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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
22 December 2016 (22.12.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/205625 A1

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

(21) International Application Number:
PCT/US2016/038050

(22) International Filing Date:
17 June 2016 (17.06.2016)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/181,451 18 June 2015 (18.06.2015) US
62/271,800 28 December 2015 (28.12.2015) US

(71) Applicant: **THE JOAN AND IRWIN JACOBS TECHNION-CORNELL INSTITUTE** [US/US]; 111 8th Avenue, Suite 302, New York, NY 10011 (US).

(72) Inventor: **ADAR, Simon**; c/o The Joan and Irwin Jacobs Technion-Cornell Institute, 111 8th Avenue, Suite 302, New York, NY 10011 (US).

(74) Agent: **BEN-SHIMON, Michael**; M&B IP Analysts, LLC, 45 S. Park Place #262, Morristown, NJ 07960 (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))*

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

(54) Title: A COMPUTING PLATFORM AND METHOD THEREOF FOR SEARCHING, EXECUTING, AND EVALUATING COMPUTATIONAL ALGORITHMS

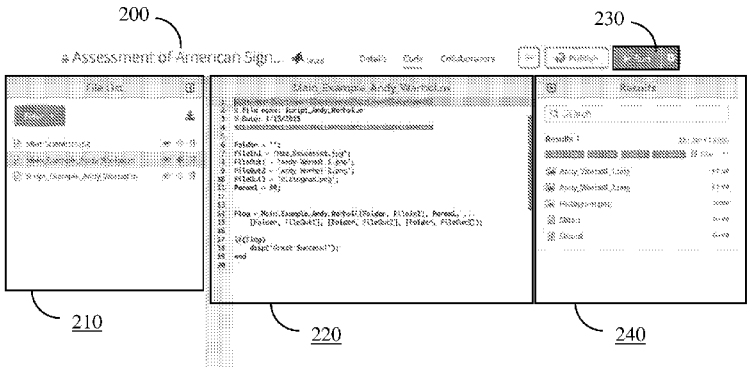

FIG. 2

(57) Abstract: A method and system for evaluating computational algorithms are provided. The method comprises receiving a textual description of a computational algorithm; generating a software container based on the received textual description; instantiating a computing environment on a computing device, wherein the computing environment includes computing resources configured to support execution of the software container; executing the software container in the computing environment; and displaying, on a user device, results that are output in response to execution of the software container.

WO 2016/205625 A1

A COMPUTING PLATFORM AND METHOD THEREOF FOR SEARCHING, EXECUTING, AND EVALUATING COMPUTATIONAL ALGORITHMS

CROSS REFERENCE TO RELATED APPLICATIONS

[001] This application claims the benefit of U.S. Provisional Application No. 62/181,451 filed on June 18, 2015 and US Provisional Application No. 62/271,800 filed on December 28, 2015, the contents of which are hereby incorporated by reference.

TECHNICAL FIELD

[002] The disclosure generally relates to the field of computer programming tools, and in particular to a computing platform and method to assist in programming, researching, evaluating, and maintenance of computational algorithms.

BACKGROUND

[003] As the development of software applications evolves, programming accurate and efficient code has become a real challenge. A typical software application interfaces with different modules, e.g., using APIs, and should be programmed to enable its execution on different platforms and operating systems. For example, a software application (app) should be available for execution over different types of operating systems, such as Linux®, Mac OS® and Windows®. Furthermore, as execution of most software applications have been migrated to cloud computing infrastructures, the code should be written such that its execution is supported by any computing environments in the cloud infrastructure. Thus, in sum, the complexity of software applications imposes a real challenge on programmers.

[004] In order to enable the development of rapid creation, setup, and deployment of applications, resources that include “off-the-shelf” functions, algorithms, and pieces of codes are available. Such resources range from academic and scientific publications to code repositories (e.g., GitHub). However, such resources are limited as they cannot provide executable code that can be easily evaluated and integrated in a software application or project being researched or

developed. For example, a scientific publication (e.g., in an IEEE® article) would include a textual description of an algorithm or at best its pseudo code.

