in any order to implement the methods or any alternative methods. Additionally, individual blocks may be deleted from the methods without departing from the spirit and scope of the subject matter described herein. Furthermore, the methods can be implemented in any suitable hardware, software, firmware, or combination thereof.

[00133] The methods may be described in the general context of computer executable instructions. Generally, computer executable instructions can include routines, programs, objects, components, data structures, procedures, modules, functions, etc., that perform particular functions or implement particular abstract data types. The methods may also be practiced in a distributed computing environment where functions are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, computer executable instructions may be located in both local and remote computer storage media, including memory storage devices.

[00134] A person skilled in the art will readily recognize that steps of the methods can be performed by programmed computers. Herein, some embodiments are also intended to cover program storage devices or computer readable medium, for example, digital data storage media, which are machine or computer readable and encode machine-executable or computer-executable programs of instructions, where said instructions perform some or all of the steps of the described methods. The program storage devices may be, for example, digital memories, magnetic storage media, such as a magnetic disks and magnetic tapes, hard drives, or optically readable digital data storage media.

[00135] Referring to Fig. 7, in an example the method 700 may be performed by the APD system 105 to assist a user in developing an application program. The application program may include multiple sections, where a section may include codes for various components, such as widgets, databases, and layouts. However, it will be understood that the certain sections may be left empty as well.

[00136] At block 705, an action pertaining to development of an application program from a user device is received. The development may include creation of a new application program or modification, which may include addition, deletion, and alteration, of any component, such as databases, structures, sections, and widgets, of the application program. For example, the action may include details for configuring a component, such as a database associated with the application program and one or more widgets.

[00137] At block 710, a development unit of an action manager, such as the action manager 150 corresponding to the received action is determined. In an example, the development unit may be determined using an action ID associated with the development unit.

[00138] At block 715, the received action is validated by at least one of a client-side and a server-side. The received action may be validated using the validation rules. In an example, the received action may be validated by the determine development unit.

[00139] At block 720, when the action is validated, inputs provided in the action may be analyzed for probable cautions and errors, based on error and caution rules. Further, the inputs may be stored and may be used for generating a source code during deployment process. The action may be analyzed for probable cautions and errors as and when the actions are validated and before the final collation and compilation process. In an example, the check for probable errors and cautions may be performed at the server-side of the concurrent checker 165 and may be provided to the client-side on a periodic basis.

[00140] At block 725, a selected portion of the application program is collated, based on a collation request. The selected portion may correspond to a specific portion of the application program or the entire source code. Accordingly, the collation request may indicate whether a portion or the entire application program is to be collated. In an example, collation may include assembling of source code of various sections of the application program. Further, the collation may also include compilation of the assembled source code. In an implementation, when the entire source code is collated, the application program, or to say, the application source code may be deployed on a target device for further actions, such as testing. Further, each deployed version of the application program may be stored for future use.

[00141] Referring to Fig. 8, in an example the method 800 may be performed by the APD system 105 to assist a user in creating a new application program.

[00142] At block 805, one or more databases to be associated with the application program may be created, based on inputs received for database attributes, such as the database name 305-1, the structures 305-2, the platform 305-3, and the database flag(s) 305-4.

[00143] At block 810, one or more structures for each of the databases may be created, based on inputs provided for structure attributes, such as the structure name 310-1, the field(s) 310-2, the structure flag(s) 310-3, and the structure element(s) 310-4.

[00144] At block 815, one or more fields for each of the structures may be created, based on inputs provided for field attributes, such as the field name 320-1, the field type 320-2, the field size 320-3, the field default 320-4, the field order 320-5, and the field flags 320-6.

[00145] At block 820, one or more sections of the application program may be created, based on inputs received for various section attributes, such as the section name 325-1, the section description 325-2, the section layout 325-3, the parent section 325-4, the corresponding program 325-5, and the section flags 325-6.

[00146] At block 825, at least one of one or more widgets and one or more widget packs are added to the sections, based on user input. The user input may indicate selection of widgets and widget packs from among a plurality of widgets and widget packs.

[00147] At block 830, as indicated, through block 805-825, the received actions are checked for validation errors, and probable cautions and errors. In an example, a validator of a corresponding development unit may check for validation errors, as mentioned at block 715. Further, the check pertaining to probable conflicts and errors may be performed by the concurrent checker 165, as indicated at block 720.

[00148] At block 835, the validation errors and other errors and cautions are provided to the user device.

[00149] At block 840, in case the action is validated and there are no errors, the sections may be collated to generate a source code of the application program. Further, it will be understood that any time during development process, the user may collate a selected portion of the application program as well.

[00150] At block 845, the collated source code of the application program may be deployed on a target device specified by the user.

[00151] Although embodiments for systems and methods for development of application programs have been described in a language specific to structural features or method(s), it is to be understood that the present subject matter is not necessarily limited to the specific features or method(s) described. Rather, the specific features and methods are disclosed as embodiments for systems and methods for development of application programs.

I/We claim:

1. An application program development (APD) system (105) comprising:

a processor (120);

a widgets and widget packs unit (216) coupled to the processor (120), the widgets and widget packs unit (216) being executable by the processor (120) to receive user inputs for configuring one or more widgets (330) to develop an application program, wherein a widget (330) includes a pre-developed code snippet to generate a part of an application source code of the application program; and

a collator (157) coupled to the processor (120) to:

initiate execution of the one or more widgets (330) to provide a corresponding source code, based on user inputs and the pre-developed code snippet;

collate the source code generated by the one or more widgets, wherein the collation is performed, before the application program is deployed; and

append the source code corresponding to the one or more widgets (330) in a corresponding section of the application program.

2. The APD system (105) as claimed in claim 1, wherein the collator (157) further comprises a project collator (615), wherein the project collator (615) loops through each of the one or more sections to create a directory hierarchy to keep a source code of the application program organized.

3. The APD system (105) as claimed in claim 2, wherein the collator (157) further comprises a program collator (620), and wherein each section of the application program includes one or more programs, and wherein while looping through each of the one or more sections, the project collator (615) calls the program collator (620) to gather a source code of each program included in the section, and wherein the program collator (620) appends the gathered source code in the corresponding section of the application program.

4. The APD system (105) as claimed in claim 1, wherein the collator (157) further comprises a widget collator (625) to initiate execution of each of the one or more widgets (330) to generate a source code using corresponding pre-developed code snippet and the user input, wherein the widget collator (625) returns the generated source code in one or more segments, the one or more segments comprising a GUI segment, a server-side segment, and a dependency segment.

5. The APD system (105) as claimed in claim 1, wherein the APD system (105) further comprises a concurrent checker (165) coupled to the processor (120) to determine probable conflicts and errors pertaining to a received action, based on error and caution rules, and wherein the probable conflicts and errors are determined before the one or more sections of the application program are collated and after the received action is validated.

6. An application program development (APD) system (105) comprising:

a processor (120);

a collator (157) coupled to the processor (120) to generate a source code of a selected portion of an application program, wherein the collator (157) comprises at least one of:

a project collator (615) to loop through each of one or more sections of the application program to create a directory hierarchy to keep an application source code of the application program organized;

a program collator (620) to gather a source code of each program included in a section, wherein the gathered source code is appended in a corresponding section of the application program; and

a widget collator (625) to initiate execution of each of one or more widgets (330), provided in a program (325-5), to generate a source code using corresponding pre-developed code snippet and user inputs; and

a concurrent checker (165) coupled to the processor (120) to determine probable conflicts and errors pertaining to a received action, based on error and caution rules, and

wherein the probable conflicts and errors are determined before the one or more sections of the application program are collated.

