# METHOD AND APPARATUS FOR CODE VIRTUALIZATION AND REMOTE PROCESS CALL GENERATION

## BACKGROUND OF THE INVENTION

### Field of the Invention

[0001] Embodiments of the present invention generally relate to a method and system for process call code-generation.

### Description of the Related Art

[0002] The growth and integration of computer programming languages has expanded many capabilities of existing languages. Each language offers particular advantages and disadvantages in terms of respective size, execution speed, and various functionality within platforms and devices. As such, some specific languages are made to only work with corresponding devices and users must often be familiar or even be experts with multiple programming languages. In addition, the nature of building modern applications has forced developers to learn multiple languages. Such that writing an application is no longer limited to using one programming language. Integrating and learning multiple languages is a difficult burden, often requiring months or years of training for a programmer/developer to reach a sufficient coding proficiency.

[0003] Applications may be composed of various languages and if deployed on servers, requires programmers to learn a front end language, a server language, and a back end language. Front end technologies are also made of several different languages (HTML, Javascript, etc.) to further add to programming complexity.

[0004] In addition, more computing occurs over a network and/or the Internet (i.e., "CLOUD computing"), programming with various software development kits (SDKs) and application programming interfaces (APIs) becomes a cumbersome necessity for developers. Emerging services such as social media (TWITTER, FACEBOOK, and the like), data storage (DROPBOX, CLOUD DRIVE, and the like), and financial transactions (PAYPAL, AMAZON, EBAY, and the like) also increase

the programming burden on programmers as each service may require its own SDK or API to implement processes offered by the service. For example, TWEETING on TWITTER requires a different code base than for posting a picture on TUMBLR. Using these services also may not be conveniently shown until after applications are deployed on the service, which also may have irreparable results. In other words, programming across multiple platforms, languages, and services remains exceedingly complex to learn in combination with understanding accompanying SDKs and APIs of remote services.

**[0005]** Thus, there is a need in the art for an integrating a local and remote programming environment.

## SUMMARY OF THE INVENTION

**[0006]** Embodiments of the present invention generally relate to a method, apparatus, and system for code virtualization and remote process call generation. Specifically, by utilizing a pre-generated code library, virtualized SDK, and snippet generator located on a remote server, a program on a user device leverages multiple services across multiple platforms and programming languages. The user device receives a code snippet and/or virtualized SDK so as to execute at least one remote process call on the remote server, and also in turn deploy the process in the native language of a different remote server substantially as shown in and/or described in connection with at least one of the figures, as set forth more completely in the claims.

**[0007]** These and other features and advantages of the present disclosure may be appreciated from a review of the following detailed description of the present disclosure, along with the accompanying figures in which like reference numerals refer to like parts throughout.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0008]** So that the manner in which the above recited features of the present invention can be understood in detail, a more particular description of the invention, briefly summarized above, may be had by reference to embodiments, some of which are illustrated in the appended drawings. It is to be noted, however, that the appended drawings illustrate only typical embodiments of this invention and are therefore not to be considered limiting of its scope, for the invention may admit to other equally effective embodiments.

**[0009]** Figure 1 is an illustration of a code virtualization system in accordance with one or more exemplary embodiments of the present invention;

**[0010]** Figure 2 illustrates a platform to execute classical service components that are interfaced with CLOUD clients using code virtual code providers (VCPs) in accordance with one or more exemplary embodiments of the present invention;

**[0011]** Figure 3 is an illustration of an overview method using the virtual code providers disclosed in Figure 2 interfacing with an integrated development environment in accordance with one or more exemplary embodiments of the present invention;

**[0012]** Figure 4 is a flow diagram for a method of creating process call snippets in accordance with one or more exemplary embodiments of the present invention;

**[0013]** Figure 5 is a flow diagram of a method for expanding a local development environment with additional remote processes in accordance with one or more exemplary embodiments of the present invention;

**[0014]** Figure 6 is a flow diagram of a method for a virtualized process call code-generator for various programming languages in accordance with one or more exemplary embodiments of the present invention; and

[0015] Figure 7 is a block diagram of modules within an exemplary computing server in accordance with one or more exemplary embodiments of the present invention.

## DETAILED DESCRIPTION

[0016] In the described following embodiments, a "process" shall refer to a ready-to-run code base or pre-generated process code that is already stored on an execution computer (e.g., server, personal computer, mobile device, etc.). Each process is uniquely identified by name, process ID, icon, or other identifier to the user of an integrated development environment (IDE). "Execution" shall refer to the launch of a process on a computer (local or remote, or CLOUD based). A "bundle" shall refer to a collection of processes or functions organized by a common attribute (API name, common parameters, SDK, parameter names, and the like). A "snippet" shall refer to a generated block of customized code to be inserted into a program with a corresponding programming language selected by a user for calling a remote process. Often, the language selected is that of a programming language specified by the user to be implemented in an application and/or IDE on a user device.

[0017] Embodiments disclosed herein relate to a code library on a remote server (e.g. "in the CLOUD") wherein the user can remotely select a process and then download a corresponding snippet and/or SDK to access the process from a local application or software program. The snippet allows for a remote process call on the remote server to execute a process on the server or deploy the process on a different remote server or remote. The user device uses a virtualized SDK such that the snippet may communicate with a remote platform on the execution computer. The remote platform includes virtualized components of additional services (e.g., compilers, e-mail, AMAZON, TWITTER, EBAY, DROPBOX, and the like) along with corresponding SDKs and APIs for the services. A computing platform includes a hardware architecture and a software framework (including application frameworks),

where the combination allows software to run. Typical platforms include a computer architecture, operating system and Runtime libraries.

**[0018]** A typical SDK includes many tools, libraries, documentation and sample code that would help a programmer to develop an application for a software package, software framework, hardware platform, computer system, video game console, operating system, or similar development platform. An API defines a set of rules that should be followed by the programs to communicate with one another. The rules specify how the routines, data structures, etc. should be defined in order for two applications to communicate with each another. The API is an interface that allows software programs to interact with each other, whereas a SDK is a set of tools that can be used to develop software applications targeting a specific platform. In some instances, an SDK may include an API.

**[0019]** Figure 1 is an illustration of a code virtualization system 100 in accordance with one or more embodiments of the present invention. The system 100 comprises at least one user device (102a...102n), a network 110, and at least one remote server (104a...104n). The at least one user device (102a...102n) comprises an electronic device capable of storing and executing a program such as a personal computer, server, virtual server, microcontroller (MCU) and controller, sensor, actuator, edge device connected to a gateway, application-specific integrated circuit (ASIC), micro-processor and processor, cellular phone, smart phone, tablet, laptop, and the like.