[005] In most cases, scientific publications do not include any code that can be immediately executed on a computing environment for which the software application is developed. Thus, to evaluate the functionality and performance of the algorithm, executable code should be written and tested on that computing environment. This is a time consuming and error prone task. At best, a scientific publication would include a link to code repository that provides a sample code. However, this would still require the programmer to integrate the code in a software program for further evaluation.

[006] Furthermore, due to the voluminous amount of academic and scientific publications, open-source codes, and other algorithm resources, it is almost impossible to effectively search for the right algorithm that would fit into a software application and would perform an intended function.

[007] It would therefore be advantageous to provide a solution that overcomes the deficiencies noted above.

SUMMARY

[008] A summary of several example embodiments of the disclosure follows. This summary is provided for the convenience of the reader to provide a basic understanding of such embodiments and does not wholly define the breadth of the disclosure. This summary is not an extensive overview of all contemplated embodiments, and is intended to neither identify key or critical elements of all embodiments nor to delineate the scope of any or all aspects. Its sole purpose is to present some concepts of one or more embodiments in a simplified form as a prelude to the more detailed description that is presented later. For convenience, the term “some embodiments” may be used herein to refer to a single embodiment or multiple embodiments of the disclosure.

[009] Certain embodiments disclosed herein includes a method for evaluating computational algorithms. The method comprises receiving a textual description of a computational algorithm; generating a software container based on the

received textual description; instantiating a computing environment on a computing device, wherein the computing environment includes computing resources configured to support execution of the software container; executing the software container in the computing environment; and displaying, on a user device, results that are output in response to execution of the software container.

[0010] Certain embodiments disclosed herein also include a system for evaluating computational algorithms. The system comprises a processing system; and a memory, the memory containing instructions that, when executed by the processing system, configure the system to: receive a textual description of a computational algorithm; generate a software container based on the received textual description; instantiate a computing environment on a computing device, wherein the computing environment includes computing resources configured to support execution of the software container; execute the software container in the computing environment; and display, on a user device, results that are output in response to execution of the software container.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] The subject matter disclosed herein is particularly pointed out and distinctly claimed in the claims at the conclusion of the specification. The foregoing and other objects, features and advantages of the disclosed embodiments will be apparent from the following detailed description taken in conjunction with the accompanying drawings.

[0012] Figure 1 is a schematic diagram of a networked system utilized to describe the disclosed embodiments.

[0013] Figure 2 is an exemplary and non-limiting screenshot of a webpage rendered for evaluating an algorithm according to one embodiment.

[0014] Figure 3 is a flowchart illustrating a method for generating a container according to one embodiment.

[0015] Figure 4 is a schematic diagram of a dependency tree generated according to one embodiment.

DETAILED DESCRIPTION

[0016] It is important to note that the embodiments disclosed herein are only examples of the many advantageous uses of the innovative teachings herein. In general, statements made in the specification of the present application do not necessarily limit any of the various claims. Moreover, some statements may apply to some inventive features but not to others. In general, unless otherwise indicated, singular elements may be in plural and vice versa with no loss of generality. In the drawings, like numerals refer to like parts through several views.

[0017] Fig. 1 is an example diagram of a networked system 100 utilized to describe the various disclosed embodiments. The networked system 100 includes a cloud computing platform 110, a plurality of data repositories 120-1 through 120-N, computing devices 130-1 and 130-2, a server 140, and a data warehouse 150. The various elements of the system 100 are communicatively connected to a network 160. The network 160 may be, for example, a wide area network (WAN), a local area network (LAN), the Internet, and the like.

[0018] The cloud computing platform 110 may be a private cloud, a public cloud, or a hybrid cloud providing computing resources to applications or services executed therein. The plurality of data repositories 120-1 through 120-N (hereinafter individually referred to as a data repository 120 or collectively as data repositories 120) are resources to computational algorithms. In the context of the disclosed embodiments, a computational algorithm is any algorithm that can be utilized in the field of mathematics and/or computer science. Such an algorithm is typically defined as a self-contained step-by-step set of operations to be performed. As an example, a computational algorithm may be designed to perform calculation, data processing and analysis (such as, image processing, transaction processing), automated reasoning, and more.