7. The APD system (105) as claimed in claim 6, wherein the APD system (105) further comprises a widgets and widget packs unit (216) coupled to the processor (120) to receive user inputs for configuring the one or more widgets to develop the application program, wherein a widget includes the pre-developed code snippet to generate a part of the application source code.

8. The APD system (105) as claimed in claim 5 or claim 6, wherein the concurrent checker (165) comprises a server concurrent checker (165-1) including:

a project checker (505) to analyze one or more of sections, structures, databases, and layouts to determine the probable conflicts and errors, based on the error and caution rules; and

a widget checker (510) to determine the probable conflicts and errors pertaining to the one or more widgets, based on inputs received from the one or more widgets (330).

9. The APD system (105) as claimed in claim 8, wherein the widget checker (510) initializes each of the one or more widgets (330) with the user input to call a corresponding validation function, the validation function being configured to check for the probable conflicts and errors.

10. The APD system (105) as claimed in claim 5 or claim 6, wherein the APD system (105) comprises a client concurrent checker (165-2) including a server requester (515) to periodically call a server concurrent checker (165-1) to receive an updated list of errors and cautions.

11. The APD system (105) as claimed in claim 1 or claim 6, wherein the collator (157) further comprises a collation recurser (610) to:

ascertain, for each of the one or more widgets (330), whether a widget (330) refers to another widget (330); and

collate a source code of the another widget to the source code of the widget (330), when the widget (330) refers to another widget (330).

12. The APD system (105) as claimed in claim 1 or claim 6, wherein the collator (157):

receives a collation request to collate a selected portion of the application program; and

creates a collator thread on fly to collate source code of the selected portion.

13. The APD system (105) as claimed in claim 1 or claim 6, wherein the APD system (105) further comprises an action manager (150) coupled to the processor (120), the action manager (150) comprising a plurality of development units, and wherein a development unit aids in development of a component, from among a plurality of components, of the application program, and wherein the action manager (150):

receives an action pertaining to development of the application program, the application program including the plurality of components, wherein the plurality of components includes at least one of widgets (330), sections, databases, and layout; and

provides the action to one of a plurality of development units for further processing, based on an action ID associated with a received action, the plurality of development units including the widgets and widget packs unit (216).

14. The APD system (105) as claimed in claim 13, wherein the action manager (150) provides for creation of a client side action manager (150-2), and wherein the client side action manager (150-2) includes a plurality of client side development units, and wherein each of the client side development unit validates the received action, based on validation rules.

15. The APD system (105) as claimed in claim 13, wherein the action manager (150) includes a server side action manager (150-1), and wherein the server side action manager (150-1) includes a plurality of server side development units, wherein each of the plurality of server side validation unit validates the received action, based on validation rules.

16. The APD system (105) as claimed in claim 13, wherein the plurality development units comprises at least one of a project settings unit (204), a databases unit (206), a structures unit (208), a sections unit (210), a programs unit (212), and a layouts unit (214).

17. The APD system (105) as claimed in claim 1 or claim 6, wherein the APD system (105) further comprises a deployer (224) to:

deploy the application source code on a target device; and

create one or more databases associated with the application program on the target device.

18. The APD system (105) as claimed in claim 1 or claim 6, wherein the APD system (105) further comprises a layouts manager (155) coupled to the processor (120) to facilitate generation of a layout of the section of the application program, based on one or more layout features selected by a user, the layout parameters comprising a document, a scope, a pallet, and a data holder, wherein the document is a collection of one or more scopes, each scope being a collection of one or more pallets, and each pallet being a collection of one or more data holders.

19. The APD system (105) as claimed in claim 18, wherein the layouts manager (155) includes a theme manager to configure a theme of at least one of the scope and the pallet.

20. The APD system (105) as claimed in claim 18, wherein the layouts manager (155), for each document:

draws one or more scopes in a loop, based on a priority associated with each of the one or more scopes;

for each scope, draws the one or more pallets in a loop, based on a priority associated with each of the one or more pallets; and

associates a selected theme with each of the one or more scopes and the one or more pallets.

21. The APD system (105) as claimed in claim 18, wherein the data holder receives data from a widget and imports the data to a corresponding section, during runtime.

22. The APD system (105) as claimed in claim 18, wherein the data holder includes one or more segments, each of the one or more segments comprising an appearance segment, a hierarchy segment, and an event segment, and wherein the appearance segment stores

user-interface settings, the hierarchy segment inherits traits from a parent data holder, and the event segment includes overridden client-side script events.

23. The APD system (105) as claimed in claim 1 or claim 6, wherein the APD system (105) further comprises a locale manager to configure the application program to display a message in a language selected by a user.

24. The APD system (105) as claimed in claim 1 or claim 6, wherein the widgets and widget packs unit (216) provides a set of widgets to a user, based on a subscription package selected by the user.

25. The APD system (105) as claimed in claim 1 or claim 6, wherein the user input to configure each of the one or more widgets (330) comprises at least one of a localization information, an internal name, client side configuration attributes, and server side configuration attributes.

26. The APD system (105) as claimed in claim 1 or claim 6, wherein the APD system (105) further includes a change management unit (160) to store each deployment instance of the application program as a version for future use.

27. A method for developing an application program comprising:

receiving one or more actions pertaining to development an application program, wherein at least one section of the application program includes one or more widgets (330), and wherein a widget includes a pre-developed code snippet to generate a part of an application source code of the application program;

providing each of the received actions to a corresponding development unit, from among a plurality of development units, based on an action ID associated with each of the one or more received actions;

determining probable conflicts and errors pertaining to each of the received action, based on error and caution rules, and wherein the probable conflicts and errors are determined before the one or more sections of the application program are collated; and

collating a source code of one or more sections of the application program, based on a collation request.

28. The method as claimed in claim 27, wherein the method further comprises:

deploying the application source code on a target device; and

creating one or more databases associated with the application program on the target device.

29. The method as claimed in claim 28, wherein the method further comprises storing each deployed version of the application source code.

30. The method as claimed in claim 27, wherein the method further comprises validating each of the one or more received actions by at least one of a client-side component of the development unit and a server-side component of the development unit, and wherein the probable conflicts and errors are determined, upon validation.

31. The method as claimed in claim 27, wherein the collating further comprises:

receiving a collation request to collate a selected portion of the application program;

creating a collator thread on the fly to collate the source code of the selected portion.

32. The method as claimed in claim 31, wherein the collating further comprises deploying the collated source code of the selected portion on a testbed server.

33. The method as claimed in claim 27, wherein the determining further comprises initializing each of the one or more widgets (330), with configuration details provided by the user, to call a corresponding validation function to check for the probable conflicts and errors.

34. The method as claimed in claim 33, wherein the determining further comprises providing an updated list of errors and cautions to a client concurrent checker.

35. The method as claimed in claim 27, wherein the collating further comprises:

initiating execution of each of the one or more widgets (330) to generate corresponding source code using corresponding pre-developed code snippet and the configuration details; and

returning the generated source code in one or more segments, the one or more segments comprising a GUI segment, a server-side segment, and a dependency segment.