**[0020]** Each user device (102a...102n) comprises a corresponding memory (112a...112n) storing code necessary to connect to and authenticate into the network 110 to connect to at least one remote server (104a...104n). In some embodiments, the memory (112a...112n) stores an IDE for receiving and deploying snippets from the at least one remote server (104a...104n).

[0021] The network 110 may comprise at least one of or combination of a wired and/or wireless network such as: a local area network (LAN), wide area network (WAN), wireless local area network WLAN, cellular network, Wi-Fi, Internet, or Intranet networks. The at least one remote server (104a...104n) may comprise a remote application server, web server, mail server, host server, database server, storage server, and the like. The at least one remote server (104a...104n) comprises memory (114a...114n) to store a platform 120 and execute particular process calls such through a SDK generator 122, a snippet generator 124, and an execution engine 126. In some embodiments, the snippet generator 124 may also be stored in the user device (102a...102n) for local snippet generation on the user device (102a...102n). The memory (114a...114n) may also store a user interface module, database of user profiles/accounts, collection of processes, SDKs, and associated parameters to link and execute the processes across other servers 104n (e.g., FACEBOOK, TWITTER, e-mail, SMS, MMS, FTP, and the like) on the network 110.

[0022] The execution engine 126 processes calls to routines in the libraries by correlating an identified routine called and running/executing the process using the call parameters. In some embodiments, the execution engine 126 may send output to another process call or the original process call.

[0023] In some embodiments, process call parameters are stored either on user device 102a or remote server 104a. In other embodiments, parameters for the process calls are specified directly by the user via pre-generated pre-defined fields presented to the user specified in a graphical user interface. In some embodiments, the graphical user interface is part of an IDE. The IDE utilizing snippets generated by the remote server 104a for integration into a program of a user selected programming language on the user device 102a.

[0024] In some embodiments, the IDE also executes remote process calls using the snippets on the server 104a so as to deploy additional processes and access

services offered on other servers 104n. The execution is facilitated by a common virtualized SDK and platform that links the remote process calls to a corresponding virtual code provider discussed further below in Figure 2.

**[0025]** Figure 2 illustrates a platform 200 to execute classical service components (CSCs) $210_1 \ldots 210_N$ that are interfaced with CLOUD clients 205 using code virtual code providers (VCPs) $245_1 \ldots 245_N$, in accordance with one or more embodiments of the present invention. The platform corresponding to platform 120 that may be stored on the remote server 104a. Hereinafter classical service components $210_1 \ldots 210_N$ are collectively referred to as CSCs 210 and similarly virtual code providers $245_1 \ldots 245_N$ are collectively referred to as VCPs 245. The platform 200 operatively uses a virtualized SDK that links remote process calls from user devices 102 to the CSCs $210_1 \ldots 210_N$ via CLOUD client(s) 205 and the VCPs 245.

**[0026]** The CSC $210_1$ comprises a remote interface $206_1$, an application layer $208_1$, virtualization layer $212_1$, and physical equipment $215_1$. Each CSC 210 represents a specific service offered by the platform 200. For example CSC $210_1$ may be an Internet service and corresponding servers (e.g., FACEBOOK, EBAY, POP3 e-mail, TWITTER, AMAZON, SMS, FTP, HTTP, and the like) and CSC $210_2$ may be a computer offering conventional local data information processing (e.g., image rendering, code compiling, data analytics, data storage, and the like).

**[0027]** Physical equipment $215_1$ or machines (CPUs, memory, network routers, and the like) are virtualized in a virtualization layer $212_1$ to make processing and storage resources available to the application layer $208_1$. Virtualization creates redundancy and increased operating bandwidth in a single computer, and even more in embodiments where multiple machines are virtualized. In such embodiments, the processing workload can also be allocated to various equipment based on priority and independent of the physical location.

**[0028]** An application layer $208_1$ allows execution runtime of programs and functionalities such as database operations (e.g., storing account profiles, warehousing data, and the like) on the virtualized machines. A remote interface layer $206_1$ comprises code to access web servers and utilize APIs. The remote interface $206_1$ communicates with the CLOUD client(s) 205 (e.g., a thin client, web browser, mobile application, terminal emulator program, and the like).

**[0029]** The CLOUD client 205 facilitates communication of data to and from the VCPs 245 with the application layer $208_1$. The CLOUD client 205 simplifies the layer interfaces $206_1$, $208_1$, $212_1$, and $215_1$ by presenting a web browser, terminal emulator or other client interface. Aforementioned embodiments of the present invention may utilize the CLOUD client 205 to facilitate remote process call code generation of snippets on the platform 200 using the virtualized SDK. The virtualized SDK links process calls generated and offered by VCPs 245 to the application layer $208_1$.

**[0030]** The VCPs 245 are enhanced hosted code libraries allowing for virtual sandboxing, normalization, universal language, results parsing, automatic credential verification, simplified oath process, platform maintenance and monitoring. Testing or “sandboxing” the program prior to full deployment of the process and application/program to a remote server has become increasingly important with the addition of multiple remote processes and services contained within a program. The VCPs 245 coordinate snippet process calls using a virtualized SDK that is downloaded to the user device 102a.

**[0031]** From the perspective of the user device 102a, the VCPs 245 allow for a universal programming language offered through the virtualized SDK by templating languages such that only component parameters are needed and inserted into pre-generated code. Process results are thus also capable of being quickly parsed. Access is also streamlined by the platform 200 using pre-authentication and stored centralized management of keys to access remote services. Pre-authentication also

produces more secure code by preventing of reverse-engineering of code communicated between the user device 102a and platform 200 that may expose sensitive information. The VCPs 245 also normalize programming by reducing processes to the snippet format. Thus, snippets represent a standardized way of writing code.

**[0032]** For example, 5 lines of code to call any process in the CLOUD is considered as code normalization. The VCPs 245 allow the platform 200 to display real-time monitoring and logging of how the created application is performing (e.g., usage statistics, error handling, and the like). The platform 200 also uses the monitoring to maintain the SDKs of respective VCPs such that APIs in the process pools 230 are continually improving to include new methods, versions and parameters.

**[0033]** The VCP $245_1$ comprises a process pool $230_1$, an authentication module $228_1$, a SDK linking module $225_1$, a snippet generator $124_1$, a user database $232_1$, and an execution engine $126_1$. In some embodiments, a single execution engine 126 may deploy all processes.

