



(43) International Publication Date
23 February 2017 (23.02.2017)

(51) International Patent Classification:
G06F 17/30 (2006.01)

(21) International Application Number:
PCT/US2016/033422

(22) International Filing Date:
20 May 2016 (20.05.2016)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/205,282 14 August 2015 (14.08.2015) US
15/054,755 26 February 2016 (26.02.2016) US

(71) Applicant: **ORACLE INTERNATIONAL CORPORATION** [US/US]; 500 Oracle Parkway, Mail Stop 50P7, Redwood Shores, California 94065 (US).

(72) Inventors: **STRAUB, Christian David**; 814 Altaire Walk, Palo Alto, California 94043 (US). **LIU, Peter**; 510 Shannon Way, Apt. #2202, Redwood City, California 94065 (US). **SUBRAMANIAM, Pavitra**; 968 Pearl Street, Alameda, California 94501 (US).

(74) Agents: **SHAFIEE, Shabnam** et al.; Miles & Stockbridge P.C., 1751 Pinnacle Drive, Suite 1500, Tysons Corner, Virginia 22102 (US).

(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, 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).

Published:

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

(54) Title: RESTORATION OF UI STATE IN TRANSACTIONAL SYSTEMS

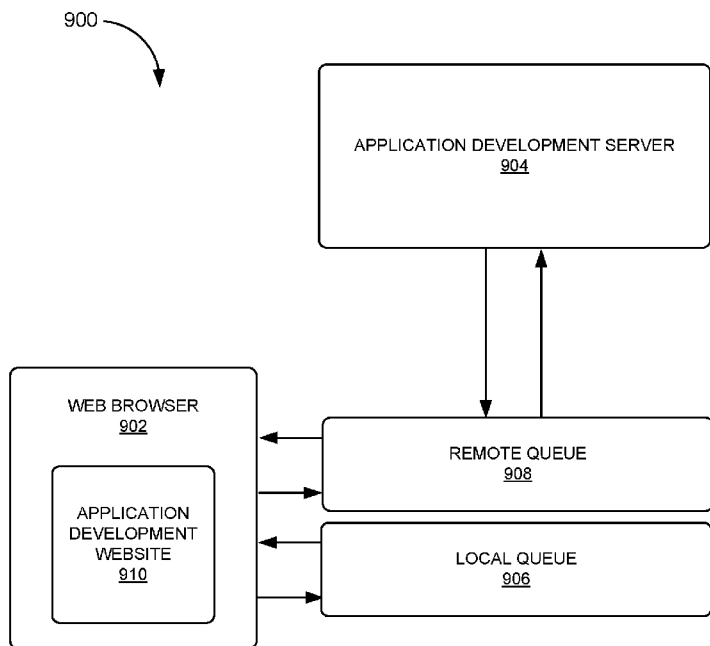


FIG. 9

(57) Abstract: A system restores a user interface ("UI") state. The system receives an action performed by a user that interacts with a UI, and determines a transaction based on the action, where the transaction is configured to modify a model corresponding to the UI. The system stores a first UI state of the UI and a first model state of the model, and then commits the transaction. The system subsequently determines to undo the transaction based on a first user interaction. The system then restores the UI to the first UI state and the model to the first model state. In one embodiment, the first model state is restored before undoing the transaction, while the first UI state is restored after undoing the transaction.

RESTORATION OF UI STATE IN TRANSACTIONAL SYSTEMS

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims priority of U.S. Provisional App. No. 62/205,282, filed August 14, 2015, entitled “RESTORATION OF UI STATE IN TRANSACTIONAL SYSTEMS,” the disclosure of which is hereby incorporated by reference.

FIELD

[0002] One embodiment is directed generally to a system that provides user interface (“UI”) restoration functionality, and in particular, to a transactional system that provides UI restoration functionality.

BACKGROUND INFORMATION

[0003] Generally, ubiquitous mobile services and wireless connections drive the demand for mobile device applications (commonly referred to as “apps”) for various personal and business needs. Such demand in turn leads to the desirability of mobile application development platforms/means that simplify and expedite mobile application development and modification, while also allowing for sophisticated application features and ensuring that business security is not compromised.

SUMMARY

[0004] One embodiment is a system that restores a user interface (“UI”) state. The system receives an action performed by a user that interacts with a UI, and determines a transaction based on the action, where the transaction is configured to modify a model corresponding to the UI. The system stores a first UI state of the UI and a first model state of the model, and then commits the transaction. The system subsequently determines to undo the transaction based on a first user interaction. The system then restores the UI to the first UI state and the model to the first model state. In one embodiment, the first model state is restored before undoing the transaction, while the first UI state is restored after undoing the transaction.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] Fig. 1 is a block diagram of a system for developing applications that use mobile cloud services, in accordance with an embodiment of the present invention.

[0006] Fig. 2 is a block diagram of a computing environment to facilitate communication between a mobile computing device and enterprise computer systems according to some embodiments of the present invention.

[0007] Fig. 3 illustrates a mobile application springboard in accordance with an embodiment of the present invention.

[0008] Figs. 3A and 3B illustrate a mobile application user interface (“UI”) in accordance with an embodiment of the present invention.

[0009] Fig. 4 is a block diagram of a mobile application framework runtime architecture in accordance with embodiments of the present invention.

[0010] Fig. 5 is a block diagram of a system for developing mobile applications in a mobile cloud infrastructure in accordance with embodiments of the present invention.

[0011] Fig. 6 is a block diagram of network components in a system for building mobile applications in accordance with embodiments of the present invention.

[0012] Fig. 7 is a block diagram of mobile security suite components in accordance with embodiments of the present invention.

[0013] Fig. 8 is a flow diagram of mobile application development in accordance with embodiments of the present invention.

[0014] Fig. 9 is a block diagram of a system for web application development in accordance with embodiments of the present invention.

[0015] Fig. 10 is a flow diagram of autosave functionality in accordance with embodiments of the present invention.

[0016] Fig. 11 illustrates an example user interface (“UI”) according to one embodiment.

[0017] Fig. 12 illustrates example commit, undo, and redo flows according to some embodiments.

[0018] Fig. 13 is a flow diagram of UI restoration functionality in accordance

with embodiments of the present invention.

DETAILED DESCRIPTION

[0019] An application refers to a software program, which on execution performs specific desired tasks. In general, several applications are executed in a run-time environment containing one or more operating systems (“OSs”), virtual machines (e.g., supporting Java™ programming language), device drivers, etc. Developers often use Application Development Frameworks (“ADF”) (which are by themselves applications) for implementing/developing desired applications. An ADF provides a set of pre-defined code/data modules that can be directly/indirectly used in the development of an application. An ADF may also provide tools such as an integrated development environment (“IDE”), code generators, debuggers, etc. In general, an ADF simplifies application development by providing re-usable components which can be used by application developers to define user interfaces (“UIs”) and application logic by, for example, selecting components to perform desired tasks and defining the appearance, behavior, and interactions of the selected components. Some ADFs, such as “Oracle ADF” from Oracle Corp., are based on a model-view-controller (“MVC”) design pattern that promotes loose coupling and easier application development and maintenance.

[0020] Generally, many companies have expressed the need to allow their employees to access secure enterprise applications with mobile devices from off-site locations, so that on-the-go employees can access information that is stored on enterprise computer systems. With such capabilities, salespeople may work from the road, service technicians may look up parts while at a customer site, employees may work from home, etc. Some companies would also like to allow end customers to access data located in enterprise computer systems. Such access may differentiate a company from competitors by improving the customer experience and lowering costs. For example, by implementing such access, a store may allow customers to remotely search store inventory for an item and shop whenever convenient, thereby improving customer experience and lowering the need for salespeople, operators, and other staff.

[0021] Different enterprise application vendors have traditionally fulfilled this need by offering specialized portals in combination with either company owned

secure mobile devices or custom mobile applications. However, with the current explosion in the variety of available personal mobile devices, these traditional solutions quickly become obsolete since vendors simply cannot keep up with all the latest OSs and hardware that become available.

[0022] Further, an application may need to connect and synchronize with different enterprise computer systems depending on the application type and/or the type of data used by the application. These enterprise computer systems may be supported by different backend computer systems which may also vary based on application type and data type. However, different backend enterprise systems may use different communication protocols and mechanisms to communicate data to devices, thereby causing mobile computing devices that run a variety of applications to encounter challenges for communicating with different backend computer systems that support an enterprise computer system.

[0023] Yet further, security may become a concern in allowing access to internal computer systems of an enterprise. The differences in communication protocols supported between the mobile computing devices and the enterprise computer systems may further complicate security access management for communications between mobile computing devices and enterprise computer systems. For example, different mechanisms may be implemented to ensure authentication of an application to access a particular enterprise computer system that has a proprietary security protocol. Some known systems have attempted to address this issues by connecting off-the-shelf consumer mobile devices with backend enterprise systems of companies. These devices may be configured with applications or OSs that connect to an enterprise network through special portals dedicated to communication with enterprise backend computer systems. However, manufacturers of mobile devices, application developers, and enterprises may benefit from more flexible and robust techniques for developing applications and connecting mobile devices to enterprise backend computer systems.

[0024] In contrast to the known systems, embodiments of the present invention provide a declarative browser based client application development tool for rapid business user friendly mobile application composition in a “cloud” service. In one embodiment, the cloud service is “Mobile Cloud Service” (“MCS”) from Oracle Corp. Embodiments allow for building mobile applications using pre-defined

templates that use the cloud service for backend services, so that a service definition can be presented to a developer during application development to allow for rapid connection between UI design and backend services.

MCS

[0025] In embodiments that use MCS, MCS facilitates communication between a mobile computing device and enterprise computer systems via a cloud computer system. MCS uses a third party cloud based interface between mobile devices and an enterprise network of a company. The cloud based interface centralizes secure adaptors for various enterprise computer systems, and translates different protocols to a standardized Representational State Transfer (“REST”) architecture. Companies can use embodiments of the present invention to create their own custom mobile applications using available tools on MCS, and such applications can be downloaded in native form onto mobile user devices. Once an application is installed, it can access the cloud based interface of MCS to reach various enterprise computer systems through the secure adaptors provided by MCS.

[0026] For application development in embodiments that use MCS, MCS provides backend services under the Mobile Backend as a Service (“MBaaS,” also referred to as “BaaS”) model. MBaaS allows Web and mobile application developers to link their applications to backend cloud storage and APIs exposed by backend applications while also providing user management, push notifications, integration with social networking services, etc. By using backend services provided in MCS under the MBaaS model, embodiments provide a declarative Web-based UI configured for mobile application development by non-technical users with no familiarity with coding.

[0027] In one embodiment, a wizard is launched when a user starts developing a new application, and the user is asked to give a name and description for the new application. Then, the user is asked to design the first page of the application by selecting from a set of pre-defined templates (e.g., tabs, bottom tabs, pagination, etc.) that can pre-seed the UI for the first page. The UI is then completed by specifying details in the template, while a preview is automatically updated to show the changes. Upon completing the UI design, the user can use a palette to browse a catalog of available services and data sources that are available to the mobile application through MCS (e.g., a service catalog). For each item of the

catalog that is added to the UI, the user is presented with a list of attributes, and using one or more gestures (e.g., drag and drop, etc.) the user can bind the attributes to UI elements. The user can repeat the process of feature definition and data-binding to create a mobile application. Other UI components such as maps, graphs, etc., can also be added to the UI. When the application is ready for testing, the user may publish the application so that corresponding binaries are created (building native executables for iOS, Android, or any other mobile device OS), and a Quick Response (“QR”) code is subsequently generated and provided to the user. If the user scans the QR code by a mobile device, the application is installed over the air onto the mobile device.

[0028] Embodiments use pre-built components in an ADF. The components offer data interaction, data visualization, and encapsulated browser side operations, and simplify rich client application development. ADF may also implement a plugin such as Apache Cordova plugin to access device features such as a camera, Global Positioning System (“GPS”), contacts, etc.

[0029] In one embodiment, when an ADF receives a request to build an application for a mobile device, it determines portions of one or more already developed applications that have been precompiled using a toolkit, and modifies declarative information associated with those existing applications. This embodiment then builds the requested application based on the modified declarative information and one or more binary artifacts of the existing applications by packaging the binary artifacts representing the requested application for a desired operating system (“OS,” such as iOS, Android, etc.). The ADF then compiles the requested application to generate one or more binary artifacts and a set of definition files. In end-user development, an artifact is an application or a complex data object that is created by an end-user without the need to know a programming language.

MOBILE SECURITY

[0030] Some embodiments use security services provided by a mobile security suite such as “Oracle Mobile Security Suite” (“OMSS”) from Oracle Corp. OMSS is a mobile device and mobile application security solution that provides an employee-centric, comprehensive Enterprise Mobility Management (“EMM”) solution and a consumer-centric mobile and social service. EMM provides mobile device management (“MDM”), mobile application management (“MAM”), mobile content

management (“MCM”), and mobile identity policies by seamlessly tying to existing user identities and leveraging advanced features of the enterprise backend identity management infrastructure for mobile access. Security policies, adhering to corporate needs, can be defined to enforce a complete device lock down (typically for corporate owned devices) and/or to separate personal applications from secure “containerized” corporate applications and data (for bring your own device (“BYOD”) cases). A mobile and social service provides a software development kit (“SDK”) allowing corporate developers to secure custom enterprise applications for iOS and Android devices, bridging the gap between mobile devices, social networks, and enterprise backend identity management infrastructure.

[0031] OMSS delivers a secure container to a mobile device for application and content security to separate, protect, and wipe corporate applications and data. All communication between the mobile device and enterprise intranet resources goes through an authenticated transport layer security (“TLS”) /secure socket layer (“SSL”) tunnel (“AppTunnel”) that can only be used by vetted (or “containerized”) applications of the mobile device. The AppTunnel is terminated at a Mobile Security Access Server located at the corporate demilitarized zone (“DMZ”). This server offers secure Intranet access to mobile devices and terminates only the AppTunnel from the secure container, thereby lessening the risk of rogue applications and the need for device level VPN.

[0032] Leveraging what is provided by an ADF, embodiments provide browser based application development, which does not require coding and which easily maps to business services. Embodiments also allow for previewing an application inline (e.g., as the application is being developed), as well as editing, testing, and publishing an application from a browser. Accordingly, instead of an IDE such as “Jdeveloper” from Oracle Corp. that is configured for use by professional developers, embodiments are configured for use by business users (e.g., non-technical users).

SERVICE CATALOG

[0033] To support embodiments of the present invention that use MCS, MCS provides access to an API catalog such as “Oracle API Catalog” (“OAC”) from Oracle Corp. OAC provides visibility to available APIs in an organization so those APIs may be reused for application development. OAC includes a simple metamodel for an API asset, automation to populate OAC with APIs, and the ability for users to search

OAC for APIs and understand the details of the APIs to assess their fit in their applications. OAC includes a harvester that creates API assets in OAC. In some embodiments, harvesting is performed at build time of projects. The harvester introspects deployed services and creates API assets representing services discovered in the project such as service oriented architecture (“SOA”) Services and Service Bus proxies, Web Services Description Language (“WSDL”) based Web services, and Web Application Description Language (“WADL”) based REST services. The created assets are collected in OAC.

[0034] After the API assets are created by the harvester, curators edit the API assets using a simple editor to provide additional metadata to facilitate the discovery and understanding of the APIs. Curators can change the name, add a description, tag keywords, or add document references to the API assets in OAC. This metadata simplifies discovery and understanding of each API asset by a user. After the API metadata is edited, curators publish the API by making it visible to users in OAC. Published assets are available in the OAC console and via the Oracle JDeveloper Oracle Enterprise Repository plug-in. Users can search OAC to discover APIs and review the metadata provided by the curator to learn more about an API.

[0035] Each OAC user is assigned a role which determines which OAC features and content are available for each user. There are predefined roles in OAC including developer, curator, and admin. Users with the developer role have the ability to search OAC for published APIs, examine the API metadata to better understand the API, declare interest in the API, and submit ratings and reviews for an API. In addition to the capabilities available to the developer role, users with the curator role can run the harvester to create new API assets in OAC, edit the APIs to update their metadata, and publish them. In addition to the capabilities available to curators and developers, users with the admin role have access to an Admin page in OAC to administer the infrastructure of OAC by editing system settings, creating new users, creating new departments, managing sessions, and using the import/export tool. Admins can also configure security features included with OAC.

[0036] In some embodiments, an application may be developed and deployed to a mobile device as either a native application or a hosted application. For native application deployment, a complete application is installed on the device. For hosted application development, a user needs to download a hosting application from an

“app store,” where such hosting application “hosts” the hosted applications that will be installed as “features” onto the hosting application. This embodiment may allow for updating a running hosting application from a server, such that declarative metadata can be sent to the device and overlaid on top of the existing application to update the application to run against this new metadata.

[0037] Fig. 1 is a block diagram of a system environment 100 for developing applications by using pre-defined templates that allow for use of MCS 122 as backend services. A service definition can be presented to a user during application development allowing rapid connection between UI design and backend services.

[0038] In the illustrated embodiment, system environment 100 includes cloud infrastructure system 102 that provides cloud services to one or more client computing devices 104, 106, and 108. Client computing devices 104, 106, and 108 may be used by users to interact with cloud infrastructure system 102. Client computing devices 104, 106, and 108 may be configured to operate a client application such as a Web browser, a proprietary client application (e.g., Oracle Forms), or some other application, which may be used by a user of the client computing device to interact with cloud infrastructure system 102 to use services provided by cloud infrastructure system 102.

[0039] Cloud infrastructure system 102 may have other components than those depicted. Further, the embodiment shown in Fig. 1 is only one example of a cloud infrastructure system that may incorporate an embodiment of the invention. In some other embodiments, cloud infrastructure system 102 may have more or fewer components than shown in FIG. 1, may combine two or more components, or may have a different configuration or arrangement of components.

[0040] Client computing devices 104, 106, and 108 may be portable handheld devices (e.g., an iPhone®, cellular telephone, an iPad®, computing tablet, a personal digital assistant (“PDA”)) or wearable devices (e.g., a Google Glass® head mounted display), running software such as Microsoft Windows Mobile®, and/or a variety of mobile OSs such as iOS, Windows Phone, Android, BlackBerry 10, Palm OS, and the like, and being Internet, e-mail, short message service (“SMS”), Blackberry®, or other communication protocol enabled. Client computing devices 104, 106, and 108 can be general purpose personal computers including, by way of example, personal computers and/or laptop computers running various versions of

Microsoft Windows®, Apple Macintosh®, and/or Linux OSs. Client computing devices 104, 106, and 108 can be workstation computers running any of a variety of commercially-available UNIX® or UNIX-like OSs, including without limitation the variety of GNU/Linux OSs, such as for example, Google Chrome OS. Alternatively, or in addition, client computing devices 104, 106, and 108 may be any other electronic device, such as a thin-client computer, an Internet-enabled gaming system (e.g., a Microsoft Xbox gaming console with or without a Kinect® gesture input device), and/or a personal messaging device, capable of communicating over network(s) 110.

[0041] Although exemplary system environment 100 is shown with three client computing devices, any number of client computing devices may be supported. Other devices such as devices with sensors, etc., may interact with cloud infrastructure system 102.

[0042] Network(s) 110 may facilitate communications and exchange of data between clients 104, 106, and 108 and cloud infrastructure system 102. Network(s) 110 may be any type of network familiar to those skilled in the art that can support data communications using any of a variety of commercially-available protocols, including without limitation transmission control protocol/Internet protocol (“TCP/IP”), systems network architecture (“SNA”), Internet packet exchange (“IPX”), AppleTalk, etc. Merely by way of example, network(s) 110 can be a local area network (“LAN”), such as one based on Ethernet, Token-Ring and/or the like. Network(s) 110 can be a wide-area network and the Internet. It can include a virtual network, including without limitation a virtual private network (“VPN”), an intranet, an extranet, a public switched telephone network (“PSTN”), an infra-red network, a wireless network (e.g., a network operating under any of the Institute of Electrical and Electronics (“IEEE”) 802.11 suite of protocols, Bluetooth®, and/or any other wireless protocol); and/or any combination of these and/or other networks.

[0043] Cloud infrastructure system 102 may comprise one or more computers and/or servers. These computer systems or servers may be composed of one or more general purpose computers, specialized server computers (including, by way of example, personal computer(“PC”) servers, UNIX® servers, mid-range servers, mainframe computers, rack-mounted servers, etc.), server farms, server clusters, or any other appropriate arrangement and/or combination. In various embodiments,

one or more computer systems or servers associated with cloud infrastructure system 102 may be adapted to run one or more services or software applications described in the foregoing disclosure. For example, one or more computer systems or servers associated with cloud infrastructure system 102 may correspond to a server for performing processing described herein according to an embodiment of the present disclosure.

[0044] One or more computer systems or servers associated with cloud infrastructure system 102 may run an OS including any of those discussed above, as well as any commercially available server OS. One or more computer systems or servers associated with cloud infrastructure system 102 may also run any of a variety of additional server applications and/or mid-tier applications, including hypertext transport protocol (“HTTP”) servers, file transfer protocol (“FTP”) servers, common gateway interface (“CGI”) servers, JAVA® servers, database servers, and the like.

[0045] In certain embodiments, services provided by cloud infrastructure system 102 may include a host of services that are made available to users of cloud infrastructure system 102 on demand, such as online data storage and backup solutions, Web-based e-mail services, hosted office suites and document collaboration services, database processing, managed technical support services, and the like. Services provided by cloud infrastructure system 102 can dynamically scale to meet the needs of its users. A specific instantiation of a service provided by cloud infrastructure system 102 is referred to herein as a “service instance.” In general, any service made available to a user via a communication network, such as the Internet, from a cloud service provider’s system is referred to as a “cloud service.” Typically, in a public cloud environment, servers and systems that make up the cloud service provider’s system are different from the customer’s own on-premises servers and systems. For example, a cloud service provider’s system may host an application, and a user may, via a communication network such as the Internet, on demand, order and use the application.

[0046] In some examples, a service instance instantiated by cloud infrastructure 102 may include protected computer network access to storage, a hosted database, a hosted Web server, a software application, or other service provided by a cloud vendor to a user, or as otherwise known in the art. For example,

a service instance instantiated by cloud infrastructure 102 can include password-protected access to remote storage on the cloud through the Internet. As another example, a service instance instantiated by cloud infrastructure 102 can include a Web service-based hosted relational database and a script-language middleware engine for private use by a networked developer. As another example, a service instance instantiated by cloud infrastructure 102 can include access to an email software application hosted on a cloud vendor's Web site.

[0047] In certain embodiments, cloud infrastructure system 102 may include a suite of applications, middleware, development service, and database service offerings that are delivered to a customer in a self-service, subscription-based, elastically scalable, reliable, highly available, and secure manner. An example of such a cloud infrastructure system as embodied in cloud infrastructure service 102 is "Oracle Public Cloud" from Oracle Corp.

[0048] Cloud infrastructure system 102 may provide the cloud services via different deployment models. For example, services may be provided under a public cloud model in which cloud infrastructure system 102 is owned by an organization selling cloud services (e.g., owned by Oracle Corp.) and the services are made available to the general public or different industry enterprises. As another example, services may be provided under a private cloud model in which cloud infrastructure system 102 is operated solely for a single organization and may provide services for one or more entities within the organization. The cloud services may also be provided under a community cloud model in which cloud infrastructure system 102 and the services provided by cloud infrastructure system 102 are shared by several organizations in a related community. The cloud services may also be provided under a hybrid cloud model, which is a combination of two or more different models.

[0049] In some embodiments, the services provided by cloud infrastructure system 102 may include one or more services provided under software as a service ("SaaS") category, platform as a service ("PaaS") category, infrastructure as a service ("IaaS") category, MBaaS category, or other categories of services including hybrid services. In some embodiments, the services provided by cloud infrastructure system 102 may include, without limitation, application services, platform services, infrastructure services, backend services, etc. In some examples, application services may be provided by cloud infrastructure system 102 via a SaaS platform.

The SaaS platform may be configured to provide cloud services that fall under the SaaS category. For example, the SaaS platform may provide capabilities to build and deliver a suite of on-demand applications on an integrated development and deployment platform. The SaaS platform may manage and control the underlying software and infrastructure for providing the SaaS services. By utilizing the services provided by the SaaS platform, customers can utilize applications executing on the cloud infrastructure system. Customers can acquire the application services without the need for customers to purchase separate licenses and support. Various different SaaS services may be provided. Examples include, without limitation, services that provide solutions for sales performance management, enterprise integration, and business flexibility for large organizations.

[0050] In some embodiments, platform services may be provided by cloud infrastructure system 102 via a PaaS platform. The PaaS platform may be configured to provide cloud services that fall under the PaaS category. Examples of platform services may include without limitation services that enable organizations (such as Oracle) to consolidate existing applications on a shared, common architecture, as well as the ability to build new applications that leverage the shared services provided by the platform. The PaaS platform may manage and control the underlying software and infrastructure for providing the PaaS services. Customers can acquire the PaaS services provided by cloud infrastructure system 102 without the need for customers to purchase separate licenses and support. Examples of platform services include, without limitation, “Oracle Java Cloud Service” (“JCS”) from Oracle Corp., “Oracle Database Cloud Service” (“DBCS”) from Oracle Corp., and others.