36. The method as claimed in claim 27, wherein the method further comprises generating a layout of a section of the application program, based on one or more layout features selected by a user, the layout parameters including a document, a scope, a pallet, and a data holder, and wherein the document is a collection of one or more scopes, each scope being a collection of one or more pallets, and each pallet being a collection of one or more data holders.

37. A method for creating an application program comprising:

receiving an action to perform at least one of:

creating one or more databases to be associated with the application program, based on inputs provided for database attributes;

creating one or more structures for each of the databases, based on inputs provided for structure attributes;

creating one or more fields for each of the structures, based on inputs provided for field attributes;

creating one or more fields for each of the structures, based on inputs provided for field attributes;

creating one or more sections of the application program, based on inputs received for section attributes; and

adding at least one of a widget and a widget packs to a section, based on user inputs; and

collating a source code pertaining to the action to create the application program, before the application program is deployed.

38. The method as claimed in claim 37, wherein the method further comprises validating the received action, by at least one a client-action manager (150-2) and a server-action manager (150-1), based on validation rules.

39. The method as claimed in claim 37, wherein the method further comprises determining probable conflicts and errors pertaining to the received action, based on error and caution rules, and wherein the probable conflicts and errors are determined before the one or more sections of the application program are collated.

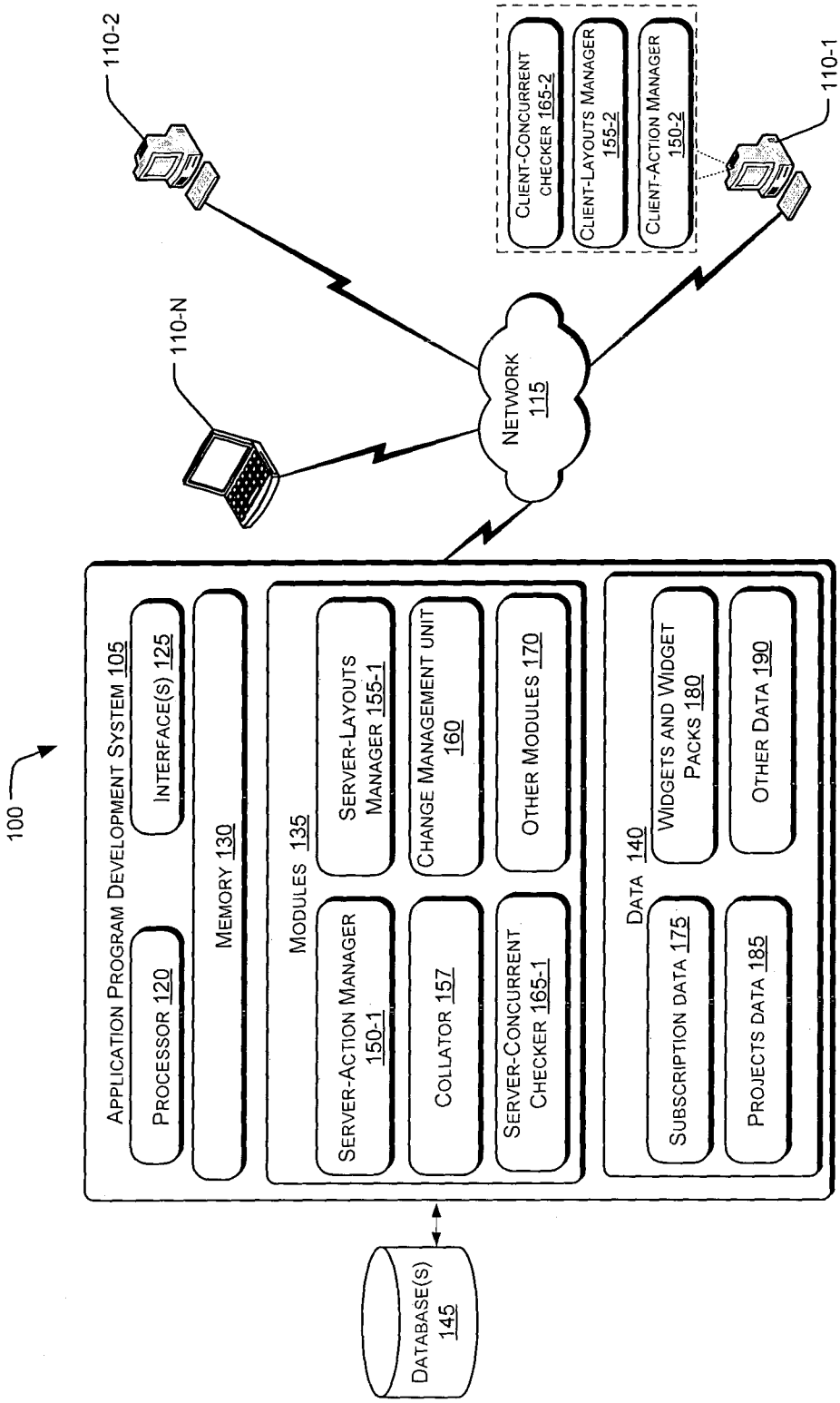
100
APPLICATION PROGRAM DEVELOPMENT SYSTEM 105
PROCESSOR 120
INTERFACE(S) 125
MEMORY 130
MODULES 135
SERVER-ACTION MANAGER 150-1
SERVER-LAYOUTS MANAGER 155-1
COLLATOR 157
CHANGE MANAGEMENT UNIT 160
SERVER-CONCURRENT CHECKER 165-1
OTHER MODULES 170
DATA 140
SUBSCRIPTION DATA 175
WIDGETS AND WIDGET PACKS 180
PROJECTS DATA 185
OTHER DATA 190
DATABASE(S) 145
NETWORK 115
110-N
110-2
110-1
CLIENT-CONCURRENT CHECKER 165-2
CLIENT-LAYOUTS MANAGER 155-2
CLIENT-ACTION MANAGER 150-2