**[0034]** For ease of explanation, each of the VCPs 245 corresponds to CSCs 210 in a one-to-one correspondence. However, alternative embodiments may include a different ratio or a single VCP 245 representing the processes of all offered by all CSCs 210. VCPs 245 are stored on remote server(s) 104a in accordance with one or more embodiments of the present invention. The VCPs 245 reduce the remote services offered on the platform 200 of the CSCs 210 to a limited number of executable pre-determined remote processes stored in the process pool 230.

**[0035]** A remote process corresponds to a desired action that may be performed on a given CSC $210_N$. For example, processes for a CSC $210_1$ of FACEBOOK would have a limited process pool $230_1$ of posting a comment, posting a picture, liking a comment or picture, declining a friend request, initiating a friend request,

deleting a post and other actions desired by a user. The number of processes in a given process pool 230 is thus limited to the actions available by a service associated with the CSC $210_1$. Continuing the example, there would not be a process in the platform 200 to place a review of a product on FACEBOOK. However, such a process would be processed by the platform 200 as available on AMAZON.

**[0036]** The process pool $230_1$ this stores all pre-generated code associated with each process offered by the remote platform 200 and corresponding CSCs 210. In some embodiments, the pre-generated code is in the native language necessary for the remote application layer $208_1$ to recognize the pre-generated code processes. In other embodiments, the programming language may be native to a user device 102a for an IDE.

**[0037]** The snippet generator $124_1$ generates a snippet by inserting parameters entered by a user as well as process names to create a snippet with a corresponding SDK such that the snippet may call the associated process from the process pool $230_1$. The snippet parameters may be entered using a GUI comprising predetermined fields associated a selected remote process. The snippet parameters may also be retrieved as previously stored in the user database $232_1$. The user database $232_1$ comprising user profiles, histories, and stored credentials for various service bundles. In other embodiments, the snippet may be generated locally on the user device 102a.

**[0038]** In alternative embodiments, the snippet generator $124_1$ generates a snippet when called using an API call such as through the simple object access protocol (SOAP), representational state transfer (REST) model, or cURL.

**[0039]** The generated snippet is in a programming language specified by the user or required by the platform 200. For example, a user may have a pre-existing program on the user device 102a written in JAVA. An associated snippet would

need to have a process call to the virtualized SDK and platform 200 in the JAVA syntax and corresponding libraries on the user device 102a in order to directly insert the snippet into the pre-existing program.

**[0040]** The authentication module $228_1$ receives credentials from a user or uses previously entered and stored credentials from a user to access a CSC $210_1$ associated with the VCP $245_1$. The authentication module $228_1$ may authenticate also into the platform 200. The user database $232_1$ is the account database for a given service of a CSC $210_1$. In other embodiments, the user database $232_1$ is a collection of user accounts for authentication into the platform 200. Subsequently, the platform 200 coordinates the authentication across multiple VCPs 245 and associated CSCs 210. For example, a user logging into the platform 200 of a remote server (e.g., TEMBOO servers) similarly will authenticate into an e-mail server and EBAY account using pre-defined credentials stored on the remote server 104a previously submitted by the user.

**[0041]** The SDK linking module $225_1$ is comprised of code to allow the authentication module $228_1$ to interface with the network 110 as well as communicating with the user device 102a and APIs to form various bundles of remote processes. The SDK linking module $225_1$ also coordinates the communication with multiple SDKs and APIs across multiple VCPs 245 for cross-platform and cross-service interactions. For example, the SDK linking module $225_1$ coordinates process pools 230 and corresponding SDKs (e.g., EBAY SDK, TWITTER SDK, SMS SDK, and the like) to form a combined single process call in the virtualized SDK. Continuing the example, by linking processes from respective process pools, a single process call with the virtualized SDK may substantially simultaneously TWEET and SMS text a winning auction on EBAY price matched to a product sold on AMAZON.

**[0042]** Lastly, the execution engine $126_1$ executes the selected process from the process pool $230_1$, deploys the process onto another remote server 104n or

corresponding CSC $210_1$ to execute the remote process. In some embodiments, the execution engine $126_1$ may also return results to the user database or the user device 102a. In further embodiments, the execution engine $126_1$ also executes the remote process from the process pool $230_1$ when the created snippet is used to call the process when imbedded into the local code base or pre-existing program on the user device 102a. Further still are embodiments requiring the execution engine $126_1$ to operates with the SDK linking module $225_1$ to use different SDKS to coordinate the communication among multiple VCPs 245 such as to link and perform several remote process calls.

**[0043]** Figure 3 is an illustration of an overview method 300 using the virtual code providers 245 disclosed in Figure 2 interfacing with an integrated development environment (IDE) 305 in accordance with one or more embodiments of the present invention. The method 300 utilizes an IDE 305 on a user device 102n, and VCPs 245 stored on remote servers 104n. The method 300 downloads language snippets in a selected programming language native to the IDE 305 to call remote processes in various programming languages and platforms of corresponding CSCs 210 necessary to execute the remote processes. In the described embodiment, a single exemplary snippet corresponding to a single VCP $245_N$ and CSC $210_N$ is shown however other embodiments include multiple snippets and corresponding VCPs 245 and CSCs 210.

**[0044]** The IDE 305 coordinates a local code base with a remote code base and allows programs or applications to be written locally but yet, include remote process calls using a virtualized SDK. In some embodiments, the virtualized SDK may be downloaded and stored in the user device 102n.

**[0045]** A user device 102a is displayed available bundles of processes offered the VCP $245_N$ and a remote process is selected. In some embodiments, the user of the IDE 305 may have an existing program written in the specific programming

language. In such an embodiment, the user also selects the same programming language for the snippet as the existing program.

**[0046]** The snippet is then downloaded to the IDE 305. In some embodiments, the snippet is generated by the snippet generator $124_N$. In other embodiments, the snippet may be previously generated and recalled with authentication into a user account located in the VCP $245_N$. In some embodiments, the snippet is edited by the user in the IDE 305 to include parameters for the process. Alternatively, the parameters may be submitted by the user when selecting the process from the process pool $230_N$ such that the snippet is downloaded with the parameters already inserted into the snippet.

**[0047]** The IDE 305 then deploys the snippet to the SDK linking module $225_N$. In some embodiments, the snippet is deployed using a virtualized SDK to associate the remote process call of the snippet to the corresponding VCP 245. In other embodiments, the deployed snippet is directly deployed to the corresponding VCP 245.