[0051] By utilizing the services provided by the PaaS platform, customers can employ programming languages and tools supported by cloud infrastructure system 102 and also control the deployed services. In some embodiments, platform services provided by cloud infrastructure system 102 may include database cloud services, middleware cloud services (e.g., Oracle Fusion Middleware services), and Java cloud services. In one embodiment, database cloud services may support shared service deployment models that enable organizations to pool database resources and offer customers a Database as a Service in the form of a database cloud. Middleware cloud services may provide a platform for customers to develop

and deploy various business applications, and Java cloud services may provide a platform for customers to deploy Java applications, in the cloud infrastructure system.

[0052] Various different infrastructure services may be provided by an IaaS platform in cloud infrastructure system 102. The infrastructure services facilitate the management and control of the underlying computing resources, such as storage, networks, and other fundamental computing resources for customers utilizing services provided by the SaaS platform and the PaaS platform.

[0053] In certain embodiments, cloud infrastructure system 102 may provide comprehensive management of cloud services (e.g., SaaS, PaaS, IaaS, and MBaaS services) in the cloud infrastructure system. In one embodiment, cloud management functionality may include capabilities for provisioning, managing and tracking a customer's subscription received by cloud infrastructure system 102, and the like. In various embodiments, cloud infrastructure system 102 may be adapted to automatically provision, manage and track a customer's subscription to services offered by cloud infrastructure system 102. A customer, via a subscription order, may order one or more services provided by cloud infrastructure system 102. Cloud infrastructure system 102 then performs processing to provide the services in the customer's subscription order.

[0054] In one embodiment, cloud management functionality may be provided by one or more modules, such as order management and monitoring module 114. These modules may include or be provided using one or more computers and/or servers, which may be general purpose computers, specialized server computers, server farms, server clusters, or any other appropriate arrangement and/or combination.

[0055] In exemplary operation, a customer using client computing devices 104, 106 or 108, may interact with cloud infrastructure system 102 by requesting one or more services provided by cloud infrastructure system 102. The customer may issue service request 134 cloud infrastructure system 102 using a variety of means. Service request 134 may include placing an order for a subscription for one or more services offered by cloud infrastructure system 102, accessing one or more services offered by cloud infrastructure system 102, or the like. In certain embodiments, the customer may access a cloud UI 132, 134, 138, and place a subscription order via

these UIs. The order information received by cloud infrastructure system 102 in response to the customer placing an order may include information identifying the customer and one or more services offered by the cloud infrastructure system 102 to which the customer intends to subscribe. After an order has been placed by the customer, the order information is received via cloud UIs, 132, 134, and/or 138.

[0056] In this example, order management and monitoring module 112 sends information received from a customer to an order database to have the order placed by the customer stored. The order database can be one of several databases operated by cloud infrastructure system 102 and operated in conjunction with other system elements. Order management and monitoring module 112 may forward information that includes all or part of the order information stored in the order database to an order management module. In some instances, the order management module may be configured to perform billing and accounting functions related to the order, such as verifying the order, and upon verification, booking the order.

[0057] In certain embodiments, cloud infrastructure system 100 may include identity management module 114. Identity management module 114 may be configured to provide identity services, such as access management and authorization services in cloud infrastructure system 102. In some embodiments, identity management module 114 may control information about customers who wish to utilize the services provided by cloud infrastructure system 102. Such information can include information that authenticates the identities of such customers and information that describes which actions those customers are authorized to perform relative to various system resources (e.g., files, directories, applications, communication ports, memory segments, etc.) Identity management module 114 may also include the management of descriptive information about each customer and about how and by whom that descriptive information can be accessed and modified.

[0058] In certain embodiments, cloud infrastructure system 102 may also include infrastructure resources 116 for providing the resources used to provide various services to customers of cloud infrastructure system 102. In one embodiment, infrastructure resources 116 may include pre-integrated and optimized combinations of hardware, such as servers, storage, and networking resources to

execute the services provided by the PaaS platform and the SaaS platform.

[0059] In some embodiments, resources in cloud infrastructure system 102 may be shared by multiple users and dynamically re-allocated per demand. Additionally, resources may be allocated to users in different time zones. For example, cloud infrastructure system 102 may enable a first set of users in a first time zone to utilize resources of the cloud infrastructure system for a specified number of hours and then enable the re-allocation of the same resources to another set of users located in a different time zone, thereby maximizing the utilization of resources.

[0060] In certain embodiments, a number of internal shared services 118 may be provided that are shared by different components or modules of cloud infrastructure system 102 and by the services provided by cloud infrastructure system 102. These internal shared services 118 may include, without limitation, a security and identity service, an integration service, an enterprise repository service, an enterprise manager service, a virus scanning and white list service, a high availability, backup and recovery service, service for enabling cloud support, an email service, a notification service, a file transfer service, and the like.

[0061] In certain embodiments, a number of external shared services 120 may be provided that are shared by different components or modules of cloud infrastructure system 102 and by the services provided by cloud infrastructure system 102. These external shared services 120 may include, without limitation, a security and identity service, an integration service, an enterprise repository service, an enterprise manager service, a virus scanning and white list service, a high availability, backup and recovery service, service for enabling cloud support, an email service, a notification service, a file transfer service, and the like.

[0062] In various embodiments, external shared services 120 may include one or more components that provide access, data transformation, automation, or the like to enterprise computer system(s) 126. Access to enterprise computer system(s) 126 may be shared by different components or modules of cloud infrastructure system 102 and by the services provided by cloud infrastructure system 102. In some embodiments, access to enterprise computer system(s) 126 may be shared by service instances provided by cloud infrastructure system 102 that are restricted to one or more subscribers.

[0063] In further embodiments, external shared services 120 may include external application programming interface (“API”) services 128 that are shared by different components or modules of cloud infrastructure system 102 and by the services provided by cloud infrastructure system 102. These external API services 128 may include, without limitation, APIs provided by other third party services or entities.

[0064] Various different mobile cloud services may be provided by MCS 122 in cloud infrastructure system 102. MCS 122 facilitates communication between a mobile computing device and enterprise computer systems (e.g., enterprise computer systems 124 and 126) according to some embodiments of the present invention. MCS 122 may include one or more memory storage devices (“local storage”) used to store enterprise data and authentication information. Enterprise data may be received from enterprise computer systems 126 or from client computing devices 104, 106, or 108 or may include enterprise data converted by cloud infrastructure system 102, or combinations thereof. Authentication information may be received from identity management system 116 and/or generated by cloud infrastructure system 102. In some embodiments, authentication information may include information indicating security authentication of a user with regard to a request for a service.

[0065] Enterprise computer systems, such as enterprise computer systems 126 may be physically located beyond a firewall of cloud infrastructure system 102 at a different geographic location (e.g., remote geographic location) than cloud infrastructure system 102. In some embodiments, enterprise computer systems 126 may include one or more different computers or servers. In some embodiments, enterprise computer systems 126 may be part of a single computer system.

[0066] In certain embodiments, enterprise computer systems 126 may communicate with cloud infrastructure system 102 using one or more different protocols. Each of enterprise computer systems 126 may communicate with cloud infrastructure system 102 using a different communication protocols. Enterprise computer systems 126 may support the same or different security protocols. In some embodiments, MCS 122 may include an agent system to handle communication with enterprise computer systems 126.

[0067] A protocol may include a communication protocol, such as SPeeDY

("SPDY"). A protocol may include an application protocol such as an HTTP-based protocol. In some embodiments, enterprise computer systems 126 may communicate with cloud infrastructure system 102 using a communication protocol such as REST or Simple Object Access Protocol ("SOAP"). For example, REST protocol may support a formats including uniform resource identifier ("URI") or uniform resource locator ("URL"). Enterprise Data formatted for communication using REST protocol may be easily converted to data formats such as JavaScript Object Notation ("JSON"), comma-separated values ("CSV"), and really simple syndication ("RSS"). Enterprise computer systems 126 and cloud infrastructure system 102 may communicate using other protocols such as remote procedure calls ("RPC") (e.g., extended markup language ("XML") RPC).

[0068] In some embodiments, MCS 122 may include an adaptor interface configured to support communication with one or more services provided by cloud infrastructure service 102, some of which may support different protocols or techniques for communications. In some embodiments, MCS 122 may include an adaptor interface configured to support communication with enterprise computer systems 126, some of which may support different protocols or techniques for communications. MCS 122 may include one or more adaptors each of which may be configured to communicate according to a communication protocol, a type of enterprise computer system, a type of application, a type of service, or combinations thereof. A communication protocol supported by an adaptor may be specific to a service or one or more of enterprise computer systems 126.

[0069] In certain embodiments, client computing devices 104, 106, and 108 may each implement an application that can provide specific UIs to communicate with MCS 122. A specific UI may be configured to communicate using a specific communication protocol. In some embodiments, specific UIs may include callable interfaces, functions, routines, methods, and/or operations that may be invoked to communicate with MCS 122. Specific UIs may accept as input parameters for communicating with a service provided by cloud infrastructure service 102 or with enterprise computer systems 126 for enterprise data and/or to request a service. In some embodiments, communication through MCS 122 may be converted for communication using a custom communication protocol. In some embodiments, specific UIs may correspond to a custom client in an application.

[0070] MCS 122 may include one or more callable interfaces, e.g., an API. Callable interfaces associated with MCS 122 may enable an application on a mobile computing device to communicate requests to MCS 122. Callable interfaces associated with MCS 122 may support a common or standard interface, which may allow requests including their parameters to be received from apps according to a standardized protocol, architectural style, and/or format (e.g., a REST protocol). Callable interfaces associated with MCS 122 may be configurable by a user of any one of computing devices 104, 106, or 108. Callable interfaces associated with MCS 122 may receive requests for services according to a communication protocol. Device application developers can connect to MCS 122 for their custom applications. In some embodiments, a callable interface associated with MCS 122 may be configured by the same person that develops an app, such that the person can implement a custom application to communicate with MCS 122.

[0071] Callable interfaces associated with MCS 122 may further enable enterprise computer systems 126 to communicate with MCS 122 according to a standardized protocol or format. Similar to application developers, those who manage enterprise computer systems can implement code (e.g., an agent system) that is configured to communicate with MCS 122 via one or more callable interfaces. Callable interfaces associated with MCS 122 may be implemented based on a type of a computing device, a type of enterprise computer systems, an app, an agent system, a service, a protocol, or other criterion. In some embodiments, callable interfaces associated with MCS 122 may support requests for services including authentication, compression, encryption, pagination with cursors, client-based throttling, non-repudiation, logging, and metrics collection. In some embodiments, callable interfaces associated with MCS 122 may be implemented for custom business-related services, such as authentication, policy enforcement, caching of responses, throttling of calls to MCS 122, translation between asynchronous and synchronous patterns, logging of calls to underlying services, or combinations thereof. In some embodiments, callable interfaces associated with MCS 122 may enable users to load custom code for implementation by cloud infrastructure system 102. The custom code may implement one or more callable interfaces associated with MCS 122 for cloud infrastructure system 102, which can enable users to access custom services or other enterprise computer systems.

[0072] Protocol translators associated with MCS 122 may process a message to determine a communication protocol for a message and/or to convert a message to a communication protocol for a destination. Protocol translators associated with MCS 122 may convert a request received from client computing devices 104, 106, or 108. The request may be converted from a format of a communication protocol supported by client computing devices 104, 106, or 108 to a format of a communication protocol supported by a service provided by cloud infrastructure service 102 or enterprise computer systems 126. Protocol translators associated with MCS 122 may convert a response received from a service provided by cloud infrastructure service 102 or enterprise computer systems 126. A response may be converted from a format of a communication protocol supported by a service provided by cloud infrastructure service 102 or enterprise computer systems 126 to a format of a communication protocol supported by client computing devices 104, 106, or 108.

[0073] Security services associated with MCS 122 may manage security authentication for requests received from any of client computing devices 104, 106, or 108. Security services associated with MCS 122 may protect the integrity of customer processes and enterprise data. To prevent system or data from being compromised, security authentication may occur when a request is received from client computing devices 104, 106, or 108. Security authentication may be performed before a request is dispatched for processing by cloud infrastructure system 102. The security authentication determined for a user may enable a user associated with a mobile computing device to have authorization to request services via MCS 122. The security authentication may reduce efforts for a user to authenticate for different requests and/or services requested via MCS 122. Security services associated with MCS 122 may be implemented as one or more functional blocks or modules configured to perform various operations authenticating security of a request.

[0074] Authentication services associated with MCS 122 may manage security authentication for requests received from client computing devices 104, 106, or 108. Authentication services associated with MCS 122 may determine security authentication for a user associated with a computing device that sends a request to MCS 122. Security authentication may be determined based on a time period, which

may be tied to operation of an application (e.g., launching an application), a request, a computing device, an enterprise computer system, other criterion related to a request, or combinations thereof. Security authentication may be verified and granted for any one of the following, such as an individual request, one or more enterprise computer systems, a particular service, a type of service, a user, a computing device, other criterion for determining security authentication, or combinations thereof. In some embodiments, cloud infrastructure system 102 may store authentication information of users received from enterprise computer systems or authentication systems supporting enterprise computer systems. Cloud infrastructure system 102 may determine authentication by performing a lookup function to determine whether an identity of a user associated with a request has authority to make such a request. The stored authentication information may include information such as the type of requests, functions, enterprise computer systems, enterprise data, or the like that a user may be authorized to access. In some embodiments, infrastructure system 102 may initiate communication with a requesting computing device to determine authentication.

[0075] In some embodiments, security authentication may be determined based on a role associated with a user requesting a service. The role may be associated with a user requesting access to MCS 122. In some embodiments, a user may request services as a subscriber or tenant of MCS 122 who may be granted access to resources and/or services provided by MCS 122. Authentication may correspond to a user's subscription to MCS 122, such that a user may be authorized to request services via MCS 122 as a subscriber. In some embodiments, the subscription may be limited to a particular set of resources provided by MCS 122. Security authentication may be based on the resources and/or services accessible to the user of MCS 122. In some embodiments, a request may be provisioned a template during execution called a "runtime environment." The runtime environment may be associated with resources that are allocated for a request, a user, or a device.

[0076] In some embodiments, authentication services associated with MCS 122 may request an identity management system to determine security authentication for the user. The identity management system may be implemented by cloud infrastructure system 102 (e.g., as identity management 114) or by another

computer system that is external to cloud infrastructure system 102. Identity management 116 may determine security authentication of the user based on the user's role or subscription for accessing MCS 122. The role or subscription may be assigned privileges and/or entitlements with respect to an enterprise computer system, a service provided by an enterprise computer system, a function or feature of an enterprise computer system, other criterion for controlling access to an enterprise computer system, or combinations thereof.

ADF

[0077] Various different ADFs 124 may be provided in cloud infrastructure system 102. ADFs 124 provide the infrastructure code to implement agile SOA based applications. ADFs 124 further provide a visual and declarative approach to development through one or more development tools (e.g., "Oracle JDeveloper 11g" development tool). One or more frameworks provided by ADFs 124 may implement an MVC design pattern. Such frameworks offer an integrated solution that covers all the layers of the MVC architecture with solutions to such areas as Object/Relational mapping, data persistence, reusable controller layer, rich Web UI framework, data binding to UI, security and customization. Extending beyond the core Web based MVC approach, such frameworks also integrate with the Oracle SOA and WebCenter Portal frameworks simplifying the creation of complete composite applications.

[0078] In certain embodiments, ADFs 124 make it easy to develop agile applications that expose data as services by coupling a service interface to built-in business services provided by cloud infrastructure system 102. This separation of business service implementation details is performed in ADFs 124 via metadata. Use of this metadata-driven architecture enables application developers to focus on the business logic and user experience, rather than the details of how services are accessed. In certain embodiments, ADFs 124 store implementation details of services in metadata in a model layer. This enables developers to exchange services without modifying the UI, making the application extremely agile. Additionally, the developer creating the UI does not need to bother with business service access details. Instead, developers can focus on developing the application interface and interaction logic. Creating the user experience can be as simple as dragging-and-dropping the desired business services onto a visual page designer

and indicating what type of component should represent that data.

[0079] In various embodiments, developers interact with ADFs 124 to create modules forming enterprise applications. The enterprise applications can be executed within the context of cloud infrastructure system 102. In various embodiments, developers interact with ADFs 124 to create modules forming mobile applications. The mobile applications can be executed within the context of cloud infrastructure system 102. Features of the present invention described below may be implemented using any desired combination of programming language and application development framework as will be apparent to one skilled in the relevant arts by reading the disclosure provided herein.

[0080] One or more frameworks provided by ADFs 124 may be embodied as Oracle ADF in one example. Accordingly, a framework in ADFs 124 can be based on an MVC design pattern. An MVC application is separated into: 1) a model layer that handles interaction with data-sources and runs the business logic, 2) a view layer that handles the application UI, and 3) a controller that manages the application flow and acts as the interface between the Model and the View layers. Separating applications into these three layers simplifies maintenance and reuse of components across applications. The independence of each layer from the others results in a loosely coupled, SOA.

[0081] In various embodiments, ADFs 124 provide tools and resources allowing developers to create an application in the form of multiple layers, each layer containing code modules/files implementing desired logic according to pre-defined specification. Thus, in one embodiment, ADFS 124 enables the application to be developed as four layers: a view layer containing code modules/files that provide the UI of the application, a controller layer containing code modules that control the flow of the application, a model layer containing data/code modules that provide an abstraction layer for the underlying data, and a business services layer containing code modules that provide access to data from various sources and handles business logic.

[0082] In certain embodiments, ADFs 124 let developers choose the technology they prefer to use when implementing each of the layers. Enterprise JavaBean ("EJB"), Web Services, JavaBeans, JPA/EclipseLink/TopLink objects, and many others can all be used as Business Services for ADFs 124. View layers can

include Web based interfaces implemented with Java Server Faces (“JSF”), Desktop Swing applications and Microsoft Office front ends, as well as interfaces for mobile devices.

[0083] In one aspect, the view layer represents the UI of the application being developed. The view layer can include desktop, mobile, and browser-based views, each of which provides all or a portion of the UI and is accessible in a variety of manners corresponding to view type. For example, Web pages may be sent by the application in response to receiving client requests containing corresponding URLs. The Web pages may then be displayed by a browser on a display unit (not shown) associated with a requesting client system, thereby enabling users of the requesting client system to interact with the enterprise application. ADFs 124 support multi-channel access to business services allowing reuse of business services and access from a Web client, a client-server swing desktop-based application, Microsoft Excel spreadsheets, mobile devices such as a smart-phone, or the like.

[0084] The code files/modules forming the view layer (such as Web pages) may be implemented using one or more of hypertext markup language (“HTML”), Java server pages (“JSP”), and JSF. Alternatively, the UI may be implemented using Java components such as Swing, and/or XML. As further noted, the UI may leverage a user’s experience and familiarity with desktop applications, such as Word and Excel by Microsoft.

[0085] As noted above, the relevant user-developed code/data modules are provided in each of the layers. However, each layer typically contains other pre-defined code/data modules provided by ADFs 124. Some of the pre-defined modules may be used during development, for example, as templates for developing the Web pages, for including desired functionality in the developed code etc. Other pre-defined modules (such as a URL rewriting module) may be deployed along with the developed application and may provide additional functionalities (mapping of requested URLs to internal names) to the user during execution of the enterprise application.

[0086] A controller layer contains code modules/files that control the flow of the application. Each controller object contains software instructions and/or data implemented according to a desired manner of presenting information in the view layer. The desired manner may include the specific Web pages to be displayed

when links in another Web page are clicked/ selected by the user, the page to be displayed when errors occur during execution, indicating the specific data to be stored/retrieved, etc.

[0087] In one aspect, the controller layer manages the application's flow and handles user input. For example, when a Search button is clicked on a page, the controller determines what action to perform (do a search) and where to navigate to (the results page). There are two controller options for Web-based applications in JDeveloper: the standard JSF controller or the ADF Controller that extends the JSF controller functionality. Whichever controller is used, application flow is typically designed by laying out pages and navigation rules on a diagram. An application's flow can be broken into smaller, reusable task flows; include non-visual components such as method calls and decision points in a flow; and create "page fragment" flows that run inside a region of a single containing page.

[0088] The code modules/files forming the controller layer are often implemented as Java servlets receiving the client requests and sending desired Web pages as corresponding responses. Controller objects may also be implemented, for example, as Apache Jakarta Struts controllers or according to the JSF standard.

[0089] A model layer contains data/code modules that connect various business services to the objects that use them in the other layers, such as to the controller objects discussed above or directly to desktop applications. Each abstract data object of the model layer provides a corresponding interface that can be used to access any type of business service executing in an underlying business service layer. The data objects may abstract the business service implementation details of a service from a client and/or expose data control methods/attributes to view components, thus providing a separation of the view and data layers.

[0090] In one aspect, the model layer consists of two components, data controls and data bindings, which utilize metadata files to define the interface. Data controls abstract the business service implementation details from clients. Data bindings expose data control methods and attributes to UI components, providing a clean separation of the view and model. Due to the metadata architecture of the model layer, developers get the same development experience when binding any type of Business Service layer implementation to the View and Controller layers.

[0091] In certain embodiments, ADFs 124 emphasize the use of the

declarative programming paradigm throughout the development process to allow users to focus on the logic of application creation without having to get into implementation details. At a high level, the development process for a Fusion Web application usually involves creating an application workspace. Using a wizard, libraries and configuration needed for technologies selected by a developer are automatically added and an application is structured into projects with packages and directories.

[0092] By modeling database objects, an online database or offline replica of any database can be created, definitions edited, and schemas updated. Using a unified modeling language (“UML”) modeler, use cases can then be created for the application. Application control and navigation can also be designed. Diagrammers can be used to visually determine the flow of application control and navigation. Then, an underlying XML file describing the flow can be automatically created. A resource library can be used to allow a developer to view and use imported libraries by simply dragging and dropping them into the application. From database tables, entity objects can be created using wizards or dialogs. From those entity objects, view objects are created to be used by pages in the application. Validation rules and other types of business logic can be implemented.

[0093] In this example, a business services layer manages interaction with a data persistence layer. It provides such services as data persistence, object/relational mapping, transaction management, and business logic execution. The business services layer can be implemented in any of the following options: as simple Java classes, EJB, Web services, JPA objects, and Oracle ADF Business Components. In addition, data can be consumed directly from files (XML or CSV) as well as REST. Thus, each business service manages interaction with a corresponding data persistence layer, and also provides such services as object/relational mapping, transaction management, business logic execution, etc. The business services layer may be implemented using one or more of simple Java classes, Enterprise Java Beans, Web services, etc.

[0094] Business components represent a business service implemented using, for example, “Oracle ADF Business Components” from Oracle Corp., to provide interaction with databases, Web services, legacy systems, application servers, and the like. In one embodiment, business components of the business

services layer contain a mixture of application modules, view/query objects, and entity objects, which cooperate to provide the business service implementation. An application module can be a transactional component/code module that UI clients communicate with for working with application/transaction data. The application module may provide an updatable data model and also procedures/functions (commonly referred to as service methods) related to user transactions.

[0095] An entity object may represent a corresponding row in a database table and simplify the manipulation (update, deletion, etc.) of the data stored in the corresponding row. An entity object often encapsulates business logic for the corresponding row to ensure that the desired business rules are consistently enforced. An entity object may also be associated with other entity objects to reflect relationships existing between rows stored in the underlying database.

[0096] Fig. 2 shows a block diagram of a computing environment 200 for facilitating communication between a mobile computing device and enterprise computer systems according to some embodiments of the present invention. For purposes of illustration, various examples are provided herein to describe techniques for enabling a mobile computing device (e.g., computing device 202) to communicate with one or more enterprise computer systems, such as a cloud enterprise computer system 240 (e.g., "serviceprovider.com") and an on-premises enterprise computer system 250. Such communications may be to exchange or transfer enterprise data, request services provided by an enterprise computer system, communicate messages, or combinations thereof.

[0097] Messages may include service invocation messages, result messages, request messages, other messages communicated internally, other messages communicated between a computing device and an enterprise computer system, or combinations thereof. A message may include a message type (e.g., a type value from a set of shared type constants), a correlation id (e.g., an id used to correlate this message with one or more other messages), priority information to support for priority based message queues, timeout, sensitivity indicator to support message data isolation, message source (e.g., a uniform resource identifier of a sender), a message destination (e.g., a uniform resource identifier that uniquely identifies the destination, a request context (e.g., request information from dispatcher), and/or a message payload. The payload may have different attributes depending upon the

type of message that is being sent, such as parameter data and result data.

[0098] Enterprise data as described herein may include data received from an enterprise computer system, data sent to an enterprise computer system, data processed by an enterprise computer system, or combinations thereof. The enterprise data may be distinguishable from data for consumer applications and/or services. In some embodiments, for example, enterprise data may change based on application or use of the enterprise data, whereas data for consumer applications (e.g., consumer data) may remain static through use. In certain embodiments, enterprise data may include or be associated with rules that indicate criteria for storing, using, and/or managing the enterprise data. For example, enterprise data may be associated with policy information that indicates one or more policies for storing, using, and/or managing the enterprise data. In certain embodiments, policy information may be included in enterprise data. In certain embodiments, enterprise data may include data processed, stored, used, or communicated by an application or a service executing in an enterprise computer system. For example, enterprise data may include business data (e.g., business objects) such as JSON formatted data from enterprise applications, structured data (e.g., key value pairs), unstructured data (e.g., internal data processed or used by an application, data in JSON format, social posts, conversation streams, activity feeds, etc.), binary large objects (“BLOBs”), documents, system folders (e.g., application related folders in a sandbox environment), data using REST techniques (referred to herein as “RESTful data”) (e.g., synchronization data made available by REST endpoints), system data, configuration data, synchronization data, or combinations thereof. In some embodiments, enterprise data may include REST-formatted enterprise data. REST-formatted enterprise data may include RESTful data. REST-formatted data may include data formatted according to REST techniques implemented by an enterprise computer system. Configuration or synchronization data may include data used for synchronization of enterprise data, such as versions, history, integration data, etc. Documents in enterprise data may include XML files, visual assets, configuration files, media assets, etc. A BLOB may include a collection of binary data stored as a single entity in a database management system, such as an image, multimedia object, or executable code, or as otherwise known in the art.