Fig. 1

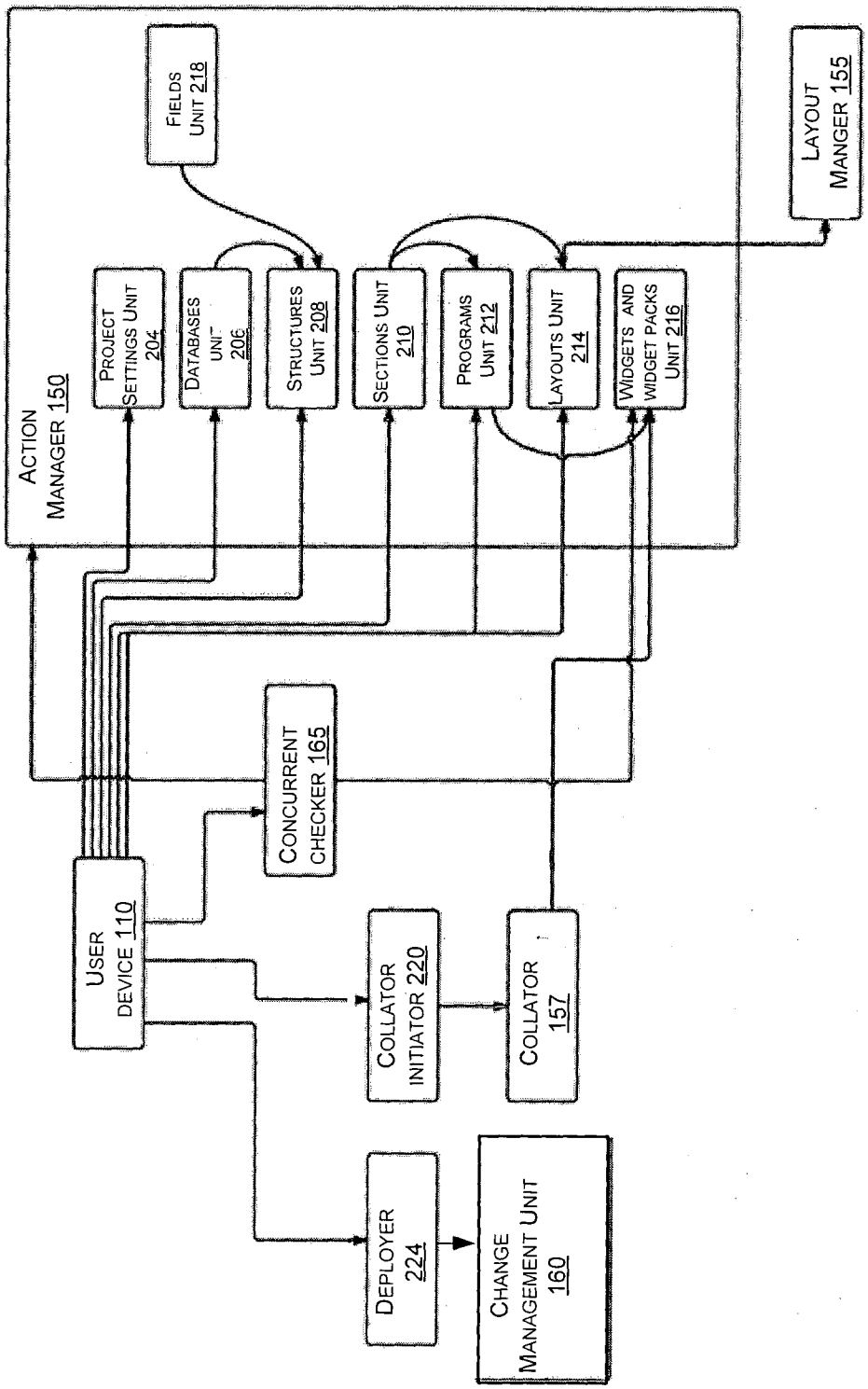
ACTION MANAGER 150
FIELDS UNIT 218
LAYOUT MANGER 155
PROJECT SETTINGS UNIT 204
DATABASES UNIT 206
STRUCTURES UNIT 208
SECTIONS UNIT 210
PROGRAMS UNIT 212
LAYOUTS UNIT 214
WIDGETS AND WIDGET PACKS UNIT 216
CONCURRENT CHECKER 165
USER DEVICE 110
COLLATOR INITIATOR 220
COLLATOR 157
DEPLOYER 224
CHANGE MANAGEMENT UNIT 160

Fig. 2

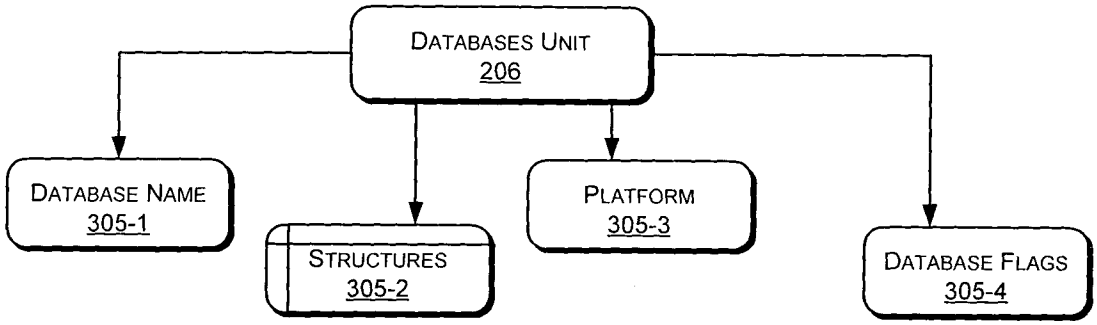
DATABASES UNIT
206
DATABASE NAME
305-1
STRUCTURES
305-2
PLATFORM
305-3
DATABASE FLAGS
305-4

Fig. 3a

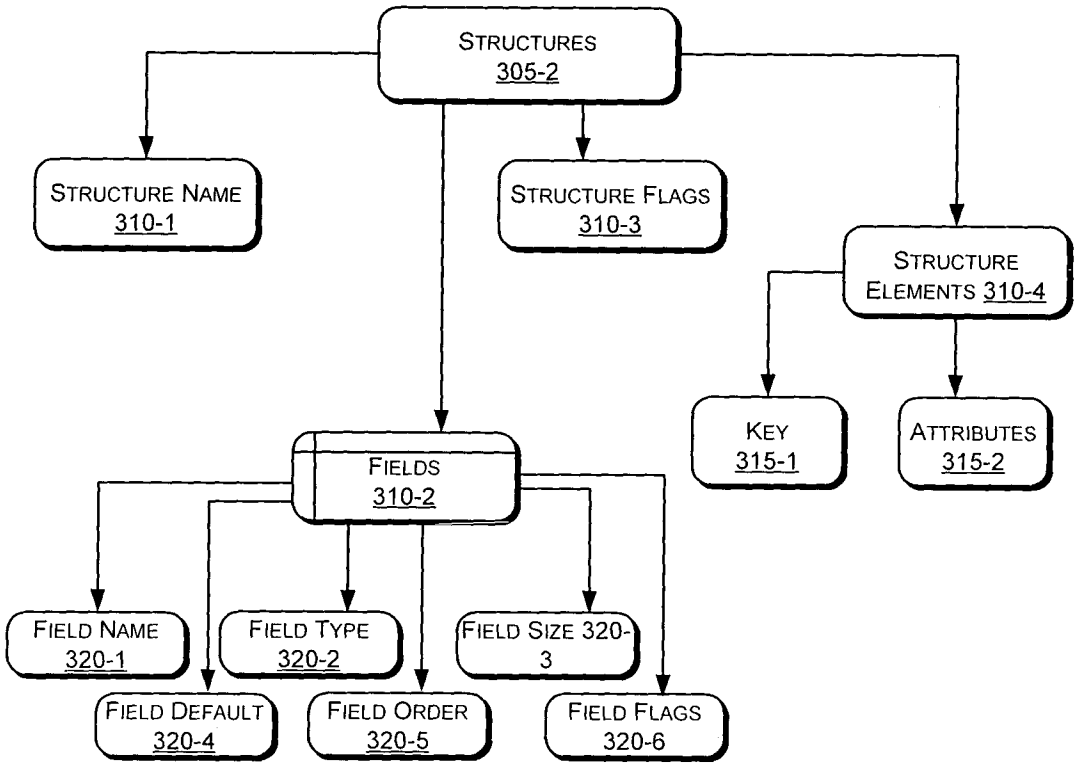
STRUCTURES
305-2
STRUCTURE NAME
310-1
STRUCTURE FLAGS
310-3
STRUCTURE ELEMENTS 310-4
KEY
315-1
ATTRIBUTES
315-2
FIELDS
310-2
FIELD NAME
320-1
FIELD TYPE
320-2
FIELD SIZE 320-3
FIELD DEFAULT
320-4
FIELD ORDER
320-5
FIELD FLAGS
320-6