[0019] The computational algorithms may be included in text books, scientific or academic publications (e.g., articles or magazines), blogs, manuals, tutorials (textual or video), and so on. A computational algorithm saved in a data repository may be described in a format of pseudo code, one or more

mathematical equations, a written description, a code sample or samples, or any combination thereof.

[0020] The computing devices 130-1 and 130-2 may include, for example, a personal computer, a laptop, a tablet computer, a smartphone, and the like. Each computing device 130 is utilized by a user to develop a software application, and as such includes an integrated development environment (IDE) 131 or any other tool that allows computer programmers to develop software.

[0021] Each computing device 130 further includes an agent 132 locally installed and executed over the computing device 130. According to various embodiments, the agent 132 is configured to scan any code developed in the IDE 131. Based on the code scanning, one or more software containers (or image containers) 133 are generated. The generated containers 133 are sent by the agent 132 to the server 140, to the data warehouse 150, or to both. The agent 132 is further configured to interface with the server 140 to retrieve the containers 133 and to integrate the retrieved containers 133 in a software program developed by the user.

[0022] Thus, according to the disclosed embodiments, an agent 132 has at least two primary roles: generating the containers 133, and integrating the generated containers 133 in a software program developed in the IDE 131. In an embodiment, the integration of the containers includes compiling or loading the containers 133 into code provided by the IDE 131.

[0023] These roles are realized by the agents 132 operable in the computing devices 130-1 and 130-2. In an example embodiment, the computing device 130-1 is operated by an author of the algorithm (e.g., the agent 132-1 generates containers 133-1), while the computing device 130-2 is used by a user of the containers (e.g., the agent 132-2 may integrate the containers 133-1 in the code).

[0024] It should be noted that the author and/or any user of the computing devices 130-1 and 130-2 may be, but is not limited to, an engineer, a programmer, a researcher, a developer, or any person with skill in the field of computer programming.

[0025] As an example, an author, using the computing device 130-1, develops an algorithm to compute an average of student grades. The agent 132-1 is configured to scan this algorithm to provide a container descriptor and to generate a container 133-1 respective of the container descriptor. The generated container 133-1 encapsulates all the software resources required for executing the algorithm. In an embodiment, the generated container 133-1 includes a data set serving as an input for the algorithm. The process for providing the container descriptor and generating the containers 133 is discussed in detail below.

[0026] It should be emphasized that the container descriptor can be generated based on any format that the algorithm is presented in, e.g., a pseudo code, a code sample, an executable code, equations, and so on. It should be further emphasized that the algorithm can be defined using the IDE 131-1, locally saved in the computing device 130-1, or downloaded from any data repositories 120.

[0027] A user of the computing device 130-2 can download the container 133-1 (as in the above example, to compute an average) and integrate the container 133-1 in any software program developed by means of the IDE 131-2. The process of the downloading, integrating, or otherwise executing the container 133-1 on the computing device 130-2 is facilitated by means of the agent 132-2.

[0028] Specifically, the agent 132-2 provides the container 133-1 with a complete set of interfaces to the computing environment. That is, any input to, or output generated by, the container 133-1 is received or sent through the agent 132-2. As an example, the agent 132-2 interfaces between the container 133-1 to the IDE 131-2, the server 140, and an operating system and file system of the computing device 130-2. The agent 132-2 is also configured to install all software resources required for the execution of the container 133-1 on the computing device 130-2.