**[0048]** The SDK linking module $225_N$ retrieves pre-generated code associated with the selected process and associated SDK of a CSC $210_N$. For example, for a CSC $210_N$ corresponding to the AMAZON servers and the process is to purchase an item, the SDK linking module $225_N$ retrieves pre-generated code from the process pool $230_N$ in a native language used to access and execute a purchase on AMAZON servers using parameters submitted from the user parsed from the snippet. Continuing the example, parsed parameters for the AMAZON purchase would include the product name and serial number, a name of the user, shipping address, and credit card information. The pre-generated code inserted with parameters is sent to the execution engine $126_N$ that subsequently deploys the remote process on the corresponding CSC $210_1$.

[0049] Figure 4 is a flow diagram for a method of creating process call snippets in accordance with one or more embodiments of the present invention. The method may be implemented by system 100 in Figure 1 and the platform 200 discussed in Figure 2 above. The method 400 generates remote process calls.

[0050] The method 400 begins at step 405 and continues to step 410 wherein processes code associated with services offered by CSCs 210 are stored into memory (e.g., process pools 230). The process code may be in a native language that is unique to a $CSC_N$. In some embodiments, the VCPs 245 generate the process code or alternatively, the process code may be pre-generated and prepared for insertion of parameter data.

[0051] Next at step 415, the process code is analyzed for attributes. Attributes include parameters (e.g., variables), programming language type, as well as necessary SDKs or APIs for interfacing and executing the process code on the CSCs 210. Attributes also include distinguishing parameters based on respective relationships between the parameters and the function of the parameter for a given process. For example, a send e-mail process requires parameters such as an originating e-mail address, destination e-mail address, subject, current time, time to send, and message body.

[0052] Next at step 420 the parameters are correlated based on relative relationships to offered processes and role in performing a process on the CSCs 210. In the example, each of the parameters has a specific respective function in the process and relation to the other parameters. A destination e-mail address is allotted to a portion of the process code requiring a destination e-mail that is related with a corresponding message subject and body with the originating e-mail address as an author of the e-mail. The time to send parameter includes a value as to when the e-mail is to be sent, and is accordingly correlated to the function of time for execution within the send e-mail process. In alternative embodiments, the analyzed process code is native to the SDKs and APIs of respective CSCs 210 (e.g.,

AMAZON, TWITTER, and the like). In some embodiments, relationship information also includes the source of the raw parameter components. Parameter data may be manually entered by the user, automatically entered (e.g., stored authentication keys), conditionally based on the result of another process, or related to a parameter of another process.

**[0053]** At step 425, the parameters are assigned to at least one process in the process pools 230. Continuing the previous example, the originating email address may be assigned to the process for sending an email as well as a login credential to a POP3 server necessary for another process to authenticate into an e-mail server.

**[0054]** At step 430, the processes in the process pools 320 are linked to a virtualized SDK. The virtualized SDK may be downloaded to a user device to call the processes in the process pools 230. In some embodiments, calling a process includes receiving parameter data from a snippet that is inserted into recalled stored process code and executing the process code with inserted parameters on corresponding CSCs 210 using APIs and SDKs for the CSCs 210 linked to the virtualized SDK. In such embodiments, the requisite APIs and SDKs are also linked to the associated process code in the process pools 320.

**[0055]** At step 435, the method 400 generates process call snippets for each process of the process pools 230 in multiple programming languages using pre-existing language templates for a selected language. The method 400 then ends at step 440.

**[0056]** Figure 5 is a flow diagram of a method 500 for expanding a local development environment with additional remote processes in accordance with one or more embodiments of the present invention. Through a virtualized SDK, the method 500 allows the code base on the user device 102a to be small and only include SDKs that are being implemented by the user. The method 500 includes

authenticating into a remote programming platform 200 that processes the remote process calls of the virtualized SDK.

**[0057]** The method 500 begins at step 505 and continues to step 510 wherein a user device 102a logs into the remote platform 200. The account of the user is authenticated and associated with previously stored keys and credentials for various remote services of CSCs 210 (e.g., data storage, e-mail, websites, social media, brokerages, banking, and the like). The authentication modules 228 coordinate with one another to authenticate into associated various CSCs 210 of the VCPs 245. In some embodiments, the platform 200 includes a centralized user account with a profile history of commonly used routines, processes, SDKs, and APIs. The profile history is recorded and updated for both the platform 200 and in the user databases 232 corresponding to associated VCPs 245 and CSCs 210.

**[0058]** The method 500 continues to step 515 wherein remote processes are populated and/or displayed to the user device 102a. In embodiments wherein the user device 102a has a display, the available processes will be shown via the IDE 305 to the user. Remote processes of process pools 230 may be displayed as bundles of processes corresponding to actions offered for a specific CSC 210. For example, a CSC 210 corresponding to TWITTER would include a FOLLOW bundle for all sub-actions (e.g., FOLLOW, stop FOLLOW, FOLLOW status, and the like) using the FOLLOWING function and a TWEET bundle for sub-functions using TWEETs.

**[0059]** The method 500 continues to step 520 wherein a virtualized SDK associated with the selected remote process is requested from the remote server 104a. The virtualized SDK may include all remote process calls or customized to include code to handle calls to selected processes. In some embodiments, the virtualized SDK is specific to the programming language on the user device 102a. The virtualized SDK provides a code snippet with all the necessary helpers/tools to call the remote process selected on the corresponding VCP 245.

[0060] In embodiments wherein no display is available, the method 500 may automate downloading the corresponding SDK of a selected process. The virtualized SDK allows remote process calls via the SDK linking modules 225 that link a process from the process pool to a corresponding SDK and API associated with at least one remote service of the CSCs 210. For example, where the CSC 210 is GOOGLE, the virtualized SDK includes linking to the GOOGLE SDK and API that has commands for a running a GOOGLE search and returning results from the search to a particular computer for a specific query.

[0061] At step 525 to download and install the associated virtualized SDK into the local IDE 305 such that the IDE 305 will have access to the remote process and integrate the virtualized SDK with existing local code base on the user device 102a. The virtualized SDK is a language-specific local code library that the developer needs to have installed in her/his local code base (and then in the deployed code for production).