[0099] An enterprise computer system may include various computing

systems that are configured to operate for an entity or an enterprise. For example, an enterprise computer system may include one or more computer systems, such as an enterprise server computer (e.g., a back-end server computer), to handle requests for services. An enterprise computer system may include applications and/or services, which can process and/or operate using enterprise data. For example, enterprise computer system 250 may provide one or more services and/or applications for managing or operating an enterprise. Services may include, without restriction, customer relationship management ("CRM"), human capital management ("HCM"), human resource ("HR") management, supply chain management, enterprise communication, email communication, business services, other enterprise management services or applications, or combinations thereof. Enterprise computer system 250 may include one or more computer systems dedicated to providing one or more services. In some embodiments, each different computer system providing a service may be located on-premise of an enterprise or may be located remotely from an enterprise. In some embodiments, multiple different computer systems supporting different services may be situated in a single geographical location, such as on-premises of an enterprise. In the example shown in Fig. 2, on-premises enterprise computer system 250 may include an HR system 254 and a CRM system 256, both of which may be located on-premises of an enterprise. In some embodiments, enterprise computer system 250 may include or implement an agent system 252 to facilitate or handle communication between cloud computer system 210 and one or more enterprise systems 254, 256. Enterprise computer systems, such as cloud enterprise computer system 240 and on-premises enterprise computer system 250 are described below in further detail.

[00100] The computer environment 200 may include MCS 212 implemented to operate as a secure intermediary computing environment that may facilitate communication between the computing device 202 and one or more enterprise computer systems because computing device 202 may not be configured to communicate with such enterprise computer systems. For example, some enterprise computer systems may be supported by legacy or back-end computer systems. Such systems may be configured to operate using different communication and/or security protocols. The protocols supported by such enterprise computer systems may be different from those supported by mobile computing devices. MCS

212 may support communication with different types of mobile computing devices. As such, MCS 212 may implement techniques to facilitate communication between enterprise computer systems and mobile computing devices to enable them to communicate with each other despite their incompatibilities in communication, such as differences between formats or communication protocols. For example, MCS 212 may translate communication protocols between mobile computing devices and enterprise computer systems.

[00101] Cloud computer system 210 may support MCS 212. Cloud computer system 210 may be implemented using hardware, software, firmware, or combinations thereof. For example, cloud computer system 210 may include one or more computing devices, such as a server computer. Cloud computer system 210 may include one or more memory storage devices and one or more processors. A memory storage device can be accessible to the processor(s) and can include instructions stored thereon which, when executed by the processor(s), cause the processor(s) to implement one or more operations disclosed herein. In some embodiments, the memory storage devices may operate as local storage (e.g., cache). Cloud computer system 210 may include different kinds of operating systems. A memory storage device may be accessible to the processor(s) and may include instructions stored thereon which, when executed by the processor(s), cause the processor(s) to implement one or more operations, methods, or processes disclosed herein. The memory storage may operate as local storage. Local storage may be implemented using any type of persistent storage device, such as a memory storage device or other computer readable storage medium. In some embodiments, local storage may include or implement one or more databases (e.g., a document database, a relational database, or other type of database), one or more file stores, one or more file systems, or combinations thereof. The local storage may store enterprise data.

[00102] In certain embodiments, cloud computer system 210 may include one or more data stores, such as a metadata repository 224, diagnostics store 226, and an analytics store 228. The data stores 224, 226, 228 may be accessible by any component in cloud computer system 210.

[00103] Metadata repository 224 may store all the metadata associated with MCS 212. This information may be composed of both run-time and design-time

data, each having their own requirements on availability and performance. A tenant or subscriber of MCS 212 may have any number of applications. Each application may be versioned and may have an associated zero or more versioned resource APIs and zero or more versioned services implementations those resource API contracts. These entities are what the run-time uses to map virtual requests (mAPIs) to the concrete service implementation (service). This mapping provides a mobile developer with the luxury of not having to know the actual implementation service when she designs and builds her application. As well as not requiring her to have to republish a new application on every service bug fix. Metadata repository 224 may store one or more callable interfaces, which may be invoked by a computing device (e.g., computing device 202). The callable interfaces may be customizable by a user (e.g., a developer) of an application to facilitate communication with MCS 212. Metadata repository 224 may store metadata corresponding to one or more configurations of a callable interface. Metadata repository 224 may be configured to store metadata for implementing a callable interface. The callable interface may be implemented to translate between a one format, protocol, or architectural style for communication and another format, protocol, or architectural style for communication. Metadata repository 224 may be modifiable by an authenticated user via the external network.

[00104] Diagnostics store 226 may store diagnostics information about processing occurring in MCS 212. Diagnostics store 226 may store messages communicated via MCS 212 and log information. Analytics store 228 may store logging and analytics data captured during processing in the system.

[00105] On behalf of MCS 212, cloud computer system 210 may utilize its computing resources to enable execution of custom code 216 (e.g., operations, applications, methods, functions, routines, or the like). Computing resources may be allocated for use with respect to a particular user associated as a subscriber or tenant to MCS 212. Resources may be allocated with respect to a user, a device, an application, or other criterion related to a subscriber. MCS 212 may be scaled in or out, depending on the demand of mobile computing devices seeking to communicate with enterprise computer systems. MCS 212 can be configured such that it is elastic to handle surges and temporary periods of higher than normal traffic between mobile computing devices and enterprise computer systems. In some embodiments, MCS

212 may include elements that support scalability such that components may be added or replaced to satisfy demand in communication.

[00106] Computing device 202 may communicate (e.g., send a request message) with MCS 212 to request service provided by an enterprise computer system. Computing device 202 (e.g., a mobile computing device) may be implemented using hardware, firmware, software, or combinations thereof. Computing device 202 may communicate with enterprise computer systems 240, 250 via MCS 212. Computing device 202 may include or may be implemented as an endpoint device, a PDA, a tablet computer, a laptop computer, a mobile computing device, a desktop computer, a wearable computer, a pager, etc. Computing device 202 may include one or more memory storage devices and one or more processors. Computing device 202 may include different kinds of operating systems. A memory storage device may be accessible to the processor(s) and may include instructions stored thereon which, when executed by the processor(s), cause the processor(s) to implement one or more operations, methods, or processes disclosed herein. The memory storage may operate as local storage. Local storage may be implemented using any type of persistent storage device, such as a memory storage device or other computer readable storage medium. In some embodiments, local storage may include or implement one or more databases (e.g., a document database, a relational database, or other type of database), one or more file stores, one or more file systems, or combinations thereof. The local storage may store enterprise data.

[00107] In various embodiments, computing device 202 may be configured to execute and operate one or more applications such as a web browser, a client application, a proprietary client application, or the like. The applications can include specific applications configured for enterprise data and/or services provided by an enterprise computer system. Client applications may be accessible or operated via one or more network(s). Applications may include a graphical UI (“GUI”) for operating the application.

[00108] Computing device 202 may communicate with MCS 212 via one or more communication networks using wireless communication. Examples of communication networks may include a mobile network, a wireless network, a cellular network, a LAN, a wide area network (“WAN”), other wireless communication networks, or combinations thereof. In certain embodiments, computing device 202

may establish a communication connection 214 with MCS 212 using a custom communication protocol (e.g., a custom protocol). Connection 214 may be established with MCS 212 through cloud computer system 210. The custom protocol may be an HTTP-based protocol. By utilizing a custom communication protocol, computing device 202 may operate on any computing device platform to communicate with cloud computer system 210.

[00109] Computing device 202 may communicate with cloud computer system 210 through one or more callable interfaces, e.g., APIs. A callable interface may be implemented on computing device 202. The callable interface may be implemented for custom applications that enable those applications to communicate with MCS 212. In some embodiments, a callable interface may be developed for MCS 212. The callable interface may enable applications to communicate with MCS 212 without having to adapt to differences in protocols (e.g., communication or development protocols) and/or architectural styles or formats.

[00110] MCS 212 may be protected by one or more firewalls 204, 230 to provide a secure environment to process requests and execute custom code 216. Communication between computing device 202 and MCS 212 may be separated by an external communication firewall 204. Firewall 204 may be connected with cloud computer system 210 to facilitate secure access to MCS 212. Firewall 204 may permit communication of messages between cloud computer system 210 and computing devices (e.g., computing device 202). Such messages (e.g., HTTP messages or REST messages) may conform to a communication protocol (e.g., HTTP or REST), which may be supported by a callable interface. In another example, a message between cloud computer system 210 and computing device 202 may conform to a communication protocol such as SPDY. MCS 212 may manage firewall 230 to secure communication between cloud computer system 210 and enterprise computer systems 240, 250. Firewall 230 may permit communication of messages between cloud computer system 210 and computing devices (e.g., computing device 202). Such messages (e.g., SPDY messages, HTTP messages or REST messages) may conform to a communication protocol (e.g., SPDY, HTTP, or REST). Communication between computing device 202 and enterprise computer systems 240, 250 may be two-way via MCS 212.

[00111] Because communication with computing device 202 and enterprise

computer systems 240, 250 may occur via an unsecure, public network, firewalls 204, 230 provide an added layer of protection for communications to and from MCS 212. Firewalls 204, 230 may enable MCS 212 to distinguish its internal network from an external network connecting computing device 202 and enterprise computer systems 240, 250. In some embodiments, firewalls 204, 230, although shown as two distinct firewalls, may be implemented as a single firewall that encapsulates MCS 212.

[00112] Cloud computer system 210 may further operate as an intermediary computing environment by communicating with enterprise computer systems, some of which may have different communication protocols. Such communication protocols may be custom or specific to an application or service in communication with cloud computer system 210. Further, cloud computer system 210 may communicate with an enterprise computer system to provide enterprise services and/or to exchange enterprise data according to a format supported by the enterprise computer system. Cloud computer system 210 may maintain local storage (e.g., local cache) of enterprise data and may use the local storage to manage synchronization of the enterprise data between mobile computing devices and enterprise computer systems 240, 250.

[00113] Computing device 202 may communicate (e.g., send a request message) with MCS 212 to request service provided by an enterprise computer system. Requests that are received through firewall 204 may be processed first by security service 232. Security service 232 may manage security authentication for a user associated with a request. Thus, a cloud computer system may provide technical advantages that include providing security mechanisms described herein which may protect the integrity of customer communications and enterprise data. Technical advantages of cloud computer system may include preventing or reducing compromised communications and/or data from being compromised, authentication may occur initially, restricting access to only those who have the required credentials. Technical advantages of cloud computer system may include the services and service invocation flow being structured such that as requests come in they may only be able to access services for which they are authorized. By decoupling authorization from the rest of the system processing, another technical advantage may include the task of authorizing “what can be done by whom” being

delegated to a dedicated provisioned security subsystem (e.g., an identity management system) that may be expanded to support whatever additional custom security measures are required by a specific corporate customer. In some embodiments, security authentication may be determined for a request, a session, a user, a device, other criterion related to the user, or combinations thereof. Security authentication may be performed for each request that is received. In some embodiments, security service 232 may determine authentication based on a previous verification of a request. Security authentication may be determined for a user or a device such that requests to different enterprise computer systems 240, 250 may be authenticated based on a single verification of security.

[00114] Further technical advantages of the invention may include a cloud computer system enabling a computing device to communicate with various enterprise computer systems, some of which may be implemented differently. For example, a computing device 202, cloud computer system 210, and enterprise computer system 250 may be located at different geographical locations, physically separated from each other. Therefore, computing device 202 can communicate with enterprise computer system 250 regardless of their location. Technical advantages may include a cloud computer system enabling a computing device to communicate requests for services to enterprise computer systems, which may support one or more distinct security protocols. In some cases, an enterprise computer system may be supported by a back-end system that is not easily adaptable to a different security protocol. In some cases, it may be desirable for developers of applications to be able to implement an application to be able to request services without knowledge of such security protocols. It may be equally desirable for a user (e.g., an administrator or an architect) of an enterprise computer system to be able to receive requests without accommodating for different types of applications, security protocols, and standards. Technical advantages may enable such desires to be met by implementation of a cloud computer system, as described herein, which can handle security authentication, such that requests can meet the security measures of different enterprise computer systems that are being requested.

[00115] In some embodiments, security service 232 may determine a security protocol for a requested enterprise computer system and accordingly generate a security token according to such security protocol. The security token may be

passed along with a request to an enterprise computer system to enable that enterprise computer system to verify authentication based on the generated security token. Enterprise computer systems may support different security protocols. A security protocol may be a standard by which security is determined. Security may be verified based on a security token that is generated by security service 232. Security service 232 may determine a security protocol for an enterprise computer system identified for a request. In some embodiments, an enterprise computer system 250 may have an agent system 252, which may be configured or implemented according to a custom or specific security protocol supported by MCS 212. As such, MCS 212 may generate a security token according to such custom security protocol.

[00116] Cloud computer system 210 may include, implement, and/or communicate with one or more load balancer systems 206, 208. Upon determining security authentication, cloud computer system 210 may request any one of load balancer systems 206, 208 to examine a request that it receives and to detect which service the request is directed to. MCS 212 may be configured with load balancers 206, 208 and updated with resources that get started up, so that when a request comes in, load balancers 206, 208 can balance a requested load across the different resources.

[00117] Cloud computer system 210 may include a dispatcher 218 that may handle requests and dispatch them to the appropriate service. A request may be routed to an appropriate service upon dispatch. In some embodiments, a service itself may route an internal request to another internal service in MCS 212 or in an enterprise computer system. In some embodiments, dispatcher 218 may resolve a request to determine its destination based on a location (e.g., an address) of a destination identified in a URI and/or URL of the request. Dispatcher 218 may parse a request and its header to extract one or more of the following information: tenant identifier, service identifier, application name, application version, request resource, operation and parameters, etc. Dispatcher 218 can use the parsed information to perform a lookup in metadata repository 224. Dispatcher 218 may retrieve a corresponding application metadata. Dispatcher 218 may determine the target service based on the requested resource and the mappings in the metadata. While initially a very basic mapping, the metadata can be enhanced to provide for more

sophisticated, rules-based dispatching. Dispatcher 218 may perform any dispatcher-specific logging, metrics gathering, etc. Dispatcher 218 may then perform initial authorization according to the application metadata. Dispatcher 218 may format the inbound request and any other necessary information and place the message on routing bus 220 for further processing. Dispatcher 218 may place a request on a queue and await the corresponding response. Dispatcher 218 may process responses received from routing bus 220 and return a response to computing device 202.

[00118] In addition to handling the dispatching for external requests, dispatcher 218 may also play a role in dispatching internal requests. Such internal requests can come in the form of composite services or custom code invocations to services. In both cases, the caller could use a logical service name as defined within the application. Dispatcher 218 may use the current execution context to determine the application and use that logical name to determine the appropriate service to invoke.

[00119] Cloud computer system 210 may include a routing bus 220 to manage deliver of messages to destinations registered with routing bus 220. Routing bus 220 may operate as a central system for managing communications in cloud service 212. Data communicated through routing bus 220 may be processed to capture and store the data. Routing bus 220 may provide a framework so that additional centralized services (additional authorization, debugging, etc.) can be plugged in easily as necessary. Data captured by routing bus 220 may be stored in diagnostics store 226 and/or analytics store 228.

[00120] Routing bus 220 may route messages to one or more destinations. In some embodiments, a message may include a request to execute custom code 216. In such embodiments, routing bus 220 may request 234 custom code 216 to be invoked. In some embodiments, routing bus 220 may pass on a request to a destination enterprise computer system identified by information in a request. Routing bus 220 may request 236 an adaptor interface 222 to perform translations, if necessary, to pass a request to an enterprise computer system, e.g., enterprise computer system 240 or enterprise computer system 250.

[00121] In certain embodiments, cloud computer system 210 may include or implement adaptor interface 222 to translate or convert a message to a protocol

supported by a receiving enterprise computer system. Adaptor interface 222 may establish separate communication connections with each of enterprise computer systems 240, 250. Cloud computer system 210 may be configured to communicate with enterprise computer systems 240, 250 via one or more networks (not shown). Examples of communication networks may include the Internet, a mobile network, a public network, a wireless network, a cellular network, a LAN, a WAN, other communication networks, or combinations thereof. In certain embodiments, communication connections may be high-speed communication connections facilitated using high-speed communication trunks. Communication with an enterprise computer system 240, 250 may pass through firewall 230 which ensures that communication with an external network is secure to prevent unauthorized access to MCS 212 via such communications.

[00122] In some embodiments, cloud computer system 210 may facilitate notifications to a user of computing device 202. Cloud computer system 210 may include an alert management service that supports stateful interaction with a user, for example to deliver an alert based on user preferences through one or more channels, wait for a response, and take action based on the response. Responses to an alert sent on one channel may be received through another channel, which the service needs to be able to handle. The platform may come with built-in state models for popular interaction patterns and be extensible with new state models. Some alert channels may include known communication resources, either one-way or two-way. Examples include SMS, Twitter®, push notifications, and Google Cloud Messaging®.

[00123] In some embodiments, cloud computer system 210 may enable computing device to access and/or request one or more services, such as an object store service, database service, access web services, social services, resource services, or combinations thereof.

[00124] Cloud computer system 210 may provide an object store service that may provide a storage facility for BLOBs. The basic unit of storage can be text, with read and write operations. A basic query facility for JSON objects may also be offered.

[00125] Cloud computer system 210 may provide a database service to allow for connectivity to hosted databases for performing queries or writes. Required

parameterization may require the full connection string for the database, the SQL string or stored procedure to execute, any parameters and possibly credentials. The necessary information can be provided at run time or be pre-configured in the application metadata.

[00126] Cloud computer system 210 may provide access to web services such as SOAP web services. Cloud computer system 210 may provide access to REST services, such as connectivity to arbitrary REST resources.

[00127] Cloud computer system 210 may provide access to social services that may provide basic integration with many of the popular social sites such as Facebook®, Twitter®, etc. These services may allow for third party authentication using the user's credentials from those sites as well as access to their services. Examples include sending a tweet or updating your status.

[00128] Cloud computer system 210 may provide a public cloud service to enable a user to simplify and optimize communication. For example, a service developer may use the generic web service of MCS 212 to talk to a resource hosted using cloud computer system's 210 cloud service.

[00129] A cloud computer system, such as one described herein, may enable mobile computing devices to communicate with enterprise computer system despite differences in computing resources. A cloud computer system may be equipped with more resources and a faster, more reliable connection to enterprise computer systems to communicate frequently to receive enterprise data. The cloud computer system may manage and coordinate requests for services from enterprise computer systems. By translating requests to a protocol supported by a recipient of a message, the cloud computer system reduces a burden on developers to configure applications for communication with different types of back-end computer systems. Enterprises are able to maintain their back-end systems without having to accommodate advances or changes in communication protocols supported for mobile devices. Different enterprise computer systems may support different security protocols based on a type of requests processed and services provided. By managing security authentication in a centralized manner for access to different enterprise computer systems, enterprise computer systems do not need to adapt to differences in security protocols. By authenticating a user of the cloud computer system, processing requests can become more efficient as authentication may not

be performed in every instance.

[00130] In some embodiments, an application may be deployed under a mobile application framework (“MAF”) that provides built in security to control access to the application and ensure encryption of sensitive data, such as Oracle MAF from Oracle Corp. MAF is a hybrid mobile architecture that uses HTML5 and Cascading Style Sheets (“CSS”) (to render the UI in the web view), Java (for the application business logic), and Apache Cordova (to access device features such as GPS activities and e-mail). Because MAF uses these cross-platform technologies, the same application can be built for both Android and iOS devices without having to use any platform-specific tools. After an application is deployed to a device, it behaves as applications created using such platform-specific tools as Objective C or the Android SDK. Further, MAF allows for building the same application for smartphones or for tablets, thereby allowing reuse of business logic in the same application and targeting various types of devices, screen sizes, and capabilities.

[00131] Fig. 3 illustrates an example mobile application springboard 300 that includes a MAF application 302 called “WorkBetter” that is deployed as a “heavy” application (e.g., a mobile application that sits in a mobile device in the same manner as a normal iPhone “app” obtained from the App store). A MAF application may include one or more embedded applications added as application features. Such added application features are represented as icons within the main application's springboard or navigation bar. Application features are essentially the building blocks of such mobile application. Each application feature that is integrated into a MAF application performs a specific set of tasks. Application features can be grouped together to complement each other's functionality. For example, an application feature that provides customer contacts may be paired with one for product inventory. Because each application feature has its own class loader and web view, application features are independent of one another, thus a single MAF application can be assembled from application features created by several different development teams. Application features can also be reused in other MAF applications. A MAF application itself can be reused as the base for another application, allowing independent software vendors (“ISVs”) to create applications that can be configured by specific customers.

[00132] In addition to hybrid mobile applications that run locally on the device,

application features may be implemented as any of the following mobile application types, depending on the requirements of a mobile application and available resources:

- For mobile web applications hosted on a server, although the code can be portable between platforms, access to device features and local storage can be limited as these applications are governed by the device's browser.
- Native applications are authored in either Xcode or through the Android SDK and are therefore limited in terms of serving both platforms. Reuse of code is likewise limited.

[00133] MAF supports authentication and access control for refined security at the feature level in an application where a developer can specify an appropriate login server, e.g., a server running “Oracle Identity Management” and/or “Oracle WebLogic” with basic authentication, a server supporting OAuth protocols, etc. At runtime users are presented with login screens and appropriate tokens are accessible for further Web service calls. With MAF, developers can build single UIs that meet the needs of users with different privileges (e.g., show/hide components based on user role or privilege).

[00134] MAF enforces communication encryption using SSL/TLS (HTTP secure (“HTTPS”)), on-device encryption to keep credentials in an encrypted key store to be used for validation when supporting offline authentication, and SQLite database encryption by using the SQLite encryption extension. Encrypting an SQLite database for an application built with MAF may be performed via a configuration option when the application is developed. In some embodiments, MAF supports offline and online modes of operation for an application so a self-contained application can run on a mobile device in connected and disconnected modes. For data access/storage, such application may leverage a local encrypted SQLite database. The application may be built such that initial access to data is performed from remote servers through Web services, and the data is then stored in the local SQLite database for offline access. The data can be replicated and synchronized to the server when connectivity is available again. MAF also supports local storage of user authentication credentials to enable offline authentication/authorization to secured applications.

[00135] Figs. 3A and 3B illustrate an HR mobile application UI 304 in

accordance with an embodiment of the present invention. UI 304 may be provided upon opening an icon on a springboard such as mobile application springboard 300 of Fig. 3. In Fig. 3A, UI 304 includes various HR related information about an employee, such as picture, title, contact information, social networking information, performance/rating information, compensation information, manager, skills, location, etc. Fig. 3B indicates various sources where the information in UI 304 may be obtained from, such as services located on premise or in the cloud. For example, basic employee information may be obtained from on premise core HR services 306 such as PeopleSoft, Systems, Applications & Products ("SAP"), etc., while location information is obtained from a map service 308 such as Google. Similarly, performance information may be obtained from a talent management cloud service 310 such as TALEO, and social networking information 312 (e.g., Twitter, Facebook, LinkedIn, etc.) may be obtained from the web. In one embodiment, the information from these various sources is channeled through MCS 212 (see Fig. 2) before being sent down to the application on the mobile device 202 (see Fig. 2).

[00136] Fig. 4 is a block diagram of a MAF runtime architecture 400 according to an embodiment. Runtime architecture 400 includes a "thin" device native container 402 deployed to a mobile device 404. Runtime architecture 400 represents the MVC development approach which separates the presentation from the model layer and the controller logic. Device native container 402 allows a MAF application to function as a native application on different platforms (e.g., iOS, Android, etc.) by interacting with a local SQLite database 406 (via SQLite 408), mobile device services 426 (via Cordova APIs of Apache Cordova 410), and server-side resources 412 such as a Configuration Server 444, Server-Generated HTML 430, Push Services 448, and Web Services 440.

[00137] Device Services 426 are services and features that are native to device 404 such as camera, GPS, e-mail, etc. Configuration Server 444 is a server based on Web Distributed Authoring and Versioning ("WebDav") and hosting configuration files used by the application configuration services. WebDav is defined in, e.g., Internet Engineering Task Force ("IETF") Request for Comments ("RFC") 4918. Configuration Server 444 is delivered as a reference implementation. Any common WebDav services hosted on a Java 2 Platform, Enterprise Edition ("J2EE") server can be used for this purpose. Server-Generated HTML 430 includes Web

content hosted on remote servers and used for browser-based application features. Push Services 448 may include, for example, Apple Push Notification Services (“APNs”) and Google Cloud Messaging (“GCM”) push services that are notification providers that send notification events to MAF applications. Web Services 440 are, for example, remotely hosted SOAP-based web services.

[00138] Device native container 402 includes Web View 416 that uses a mobile device's web engine to display and process web-based content. In a MAF application, Web View 416 delivers the UI by rendering the application markup as HTML 5. The UI may be created for a mobile application feature by implementing any of the following content types: MAF Application Mobile XML (“AMX”) Views 420, Controller 422, local HTML 424, or server HTML 428, where MAF AMX Views 420, Controller 422, and local HTML 424 provide HTML5 and JavaScript presentation 418. Application features implemented from various content types can coexist within the same mobile application and can also interact with one another.