Fig. 3b

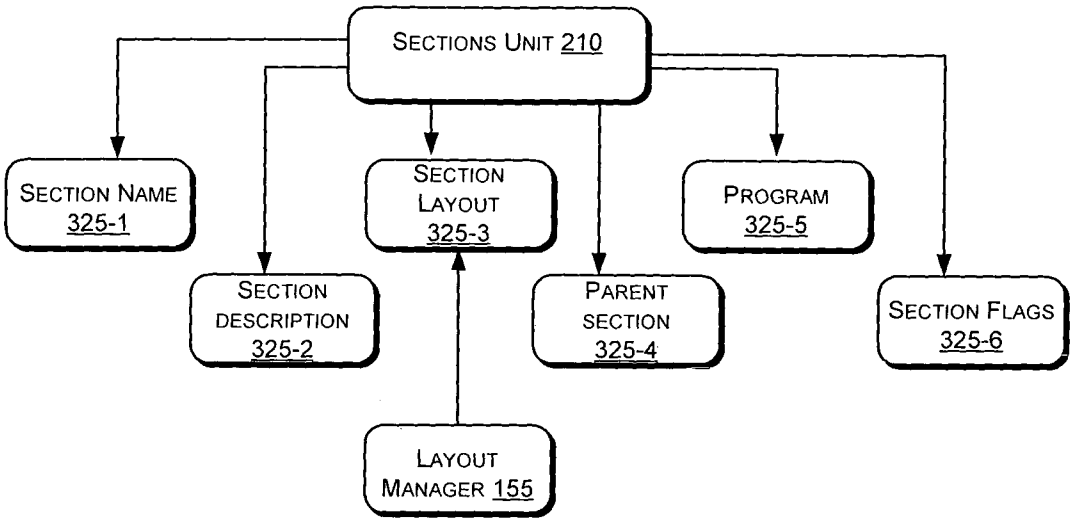
SECTIONS UNIT 210
SECTION NAME 325-1
SECTION LAYOUT 325-3
PROGRAM 325-5
SECTION DESCRIPTION 325-2
PARENT SECTION 325-4
SECTION FLAGS 325-6
LAYOUT MANAGER 155

Fig. 3c

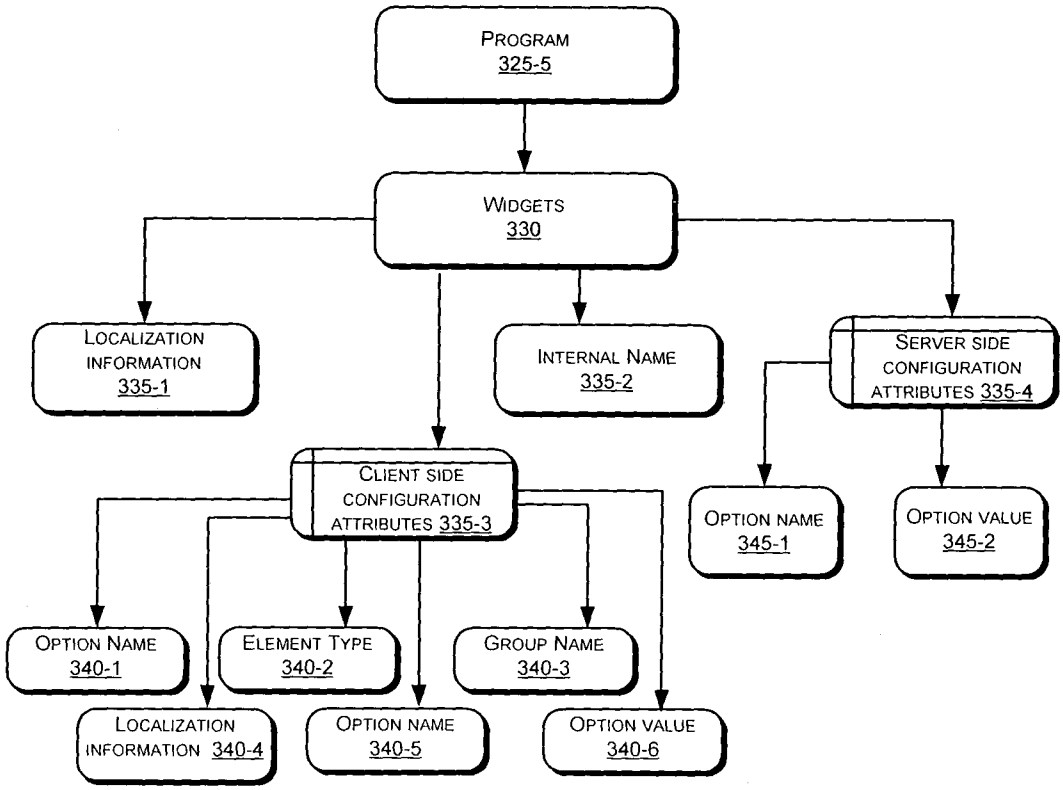
PROGRAM 325-5
WIDGETS 330
LOCALIZATION INFORMATION 335-1
INTERNAL NAME 335-2
SERVER SIDE CONFIGURATION ATTRIBUTES 335-4
CLIENT SIDE CONFIGURATION ATTRIBUTES 335-3
OPTION NAME 345-1
OPTION VALUE 345-2
OPTION NAME 340-1
ELEMENT TYPE 340-2
GROUP NAME 340-3
LOCALIZATION INFORMATION 340-4
OPTION NAME 340-5
OPTION VALUE 340-6