[0062] Next at step 530, the downloaded virtualized SDK is used to allow a local program (e.g., created on the user device 102a) to call and execute a corresponding remote process on the server 104a using parameters entered by the user in the IDE 305. Parameters may be entered via pre-defined fields presented to the user via a graphical user interface. The pre-defined fields limit and simplify the information necessary to create the remote call snippet. The data of the parameters will then be transmitted to the remote server 104a and inserted into pre-generated code in the process pools 230 comprising the remote processes.

[0063] As will be further discussed below in Figure 6, the virtualized SDK also provides support for generated snippets. The snippets generated in a selected programming language that is native to the operating and/or programming environment (e.g., IDE 305) of the user device 102.

[0064] At step 535, the remote server 104a deploys through the VCP 245 and execution engine 125 the correlated remote process to the CSC 210 associated with the remote process and VCP 245.

[0065] At step 540 real-time results of the executed process are also displayed from the remote server 104a in the IDE 305. Thus, a successful FACEBOOK post may return a message to the remote server 104a that processes a response to notify the IDE 305. In some embodiments, the results are returned directly from the CSCs 210 associated with the process call. For example, the results from a FACEBOOK post may be returned directly to the IDE 305. The method 500 then ends at step 545.

[0066] Figure 6 is a flow diagram of a method 600 for a virtualized process call code-generator for various programming languages in accordance with one or more embodiments of the present invention. The method 600 applies the virtualized SDK and allows for conversion between various programming languages by generating a snippet tailored for the user device 102a and/or IDE 305 to call remote processes on CSCs 210. In some embodiments, the method 600 is a continuation of method 500 in calling a remote process with a virtualized SDK.

[0067] The method 600 begins at step 605 and continues to step 610 wherein the collection of remote processes are populated for the user device 102a. In some embodiments the user device 102a must login to the remote platform 200 such that the user account has access to the remote processes. The login process described in further detail above with respect to Figure 5.

[0068] Next at step 615, at least one remote process is selected. In some embodiments, the remote process is selected through a graphical user interface (GUI) on the user device 102a. In other embodiments, the remote process is specified by calling to the remote process through a function in a virtualized SDK downloaded to the user device 102a.

**[0069]** At step 620, parameters are input from the user for the selected process in the user device 102a. The parameters may include authentication keys for services, timing information, payload data, file extensions, and other information necessary to successfully execute a remote process. The parameters are inserted into a snippet generated in association with a remote process. In some embodiments, the parameters are input by the user via the local IDE 305 and in further embodiments, some parameter entries may be completed remotely using stored credentials or data on the remote server 104a. The IDE 305 is in communication with the platform 200 stored on remote server 104a.

**[0070]** At step 625 a program (e.g., IDE 305) on the user device 102a locally triggers deployment of the selected remote process with associated process parameters to the remote platform 200. The deployment utilizes the snippets and virtualized SDK to call the appropriate VCPs 245 and process pools 230 on the remote platform 200.

**[0071]** At step 626, the parameters are parsed from the snippets. Parsing includes directing the parameters to the VCPs for the selected process as linked through the virtualized SDK.

**[0072]** At step 628, the parsed parameters are inserted into pre-generated code. In some embodiments, the pre-generated code in the native language of the CSCs 210 and in process pools 230.

**[0073]** At step 630, the process is executed on an actual remote environment such as CSCs 210. For example, a purchase using AMAZON servers or sending a POP3 email, and the like. However, alternative embodiments may execute the selected process in a test environment for testing or "sandboxing" the program prior to full deployment of the process and program to a remote server. Sandboxing becomes increasingly important with the addition of multiple remote processes and services within a program.

[0074] Next at step 635, results are displayed in real-time to the user IDE. In some embodiments, results of process calls deployed on CSCs 210 also provide monitoring and maintenance using the remote platform 200 to VCPs 245. The snippets of the application are processed such that native code(s) are executed on remote servers (e.g., CSCs 210) using parameter data from the snippet. In such an embodiment, the platform 200 then monitors and logs the application to returns statistics such as usage and error data to the IDE 305 and update APIs and SDKs in the VCPs 245. The updates are then integrated into the corresponding pre-generated code in the process pools 235 to reduce subsequent runtime errors.

[0075] At step 640 the code snippet for triggering the selected remote process call is generated in a first default programming language by the corresponding VCPs 245. In some embodiments, the code snippet may be generated simultaneous to the deployment of the selected remote processes on the CSCs 210. Further still are embodiments wherein the code snippet may be generated without deployment or execution of the process.

[0076] In some embodiments a code snippet may call/access remote services also on the network 110. For example, a program may be written in ECLIPSE used to simultaneously TWEET a user specified message and check-in on FOURSQUARE using pre-authenticated keys/credentials of the associated user account. To access such services, the snippet generator generates a code snippet to be inserted into ECLIPSE to call the remote TWITTER and FOURSQUARE services and execute the aforementioned processes. As will be described further below, in some embodiments the parameters from the IDE can then be extracted to build the snippet format via the snippet generator 124.

[0077] In some embodiments, the code snippet may also include execution timing information. The code snippet is formed from a predefined template for a particular programming language on the user device 102a that communicates process calls to the remote platform 200 using the installed virtualized SDK.

[0078] At step 645, the method 600 determines whether the user has selected a different programming language in the IDE 305. In some embodiments, the method 600 may automatically detect the desired programming language by comparing existing programming code input by the user as well as via stored user profile preferences or histories by the platform 200. If the method 600 determines no other language is selected or the default language is the correct language the snippet sent to the user device 102a at step 655. However, if the method 600 determines another language is selected or detected, the snippet is regenerated or reformatted using a template for the corresponding selected language at step 650. In some embodiments, the templates supplement user entered parameters with previously stored parameters on the user device 102a or the server 104a such as variables, access keys to remote APIs, tokens, credentials, and the like.

[0079] From step 650, the method also continues to step 655 and sends the new snippet to the user device 102a and/or IDE 305. Next, at step 660, the snippet is inserted into the local code base such that a local program written by the user may call remote processes on the server 104a over the network 110 in addition to code that is to be run locally on the machine (e.g., solely on the user device 102a). In some embodiments, the local code base may also access other networked machines. In further embodiments, the local code base can be on the developer workstation or already deployed on servers (directly or through a code management systems such as GITHUB).

[0080] At step 665, the local code and remote process calls may be executed within the user IDE or in the native development environment chosen by the user. The method 600 then ends at step 670.

[0081] Figure 7 is a simplified block diagram of modules within an exemplary computing server in accordance with one or more embodiments of the present invention. The computer system 700 comprises substantially similar structure comprising servers or electronic devices in the aforementioned embodiments.