[00139] Applications whose contents are implemented as MAF AMX views 420 reside on device 404 and provide the most authentic device-native user experience, similar to an application authored in the language specific to the device's platform. MAF provides a set of code editors that enable a user to declaratively create a UI from components that are tailored to the form factors of mobile devices. These components can be used to create the page layout (e.g., list view) as well as input components (e.g., input fields). When a user develops MAF AMX views 420, they can leverage data controls which enable the user to declaratively create data-bound UI components and access a Web service and the services of a mobile device (e.g., camera, GPS, or e-mail). At runtime, a JavaScript engine in Web View 416 renders MAF AMX view definitions into HTML5 and JavaScript.

[00140] For applications whose contents are implemented as Controller 422, Controller 422 governs the flow between pages in the mobile application. Controller 422 enables a user to break an application's flow into smaller reusable task flows and include non-visual components such as method calls and decision points. In the embodiment of Fig. 4, Controller 422 is included in MAF AMX Views 420 and is called by MAF AMX Views 420 to, e.g., transition a page and/or activate actions. However, in alternative embodiments, Controller 422 may be implemented as a peer of MAF AMX Views 420.

[00141] For applications whose contents are implemented as local HTML 424, HTML pages run on the device as part of the MAF application. Local HTML files can access device-native features and services through Apache Cordova 410 and JavaScript APIs.

[00142] For applications whose contents are implemented as server HTML 428, the UI is delivered from server-generated Web pages (Server-Generated HTML 430) that can open within the application feature's Web View 416. Within the context of MAF, this content type is referred to as remote URL. The resources for these browser-based applications do not reside on the device 404. Instead, the UI, page flow logic, and business logic are delivered from a remote server.

[00143] When one of these remotely hosted Web applications is allowed to open within Web View 416, it can use Cordova JavaScript APIs to access any designated device-native feature or service such as the camera or GPS capabilities. When implementing an application using the remote URL content, a user can leverage an existing browser-based application that has been optimized for mobile use, or use one that has been written specifically for a specific type of mobile device. For applications that can run within the browsers on either desktops or tablets, the user can implement the remote URL content using applications created through rich client-based components such as those provided by "Oracle ADF Faces" from Oracle Corp. For applications specifically targeted to mobile phones, the remote URL content can be delivered from Web pages created using MAF. Not only can applications authored with MAF render on a variety of smartphones, but they can gracefully degrade to the reduced capabilities available on feature phones through UIs constructed with Apache Trinidad JSF components and dynamically selected style sheets. Because the content is served remotely, the application is available only as long as the server connection remains active.

[00144] Device native container 402 further includes Apache Cordova 410 that provides JavaScript APIs that integrate the device's native features and services into a mobile application. Although a user can access these APIs programmatically from Java code (or using JavaScript when implementing a MAF mobile application as local HTML 424), the user can add device integration declaratively when creating MAF AMX pages because MAF packages these APIs as data controls.

[00145] Device native container 402 further includes a Java Virtual Machine

("JVM") 432. Java provides a Java runtime environment for a MAF application. JVM 432 is implemented in device-native code, and is embedded (or compiled) into each instance of the MAF application as part of the native application binary. JVM 432 is based on the Java Platform, Micro Edition ("Java ME") Connected Device Configuration ("CDC") specification. In runtime architecture 400, JVM 432 includes business logic 434, model 436, and Java database connectivity ("JDBC") 438. Java enables business logic 434 in MAF applications. Managed Beans ("MBeans") are Java classes that can be created to extend the capabilities of MAF, such as providing additional business logic for processing data returned from the server. MBeans are executed by the embedded Java support, and conform to the Java ME CDC specifications. Model 436 includes the binding layer that connects the business logic components with the UI. In addition, the binding layer provides the execution logic to invoke web services 440 such as remotely hosted SOAP-based web services. These services are accessed through the Java layer (JVM 432). Application features authored in MAF AMX access SOAP-based data services through data controls. JDBC 438 is an API that enables the model layer to access the data in encrypted SQLite database 406 through Create, Read, Update, and Delete ("CRUD") operations.

[00146] Device native container 402 further includes Application Configuration 442 which refers to services that allow application configurations to be downloaded and refreshed, such as URL endpoints for a web service or a remote URL connection of a configuration server 444. Application configuration services download the configuration information from a server-side WebDav-based service.

[00147] Device native container 402 further includes module 446 that provides Credential Management, Single Sign-on ("SSO"), and Access Control. MAF handles user authentication and credential management through the "Oracle Access Management Mobile and Social" ("OAMMS") identity manager ("IDM") SDKs. MAF applications perform offline authentication, meaning that when users log in to the application while connected, MAF maintains the username and password locally on device 404, allowing users to continue access to the application even if the connection to the authentication server becomes unavailable. MAF encrypts the locally stored user information as well as the data stored in local SQLite database 406. After authenticating against the login server, a user can access all of the

application features secured by that connection. MAF also supports the concept of access control by restricting access to application features (or specific functions of application features) by applying user roles and privileges. For remotely served Web content, MAF uses whitelists to ensure that only the intended URIs can open within the application feature's web view 416 (and access the device features).

[00148] Device native container 402 also enables push notifications via a Push Handler 414 that communicates with push services 448 included in server side resources 412 and enables MAF applications to receive events from notification servers such as the iOS or Android notification servers. The Java layer (JVM 432) handles the notification processing.

[00149] In runtime architecture 400, device native container 402 interacts with encrypted SQLite Database 406 that is an embedded SQLite database that protects locally stored data and is called by the model layer using JDBC 438. The MAF application generates this lightweight, cross-platform relational database 406. Because database 406 is encrypted, it secures data if the device is lost or stolen. Only users who enter the correct user name and password can access the data in this database.

[00150] Fig. 5 is a block diagram of a system 500 for developing mobile applications in a mobile cloud infrastructure in accordance with embodiments of the present invention. In system 500, a user may use a user device 528 to develop and build applications in a cloud infrastructure 506 via a Web based tool. In one embodiment, the applications may be downloaded on a mobile device 526 over the air, thus obviating the need for an App store. The native applications talk to backends 504 created in MCS 502. In one embodiment, MAF runtime architecture 400 of Fig. 4 may be used to deliver an application to mobile device 526. In one embodiment, a declarative syntax of an application is deployed on mobile device 526 over the air, and the declarative syntax is interpreted on mobile device 526 by MAF runtime architecture 400 of Fig. 4.

[00151] Cloud infrastructure 506 includes MCS 502 that provides an admin UI 516 through which application development may be performed. MCS 502 further includes production environments 512 and testing environments 514 in which a mobile application may be developed and tested, respectively. These environments provide production/testing functionality by talking to corresponding backends 504 via

connectors. An application is first developed in testing environments 514. Once published, the application moves to production environments 512.

[00152] In one embodiment, a mobile application is developed by using user device 528 to communicate with MCS admin UI 516 (also referred to as portal) through a security layer 524. MCS admin UI 516 includes an application development server 518 that can be interfaced via MCS admin UI 516. An application that is developed in MCS admin UI 516 can be run on a browser of user device 528 or on mobile device 526 by communicating with production environments 512 and/or testing environments 514. In one embodiment, when an application is deployed on mobile device 526, mobile device 526 communicates with testing environments 514. However, if the application is updated on mobile device 526, such updates are performed through MCS admin UI 516.

[00153] An application that is developed in system 500 may be built as a light application or a heavy application. A heavy application is a full application such as apps that are downloaded from an App store. A light application is an application that is deployed as an added feature to an already deployed full application (i.e., a hosting application) such as an Oracle app. The hosting application acts as a container that holds the light applications. Both heavy and light applications can be further containerized by a security container, as described herein with reference to Fig. 7.

[00154] Fig. 6 is a block diagram of network components in a system 600 for building mobile applications in accordance with embodiments of the present invention. In system 600, a first device 602 interacts with an MCS website (denoted as “https://mcs-tenant-a.cloud.oracle.com” in the example embodiment of Fig. 6) to initiate build requests, and a second device 604 communicates with the MCS website to perform an over the air install of native applications. Generally, the over the air install includes downloading a file such as a property list file (a “p-list” file with extension “.plist”) that describes the application and the location from which to download the corresponding application archive file (a file with “.ipa” extension and storing the application), and then downloading the application archive file from that location.

[00155] First device 602 and second device 604 interact with the MCS website by communicating with an MCS Portal VM 612 of a server 610 through a

public Oracle HTTP Server (“OHS”) 606. Public OHS 606 is a public facing HTTP server that directs traffic to MCS Portal VM 612 located behind a firewall 608. Public OHS 606 implements a WebGate that is a web server plugin for Oracle Access Manager (“OAM”) to intercept HTTP requests and forward them to the corresponding Access Server for authentication and authorization. Accordingly, public OHS 606 authenticates the user of first device 602, passes along user credentials to MCS Portal VM 612, and terminates the SSL connection with first device 602. In the example embodiment of Fig. 6, first device 602 and second device 604 access public OHS 606 at “https://mcs-tenant-a.cloud.oracle.com” with port 443 used for https.

[00156] MCS Portal VM 612 is a standard WebLogic Server (“WLS”) application whose data is backed by a single tenant schema in schema service 614 and its corresponding application development client is written using the Oracle Jumpstart Enterprise Toolkit (“JET”) framework. WebLogic Server is a Java EE application server developed by Oracle Corp. A database schema is a container of objects (e.g., tables, views, stored procedures, etc.) to logically group them.

[00157] MCS Portal VM 612 is a single tenant and its security is provided via Oracle Web Services Manager (“OWSM,” described herein with reference to Fig. 7). Accordingly, MCS Portal VM 612 runs WLS in a trusted zone. MCS Portal VM 612 handles requests by first device 602 and has a connection to a schema service 614. MCS Portal VM 612 is also connected to a build server farm 618 via a load balancer 616. In the embodiment of Fig. 6, MCS Portal VM 612 uses open port 80 (or equivalent) for http communication to/from Public OHS 606, to load balancer 616, and from individual servers in server farm 618.

[00158] Schema service 614 interacts with the MCS Portal VM 612 and stores application data, enterprise signing certificates, and provisioning profiles for the tenant. Load balancer 616 routes farm tasks to servers in server farm 618. The routing may initially be performed in a round-robin fashion. In the embodiment of Fig. 6, load balancer 616 is a BIG-IP appliance from F5 Corp. which uses open port 80 (or equivalent) and provides redundancy. Server farm 618 includes a number of servers (e.g., 20 servers) that handle build jobs. It is connected to a filer (not shown) for storing application binaries (e.g., 5TB). In one embodiment, connections of server farm 618 are handled via a local Tomcat instance running locally on a server, and build tools and processes are handled by native OSX calls.

BUILDING AN APPLICATION

[00159] In one embodiment, once a user of first device 602 has created an application and wishes to produce a native binary, the user initiates a build POST request at the MCS website (e.g., at “https://mcs-tenant-a.cloud.oracle.com/max/build”) via a UI of first device 602. POST is a request method supported by the HTTP protocol for requesting a web server to accept and store the data enclosed in the body of the request message. The payload of the build POST request includes the application identifier (“ID”) for the application. Public OHS 606 receives the request, terminates the SSL, authenticates and authorizes the user against OAM (assuming the user is logged in), places the user identity into HTTP headers of the request, and forwards the request past firewall 608 to the WLS server of MCS Portal VM 612 (e.g., the WLS server running at “http://mcs-tenant-a.internal/max/build”).

[00160] MCS Portal VM 612 receives the request, authorizes the user for privileges against the requested application, and sends a query to the tenant schema service 614 for application data, tenant enterprise certificate, encrypted certificate password, and tenant provisioning profile. Once schema service 614 returns the requested items, MCS Portal VM 612 creates a new entry in a table of build jobs (stored at schema service 614) to record the build attempt and capture the primary key of the corresponding new build record. MCS Portal VM 612 also creates a new POST request against build server farm 618 behind load balancer 616 (e.g., at “http://max-mini-farm.internal/build/initiate”), passing in the corresponding parameters (application data, signing certificate and password, and provisioning profile) into the body of the request as well as a callback URL for job completion, where the callback URL encodes the primary key of the corresponding build record in the table of build jobs. The following functionality provides an example of a build POST request payload that includes the corresponding parameters:

```
applicationData: (app data)
signingCertificate: (cert) *
signingPassword: (password) *
provisioningProfile: (profile)
callbackUrl: http://mcs-tenant-a.internal:3000/maxbuild/complete?jobId=(BuildJobId)**
```

In this example, the certificate and the password are created by the user of first

device 602 exclusively for building mobile application according to this embodiment (i.e., the certificate and the password are not shared with services other than building mobile application according to this embodiment), and port 3000 is not publicly accessible.

[00161] Load balancer 616 maintains a list of healthy servers in server farm 618. In one embodiment, this is done via a health check that performs sanity checks at certain time intervals (e.g., every few minutes). Upon receiving a build job request, load balancer 616 selects a server from the healthy server pool in the list and routes the build job request to that server (e.g., routes the job to “http://mac-mini1.internal/build/initiate”). In one embodiment, selecting the server is according to a round robin process for build jobs that are of equal complexity.

[00162] In one embodiment, a Tomcat web server is running on the selected server in server farm 618. The Tomcat web server receives the build job request and starts an external process that runs on an asynchronous servlet to prevent the input/output from blocking the request thread pool. When the process completes, the Tomcat web server creates a POST request to the callback URL in the request payload. The following functionality provides an example payload for this new request:

result: (success if successful, etc)
binaryKey: (jobId)

[00163] MCS Portal VM 612 receives the new request and updates the corresponding record in the table of build jobs with the binary key from the payload if the event has been successful. It also informs the client (i.e., first device 602) that the build job was completed (e.g., via polling on Oracle Business Intelligence Enterprise Edition (“OBIEE”) 11g push or asynchronous servlet with OBIEE 12c planned) and produces a QR code with an encoded link for downloading the application (e.g., “https://mcs-tenant-a.cloud.oracle.com/max/native-application/(binaryKey)”).

INSTALLING AN APPLICATION

[00164] In one embodiment, once a user of second device 604 scans the QR code on second device 604, an “over the air” install is initiated. Scanning the QR code opens the URL encoded in the QR code (e.g., “https://mcs-tenant-a.cloud.oracle.com/max/native-application/(binaryKey)”). Public OHS 606 receives

the request, terminates the SSL, authenticates and authorizes the user against OAM (assuming the user is logged in), places the user identity into the HTTP headers of the request, and forwards the request past firewall 608 to the WLS server of MCS Portal VM 612 (e.g., running at “http://mcs-tenant-a.internal/max/build”).

[00165] MCS Portal VM 612 receives the request, authorizes the user for privileges against the requested application, determines user-agent (in this context, the OS framework of the device, e.g., iOS vs Android) of the requesting device (second device 604), identifies the platform of second device 604 (e.g., iOS), and forwards the request to a corresponding URL (e.g., “https://mcs-tenant-a.cloud.oracle.com/max/native-application/plist/(binaryKey)”) by directing to Public OHS 606, which will in turn forward the request to MCS Portal VM 612 to authorize (as performed during the build process described herein to ensure that the user is allowed to download the application). MCS Portal VM 612 receives the forwarded request and generates a property list file (e.g., an iOS “p-list” file) that includes application information for a corresponding platform (e.g., iPhone) as well as a link to the binary (e.g., “https://mcs-tenant-a.cloud.oracle.com/max/native-application/ios/(binaryKey)”).

[00166] Second device 604 then prompts the user if they want to install the application. Assuming yes, second device 604 follows the link to the binary (e.g., “https://mcs-tenant-a.cloud.oracle.com/max/native-application/ios/(binaryKey)”) by directing to Public OHS 606 which will in turn forward the request to MCS Portal VM 612 to authorize (as performed during the build process described herein to ensure that the user is allowed to download the application). MCS Portal VM 612 receives the request and generates a new build job request to build server farm 618 (e.g., at “http://max-mini-farm.internal/download/ios/(binaryKey)”) behind load balancer 616. Load balancer 616 selects a server in build server farm 618 (e.g., via a round robin process) from a healthy server pool and routes the build job request to that server (e.g., to “http://mac-mini1.internal/download/ios/(binaryKey)”). An application server (e.g., a Tomcat) on the selected server receives the request, determines if corresponding content exists, and streams the binary from the network (e.g., from “Filer:/filer_mnt/generated_binaries/(binaryKey)/result.ipa”). Load balancer 616 returns the streamed response back to MCS Portal VM 612, which receives the response and copies it into its request’s output stream to second device 604.

Finally, second device 604 receives the binary and performs the install.

[00167] Fig. 7 is a block diagram of mobile security suite components in an embodiment that uses security services provided by a mobile security suite 700 such as OMSS. OMSS components are distributed across the corporate DMZ 740 and the enterprise intranet (or corporate network 750). Under OMSS, a security container 706 such as “Oracle Mobile Security Container” from Oracle Corp. is installed on a mobile device 702 and is configured to hold “containerized” applications 708, e.g., applications that have been securely linked to their specific container. Mobile device 702 may also include other personal applications 704 held outside security container 706.

[00168] Security container 706 includes a secure web browser 712, a file manager (not shown), a document editor (not shown), and an optional secure mobile mail manager 710. Secure mobile mail manager 710 includes personal information management (“PIM”) applications such as a mail client, calendar, contacts, tasks, and notes synchronizing with corporate mail servers via the “Microsoft Exchange ActiveSync” (“EAS”) protocol. Many applications such as “Oracle Business Intelligence” (“BI”), “Oracle Fusion Tap,” “Oracle Social Network,” “Oracle Enterprise Manager Cloud Control,” “Oracle WebCenter Spaces,” etc., and a broad range of third-party enterprise applications can be containerized with security container 706. All data at rest inside containerized applications 708 on mobile device 702 is encrypted. Encrypted data storage includes database, file store, cache, and user preferences. Security container 706 uses a secure channel such as an “AppTunnel” 714 (as described in U.S. Pat. No. 8,332,464, the disclosure of which is incorporated herein in its entirety) to communicate with corporate network 750 behind corporate DMZ 740. In one embodiment, data in transit through AppTunnel 714 is encrypted using TLS/SSL with Federal Information Processing Standard (“FIPS”) approved algorithms.

[00169] In one embodiment, when a web browser or other client program makes an unauthenticated request to a security access server such as “Oracle Mobile Security Access Server” (“MSAS”) from Oracle Corp., the security access server responds with a redirect to the appropriate security container. Security containers use a key hierarchy to protect data. All keys are derived from user credentials that are never stored. The key hierarchy involves multiple keys to

support different sensitivity of data. For example, a unique key is used for the user's authentication certificate, which is allowed to be open for a very short period of time. A different key is used for the browser cache, which must remain decrypted for an entire session. The main security container distributes and manages keys for the complete set of apps in the user's secure enterprise workspace.

[00170] Secure container 706 has at least three distinctive benefits over conventional mobile virtual private network ("VPN") solutions: device trust vs. gateway, secure container password vs. device password, and secure container AppTunnel vs. device-level VPN. OMSS extends a network's Kerberos authentication trust directly to the user's device instead of stopping at a gateway server sitting in the DMZ. OMSS is significantly more efficient and secure than implementing "constrained delegation" offered by VPN providers. A constrained delegation solution is not only less secure but also more cumbersome to set up and maintain. Further, the tradeoff between usability and security is magnified when dealing with consumer devices and BYOD programs. Corporate IT requires strong passwords to protect corporate data on BYOD devices. Conversely, users want simple passwords—or preferably no device password at all—so they can easily access social networks and other consumer applications. Requiring a device password is frustrating for users, as they are constantly using the device for non-enterprise purposes that don't require enterprise authentication. Embodiments provide the necessary balance between security and usability when dealing with BYOD programs by requiring a password only to access corporate applications.

[00171] Yet further, device-level VPNs provide a trusted secure tunnel between a user's device and the enterprise's network. However, device-level VPN solutions are more appropriate for corporate-owned and secured endpoint devices such as laptops than for consumer mobile devices. Once a mobile-device VPN tunnel is open to the network, any application on a device has access to this secure tunnel, causing significant security vulnerability. With embodiments, however, the connection from mobile device 702 to enterprise intranet 750 exists only between security container 706 and enterprise servers.

[00172] In mobile security suite 700, MSAS 716 is typically deployed in corporate DMZ 740 and multiple server instances can be deployed behind a load balancer for high availability and scalability. MSAS 716 provides tunneled

connections between the server and containerized apps 708. MSAS 716 brokers authentication (strong authentication leverages HTTPS connections to “Oracle Access Manager” (“OAM”) 722 or Kerberos connections to Kerberos Domain Controllers 718), authorizes, audits, and enables SSO for, and proxies requests to, their destination (resources in corporate intranet 750 such as web applications and web services 724). MSAS 716 acts as the terminating end-point of the tunneled connections initiated by security container 706 and containerized applications 708.

[00173] MSAS 716 supports “Oracle API Gateway” (“OAG”) from Oracle Corp. and OWSM from Oracle Corp. to add security, threat protection, and throttling policies to an organization’s REST API infrastructure. SSO is supported through OAuth, OAM tokens, Kerberos, and NT LAN Manager (“NTLM”). SAML is supported through OAM 722 or Kerberos integration with SAML identity providers such as Oracle, CA, or Ping Identity. MSAS 716 is integrated with the OAM platform and supports the retrieval of OAM and OAuth tokens for SSO to backend resources protected by OAM, OAG, and OWSM. MSAS 716 also supports “virtual smart card” authentication by performing public key infrastructure (“PKI”) authentication to Microsoft Active Directory protected by a PIN. Digital certificates are provisioned inside the security container app and only accessed after successful PIN validation. MSAS integration with OAM allows for context aware, risk based, step-up authentication.

[00174] Mobile security suite 700 further implements OWSM which is a component of the SOA Suite and addresses web-services-based SOA security and management. The purpose of a SOA infrastructure is to allow consumers to invoke services exposed by providers. OWSM offers a solution for policy management and security of such service infrastructure. It provides visibility and control of the policies through a centralized administration interface offered by “Oracle Enterprise Manager” from Oracle Corp. OWSM allows companies to (1) centrally define and store declarative policies applied to the multiple web services making up a SOA infrastructure, (2) locally enforce security and management policies through configurable agents, and (3) monitor runtime security events such as failed authentication or authorization. It also provides business agility to respond to security threats and security breaches by allowing policy changes to be enforced in real time without the need to interrupt the running business processes.

[00175] Mobile security suite 700 further implements “Oracle Mobile Security Manager” (“MSM”) 720 within corporate network 750. MSM 720 is a “WebLogic” managed server running on either Oracle Linux or Red Hat Enterprise Linux. MSM 720 integrates with Microsoft Exchange Servers 728 in corporate network 750 to provide access to corporate email services. MSM 720 also integrates with LDAP servers 732 to provision users, assign and manage policies for Mobile Device Management and for accessing security container 706, manage the app catalog, control the remote lock or wipe of the device and secure workspace apps (wiping security container 706 removes all data and configuration for workspace apps), and set access control policies for the security container. Policies are assigned to users by associating policy templates with users and user groups. Available policy controls include device restrictions, authentication (authentication frequency, failed attempt threshold, PIN strength for PKI), catalog (apps, URLs, file shares), container/apps (compromised platform, location services, offline status, inactivity duration, data leak prevention (“DLP”)), time access (lock if outside time window), Geo Access (lock if outside geo-fence (city, state, country)), devices (whitelist specific device models, specify minimum OS level), browser (disable address bar, disable download), file browser (allow/disallow, disable download, specify file server URL), personal information manager (“PIM,” mail server URL), provisioning (invite template, PKI details), etc. If a user is in multiple groups and has multiple policies, policy combinations are resolved following specific rules.

[00176] MSM 720 maintains the EMM policies, which are then associated to one or more user groups in the directory. MSM 720 does not perform any user or group management but leverages these identities and groups directly (no synchronization) from the directory store. MSM 720 uses APNS and CGN over HTTPS to send notifications to devices. MSM 720 also exposes a WebDAV front-end to internal CIFS/SMB-enabled File Systems 730 or “Microsoft SharePoint servers,” and enables browsing intranet file shares from the client.

[00177] With more and more organizations establishing a presence on social networks, IT departments need support for social identities, which rely on more lightweight security standards than enterprise identities but are better adapted to the requirements of social networks. For example, some websites may require users to provide access tokens obtained from Facebook or Google in order to be

authenticated to their services. Accordingly, Mobile security suite 700 further implements OAMMS that includes a server that interfaces with existing backend identity management infrastructures. The server acts as an intermediary between supported mobile client apps and backend identity services. This decouples the client apps from the backend infrastructure so that backend infrastructure can be modified without having to update mobile client programs. OAMMS includes the following functionality:

- Delegated authorization leveraging the OAuth standard.
- Mobile Services connecting browser-based (HTML5) and native mobile apps to the enterprise identity management infrastructure, typically the “Oracle Access Management platform.”
- Internet Identity Services that let OAMMS be used as the relying party when interacting with popular, cloud-based identity authentication and authorization services such as Google, Yahoo, Facebook, Twitter, or LinkedIn. By deploying OAMMS, the user is provided with multiple login options without the need to implement access functionality for each identity provider individually. User Profile Services providing a REST interface for LDAP CRUD operations (customers use the same REST interface to build graphical UIs for apps), user self service functions such as self-registration, profile maintenance, password management, and account deletion. User Profile Services are also available as an OAuth resource.
- Access Management Integration Services for leveraging OAM 722 through a runtime REST interface provided by an agent SDK.