[0029] According to the disclosed embodiments, the container 133-1 includes such software resources and the agent 132-2 is configured to select those resources that are needed for the proper execution of the container's 133-1 code. It should be noted that the agent 132-2 can check which resources are required for installation and which are not. For example, assuming the container's 133-1 code

is in Python, the agent installs a Python native in the container 133-1. As such, the container 133-1 including the Python installation can be executed on any computer regardless of whether the computer includes Python native. Alternatively, the container 133-1 may call to install Python native on-the-fly. Thus, for example, if the agent 132-2 detects that the operating system of the device 130-2 does not support Python, then the agent 132-2 causes installation of the native Python on the device 130-2, on the fly. Thus, in some embodiments, the user is not required to install any files, and these techniques may further allow seamless sharing and execution across different devices.

[0030] In another embodiment, the agent 132-2 is further configured to provide the IDE 133-1 with the code included in the container 133-2 to enable the display of such code in the IDE 133-1.

[0031] Certain tasks performed by the agents 132 can be carried out, alternatively or collectively, by the server 140. Specifically, in an embodiment, the server 140 is configured to receive a container descriptor from an agent 132-1 and to generate the container 133-1 respective thereof. In yet another embodiment, containers 133 can be executed by the server 140, and their inputs/outputs will be communicated to the respective agent 132 through an API. As an example, a container 133-1 for computing averages can receive an input data set (e.g., a list of student grades) provided to the server 140 by the agent 132-2, and upon execution of the container 133-1 by the server 140, the average is returned to the agent 132-2. The input data set is part of the container 133-1.

[0032] In order to allow execution of a container 133, the server 140 is configured to create or instantiate a computing environment that supports the execution of the container 133. For example, if a container 133 includes a Python code, a virtual machine (not shown) that can run Python is instantiated by the server 140. In an embodiment, the creation of computing environments for running containers 133 can be performed on-demand (e.g., upon receiving a request to run a container). In another embodiment, computing environments utilized by “popular” containers (e.g., computing environments utilized above a predefined threshold number of times during a given time period) are always available.

[0033] In a further embodiment, the resources produced during the execution of a container 133 are cached, for example, in a cache memory (not shown) connected to the server 140. The cached resources may include, for example, installation files, compilation files, results output by the container, combinations thereof, and the like. In another embodiment, the cache memory can be shared among different computing environments (e.g., "workers") created to execute the containers. This sharing of cache memory allows for sharing cached resources among the computing environments. It should be appreciated that the cached resources produced in one worker can be used by other workers without requiring re-compilation, thereby accelerating the execution and accessibility to containers. It should be noted that the cached resources are available to containers executed by the agents 132 on user devices 130 or to the server 140.

[0034] Specifically, cached resources is based in part on the memory's state during code execution including all variables, matrixes, figures, intermediate/temporarily files, result files. The cached resources can be saved in the platform's centralized data repository, on a ready to execute "warm" worker, or both with a synchronizing mechanism. Thus, caching the resources include sharing the resources between executions of a container by the same or different accounts and users.

[0035] The trigger to use cached resources is when, for example, the state of code, input data, packages, versions and configuration are detected as the same or similar enough to generate the same resources.

[0036] As an example, execution of empty spaces or lines, comments, or certain pieces of code can be avoided using the cached resources. That is, such resources can be utilized instead of executing their respective pieces of code. The execution can them resume from that point to either execute new code according to user request or predefined code that will be executed from the shared resources point and forward automatically.

[0037] In an embodiment, shared resources can also be utilized during the compilation stage, where certain compilation steps are not performed. To this end, the state of the code and cached (shared) resources are identified, and

compilation, execution and/or synchronization of shared resources is not performed.

[0038] In another embodiment, the server 140 is further configured to create the containers 133 based on the container descriptors provided by the agents 132. The generated containers 133 are saved in the data warehouse 150. The process for generating the container descriptors and containers is described below.

[0039] According to some embodiments, a search for containers 133 and the respective algorithms they carry out can be performed by any device communicatively connected to the server 140. As noted above, the containers are saved in the data warehouse 150.