Fig. 3d

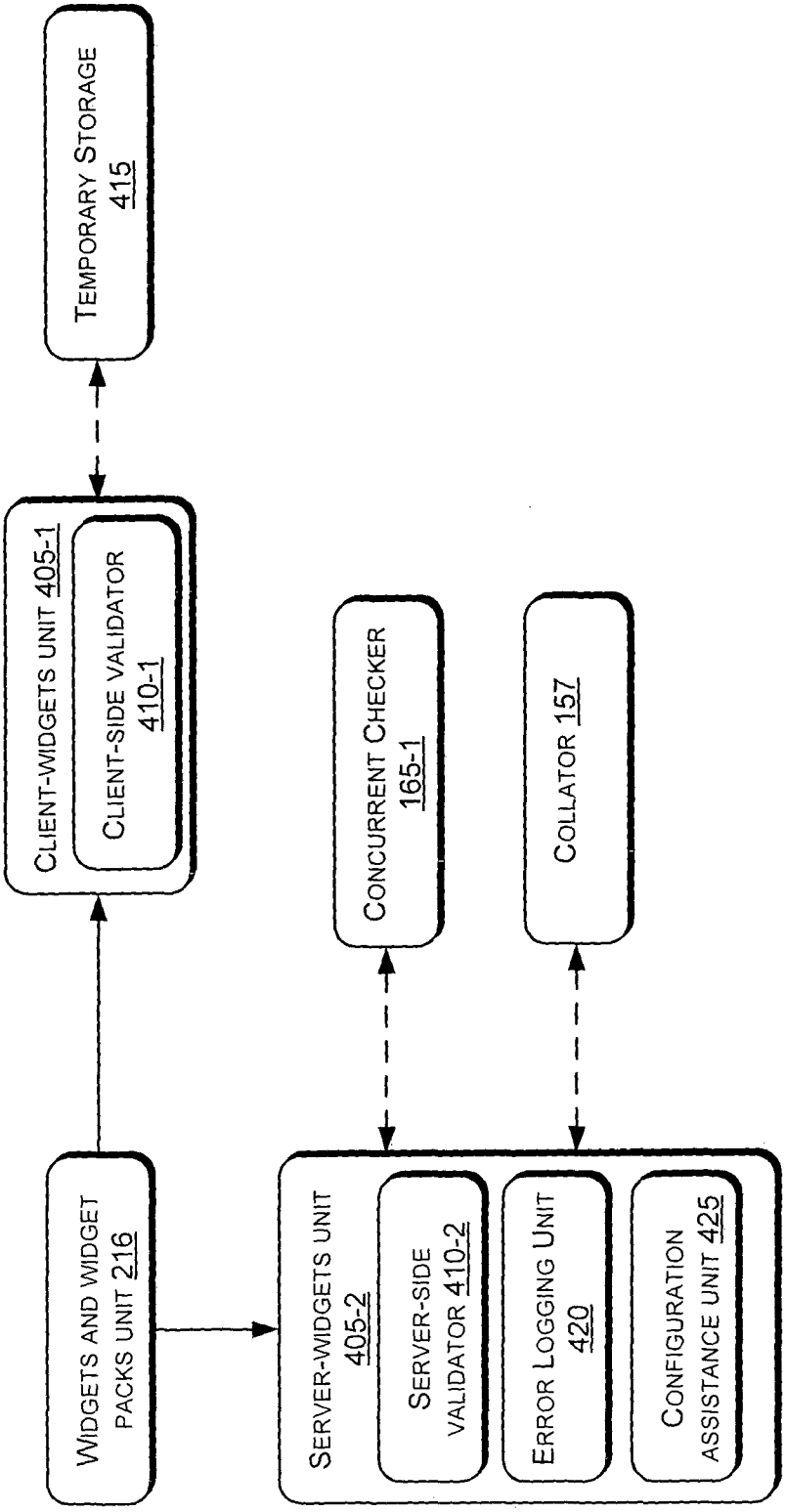
TEMPORARY STORAGE 415
CLIENT-WIDGETS UNIT 405-1
CLIENT-SIDE VALIDATOR 410-1
CONCURRENT CHECKER 165-1
COLLATOR 157
WIDGETS AND WIDGET PACKS UNIT 216
SERVER-WIDGETS UNIT 405-2
SERVER-SIDE VALIDATOR 410-2
ERROR LOGGING UNIT 420
CONFIGURATION ASSISTANCE UNIT 425

Fig. 4

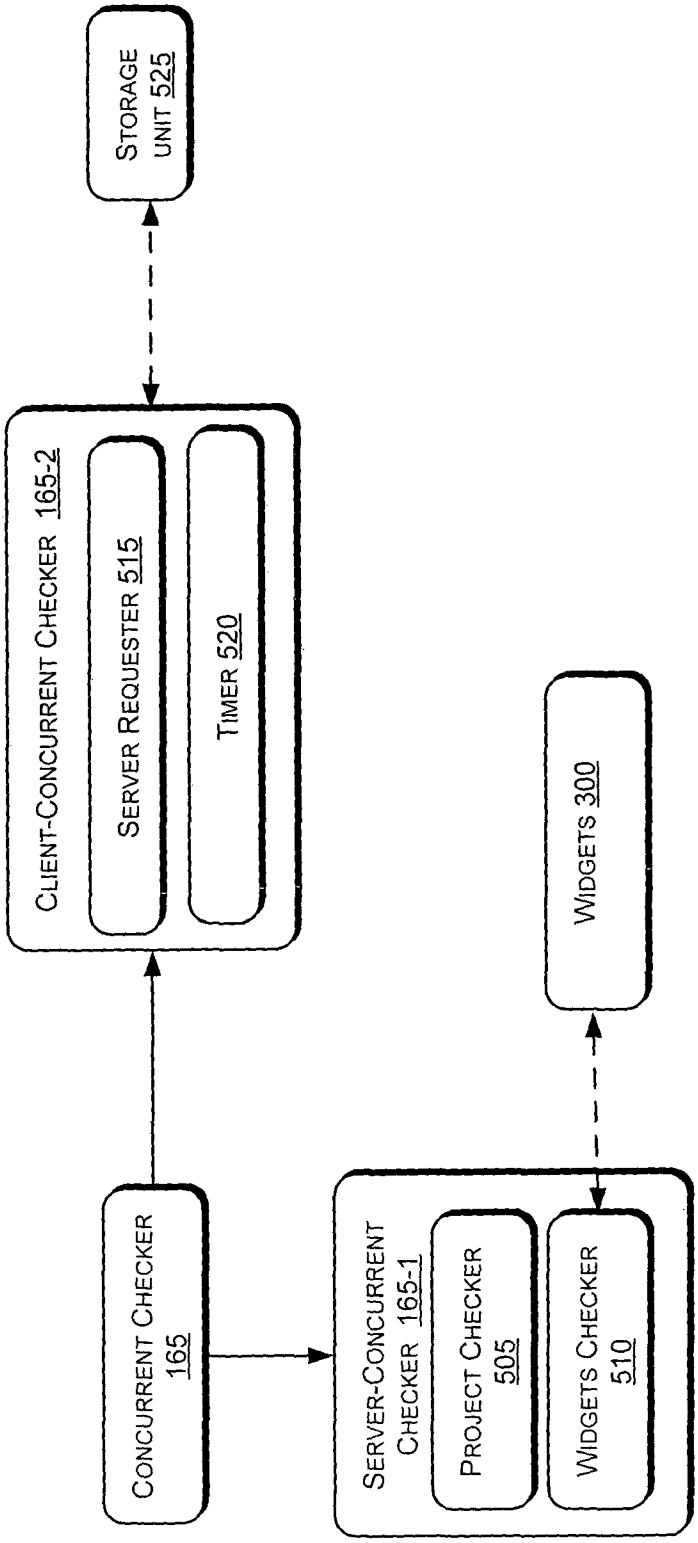
Storage Unit 525
Client-Concurrent Checker 165-2
Server Requester 515
Timer 520
Widgets 300
Concurrent Checker 165
Server-Concurrent Checker 165-1
Project Checker 505
Widgets Checker 510