[00178] Fig. 8 is a flow diagram of mobile application development in accordance with embodiments of the present invention. In one embodiment, the functionality of the flow diagram of Fig. 8 (and Fig. 10 described below) is implemented by software stored in memory or other computer readable or tangible medium, and executed by a processor. In other embodiments, the functionality may be performed by hardware (e.g., through the use of an application specific integrated circuit (“ASIC”), a programmable gate array (“PGA”), a field programmable gate array (“FPGA”), etc.), or any combination of hardware and software. An example of cloud based mobile application development is provided in U.S. Provisional App. No. 62/186,080 (Attorney Docket No.: 88325-924721(165701US), Client Reference No.:

ORA150600-US-PSP), filed June 29, 2015, entitled "CLOUD BASED EDITOR FOR GENERATION OF INTERPRETED ARTIFACTS FOR MOBILE RUNTIME," the disclosure of which is hereby incorporated by reference.

[00179] At 810 an application definition wizard is generated. An application definition wizard as used herein represents a set of one or more UIs that guide a user during the definition process of a mobile application that utilizes one or more pre-defined cloud-accessible services. The application definition wizard can implement one or more workflows each associated with a part of the application definition process. In one embodiment, the application definition wizard can prompt or otherwise guide a user to specify application defaults, such as application identifier prefixes, default icons, splash screens, default application/feature templates, setup enterprise provisioning profile / keystore, or the like.

[00180] In certain embodiments, the application definition wizard can prompt or otherwise guide a user to specify an application name, a form factor (such as a phone or tablet device), a navigation type (e.g., none meaning a single feature or UI, as springboard, a navigation bar ("NavBar"), a Spring/Nav combo, or the like), and any application preferences.

[00181] At 820 an application definition is received. As discussed herein, the application definition can include any information needed in order to create at least a minimally functional mobile application. At 830 a mobile application is generated based on the application definition. In one embodiment, the mobile application is represented in a simulator of the target device and can include a set of definitions that when interpreted, function as a compiled mobile application.

[00182] At 840 a feature selection wizard is generated. A feature selection wizard as used herein represents a set of one or more UIs that guide a user during the development process of a mobile application that utilizes one or more pre-defined cloud-accessible services. The feature selection wizard can implement one or more workflows each associated with a part of the application development process. In one embodiment, the feature selection wizard can prompt or otherwise guide a user to specify features, UI modules, Business Object, or the like that can be used with the mobile application.

[00183] In certain embodiments, the feature selection wizard can prompt or otherwise guide a user to specify components of the first screen of the mobile

application. A component can be selected from a catalog of components.

[00184] In certain embodiments, the feature selection wizard can prompt or otherwise guide a user to specify components of other screens of the mobile application. These other screens can form part of one or more UI modules. In certain embodiments, the feature selection wizard can prompt or otherwise guide a user to specify one or more UI modules of the mobile application. A UI module represents a processor, task, or flow that can be performed with respect to the mobile application. A UI module can be selected from a catalog of UI modules or a set of templates that provide cohesive collections of UI elements and page flows. Some examples of UI modules are approval workflows, worker tasks, data entry tasks, reports builders, or the like. A template provides a pre-set arrangement/binding of a set of UI elements so that a user only needs to configure those UI elements and bind the template instead of having to arrange and bind individual UI elements. In one embodiment, a user may contribute their own templates to the set of templates available to another user. The user can configure or otherwise specify a series of pages that represent the UI Module. For each page, the user can be presented with a set of layout templates much like before. Each layout template might have several facets such as choosing a secondary template.

[00185] In some embodiments, the feature selection wizard can prompt or otherwise guide a user to specify additional features of the mobile application, such as business objects previously defined. The user can specify what resource of the backend service, API, or connector is to be used or otherwise associated with UI elements of each component, screen, UI module, or the like.

[00186] At 850 feature definitions are received, and at 860 a data binding wizard is generated. A data binding wizard as used herein represents a set of one or more UIs or UI elements of an existing UI that guide a user during the data binding process of a mobile application that utilizes one or more pre-defined cloud-accessible services. The data binding wizard can implement one or more workflows each associated with a part of the application development process. In one embodiment, the data binding wizard can prompt or otherwise guide a user to specify how features, screens, UI modules, etc., are bound to business objects, services, APIs, or the like that can be used with the mobile application. In certain embodiments, the data binding wizard can prompt or otherwise guide a user to

specify a business object of the mobile application. A business object can be selected from a catalog or set of services, APIs, etc. that are available to the mobile application.

[00187] At 870 data binding definitions are received. In various embodiments, steps 840-870 can be performed in series or in parallel. Individual steps in 840-870 can be performed on individual element of a mobile application or to a group of elements. As illustrated, a user can repeat the process of feature definition and data binding to create a mobile application.

[00188] At 880 the mobile application is deployed. The user can test the application using a testing application deployed on a target device, or as a native application deployed on a target device.

TRANSACTIONAL AUTOSAVE

[00189] Currently, in most web applications, there is no explicit save action performed by a user. Instead, the application autosaves its content on behalf of the user. Accordingly, the application needs to determine when certain save boundaries occur and then perform autosave accordingly. Save boundaries are conditions (e.g., time instances/intervals, user actions, application variable values, application/system events, etc.) that are configured to trigger an autosave of an application.

[00190] Some known systems perform autosave based on a simple time based control functionality (e.g., autosave after a set period of time) but are not ideal in case of undo/redo actions which demand more logical save boundaries. Some known systems use action based boundaries where an autosave is triggered when the user performs an action. These systems can provide autosave functionality in response to undo/redo actions but result in increased complexity for determining when to perform the autosave and how to coordinate model changes that are tied to the same user action. For example, when a user binds some text on a screen to a backend service, respective changes are made to a UI definition file and a binding definition file, and each of these files is modified by their respective model. Accordingly, providing autosave that is triggered by undo/redo actions requires coordination of these model changes which leads to increased complexity. A model is a “source of truth” which is the abstraction over the actual data, such as a page schema including the declarative syntax that describes UI components on a screen.

[00191] Further, with regards to performance and data integrity, a client

needs to confirm with the server whether or not a save action was performed successfully, so that the client knows whether it needs to retry the save action. Some known systems wait for the server response before allowing further user input. However, this waiting blocks the UI to the user and significantly impairs performance if the connection to the server is slow. Some known systems do not wait for the server and do not block the client UI. However, if the server fails, the client is often unable to recover as the states between the server and the client no longer match.

[00192] In contrast, embodiments implement a transaction system that correlates user actions with specific model changes on the client and provides corresponding action based save boundaries while ensuring data integrity. In addition, embodiments do not need to wait for server responses for save actions (and thus do not block the client UI) while also allowing for self-healing if a server error happens while attempting a save action.

[00193] One embodiment defines client side transactions that are triggered by user actions that affect client side models. A transaction is a series of related changes caused by a single user action. For example, for a user action to bind some text on a screen to a backend service requiring changes to be made to a UI definition file and a binding definition file, instead of applying respective model changes for each of these files, one embodiment captures a transaction that encompasses changes to both of these models and uses the transaction to support autosave functionality.

[00194] In one embodiment, when a model is modified, a change record is added to a current transaction. After the user action completes, the transaction is committed and the modifications are recorded in the change records. An example of a model is a page schema which is the declarative syntax that describes UI components on a current screen. This model may be implemented as an application programming interface (“API”) that manipulates the extensible markup language (“XML”) code that defines a screen. An example model change for this model is:

`some_component.setAttribute(“style”, “color: red”)`

where “some_component” is a UI component in the page model and the change is to modify a style attribute to make it red. An example transaction for this model is the series of model changes (e.g., the aforementioned change to a page schema and the corresponding bindings), which may be caused by clicking a button by a user.

Examples of user actions that can be configured to trigger an autosave action for this model are user gestures such as pressing a button, entering some text, dragging and dropping some piece of UI, etc. For example, when completing a “new application wizard” to develop an application, the user may modify the application and its screen models, and an autosave can be triggered when the user hits a “finish” button to finish developing the application.

[00195] One embodiment defines save boundaries to autosave local content to a remote persistence store of a server. A persistence store is a storage (e.g., a file system, a database, etc.) that stores data for long term. This embodiment implements two separate lifecycles on two separate queues. The change records are first applied to a local lifecycle on a local queue to commit the changes to local models. This lifecycle happens very quickly and allows the client UI to reflect these changes in near instantaneous manner. The change records are also run under a remote lifecycle where they are queued in a remote queue to be sent to the server. The remote queue is a persistence queue that manages what gets stored to the remote persistence store of the server. Change records queued on the remote queue are “synchronously” sent to the server in the order they were created. That is, change records on the remote queue are processed one at a time and each change record is sent to the server only if the prior change record has been successfully recorded on the server in one embodiment. This ensures that the order of the changes is maintained at the server.

[00196] When a save request to save a change to the server fails, embodiments are aware of the point at the remote queue from which they need to resend the queued save requests. Accordingly, embodiments retry the failed record in the remote queue and then continue with subsequent records. If the server fail persists, embodiments transition to off-line mode and continue recording change records on the remote lifecycle. These records are then saved at the server in the correct order when the connection with the server has been restored or server instability has been corrected.

[00197] By allowing for almost instantaneous UI updates, embodiments improve client UI performance without sacrificing data integrity. For example, actions performed by a user (including complicated actions such as creating complex artifacts) are completed immediately for the user. Further, since the server saves

are synchronous, embodiments can handle server outages and ensure that data is not lost and contents will have eventual consistency with the server.

[00198] In one embodiment, a transaction is created when a user action is triggered and is either committed or rolled back when the action completes. An action completes when the system has finished processing the result of that action. In between these two events (the user action and completion of the user action), model changes can occur in response to the user action. For example, when the user action is pressing a button, the system responds to such pressing and the action completes when the system completes the processing of the pressing. If the button is pressed to, e.g., add a new attribute to some part of a page but it is deemed to be illegal to add that attribute, the transaction is cancelled. If on the other hand the attribute is in fact legal, then the transaction is committed.

[00199] As another example, when the user attempts to add a button to a screen by “drag and drop” functionality, the transaction completes if the user can successfully drop the button to the screen. However, if the user tries to perform an unacceptable action such as adding a button within another button, the page model detects that this action is unacceptable and notifies the system. This causes the transaction to fail and all related activities (all model changes) of the transaction are rolled back (reverted). In some embodiments, additional model changes may be made in response to a single model change initiated by a user gesture. For example, when a user drops a button in a location that is acceptable, the system may automatically add text to the button (e.g., by default) as a separate/additional model change.

[00200] One embodiment implements transactional functionality as provided in transactional databases. A transactional database is a database management system (“DBMS”) where write transactions on the database can be rolled back if they are not completed properly (e.g., due to power or connectivity loss). Some relational DBMS’s support transactions that include one or more data manipulation statements and queries each reading and/or writing information in the database. Transactions may also be implemented as nested transactions that include statements within them that start new transactions (i.e., sub-transactions). A transaction is committed if no errors occur during its execution. A transaction commit operation applies all data manipulations within the transaction and persists the results to the corresponding

model. If an error occurs during the transaction or if the user specifies a rollback operation, data manipulations within the transaction are not persisted to the model. A partial transaction cannot be committed to the model since that would leave the model in an inconsistent state.

[00201] In one embodiment, each model is configured to record its changes to the transaction, including a minimal set of instructions to perform the change and undo that change. Such minimal set of instructions can be used to implement undo/redo actions. For example, for a page model that is backed by XML, calling “some_component.setAttribute(…)” immediately modifies the XML, and “[set attribute “foo” on element “X”]” can be a singular instruction that can be used to implement undo/redo actions.

[00202] In one embodiment, when a transaction is committed, it is sent to the local lifecycle where transactions are recorded in the order they are completed. A transaction is considered “recorded” when its changes are applied to corresponding client side models and any necessary UI is updated to reflect the changes. Once a transaction is recorded locally, its change record is added to the remote persistence queue of the remote lifecycle. The remote lifecycle is a queue of transaction change records that need to be saved to the server. These records are processed one at a time and indicate how content is persisted on the server. If the persisting of a change record to the server completes, the change record is removed from the remote persistence queue. However, if a server save fails or a response is not received after a period of time (e.g., 20 seconds after attempting the persist operation), the persist operation is determined to have failed. In this case, one embodiment retries the save action using an exponential back-off algorithm. For example, one embodiment attempts a first retry of the save action after 1 second, and if the first retry fails waits 2 seconds to attempt a second retry, and if the second retry fails waits 4 seconds to attempt a third retry, and so on. If the failure persists, one embodiment transitions to offline mode and checks at longer time intervals (e.g., every minute) to determine if the server connection has been restored. This may be performed by retrying the save of the transaction at the head of the persistence queue.

[00203] Embodiments are applicable to any web based applications such as those in a cloud environment. For example, when a user of a web based application

in a cloud based IDE drags and drops an object onto a design canvas in a UI, a number of separate but logically related files are updated and the changes to these files are treated as a single logical transaction to allow for undo/redo of the combined action as a single atomic transaction. In the distributed system that provides the IDE, some of the objects that are part of a specific transaction may be remote to the user. Accordingly, embodiment ensure that coordination of these remote updates is not blocking from user perspective, and changes sent to remote systems are coordinated in terms of ordering so that failed transactions can be replayed.

[00204] One embodiment implements a “TransactionManager” JavaScript module that manages an undo history list including a history of committed transactions that can be undone or redone. TransactionManager is a singleton (i.e., only one instance of this JavaScript module is created) and provides callers with a “startTransaction” JavaScript method so that they can start recording changes. At any time, callers can retrieve the current transaction using a “getCurrentTransaction()” JavaScript method. Once all changes are done, “TransactionManager” calls a “commitTransaction()” JavaScript method to save the change records into the undo history. If for some reason during the lifetime of the transaction there are errors or if the commit operation fails, then “TransactionManager” calls a “rollbackTransaction()” JavaScript method which removes the recorded changes. To undo the latest transaction, an “undo()” JavaScript method can be called. Calling “undo()” repeatedly walks back the history of transactions by undoing each one. At any point, a “redo” JavaScript method can be called to redo the most recent undo transaction. “redo” can be called as many times as “undo” was called. Each “undo”/“redo” triggers a “TransactionEvent” JavaScript method of type undo/redo. “undo” and “redo” JavaScript methods should not be called within a transaction.

[00205] One embodiment implements a “ChangeRecord” JavaScript module as the base JavaScript module for a change record that constitutes part of a transaction. Each transaction can include one or more change records, and all records are transacted atomically (i.e., either all occur or none occur). Should one change record roll back, all change records in that same transaction are subsequently rolled back. A change record has two life cycles: local and remote.

[00206] The local life cycle includes non-blocking operations that occur on the

local client. A non-blocking operation is an operation that does not halt the UI while processing is occurring, such as not waiting on a server response to continue. There are three lifecycle JavaScript methods that occur on the client: “localCommit,” “localRevert,” and “localReplay.” “localCommit” is called exactly once when the change record is initially committed. “localRevert” is called whenever the record is reverted (e.g., in an undo or roll back operation). “localReplay” is called whenever the record is asked to be replayed (e.g., in a redo operation).

[00207] The remote lifecycle occurs in parallel with the local lifecycle, but since the remote lifecycle JavaScript methods are performed asynchronously (i.e., at the same time or in parallel), they may be delayed as compared to the local lifecycle JavaScript methods. There are two JavaScript methods for remote operations: “remotePersist” and “remoteRestore.” “remotePersist” persists the result of the change to a remote server (i.e., saves the change to a database), while “remoteRestore” restores the original state (i.e., before the change) to the remote server.

[00208] In one embodiment, a change record is identified by a unique type that represents a single resource in the system. A resource may be a model such as a screen, bindings related to a screen, navigation flow of the application, application metadata, etc. The type also identifies the JavaScript method of the operation. In one embodiment, the type is left non-specific in order to allow change records and listeners to define the contract of the operation. A type is a tuple including the operation of the change as well as the ID of the resource. The operation may also uniquely identify what type of resource is being modified. For example, in the following functionality, “resourceType” indicates what type of model is affected (i.e., correlates to a table in the database) and the “id” identifies a specific instance of that resource (i.e., a row in that table):

```
{
  resourceType: page
  id: pageId
}
```

[00209] For certain types, the persist operation may combine the remote lifecycle of multiple change records. For example, if two persist operations occur for the same type, the first may be skipped with the assumption that the second will override data of the first. In one embodiment, a single change record affects a single

resource, and multiple change records are used if multiple resources are modified in the same transaction. For example, when a user action to bind some text on a screen to a backend service requires changes to be made to a UI definition file and a binding definition file, one embodiment captures a transaction that encompasses changes to both of these models and adds respective change records for changes to each model.

[00210] One embodiment also provides a JavaScript method that allows for UI-only change records to be added in the local lifecycle. Such change records are only used in the local life cycle to affect UI changes locally. An example of a change that only requires UI-only change records is changing tabs on a screen. This change does not require a data change and therefore nothing needs to be synchronized back to the server. For UI handling that is incidental (e.g., capturing UI state before/after a transaction), transaction metadata available during the local lifecycle eventing is used. For example, when a user switches tabs on a screen, embodiments need the tabs to switch back if the user performs an undo. The metadata recorded in a UI-only transaction would capture what tab was active so that it can be restored during the undo. This metadata is stored via the transaction metadata (e.g., `the_current_transaciton.setMetadata(CurrentTab, "some tab")`).

[00211] A `ChangeRecord::localCommit` JavaScript method commits the transaction locally. Some models perform the operations immediately on command, and thus there is nothing to commit. In these cases, this JavaScript method can serve to inform the model when a commit has been completed. This JavaScript method can optionally return a Boolean. In some models, operations are performed on a tentative basis and there may be no actual changes committed. In those cases, this JavaScript method returns `"false"` to indicate that no change has been made to the underlying model. If this JavaScript method returns `"false,"` the transaction is not aborted but this particular change is ignored by the transaction. A transaction that has all of its records return `"false"` for this JavaScript method is silently disposed of but not rolled back. A UI-only change record is not affected by what is returned by this JavaScript method.

[00212] An example of a model that makes its changes immediately when a caller modifies a property on that model is a page model that is backed by XML. For this model, calling `"some_component.setAttribute(...)"` immediately modifies the

XML. Since the model is modified before the transaction completes, there is no work to be done during the commit operation when the transaction completes.

[00213] Some models do not modify their underlying sources immediately. Some models capture the changes that are done to that model in a temporary storage, and when the transaction completes during a commit operation, those changes are read from that temporary storage and written to the model. An example of a model where no changes are necessary is when a user changes the text of a button from “a” to “b” and then changes it back from “b” to “a” in the same transaction. Since the value “a” has not changed from the start of the transaction to when the transaction is committed, the model can choose to do nothing since an undo of that operation would not have any effect on the model.

[00214] A “ChangeRecord::remoteCommit” JavaScript method persists the changes to a remote server. When the persist operation completes, this lifecycle JavaScript method indicates that it has completed. When the persist operation is rejected, this lifecycle JavaScript method indicates that an error has occurred and schedules the persist operation to be retried in the future.

[00215] Fig. 9 is a block diagram of a system 900 for web application development in one embodiment. A user develops an application by using a web browser 902 running an application development website 910 corresponding to an application development server 904. The actions performed by the user are reflected at application development server 904 through a request/response mechanism in which each change is sent to application development server 904, which in turn sends a corresponding response back indicating whether the change has been persisted at application development server 904.

[00216] In some known systems, a user is not allowed to continue interacting with application development website 910 and making further changes unless a response is received from application development server 904 indicating that the change that was last made is persisted at application development server 904. However, this may result in a poor user experience and overall performance when there is a delay in receiving the response due to, for example, application development server 904 being loaded, the connection to application development server 904 being slow, poor connection with application development server 904 in cellular/wireless scenarios, etc. Indeed, in some known systems, a user may need

to wait 2-3 seconds after each change before being able to continue. This problem is amplified when a series of actions are performed by the user and the user then intends to undo all those actions, since the user will have to wait 2-3 seconds for performing each action and then another 2-3 seconds for undoing each action, only to end up with no changes.

[00217] In contrast, system 900 maintains a transactional system at the client side (i.e., web browser 902) by implementing a local queue 906 and a remote queue 908. For example, when a change “A” is made by the user in application development website 910, this change is viewed in terms of a transaction (i.e., a unit/block of work with clear start and end points). For example, “A” may be the addition of a form to a screen on application development Website 910. Accordingly, every discrete operation is its own transaction. When a transaction gets committed (i.e., the transaction has entered the system and the user has decided to perform the corresponding action), “A” is placed in local queue 906. Application development website 910 processes the contents of local queue 906 one at a time and performs the corresponding changes. For example, when “A” is the oldest unprocessed item in local queue 906, application development website 910 takes “A” out of local queue 906 and shows its effect on the screen of web browser 902.

[00218] The transactions are also queued in remote queue 908 independently of local queue 906. Remote queue 908 is also ordered and its contents are processed one at a time. For example, for change “A” to add a form to the screen of Web browser 902, the user views the result immediately while “A” may still be waiting in remote queue 908 to be communicated with application development server 904. If application development server 904 is slow (e.g., if it takes 5 seconds to upload a change to application development server 904), instead of waiting for “A” to be successfully sent to application development server 904, the user can go ahead and perform other changes “B,” “C,” and “D” locally. On the client side, despite a slow server, changes “A” through “D” happen instantly and are recorded. Accordingly, local queue 906 is processed very quickly.

[00219] If a catastrophic event happens at application development server 904 (e.g., the server goes down), system 900 transitions to offline mode and the changes that are not yet reflected at application development server 904 stay in remote queue 908 so they can be communicated with application development

server 904 once it comes back on. Remote queue 908 holds such changes even if web browser 902 is terminated. By referring to remote queue 908 after application development server 904 is available again, system 900 identifies which change is the next one to be communicated with application development server 904. An item in remote queue 908 is eliminated only after receiving a corresponding response from application development server 904 indicating that the item has been successfully stored at application development server 904. Since the items are communicated with application development server 904 one at a time, at every instance system 900 knows which items have been successfully communicated with application development server 904.

[00220] In one embodiment, when a most recent change saves the totality of a resource, previous changes to that resource can be eliminated from local queue 906 and/or remote queue 908. For example, in a series of changes “A” through “D,” if “B” and “D” modify the same resource, and “C” and “D” are independent, “B” can be eliminated from the queues. An example of such resource is a UI on web browser 902, where moving one form field to another form field modifies the same resource.

[00221] In one embodiment, when processing local queue 906 and/or remote queue 908, a respective pointer is moved instead of moving the transactions themselves. In one embodiment, undo operations are scheduled in remote queue 908 just the same as other transactions. However, if an undo transaction is added to remote queue 908 while the corresponding original transaction is still waiting in remote queue 908, both transactions may be cancelled in remote queue 908.

[00222] In one embodiment, undo history is maintained in local queue 906 from the beginning of time (e.g., the first time that application development website 910 was reached). In this embodiment, local queue 906 is a persistent object.

[00223] Fig. 10 is a flow diagram of autosave functionality in accordance with embodiments of the present invention.

[00224] At 1010 a web browser of a client device receives a user action performed by a user that interacts with a website corresponding to a server. The website may be an application development website and the server may be an application development server.

[00225] At 1020 a change record corresponding to the user action is

determined and at 1030 the change record is queued in a first queue to commit corresponding changes to local models. In one embodiment, the first queue is an ordered persistence queue that maintains a history of change records for performing undo and redo operations in interacting with the website.

[00226] At 1040 the change record is queued in a second queue that communicates with the server to persist the change record at the server. In one embodiment, the second queue is an ordered queue where change records are processed one at a time and each change record is sent to the server only if a prior change record in the second queue has been successfully recorded on the server. The change record is removed from the second queue after being successfully communicated with the server. The second queue communicates with the server through a request/response mechanism in which each change record in the second queue is sent to the server which in turn sends a corresponding response back indicating whether the change record has been persisted at the server.

[00227] In one embodiment, the change record reflects model changes caused by the user action in the website and is added in a transaction that constitutes related model changes. Change records in the transaction are placed in the first queue when the transaction is committed. The transaction is committed if no errors occur during its execution. The website processes contents of the first queue one at a time and performs corresponding changes. Change records in the transaction are placed in the second queue when the transaction is recorded. The transaction is recorded when its changes are applied to corresponding client side models and any necessary user interface is updated to reflect corresponding changes.

[00228] One embodiment determines that a response to a request is not received from the server after expiry of a time threshold and transitions to offline mode while preserving contents of the second queue and continuing the queuing of new change records in the second queue. The embodiment then determines that communication with the server is restored, transitions back to online mode, and resumes to persist the contents of the second queue at the server. The second queue holds its contents even when the web browser is terminated.

[00229] One embodiment determines that the change record saves a totality of a resource and removes previous change records to the resource from the first

queue and the second queue.

[00230] In one embodiment, when an undo transaction is added to the second queue while a corresponding original transaction is still waiting in the second queue, both the undo transaction and the corresponding original transaction are cancelled from the second queue.

[00231] As disclosed, embodiments provide a transactional autosave system that allows for client side changes to be almost instantaneous while also persisting those changes at a remote server without losing data or causing inconsistency between the client and the server, even in case of server connection problems or undo/redo actions performed at the client. Accordingly, since there is no need for blocking the user while persisting user changes to a remote server, embodiments greatly enhance performance and improve user experience.