[0082] Various embodiments of methods and system for snippet generation and remote process calls, as described herein, may be executed on one or more computer systems, which may interact with various other devices. One such computer system is computer system 700 illustrated by Figure 7, which may in various embodiments implement any of the elements or functionality illustrated in Figures 1-6. In various embodiments, computer system 700 may be configured to implement methods described above. The computer system 700 may be used to implement any other system, device, element, functionality or method of the above-described embodiments. In the illustrated embodiments, computer system 700 may be configured to implement methods 400, 500, and 600 as processor-executable executable program instructions 722 (e.g., program instructions executable by processor(s) 710) in various embodiments.

[0083] In the illustrated embodiment, computer system 700 includes one or more processors 710a-710n coupled to a system memory 720 via an input/output (I/O) interface 730. Computer system 700 further includes a network interface 740 coupled to I/O interface 730, and one or more input/output devices 760, such as cursor control device 760, keyboard 770, and display(s) 780. In some embodiments, the keyboard 770 may be a touchscreen input device.

[0084] In various embodiments, any of the components may be utilized by the system to route and execute remote process calls as described above. In various embodiments, a user interface may be generated and displayed on display 780. In some cases, it is contemplated that embodiments may be implemented using a single instance of computer system 700, while in other embodiments multiple such systems, or multiple nodes making up computer system 700, may be configured to host different portions or instances of various embodiments. For example, in one embodiment some elements may be implemented via one or more nodes of computer system 700 that are distinct from those nodes implementing other elements. In another example, multiple nodes may implement computer system 700 in a distributed manner.

[0085] In different embodiments, computer system 700 may be any of various types of devices, including, but not limited to, personal computer systems, mainframe computer systems, handheld computers, workstations, network computers, application servers, storage devices, a peripheral devices such as a switch, modem, router, or in general any type of computing or electronic device.

[0086] In various embodiments, computer system 700 may be a uniprocessor system including one processor 710, or a multiprocessor system including several processors 710 (e.g., two, four, eight, or another suitable number). The uniprocessor system including a one processor 710 may split functionality of the single processor to appear as multiple processors for processing and executing simultaneous data of a program. Processors 710 may be any suitable processor capable of executing instructions. For example, in various embodiments, processors 610 may be general-purpose or embedded processors implementing any of a variety of instruction set architectures (ISAs). In multiprocessor systems, each of processors 710 may commonly, but not necessarily, implement the same ISA.

[0087] System memory 720 may be configured to store program instructions 722 and/or data 732 accessible by processor 710. In various embodiments, system memory 720 may be implemented using any suitable memory technology, such as static random access memory (SRAM), synchronous dynamic RAM (SDRAM), nonvolatile/Flash-type memory, or any other type of memory. In the illustrated embodiment, program instructions and data implementing any of the elements of the embodiments described above may be stored within system memory 720. In other embodiments, program instructions and/or data may be received, sent or stored upon different types of computer-accessible media or on similar media separate from system memory 720 or computer system 700.

[0088] In one embodiment, I/O interface 730 may be configured to coordinate I/O traffic between processor 710, system memory 720, and any peripheral devices in the device, including network interface 740 or other peripheral interfaces, such as

input/output devices 750. In some embodiments, I/O interface 730 may perform any necessary protocol, timing or other data transformations to convert data signals from one component (e.g., system memory 720) into a format suitable for use by another component (e.g., processor 710). In some embodiments, I/O interface 730 may include support for devices attached through various types of peripheral buses, such as a variant of the Peripheral Component Interconnect (PCI) bus standard or the Universal Serial Bus (USB) standard, for example. In some embodiments, the function of I/O interface 730 may be split into two or more separate components, such as a north bridge and a south bridge, for example. Also, in some embodiments some or all of the functionality of I/O interface 730, such as an interface to system memory 720, may be incorporated directly into processor 710.

**[0089]** Network interface 740 may be configured to allow data to be exchanged between computer system 700 and other devices attached to a network (e.g., network 790), such as one or more external systems or between nodes of computer system 700. In various embodiments, network 790 may include one or more networks including but not limited to Local Area Networks (LANs) (e.g., an Ethernet or corporate network), Wide Area Networks (WANs) (e.g., the Internet), wireless data networks, cellular networks, Wi-Fi, some other electronic data network, or some combination thereof. In various embodiments, network interface 740 may support communication via wired or wireless general data networks, such as any suitable type of Ethernet network, for example; via telecommunications/telephony networks such as analog voice networks or digital fiber communications networks; via storage area networks such as Fibre Channel SANs, or via any other suitable type of network and/or protocol.

**[0090]** Input/output devices 750 may, in some embodiments, include one or more display terminals, keyboards, keypads, touchpads, touchscreens, scanning devices, voice or optical recognition devices, or any other devices suitable for entering or accessing data by one or more computer systems 700. Multiple input/output devices 750 may be present in computer system 700 or may be distributed on

various nodes of computer system 700. In some embodiments, similar input/output devices may be separate from computer system 700 and may interact with one or more nodes of computer system 700 through a wired or wireless connection, such as over network interface 740.

**[0091]** In some embodiments, the illustrated computer system may implement any of the methods described above, such as the methods illustrated by the flowchart of Figures 3-6. In other embodiments, different elements and data may be included.

**[0092]** Those skilled in the art will appreciate that computer system 700 is merely illustrative and is not intended to limit the scope of embodiments. In particular, the computer system and devices may include any combination of hardware or software that can perform the indicated functions of various embodiments, including computers, network devices, Internet appliances, smartphones, tablets, PDAs, wireless phones, pagers, and the like. Computer system 700 may also be connected to other devices that are not illustrated, or instead may operate as a stand-alone system. In addition, the functionality provided by the illustrated components may in some embodiments be combined in fewer components or distributed in additional components. Similarly, in some embodiments, the functionality of some of the illustrated components may not be provided and/or other additional functionality may be available.

**[0093]** Those skilled in the art will also appreciate that, while various items are illustrated as being stored in memory or on storage while being used, these items or portions of them may be transferred between memory and other storage devices for purposes of memory management and data integrity. Alternatively, in other embodiments some or all of the software components may execute in memory on another device and communicate with the illustrated computer system via inter-computer communication. Some or all of the system components or data structures may also be stored (e.g., as instructions or structured data) on a computer-accessible medium or a portable article to be read by an appropriate drive, various

examples of which are described above. In some embodiments, instructions stored on a computer-accessible medium separate from computer system 700 may be transmitted to computer system 700 via transmission media or signals such as electrical, electromagnetic, or digital signals, conveyed via a communication medium such as a network and/or a wireless link. Various embodiments may further include receiving, sending or storing instructions and/or data implemented in accordance with the foregoing description upon a computer-accessible medium or via a communication medium. In general, a computer-accessible medium may include a storage medium or memory medium such as magnetic or optical media, e.g., disk or DVD/CD-ROM, volatile or non-volatile media such as RAM (e.g., SDRAM, DDR, RDRAM, SRAM, and the like), ROM, and the like.