Fig. 5

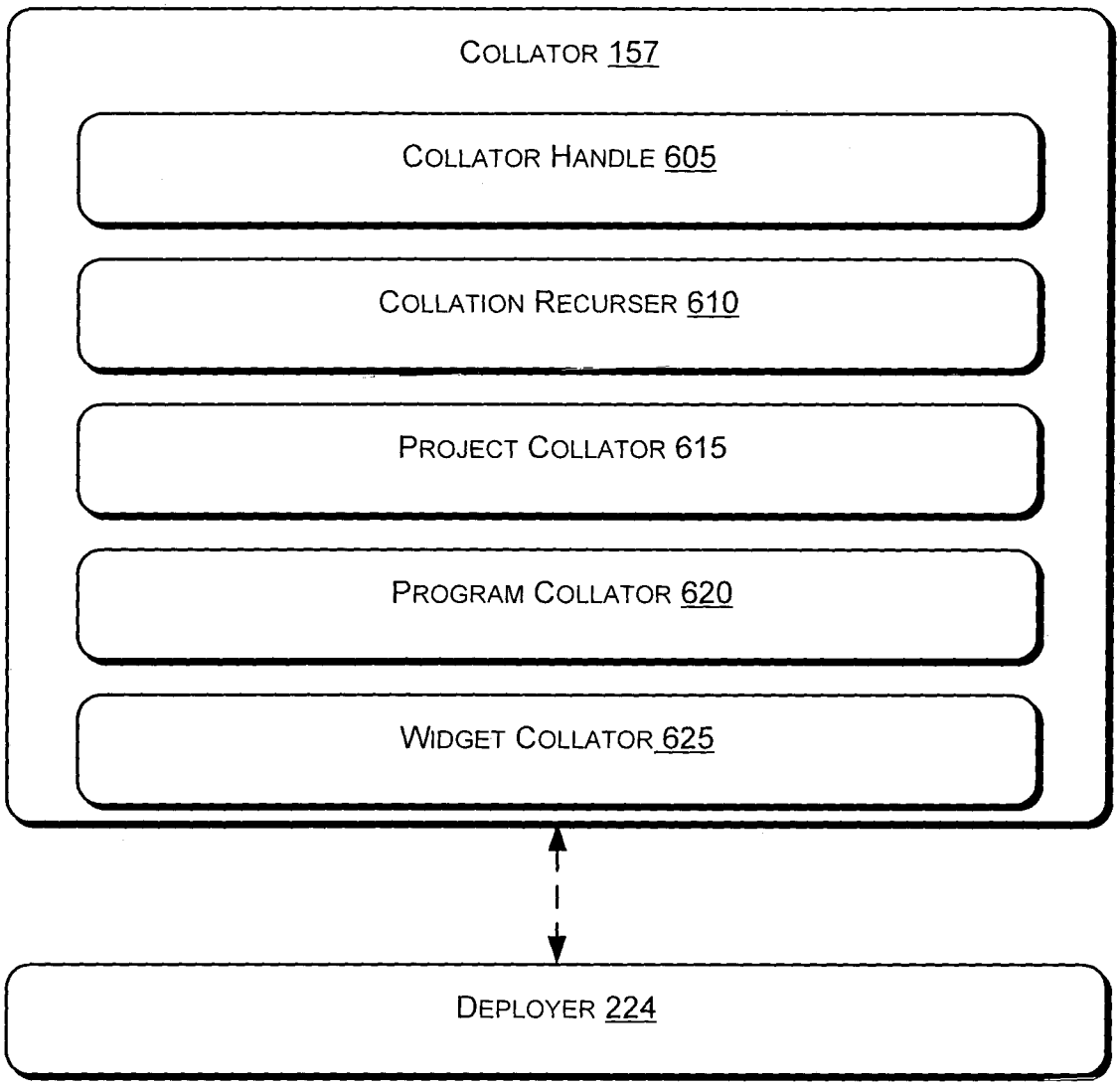
COLLATOR 157
COLLATOR HANDLE 605
COLLATION RECURSER 610
PROJECT COLLATOR 615
PROGRAM COLLATOR 620
WIDGET COLLATOR 625
DEPLOYER 224

Fig. 6

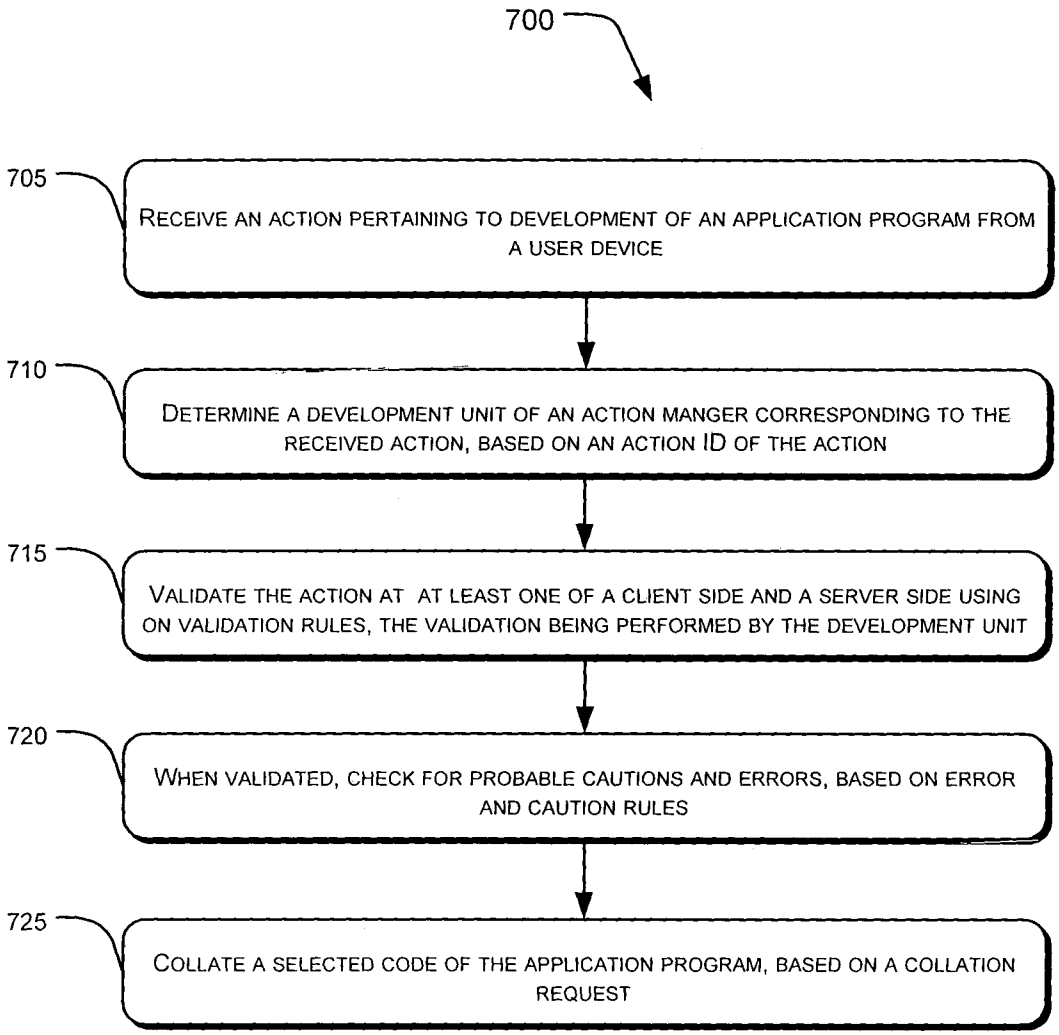
700
705
RECEIVE AN ACTION PERTAINING TO DEVELOPMENT OF AN APPLICATION PROGRAM FROM A USER DEVICE
710
DETERMINE A DEVELOPMENT UNIT OF AN ACTION MANGER CORRESPONDING TO THE RECEIVED ACTION, BASED ON AN ACTION ID OF THE ACTION
715
VALIDATE THE ACTION AT AT LEAST ONE OF A CLIENT SIDE AND A SERVER SIDE USING ON VALIDATION RULES, THE VALIDATION BEING PERFORMED BY THE DEVELOPMENT UNIT
720
WHEN VALIDATED, CHECK FOR PROBABLE CAUTIONS AND ERRORS, BASED ON ERROR AND CAUTION RULES
725
COLLATE A SELECTED CODE OF THE APPLICATION PROGRAM, BASED ON A COLLATION REQUEST