RESTORATION OF UI STATE

[00232] Currently, some transactional systems provide undo and redo functionality to revert or re-apply model changes. However, while it may also be desirable to restore UI states (i.e., what UIs the user is seeing), the UI state information may not be part of the model data, and therefore, restoring the model may not automatically restore the UI state. For example, a user may be on a page “X” in a system, make a model change to page “X,” and then switch/navigate to a page “Y.” If the user performs an undo on the model change to page “X,” it would be highly desirable to switch the view back to page “X” since that is where the user made that model change. However, undoing just the model change to page “X” does not automatically bring the user from page “Y” to page “X.” As a system gets more complex, the amount of UI state that preferably needs restoration also increases.

[00233] Some known systems either do not address the issue at all (i.e., the UI is not changed on undo/redo) or only restore the UI state to a certain extent. For example, in some known systems, when an undo or redo action is received, the user is brought back to some main/default UI but not a detailed UI specific to the undo/redo action.

[00234] Some known systems attempt to inspect the changes to determine what UI to show to the user. However, implementing this functionality complicates the system and is difficult to maintain.

[00235] In contrast to the known systems, embodiments provide a transactional system where the UI state can be restored when a transaction is rolled back or when it is subsequently re-committed (i.e., operations typical in undo/redo functionality). One embodiment stores UI and model states both before and after committing a transaction, and uses the stored states to restore appropriate UI and model states when an undo or a redo operation is performed on the transaction. Accordingly, embodiments make UI restoration manageable and automatic within a transactional environment, even for complex systems where a substantial amount of UI state needs restoration.

[00236] In one embodiment, upon undoing an action, the user sees the same UI they were looking at before performing that action. One embodiment provides similar functionality for redo. That is, upon redoing an action that has been undone, the user sees the same UI they were looking at after performing that action. Accordingly, embodiments significantly improve the user experience by preventing the user from losing context within the UI.

[00237] Embodiments are extensible to allow for any piece of UI to participate in the restoration functionality and do not require the UI to understand the semantics of a transactional system. That is, the UI does not need to understand what model changes are actually performed by a transaction, what that transaction means to the system, or how transactions work in general (e.g., how a commit operation, a rollback operation, etc., works).

[00238] In one embodiment, when a web page is rendered on a browser, components of the web page are initialized. That is, when a web page is rendered, UI elements/actors are first downloaded to the browser and then executed. In this embodiment, UI actors in the system can register with the transactional framework at initialization time. The registrations can also be dynamic so that a UI that is created during a work session can register with the transactional framework and then deregister when that UI is removed.

[00239] In one embodiment, when a transaction is started, the transactional framework queries the registered UIs and retrieves a snapshot of their current UI states. In one embodiment, a UI state includes the semantical meaning of the change resulting from a transaction. The UI actors involved in the transaction know how to restore such UI state since these actors created the UI state in the first place.

The following is an example functionality that may be included in a UI state to record that the current color of a button is blue:

```
{
  "color": "blue"
}
```

[00240] The transaction then commits and updates the corresponding models in the system. Once these changes are reflected in the UIs, the registered UIs are queried again and another snapshot of their current UI states is taken. In one embodiment, the snapshots do not indicate what lifecycle they are in within a transaction (e.g., whether the lifecycle state of a transaction is a commit, undo, redo, etc.). For example, the UI just takes its current snapshot regardless of whether the user performed an action, an undo operation, a redo operation, etc. In one embodiment, “before transaction” and “after transaction” snapshots of UI states are stored along with corresponding transaction change records.

[00241] In one embodiment, a change record is stored independently of a corresponding model state, and stores instructions for modifying that model state. The model state is assumed to be correct at any given point in time (e.g., match the UI that is being provided to the user at any given point in time). In one embodiment, change records are stored in an “undo” stack. Each change record includes a set of instructions that can perform a corresponding action or undo it. Accordingly, when a user undoes an action, the last change record is retrieved from the undo stack and called to perform the corresponding undo instructions.

[00242] In one embodiment, when model changes are applied in a transaction which is subsequently rolled back (e.g., by performing an undo operation), the registered UIs are provided with their corresponding UI snapshots taken when the transaction started, and the UIs are prompted to restore their corresponding UI snapshots. Similarly, when model changes are restored in a rolled back transaction which is subsequently replayed (e.g., by performing a redo operation), the registered UIs are restored to their corresponding UI snapshots taken when the transaction was committed.

[00243] In one embodiment, restoring a snapshot of one piece of a UI does not affect other pieces of the UI. For example, one UI piece does not event to another UI piece when a snapshot is restored since updating the other UI piece outside its own lifecycle may introduce unpredictable behavior. Accordingly, in one

embodiment, each piece of an affected UI registers and manages its UI state independently of other UI pieces, does not react to events of other UI pieces, and does not send any events to other UI pieces during this process.

[00244] Embodiments are applicable to any system or web application that supports undo/redo functionality.

[00245] Fig. 11 illustrates an example UI 1100 according to one embodiment. UI 1100 may be used, for example, to configure/design a mobile application or another type of application. UI 1100 includes three UI pieces: a template selection piece 1102, a template piece 1104, and a component selection piece 1106. In one example, template selection piece 1102 may provide options to select a template A 1110, a template B 1112, or a template C 1114. Upon selecting a template option in template selection piece 1102, the selected template is shown in template piece 1104. In one embodiment, the UI component in template selection piece 1102 that corresponds to the selected template in template piece 1104 may be highlighted in template selection piece 1102.

[00246] Template piece 1104 may be further defined and/or modified by components provided in component selection piece 1106. The options provided in component selection piece 1106 may be related to the choice of template being shown in template piece 1104. For example, component selection piece 1106 may provide options for adding a button D 1116, a button E 1118, or a button F 1120 to template piece 1104. In one example, a button 1108 may be shown in template 1104 upon selection of a corresponding option in component selection piece 1106. In one embodiment, the UI component in component selection piece 1106 that corresponds to a button added in template piece 1104 may be highlighted in component selection piece 1006.

[00247] In one embodiment, although UI pieces in UI 1100 are related to each other, the UI state of each UI piece is stored independently of the other UI pieces, and restoration of each UI piece does not affect or event restoration of the other UI pieces. Accordingly, each UI piece may be restored independently and the UI pieces do not receive events from other UI pieces or send events to other UI pieces. For example, if a highlighted UI component in template selection piece 1102 corresponds to a selected template in template piece 1104, the UI state of template selection piece 1102 includes such highlighting information but its restoration does

not event template piece 1104 to show the corresponding template. In this scenario, the UI state of template piece 1104 is independently saved and restored to show the appropriate template, thereby obviating the need for inter-related eventing between the two related UI pieces. A similar independent restoration functionality is implemented for template piece 1104 and component selection piece 1106 which hold inter-related information but do not event each other at the time of restoration. Accordingly, every UI piece of UI 1100 stores its own state independently of the other UI pieces, and therefore no UI piece needs to wait for another UI piece for restoration.

[00248] In one embodiment, all pieces of the UI can save their respective states but only those pieces of the UI that are affected by an action need to be saved/restored. In one embodiment, some UI pieces may not know what the action is doing (i.e., what the effect of the action will be), therefore, each visible UI piece has its state saved/restored even if they are not affected by the action. In one embodiment, if a UI piece is not visible when the action is performed, then such UI piece knows that it is not affected by the action since it was not even being displayed at the point in time when the action was performed. Accordingly, the state of UI pieces that are not visible during the action may not need to be saved/restored.

[00249] The following provides an example of saving and restoring a UI for a page that has a main content area and a sidebar, where the sidebar should have a slightly darker background color than the main content area. The page may start out with the main content area being “gray” and the sidebar being “dark gray.” The background color of the main content area may be changed to “red” as follows:

- A user makes a gesture to change the background color of the main content area, and a corresponding transaction is started.
- The framework first records all the UI states. Assuming there are only two UIs including a color picker for the main content area and a color picker for the sidebar, each color picker records its current state according to the following functionality:

```
main content color picker:
{
  "background-color": "gray"
}
side bar color picker:
{
  "background-color": "dark gray"
```

}

- The main content color picker includes code that knows that when its color changes, the side bar color picker should also automatically change to be a darker version of the new color of the main content color picker. Accordingly, the main content color picker tells the side bar color picker to make its color “dark red.”
- The transaction is committed and the model state becomes:

```
main-content-color: red
side-bar-color: dark red
```

[00250] If the user subsequently undoes this transaction:

- The model state is reverted to:

```
main-content-color: gray
side-bar-color: dark gray
```

At this point the model is correct and reflects the undo action, but the color pickers are still showing the wrong “red” colors.

- The pieces of UI (i.e., the color pickers) are told to restore their state to the states before the transaction, i.e.:

```
main content color picker:
{
  "background-color": "gray"
}
side bar color picker:
{
  "background-color": "dark gray"
}
```

[00251] The main content color picker then restores its state back to “gray” and the side bar color picker restores its state back to “dark gray.” In restoring these states, changing the main color picker no longer automatically tells the side bar color picker to be a darker version of the restored color of the main content color picker. That is, there is no “eventing” similar to that of normal UI changes, and each UI piece manages its state by itself. In this embodiment, the UI pieces may read their corresponding models to know what their states are. However, tracking a UI state of a UI piece also tracks/stores UI changes that may not be recorded in the corresponding model, such as whether a color picker should be visible or not.

[00252] Fig. 12 illustrates an example commit flow 1202, an example undo flow 1204, and an example redo flow 1206, according to some embodiments.

[00253] In commit flow 1202, when a transaction occurs, corresponding model and UI states are recorded at 1208 before the transaction starts (i.e., at “INIT”

state). Such model and UI states are denoted, respectively, as “INIT MODEL” and “INIT UI” in Fig. 12. Then, the pre-commit state at 1210 provides outside listeners with the opportunity to perform actions before the commit state. At 1212 the transaction finishes (i.e., the transaction is committed at “COMMIT” state) and corresponding model and UI states are recorded again. Such model and UI states are denoted, respectively, as “COMMIT MODEL” and “COMMIT UI” in Fig. 12.

[00254] In undo flow 1204, an undo (revert) operation attempts to restore the UI and model states prior to the transaction (i.e., the “INIT MODEL” and “INIT UI” states captured at 1208). In order to do so, at 1214 the “INIT MODEL” state is restored at a pre-revert state before the revert operation. At 1216 the revert operation is performed. The revert operation includes performing the undo action. For example, if a text is changed from “A” to “B” and then an undo action is indicated, the revert operation changes “B” back to “A.” After the revert operation completes, at 1218 the “INIT UI” state is restored at a post-revert state.

[00255] In redo flow 1206, a redo (replay) operation attempts to restore the UI and model states after the transaction was committed (i.e., the “COMMIT MODEL” and “COMMIT UI” states captured at 1212). In order to do so, at 1220 the “COMMIT MODEL” state is restored at a pre-replay state before the redo operation. At 1222 the replay operation is performed. The replay operation includes performing the redo action. For example, if a text is changed from “A” to “B” and then an undo action changes “B” back to “A,” the redo action changes “A” back to “B” again. After the replay operation completes, at 1224 the “COMMIT UI” state is restored at a post-replay state.

[00256] Fig. 13 is a flow diagram of UI restoration functionality in accordance with embodiments of the present invention.

[00257] At 1310 an action performed by a user that interacts with a UI is received. Examples of user actions are user gestures such as pressing a button, entering some text, dragging and dropping some piece of UI, etc. In one embodiment, the action is received by a web browser of a client device of the user, and the user interacts with a website corresponding to a server that hosts the UI. In one embodiment, the website is an application development website and the server is an application development server.

[00258] At 1312 a transaction is determined based on the action, where the

transaction is configured to modify a model corresponding to the UI. In one embodiment, the transaction constitutes a number of related model changes.

[00259] An example of a model is a page schema which is the declarative syntax that describes UI components on a current screen. This model may be implemented as an API that manipulates the XML code that defines a screen. An example model change for this model is:

```
some_component.setAttribute("style", "color: red")
```

where "some_component" is a UI component in the page model and the change is to modify a style attribute to make it red. An example transaction for this model is the series of model changes (e.g., the aforementioned change to a page schema and the corresponding bindings), which may be caused by clicking a button by a user.

[00260] At 1314 a first UI state of the UI and a first model state of the model are stored.

[00261] At 1316 the transaction is committed. In one embodiment, the transaction is committed if no errors occur during its execution.

[00262] At 1318 it is determined to undo the transaction based on a first user interaction.

[00263] At 1320 the UI is restored to the first UI state and the model is restored to the first model state. In one embodiment, the first model state is restored before undoing the transaction, and the first UI state is restored after undoing the transaction. In one embodiment, the UI includes two or more UI pieces, and the first UI state includes a UI state for each piece of the UI. In one embodiment, the UI state of each piece of the UI is stored and restored independently of other UI pieces of the UI.

[00264] In one embodiment, a second UI state of the UI and a second model state of the model are stored after the transaction is committed and before the first user interaction.

[00265] In one embodiment, it is further determined to redo the transaction based on a second user interaction, and then the UI is restored to the second UI state and the model is restored to the second model state. In one embodiment, the second model state is restored before redoing the transaction, and the second UI state is restored after redoing the transaction.

[00266] As disclosed, embodiments provide a transactional system that uses

stored snapshots of UI and model states before/after committing a transaction and restores appropriate UI and model states when undo/redo operations are performed. In one embodiment, upon undoing an action, the user is provided with the same UI they were looking at before performing that action. In one embodiment, upon redoing an action, the user is provided with the same UI they were looking at after performing that action. Accordingly, embodiments significantly improve the user experience and improve the efficiency of a UI design process.

[00267] Several embodiments are specifically illustrated and/or described herein. However, it will be appreciated that modifications and variations of the disclosed embodiments are covered by the above teachings and within the purview of the appended claims without departing from the spirit and intended scope of the invention.

What is claimed:

1. A method of autosave comprising:
 - receiving, by a web browser of a client device, a user action performed by a user that interacts with a website corresponding to a server;
 - determining a change record corresponding to the user action;
 - queuing the change record in a first queue to commit corresponding changes to local models; and
 - queuing the change record in a second queue that communicates with the server to persist the change record at the server.
2. The method of claim 1, wherein the server hosts a user interface (UI), wherein the user action is performed by the user when interacting with the UI, the method further comprising:
 - determining a transaction based on the user action, wherein the transaction is configured to modify a model corresponding to the UI;
 - storing a first UI state of the UI and a first model state of the model;
 - committing the transaction;
 - determining to undo the transaction based on a first user interaction; and
 - restoring the UI to the first UI state and the model to the first model state.
3. The method of claim 2, wherein the first model state is restored before undoing the transaction, wherein the first UI state is restored after undoing the transaction.
4. The method of claim 2, wherein a second UI state of the UI and a second model state of the model are stored after the transaction is committed and before the first user interaction.
5. The method of claim 4, further comprising:
 - determining to redo the transaction based on a second user interaction; and
 - restoring the UI to the second UI state and the model to the second model state.

6. The method of claim 5, wherein the second model state is restored before redoing the transaction, wherein the second UI state is restored after redoing the transaction.

7. The method of claim 1, wherein the website is an application development website and the server is an application development server.

8. The method of claim 2, wherein the transaction constitutes a number of related model changes.

9. The method of claim 2, wherein the transaction is committed if no errors occur during its execution.

10. The method of claim 2, wherein the UI comprises two or more UI pieces, wherein the first UI state includes a UI state for each piece of the UI.

11. The method of claim 10, wherein the UI state of each UI piece of the UI is stored and restored independently of other UI pieces of the UI.

12. The method of claim 1, wherein the first queue is an ordered persistence queue that maintains a history of change records for performing undo and redo operations in interacting with the website.

13. The method of claim 1, wherein the second queue is an ordered queue where change records are processed one at a time and each change record is sent to the server only if a prior change record in the second queue has been successfully recorded on the server.

14. The method of claim 1, wherein the change record is removed from the second queue after being successfully communicated with the server.

15. The method of claim 1, wherein the second queue communicates with the server through a request/response mechanism in which each change record in the

second queue is sent to the server which in turn sends a corresponding response back indicating whether the change record has been persisted at the server.

16. The method of claim 1, wherein the change record reflects model changes caused by the user action in the website and is added in a transaction that constitutes related model changes.

17. The method of claim 1, wherein change records in the transaction are placed in the first queue when the transaction is committed, wherein the transaction is committed if no errors occur during its execution.

18. The method of claim 17, wherein change records in the transaction are placed in the second queue when the transaction is recorded, wherein the transaction is recorded when its changes are applied to corresponding client side models and any necessary user interface is updated to reflect corresponding changes

19. A computer readable medium having instructions stored thereon that, when executed by a processor, cause the processor to perform autosave, the performing comprising:

receiving, by a web browser of a client device, a user action performed by a user that interacts with a website corresponding to a server;

determining a change record corresponding to the user action;

queuing the change record in a first queue to commit corresponding changes to local models; and

queuing the change record in a second queue that communicates with the server to persist the change record at the server.

20. An autosave system comprising:

a receiving module that receives, by a web browser of a client device, a user action performed by a user that interacts with a website corresponding to a server;

a determining module that determines a change record corresponding to the user action;

a first queuing module that queues the change record in a first queue to commit corresponding changes to local models; and

a second queuing module that queues the change record in a second queue that communicates with the server to persist the change record at the server.