**[0094]** An exemplary snippet generated for insertion into a JAVA program calls the TEMBOO virtualized SDK includes::

```
package TembooDemoECOs;

import com.temboo.Library.NYTimes.ArticleSearch.QueryArticles;

import
com.temboo.Library.NYTimes.ArticleSearch.QueryArticles.QueryArticlesInputSet;

import
com.temboo.Library.NYTimes.ArticleSearch.QueryArticles.QueryArticlesResultSet;

import com.temboo.core.TembooException;

import com.temboo.core.TembooSession;

public class TembooDemoECOs {
```

**[0095]**

```
    /**

     * @param args
```

```
     * @throws TembooException
     */
    public static void main(String[] args) throws TembooException {
```

[0096] // Instantiate a Temboo session

```
        TembooSession session = new TembooSession("demo", "myFirstApp", "6d975bf6-e761-47c7-8");
```

[0097] // Instantiate the Choreo, using a previously instantiated TembooSession object

```
    QueryArticles queryArticlesChoreo = new QueryArticles(session);
```

[0098] // Get an InputSet object for the choreo

```
    QueryArticlesInputSet queryArticlesInputs = queryArticlesChoreo.newInputSet();
```

[0099] // Set credential to use for execution

```
        queryArticlesInputs.setCredential("NYTimesCred");
```

[00100] // Set inputs

```
        queryArticlesInputs.set_Query("obama");
```

[00101] // Execute Choreo

```
        QueryArticlesResultSetqueryArticlesResults= queryArticlesChoreo.execute(queryArticlesInputs);
```

[00102] // Print results using ECOs

```
for (int i = 0; i < queryArticlesResults.getArticles().length; i++)
{System.out.println(queryArticlesResults.getArticles()[i].getTitle() + " :: " +
queryArticlesResults.getArticles()[i].getByLine());

        }

    }

}
```

**[00103]** In the above example, the snippet comprises five lines of code capable of being inserted into any Java program. Beginning with code for "Instantiate a Temboo session" and ending with code for "Execute Choreo". These lines of code are capable of being edited in a local IDE as text and edited in the future so as the user knows the names of choreos he or she would like to call. The line "**package** TembooDemoECOs;" calls the virtualized SDK by doing an import of a specific class: QueryArticles. The above example is possible with installation this exemplary SDK that comprises the code for QueryArticles:

**[00104]**

```
public class QueryArticles extends Choreography {

    /**  Create a new instance of the QueryArticles Choreo. A
TembooSession object, containing a valid

    set of Temboo credentials, must be supplied.

    */

    public QueryArticles(TembooSession session) {

    super(session,
TembooPath.pathFromStringNoException("/Library/NYTimes/ArticleSearch/QueryAr
ticles"));
```

```
}
```

[00105] /** Obtain an InputSet object, used to define inputs for an execution of this Choreo.

```
@return QueryArticlesInputSet
*/
public QueryArticlesInputSet newInputSet() {
    return new QueryArticlesInputSet();
}
```

[00106] /**

```
* Execute the Choreo using the specified InputSet as parameters, wait for the Choreo to complete
* and return a ResultSet containing the execution results.
* @param choreoInputs
* @return
* @throws TembooException
*/
@Override
public QueryArticlesResultSet execute(InputSet choreoInputs) throws TembooException {
    JSONObject result = super.executeWithResults(choreoInputs);
```

```
        return new QueryArticlesResultSet(result);
    }
```

[00107] /** An InputSet with methods appropriate for specifying the inputs to the QueryArticles Choreo. The InputSet object is used to specify input parameters when executing this choreo.

```
    */
    public static class QueryArticlesInputSet extends InputSet {
```

[00108] /** Set the value of the APIKey input for this Choreo.

@param String - (required, string) The API Key provided by NY Times.

```
        */
        public void set_APIKey(String value) {
            this.setInput("APIKey", value);
        }
```

[00109] /** Set the value of the BeginDate input for this Choreo.

@param String - (optional, date) Sets the starting point (which is inclusive) of the range of publication dates to return. Must be used with EndDate. Date should be formatted like YYYYMMDD.

```
        */
        public void set_BeginDate(String value) {
            this.setInput("BeginDate", value);
```

```
}
```

[00110] /** Set the value of the EndDate input for this Choreo.

@param String - (optional, date) Sets the end point (which is inclusive) of the range of publication dates to return. Must be used with BeginDate. Date should be formatted like YYYYMMDD.

```
*/

public void set_EndDate(String value) {

        this.setInput("EndDate", value);

}
```

[00111] /** Set the value of the Facets input for this Choreo.

@param String - (optional, string) A comma-delimited list of up to 5 facets. This indicates the sets of facet values to include in the response. See Choreo documentation for more information about accepted values for this input.

```
*/

public void set_Facets(String value) {

        this.setInput("Facets", value);

}
```

[00112] /** Set the value of the Fields input for this Choreo.

@param String - (optional, string) A comma-delimited list of fields to return. These fields are returned by default: body, byline, date, title, and url.

```
*/
```

```
public void set_Fields(String value) {
        this.setInput("Fields", value);
}
```

[00113] /** Set the value of the Offset input for this Choreo.

@param Integer - (optional, integer) This corresponds to which set of 10 results is returned. Used to page through results. Set to 0 to return records 0-9, set to 1 to return records 10-19, etc.

```
*/
public void set_Offset(Integer value) {
        this.setInput("Offset", value);
}
```

[00114] /** Set the value of the Offset input for this Choreo as a String.

@param String - (optional, integer) This corresponds to which set of 10 results is returned. Used to page through results. Set to 0 to return records 0-9, set to 1 to return records 10-19, etc.

```
*/
public void set_Offset(String value) {
        this.setInput("Offset", value);
}
```

[00115] /** Set the value of the Query input for this Choreo.

@param String - (required, string) Search keywords (optionally applied to specific fields) and/or facets. See Choreo documentation for syntax examples.