Fig. 7

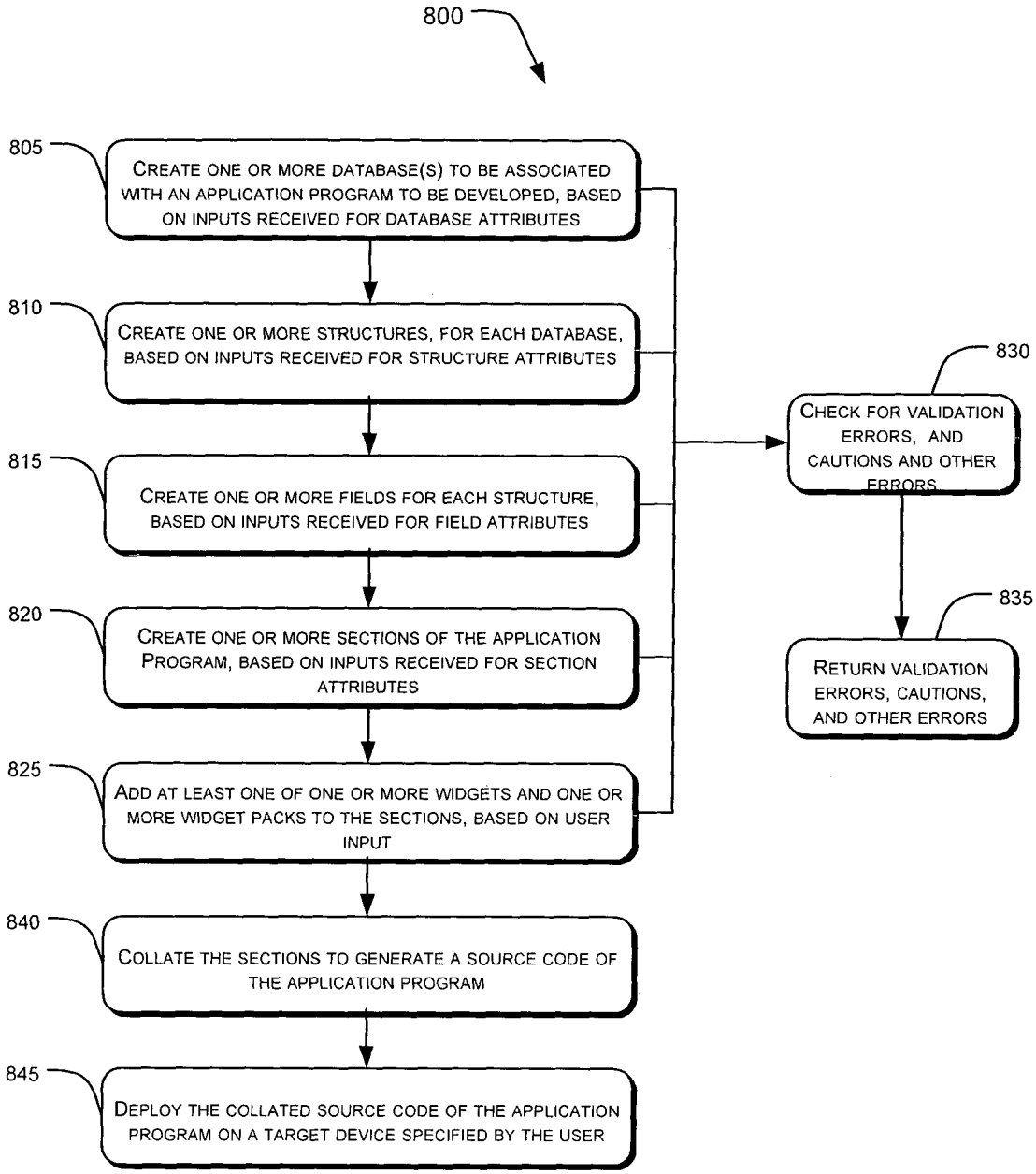
800
805
CREATE ONE OR MORE DATABASE(S) TO BE ASSOCIATED WITH AN APPLICATION PROGRAM TO BE DEVELOPED, BASED ON INPUTS RECEIVED FOR DATABASE ATTRIBUTES
810
CREATE ONE OR MORE STRUCTURES, FOR EACH DATABASE, BASED ON INPUTS RECEIVED FOR STRUCTURE ATTRIBUTES
815
CREATE ONE OR MORE FIELDS FOR EACH STRUCTURE, BASED ON INPUTS RECEIVED FOR FIELD ATTRIBUTES
820
CREATE ONE OR MORE SECTIONS OF THE APPLICATION PROGRAM, BASED ON INPUTS RECEIVED FOR SECTION ATTRIBUTES
825
ADD AT LEAST ONE OF ONE OR MORE WIDGETS AND ONE OR MORE WIDGET PACKS TO THE SECTIONS, BASED ON USER INPUT
840
COLLATE THE SECTIONS TO GENERATE A SOURCE CODE OF THE APPLICATION PROGRAM
845
DEPLOY THE COLLATED SOURCE CODE OF THE APPLICATION PROGRAM ON A TARGET DEVICE SPECIFIED BY THE USER
830
CHECK FOR VALIDATION ERRORS, AND CAUTIONS AND OTHER ERRORS
835
RETURN VALIDATION ERRORS, CAUTIONS, AND OTHER ERRORS

Fig. 8

INTERNATIONAL SEARCH REPORT

International application No.
PCT/IB2014/002787

A. CLASSIFICATION OF SUBJECT MATTER
G06F9/44,G06Q10/00,G06F17/30 Version=2014.01

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F, G06Q

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

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

Databases: PatSeer, IPO Internal
Search terms: Application Program Development, Widget, Layout

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US20110088011A1 (Vermeg Sarl), 14 April 2011 (Abstract; Paragraphs: - [0002]-[0009]; Claims 1-7)	1-39
A	US20080109785A1 (Bailey Bendrix L), 8 May 2008 (Abstract; Paragraphs: -[0012], [0021])	1-39
A	US5495567A (Ricoh Company Ltd.), 27 February 1996 (Abstract; Column 6, Line 10-Column 7, Line 5)	1-39
A	US20090055757A1 (International Business Machines Corporation), 26 February 2009 (Abstract; Paragraphs: - [0004], [0009], [0012])	1-39

☐ Further documents are listed in the continuation of Box C. ☒ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
01-05-2015	01-05-2015
Name and mailing address of the ISA/ Indian Patent Office Plot No.32, Sector 14,Dwarka,New Delhi-110075 Facsimile No.	Authorized officer Pranav Kumar Telephone No. +91-1125300200

International application No.
PCT/IB2014/002787

Citation	Pub.Date	Family	Pub.Date
US 20110088011 A1	14-04-2011	EP 2488941 A1	22-08-2012