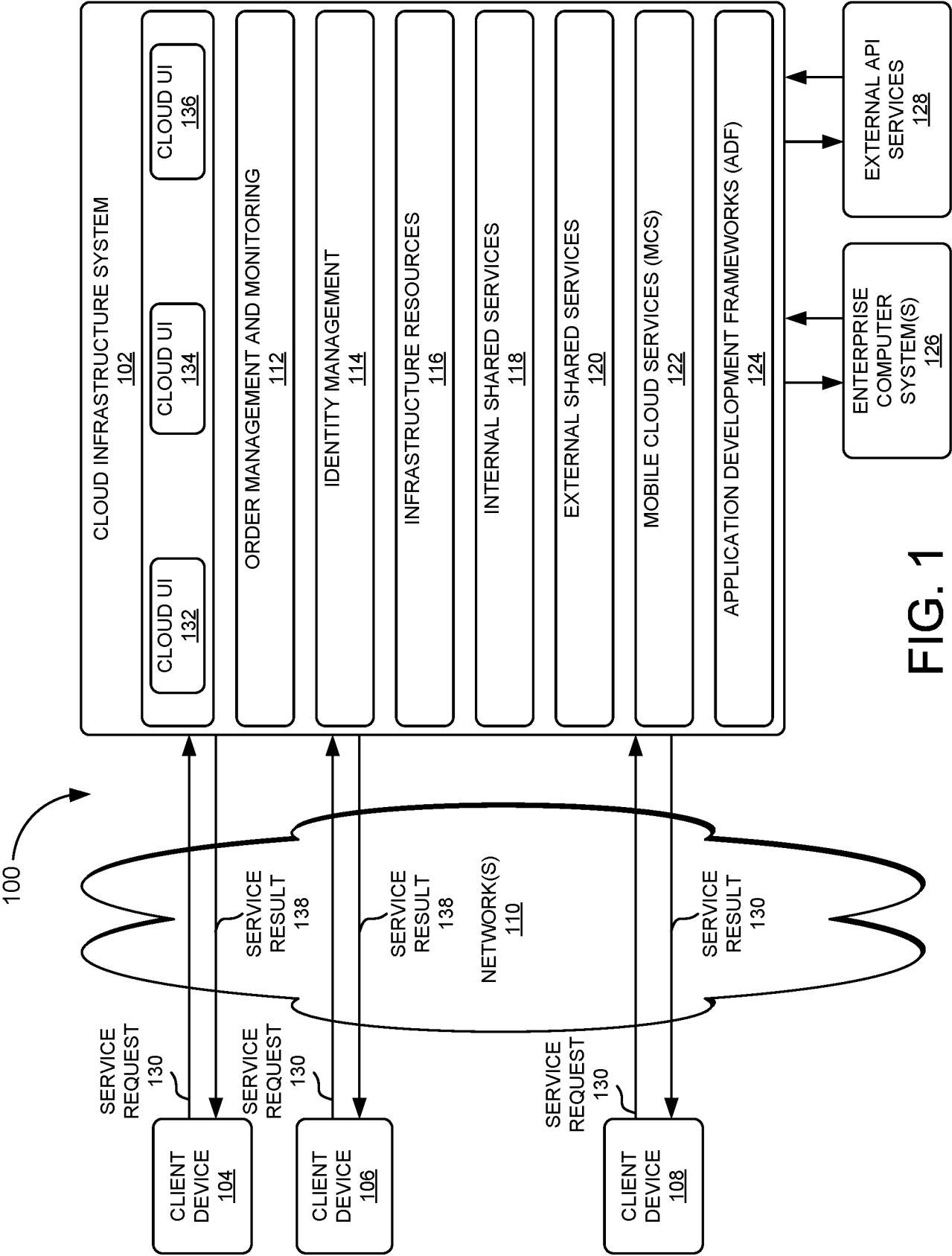


FIG. 1

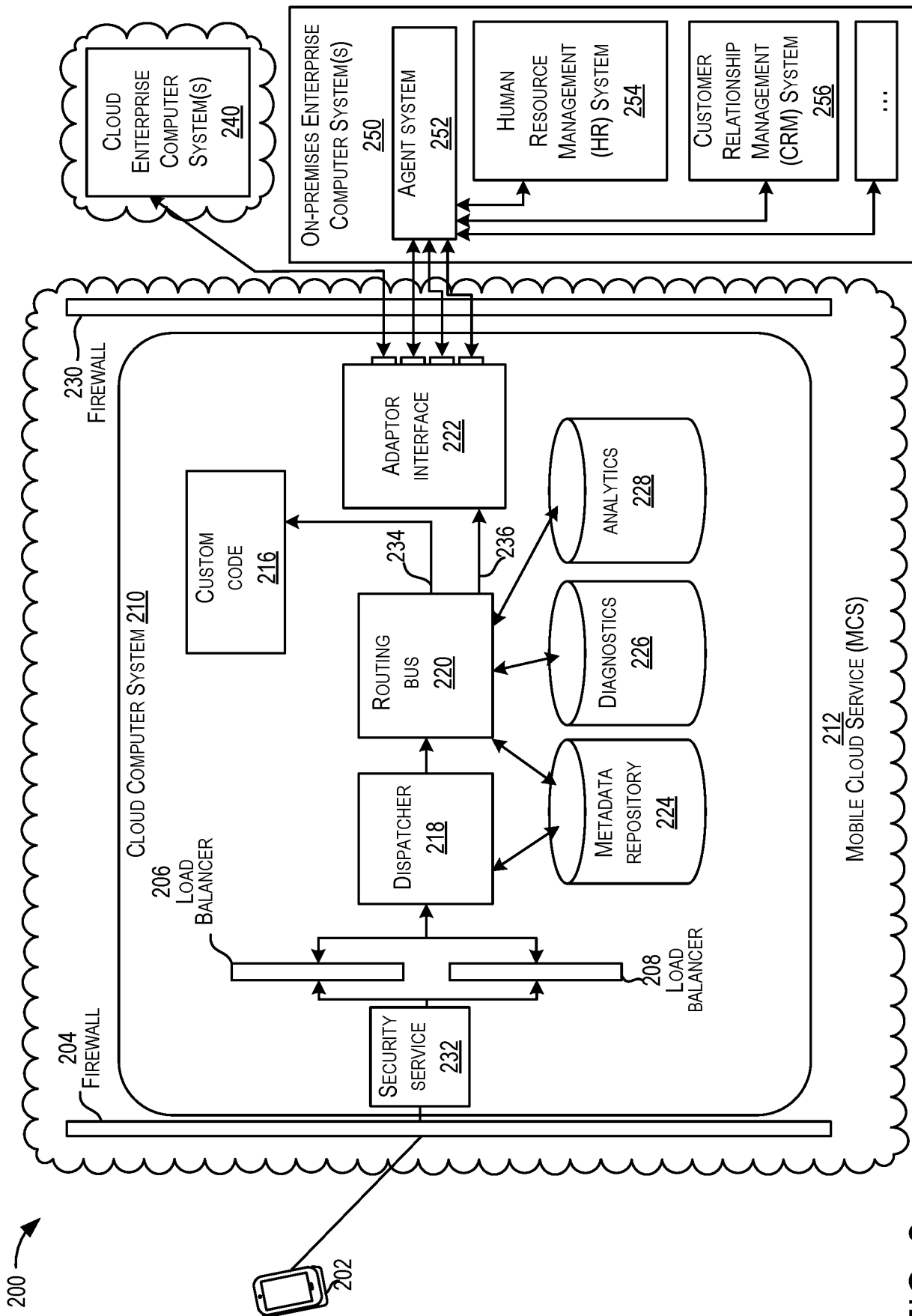


FIG. 2

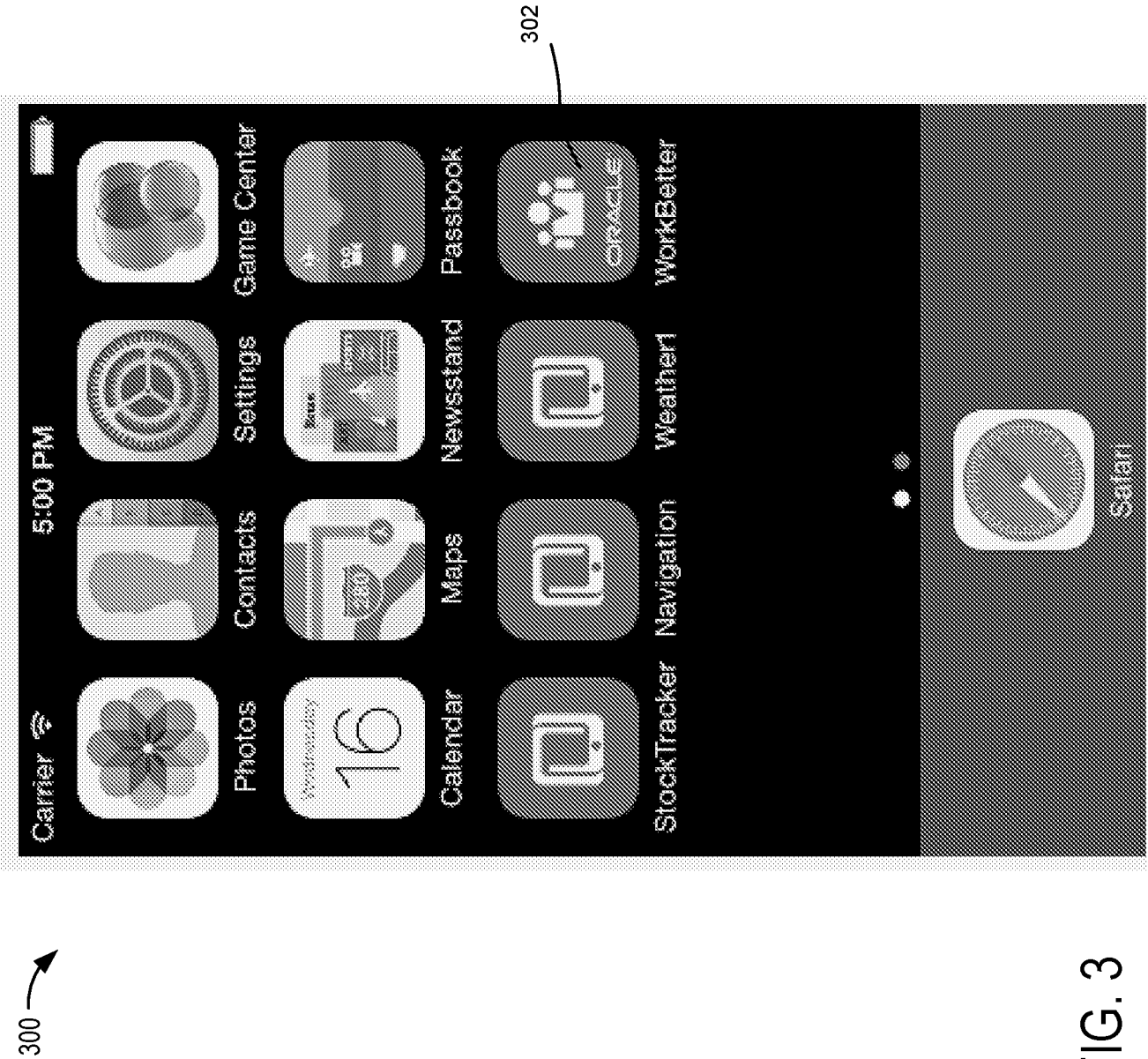


FIG. 3

304 →

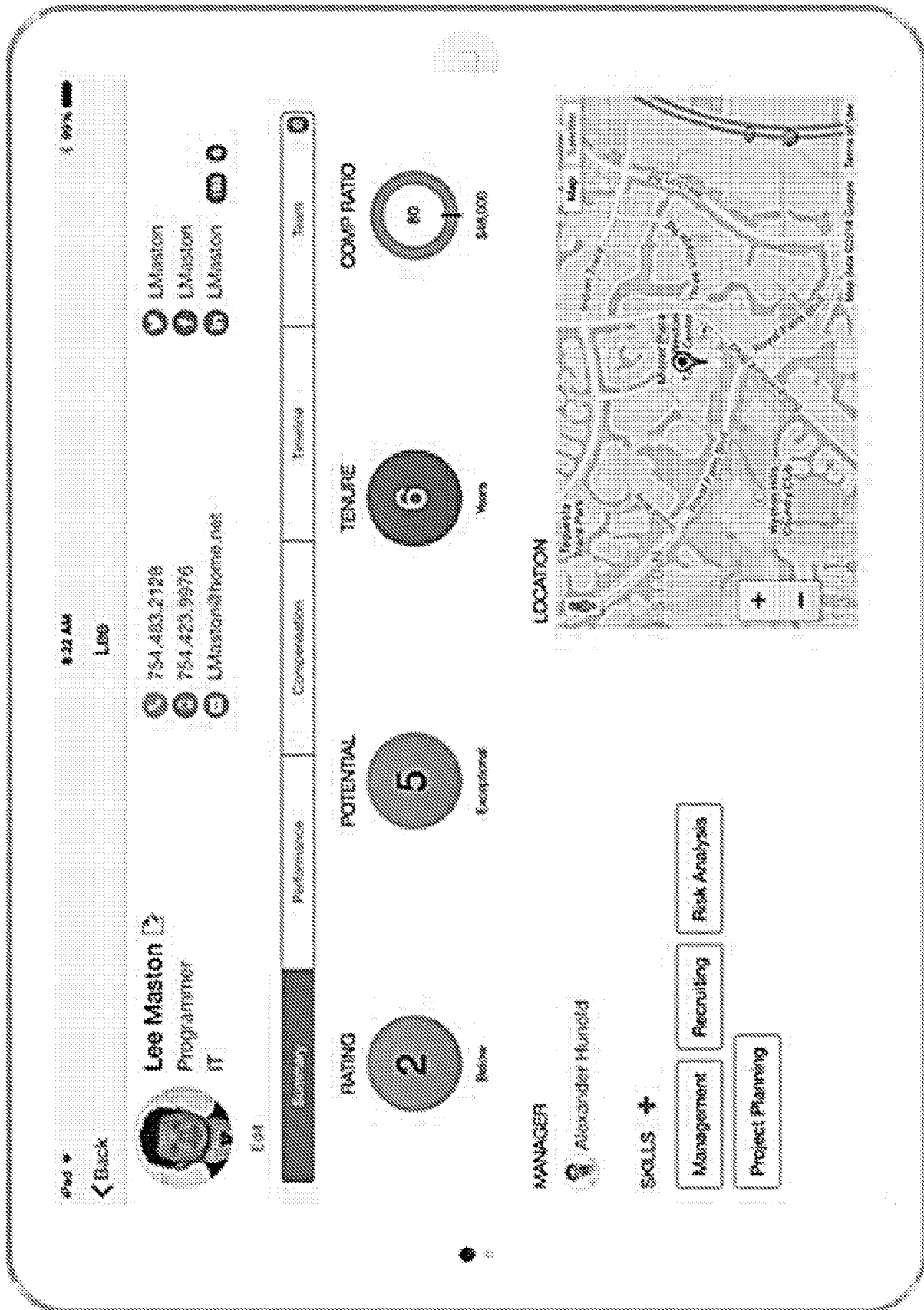


FIG. 3A

304

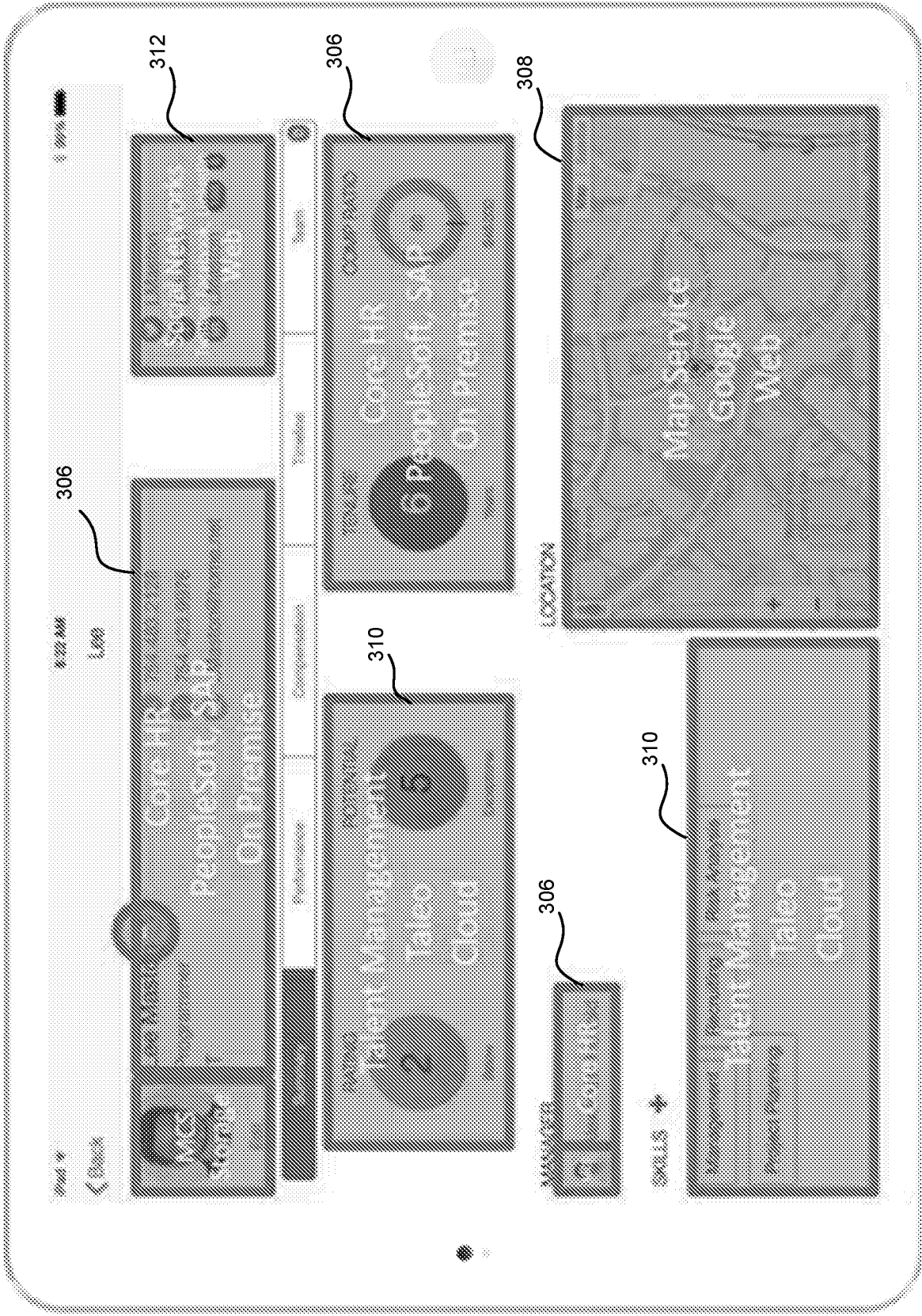


FIG. 3B

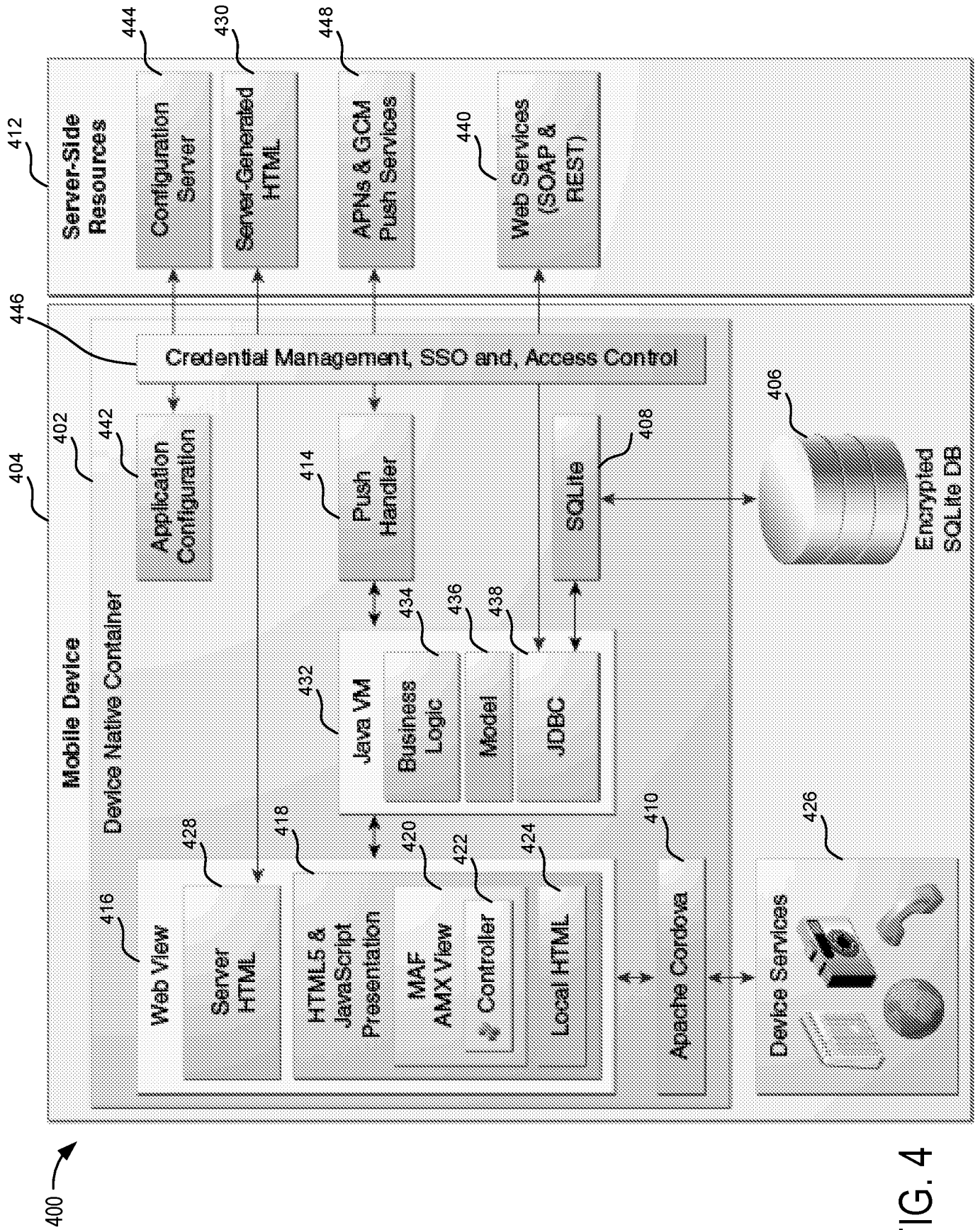


FIG. 4

500

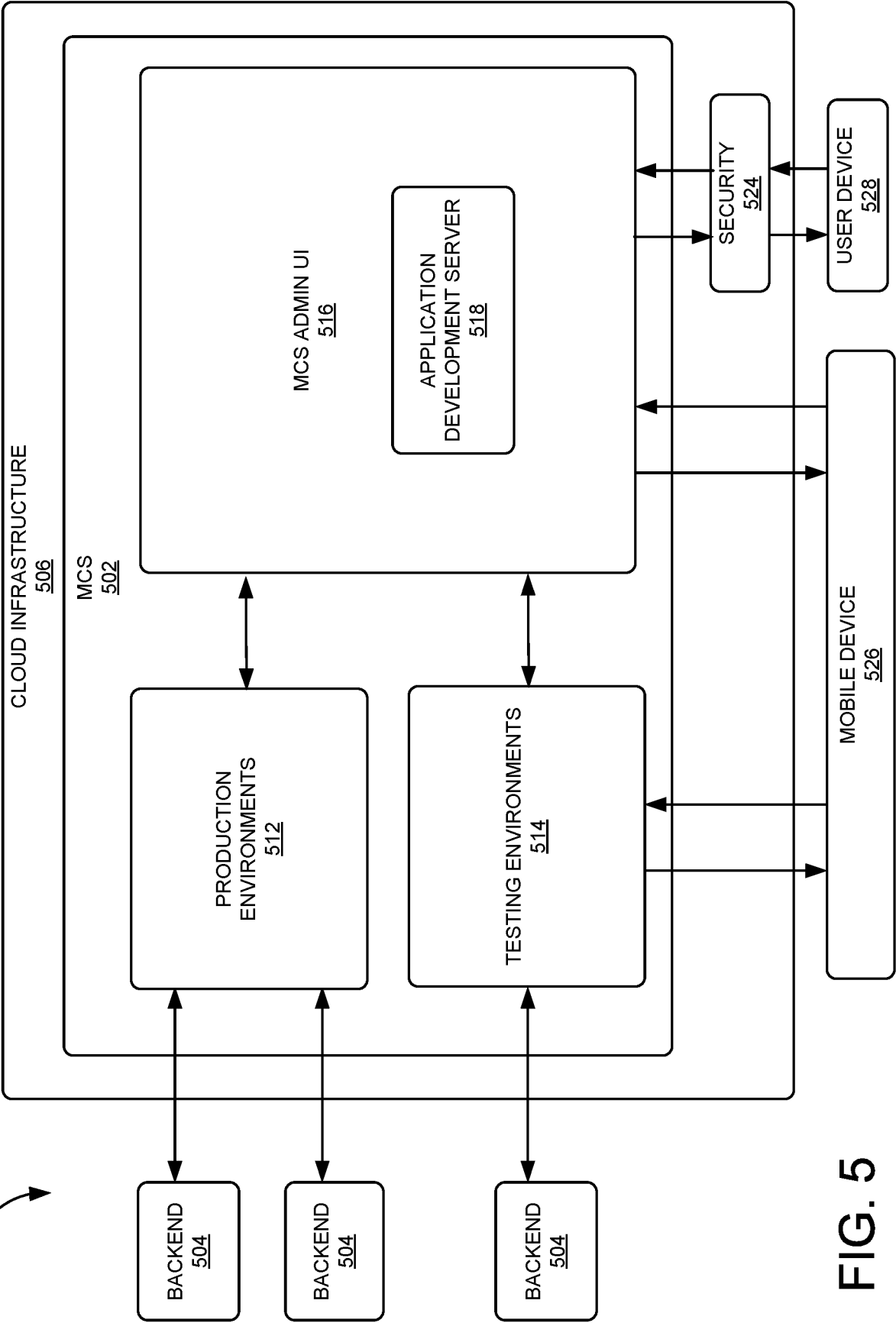


FIG. 5

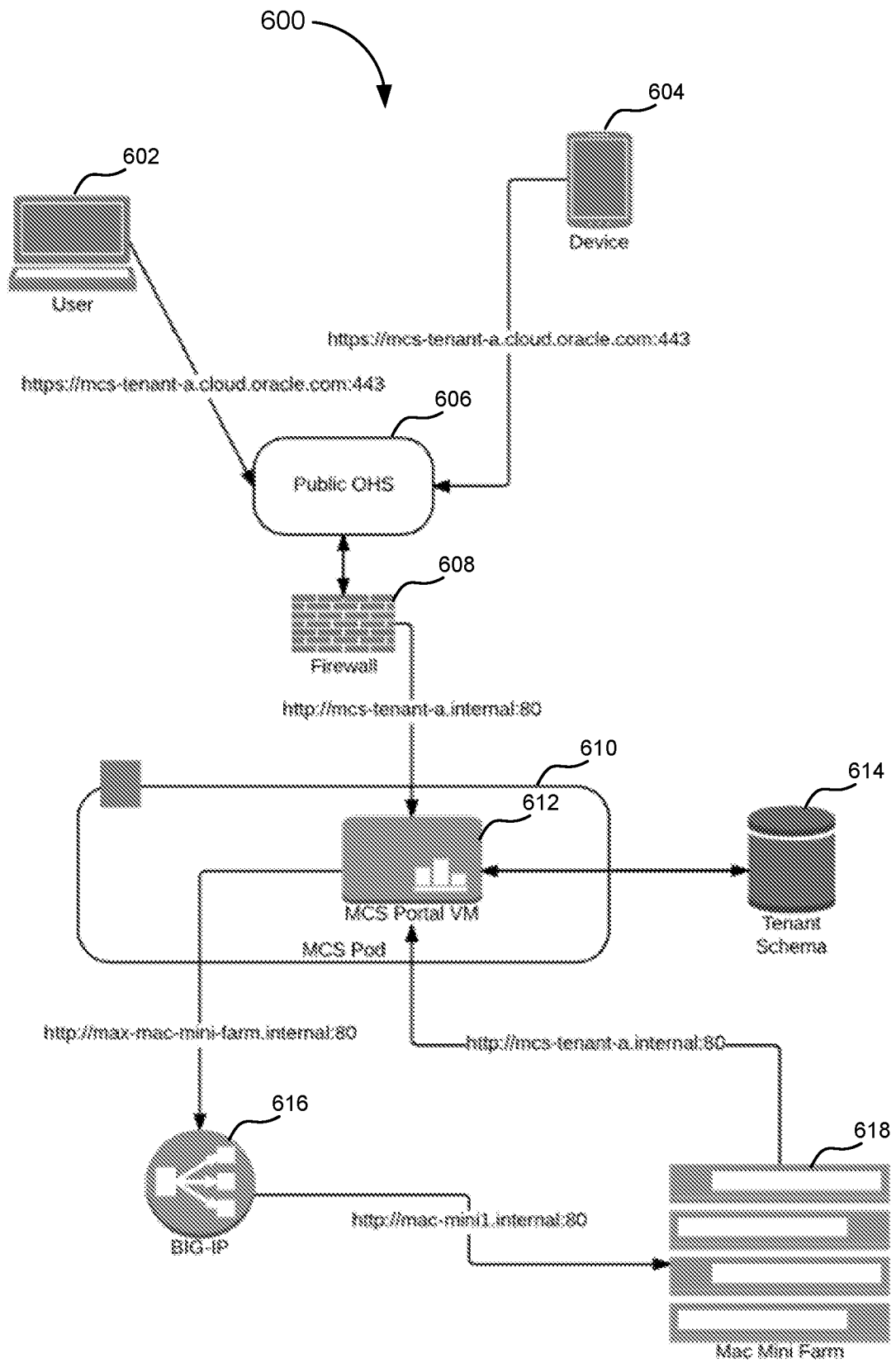


FIG. 6

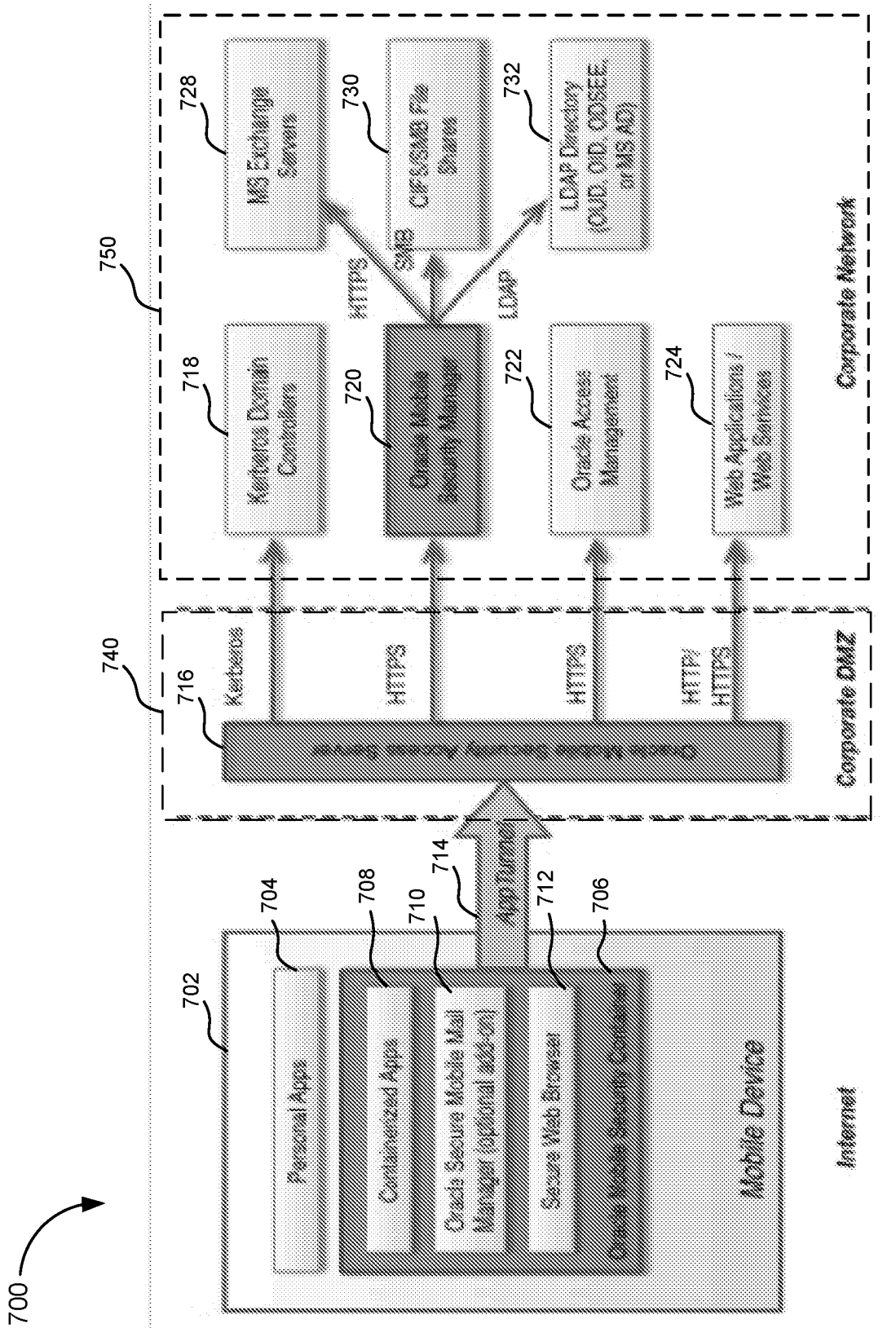
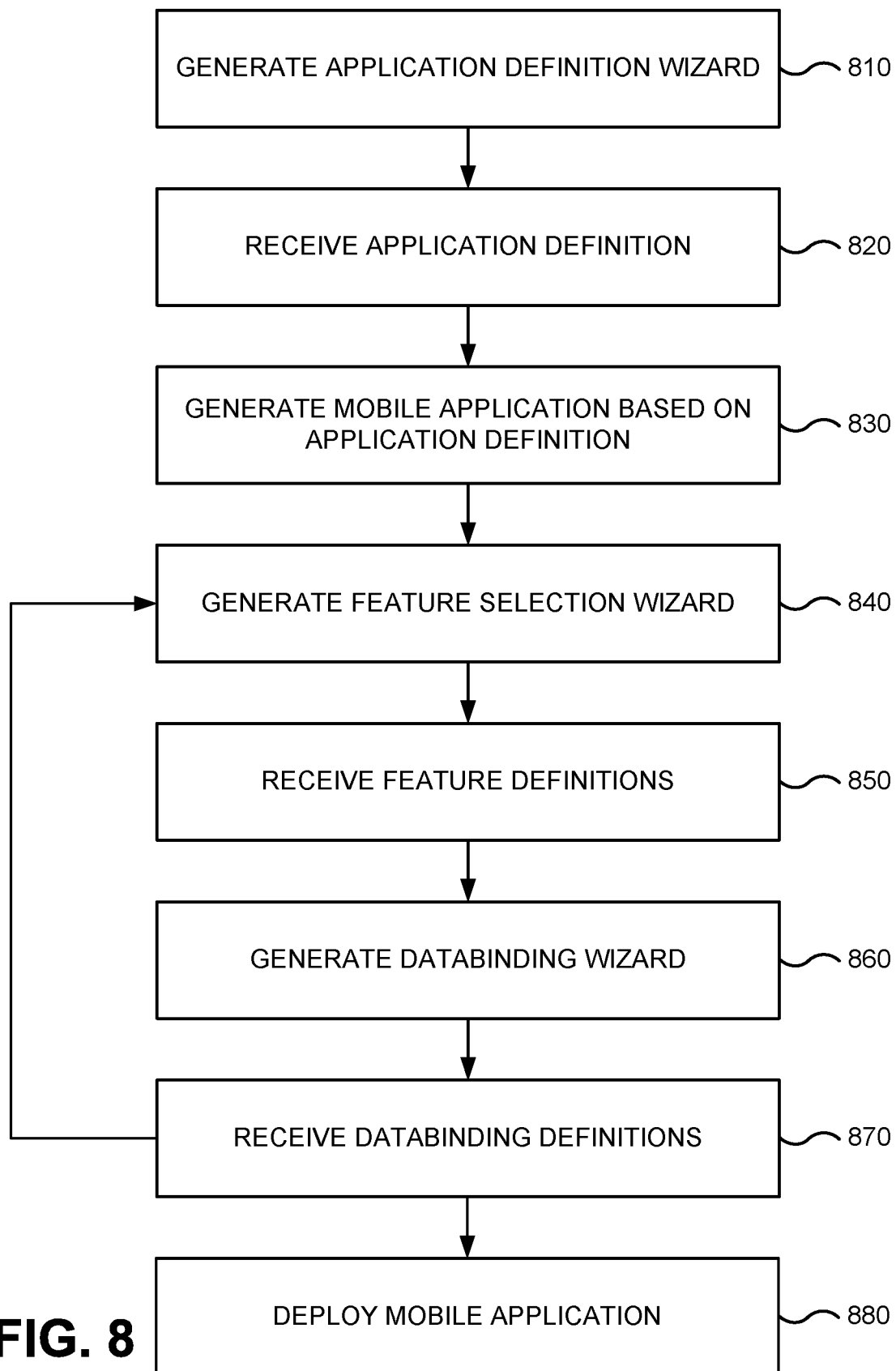
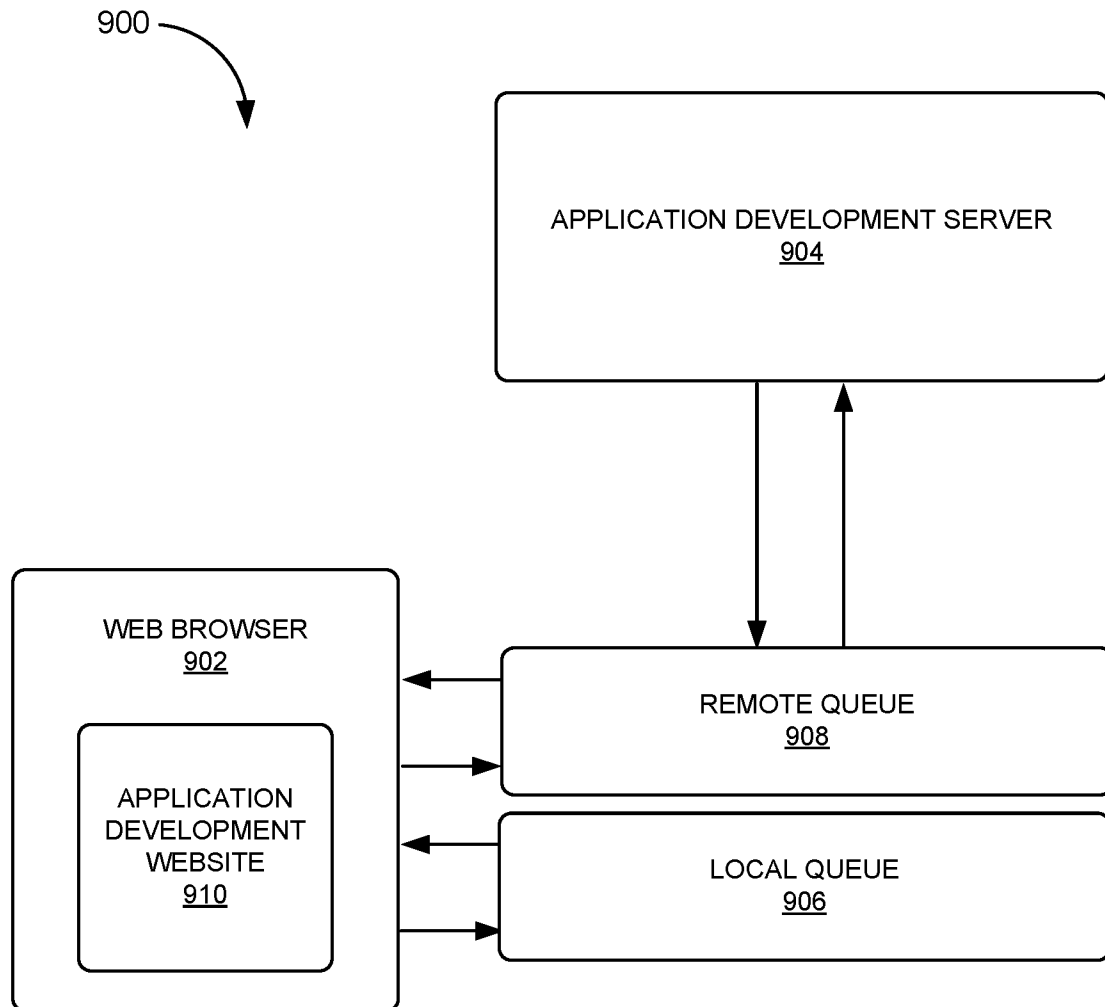
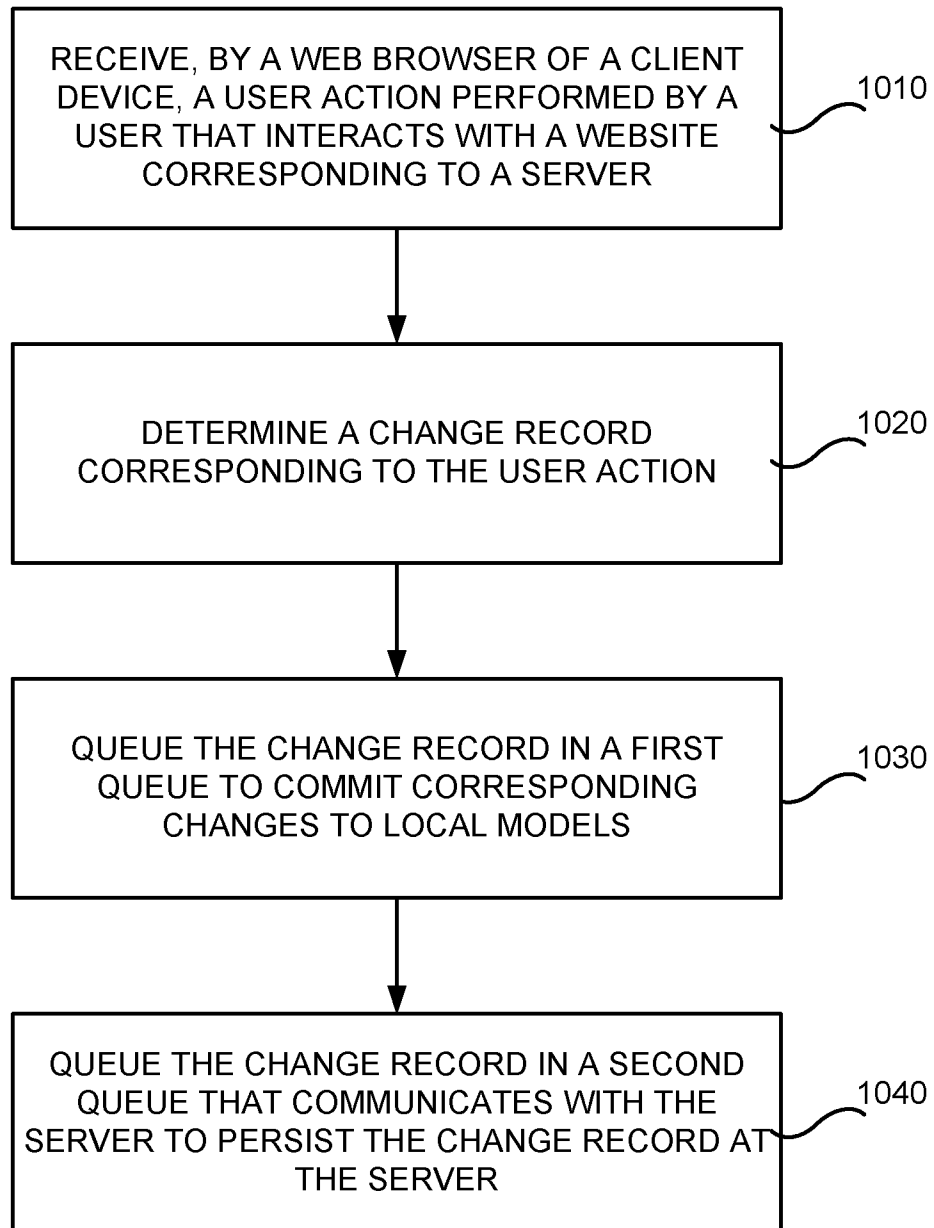


FIG. 7



**FIG. 9**

**FIG. 10**

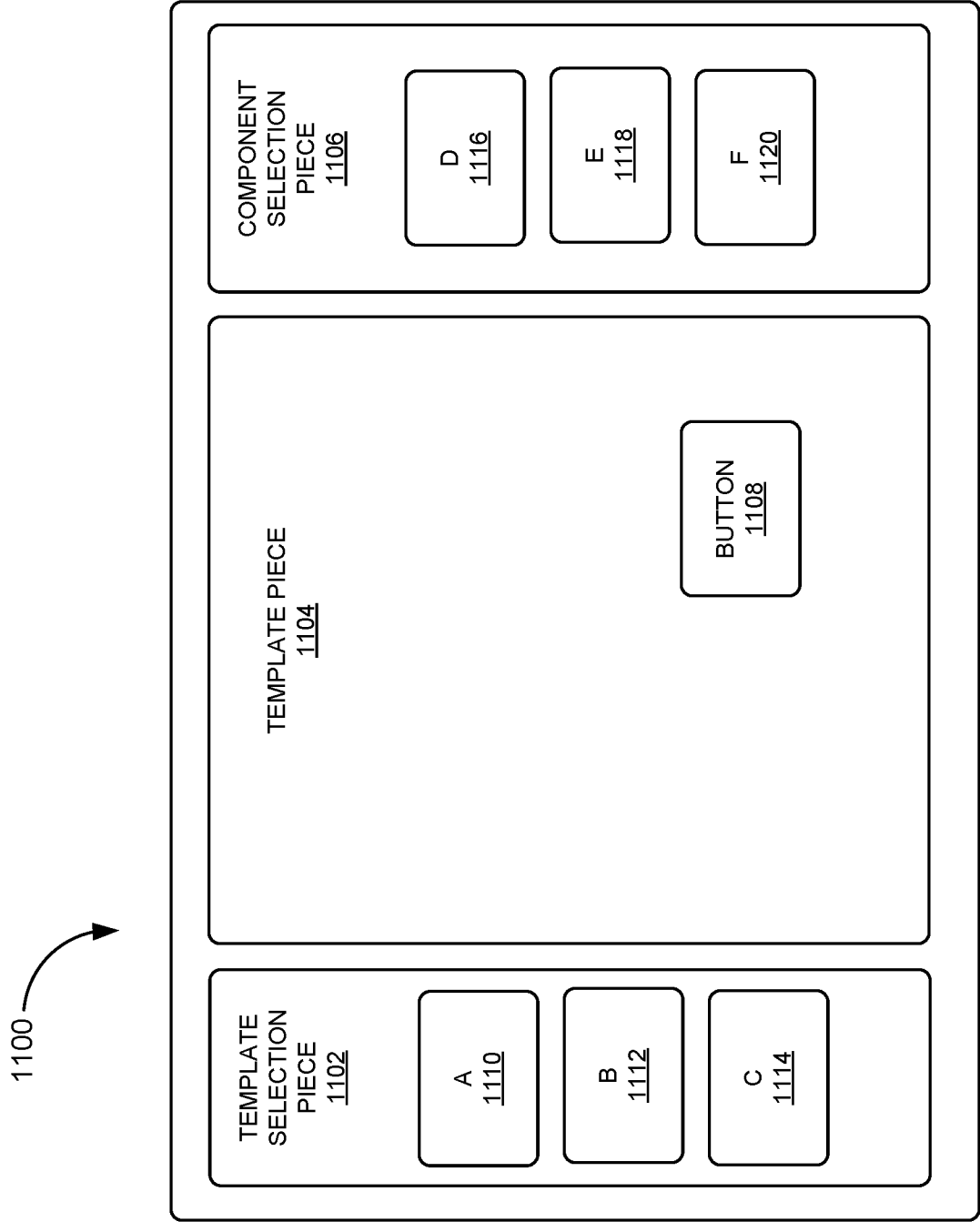
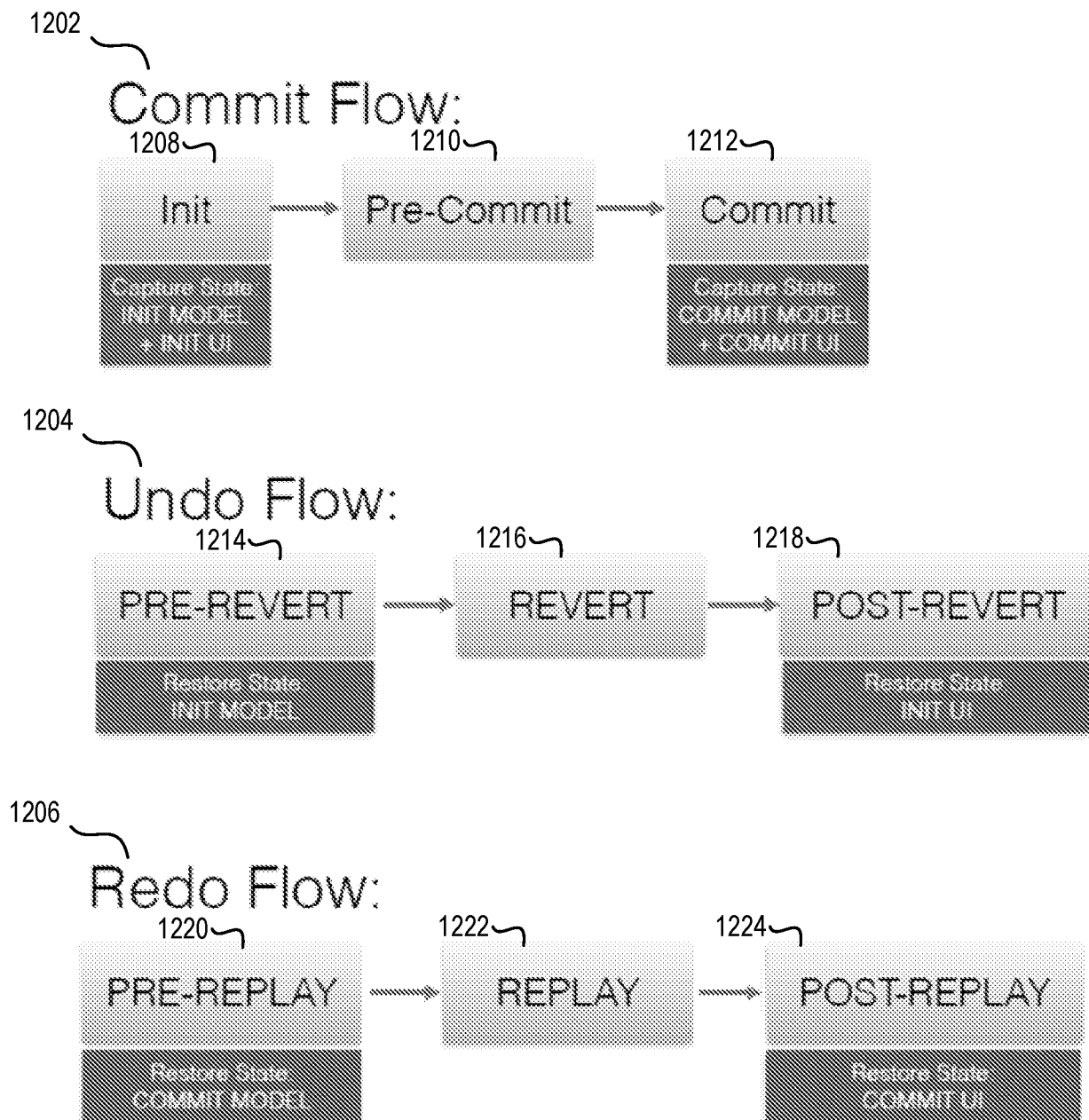
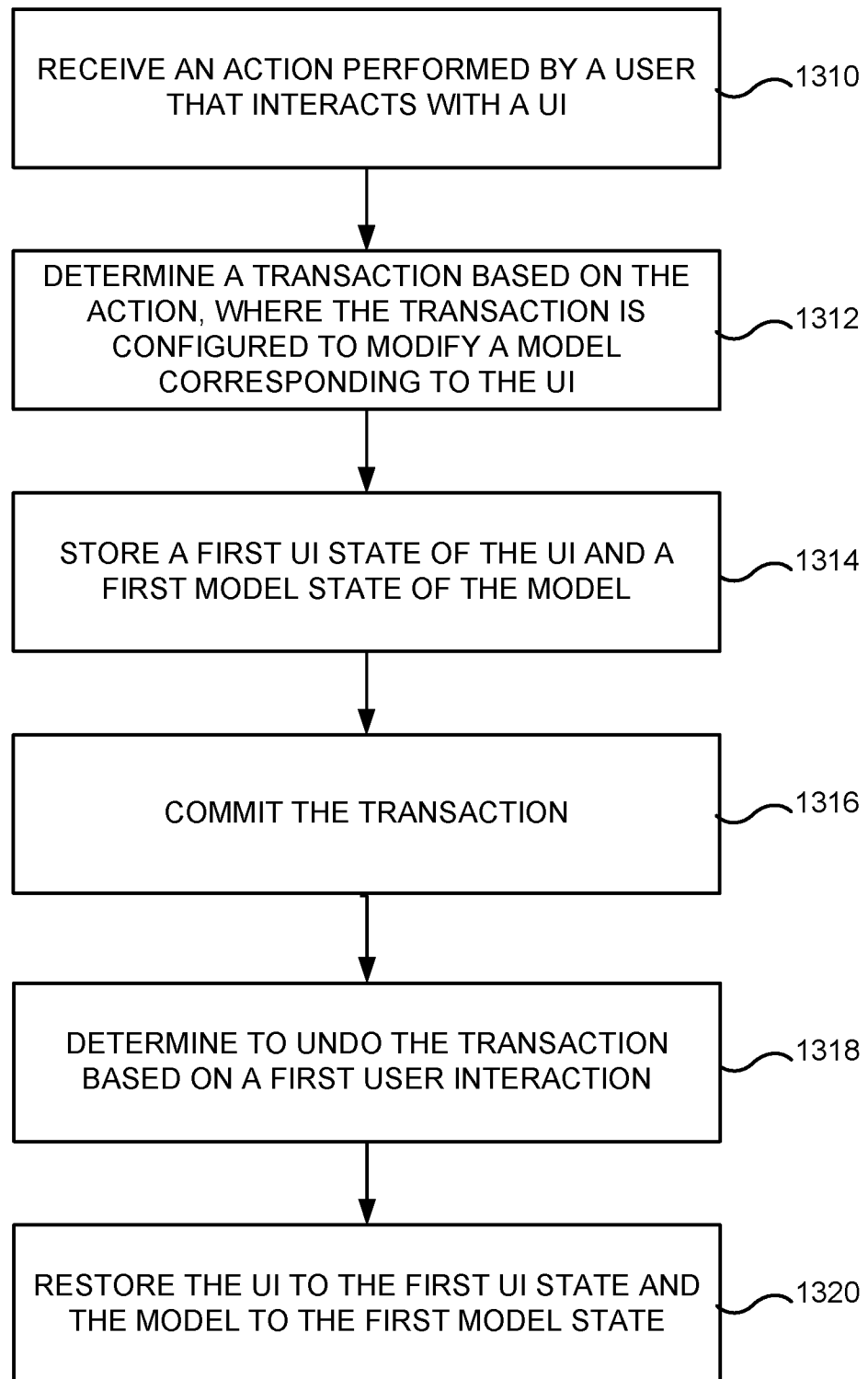


FIG. 11

**FIG. 12**

**FIG. 13**

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2016/033422

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F17/30
ADD.

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

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)

EPO-Internal, INSPEC

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2010/293080 A1 (SHAH MOHAMMED KAMRAN [US]) 18 November 2010 (2010-11-18) abstract paragraph [0009] - paragraph [0015]; figures 5A, 5B paragraph [0353] - paragraph [0364] -----	1-20
A	US 2009/157811 A1 (BAILOR JONATHAN BECKETT [US] ET AL) 18 June 2009 (2009-06-18) abstract paragraph [0028] - paragraph [0029]; figures 1, 2 paragraph [0046] - paragraph [0051]; figure 5 paragraph [0056] - paragraph [0062]; figure 7 paragraph [0092] - paragraph [0098]; figure 20 -----	1-20