*/

public void set_Query(String value) {

this.setInput("Query", value);

}

[00116] /** Set the value of the Rank input for this Choreo.

@param String - (optional, string) Sets the order of the results. Accepted values are: newest (the defaults), oldest, or closest.

*/

public void set_Rank(String value) {

this.setInput("Rank", value);

}

}

[00117] /** A ResultSet with methods tailored to the values returned by the QueryArticles Choreo.

The ResultSet object is used to retrieve the results of a Choreo execution.

*/

public static class QueryArticlesResultSet extends ResultSet {

```
public QueryArticlesResultSet(JSONObject doc) throws
TembooException {
        super(doc);
    }
```

[00118]
```
    /**  Retrieve the value for the "Response" output from
this Choreo execution
    @return String - (json) The response from the NY Times API.
    */
    public String get_Response() {
        return this.getResult("Response");
    }
```

[00119]
```
    /** The value of offset corresponds to a set of 10 results (i.e.
offset=0 for results 0-9, offset=1 for results 10-19, etc)
    */
    public String getOffset() {
      return (String)
((JSONObject)getJSONFromString(this.getResult("Response"))).opt("offset");
    }
```

[00120]
```
    /** Get an article matching the search criteria
    */
    public NYTimesArticle[] getArticles() {
```

```
                    JSONArray list = (JSONArray)
((JSONObject)getJSONFromString(this.getResult("Response"))).opt("results");

                    NYTimesArticle[] resultArray = new NYTimesArticle[(list !=
null) ? list.length() : 0];

                    if(list != null) {

                        for(int i=0; i<list.length(); i++) {

                            try {

                                resultArray[i] = new NYTimesArticle(list.get(i));

                            } catch (JSONException e) {}

                        }

                    }

                    return resultArray;

                }
```

[00121]

```
                    /** Contains the search tokens provided for the query

                    */

                    public String[] getTokens() {

                        JSONArray list = (JSONArray)
((JSONObject)getJSONFromString(this.getResult("Response"))).opt("tokens");

                        String[] resultArray = new String[(list != null) ? list.length() : 0];

                        if(list != null) {
```

```
                    for(int i=0; i<list.length(); i++) {
                        try {
                            resultArray[i] = (String) list.get(i);
                        } catch (JSONException e) {}
                    }
                }
                return resultArray;
            }
```

[00122]

```
            /** The total number of articles for this search
            */
            public Number getTotal() {
                return (Number)
((JSONObject)getJSONFromString(this.getResult("Response"))).opt("total");
            }
        }
    }
```

[00123] By installing the above exemplary virtualized SDK, the snippet may now execute calls to remote processes. The code snippet links to the virtualized SDK code to call the actual process on the remote server 104a through the associated VCPs 245. In some embodiments, the SDK comprises insuring the input parameters are in the proper format and then calling the corresponding remote process. In the

above example a “choreo” is a graphical representation of a remote process on the user device 102a and/or local IDE 305.

**[00124]** The methods described herein may be implemented in software, hardware, or a combination thereof, in different embodiments. In addition, the order of methods may be changed, and various elements may be added, reordered, combined, omitted or otherwise modified. All examples described herein are presented in a non-limiting manner. Various modifications and changes may be made as would be obvious to a person skilled in the art having benefit of this disclosure. Realizations in accordance with embodiments have been described in the context of particular embodiments. These embodiments are meant to be illustrative and not limiting. Many variations, modifications, additions, and improvements are possible. Accordingly, plural instances may be provided for components described herein as a single instance. Boundaries between various components, operations and data stores are somewhat arbitrary, and particular operations are illustrated in the context of specific illustrative configurations. Other allocations of functionality are envisioned and may fall within the scope of claims that follow. Finally, structures and functionality presented as discrete components in the example configurations may be implemented as a combined structure or component. These and other variations, modifications, additions, and improvements may fall within the scope of embodiments as defined in the claims that follow.

**[00125]** While the foregoing is directed to embodiments of the present invention, other and further embodiments of the invention may be devised without departing from the basic scope thereof, and the scope thereof is determined by the claims that follow.

Claims:

1. A method for code virtualization comprising:

receiving a selection of a remote process on a remote server, the remote process correlated to at least one remote service;

generating from at least one virtual code provider (VCP) on the remote server, a code snippet with parameters from pre-defined fields to call the remote process; and

providing the code snippet for insertion into a local code base on a user device such that the code snippet calls the remote process from the user device with an installed software development kit (SDK) on the user device.

2. The method of claim 1, wherein the code snippet and the local code base are of a same programming language.

3. The method of claim 1, wherein generating the code snippet further comprises coordinating multiple SDKs for multiple classical service components (CSCs).

4. The method of claim 3, further comprising receiving the code snippet with parameter data and extracting parameter values of the parameter data to execute commands corresponding to the processes on the CSCs.

5. The method of claim 1, further comprising authenticating into a remote platform hosting the at least one VCP.

6. The method of claim 1, displaying real-time results of deploying the snippet to call the remote process from the user device.

7. The method of claim 1, wherein receiving the selection of the remote process further comprises receiving a selection of a programming language for generating the code snippet.

8. A method for remote process call generation in at least one virtual code provider (VCP) comprising:

storing a process code for at least one remote process as offered by at least one classical service component (CSC);

analyzing the process code corresponding to at least one process;

correlating parameters in the process code based on relationships to other parameters and at least one associated function of the process code;

assigning parameters to processes requiring the parameters in a process pool;

linking each process from the process pool to a virtualized software development kit (SDK); and

generating a code snippet to insert parameter values and execute an associated process from the process pool when called using the virtualized SDK.

9. The method of claim 8, wherein the generated process code is in a programming language native to the at least one CSC.

10. The method of claim 8, wherein correlating based on relationships further comprises designating a source for parameters in the process code.

11. The method of claim 8, further comprising recalling the stored remote process code from a process pool of the at least one VCP and inserting parameters into the stored process code.

12. The method of claim 11, further comprising deploying the process code to at least one corresponding CSC.

13. The method of claim 11, wherein the at least one VCP links the stored process code to a SDK and API native to the at least one CSC.

14. The method of claim 8, further comprising receiving a request for the snippet in a programming language that is of a same programming language as a program on a user device.

15. A system for remote process call generation comprising:

a) at least one processor;

b) at least one input device coupled to at least one network; and

c) at least one storage device storing processor executable instructions which, when executed by the at least one processor, performs a method of any of claims 1-7.