Further documents are listed in the continuation of Box C.



See patent family annex.

* Special categories of cited documents :

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

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

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

14 September 2016

Date of mailing of the international search report

27/09/2016

Name and mailing address of the ISA/

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040,
Fax: (+31-70) 340-3016

Authorized officer

Barieux, Marc

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/US2016/033422

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2010293080	A1	18-11-2010	EP 2433213 A1 28-03-2012
			EP 2433214 A1 28-03-2012
			US 2010293080 A1 18-11-2010
			US 2010293481 A1 18-11-2010
			US 2010293482 A1 18-11-2010
			US 2010293483 A1 18-11-2010
			US 2010293484 A1 18-11-2010
			US 2010293521 A1 18-11-2010
			US 2010293522 A1 18-11-2010
			US 2010293525 A1 18-11-2010
			US 2010293526 A1 18-11-2010
			US 2010293527 A1 18-11-2010
			US 2010293528 A1 18-11-2010
			US 2010293529 A1 18-11-2010
			US 2010293591 A1 18-11-2010
			US 2014096108 A1 03-04-2014
			WO 2010135336 A1 25-11-2010
			WO 2010135353 A1 25-11-2010
US 2009157811	A1	18-06-2009	AU 2008338826 A1 25-06-2009
			BR PI0819825 A2 26-05-2015
			CN 101896915 A 24-11-2010
			EP 2220581 A2 25-08-2010
			JP 5250637 B2 31-07-2013
			JP 2011507100 A 03-03-2011
			KR 20100095583 A 31-08-2010
			RU 2010123793 A 20-12-2011
			TW 200925887 A 16-06-2009
			TW 201506641 A 16-02-2015
			US 2009157811 A1 18-06-2009
			US 2014373108 A1 18-12-2014
			WO 2009079116 A2 25-06-2009