[0040] In order to create searchable containers 133, metadata is generated and associated with each container 133 saved in the data warehouse 150. The metadata may be generated by the server 140 and/or by an agent 132. The metadata may include, but is not limited to, a container name, a container creation date, an algorithm name, an author of the algorithm, similar or related algorithms, user interaction, combinations thereof, and so on. The user interaction metadata lists how users downloaded or executed a specific container, the number of repetitions of a container per user, a benchmark, combinations thereof, and so on. In an exemplary embodiment, a ranking process is implemented to rank matching containers to a user query respective of the generated metadata. The search may be performed through a web interface (e.g., a portal) communicatively connected to the server 140.

[0041] In an embodiment, the ranking may be based, in part, on the user interaction information contained in the metadata. For example, a ranking score indicative of the popularity of a container may be computed using the number of runs, the number of views, and the number of downloads. The containers and/or their respective printed publications will be ranked based on their computed scores.

[0042] According to some disclosed embodiments, the evaluation of the performance and/or functionality of an algorithm can be also performed through the web interface. An exemplary and non-limiting screenshot of a webpage 200 for

evaluating an algorithm is provided in Fig. 2. The webpage 200 is rendered by the web interface.

[0043] In the example shown in Fig. 2, the algorithm is for a recognition program written in Matlab® code. The webpage 200 may display the code associated with a container generated for the algorithm. In this example, the algorithm is composed from 3 files listed in the file list 210. The contents of each file can be viewed and edited in the section 220 of the webpage 200. Any changes to the code are saved to the data warehouse 150. Also, the changes are associated with the respective container.

[0044] Once the user clicks the "Run" button 230, the respective container generated for the recognition algorithm (and any changes therein) is executed by the server 140. The results of the execution are displayed in the results section 240.

[0045] According to various embodiments, the server 140 is configured to benchmark an algorithm through its respective container. To this end, an input data set is predefined to be utilized by containers designed to execute similar algorithms. The server 140 is configured to execute each container and to measure performance parameters. The parameters may be global parameters and/or specific parameters. The global parameters are agnostic to the purpose of the algorithm, while the specific parameters are related to the purpose/function of the algorithm. For example, global parameters may include execution time, memory, CPU and/or bandwidth utilization, and so on. The specific parameters may be, for example, a compression ratio for lossless compression algorithms.

[0046] In a further embodiment, the agent is configured to generate a user interface, e.g., a graphical user interface, based on the container's code. The developer of the container would not be required to code any GUI for inputting data or outputting the results. The developer is merely required to specify which variables are input variables.

[0047] The agent is configured to scan the code, to identify the specified input variables and their types, to generate GUI elements respective of such variables, and to display the generated GUI elements according to a specific order in which

the input should be received. The agent is further configured to provide an output window for displaying the execution's results.

[0048] The generated GUI enables easy and fast interaction with the executed computational algorithm. For example, a user evaluating the computational algorithm is not required to enter a series of datasets through a command line.

[0049] In an embodiment, the number of downloads of containers and/or the execution of the containers by a user are tracked. This allows for monetization of the usage of containers, for example, via compensation per download or per execution.

[0050] As noted above, each container generated according to the disclosed embodiments can be shared among different users, thereby allowing collaboration among users. The sharing of containers can be on many levels, i.e., the entire container, input data sets, results, the container's code, or a combination thereof, can be shared. In an embodiment, two or more users can work on code of a single container. To this end, a version (or source) control is implemented to record changes to a container's code so that a user can recall specific versions as needed.

[0051] It should be noted that the disclosed embodiments allow a programmer to develop a software application including a plurality of containers that may execute the same or different algorithms. Specifically, in an embodiment, a global container is created and integrated in the code. The global container can be executed on any computing (or programming) environment. When executed, the global container can call other different containers to run the respective algorithm.

[0052] Typically, a software license is required to execute a code. For example, a Matlab® license is needed to run Matlab® code. According to various disclosed embodiments, a license manager is implemented by any of the agents 132 on the server 140. The license manager checks if a valid license exists on the local device before running a container that contains a code that requires a license. If so, execution of the container on the local device is enabled. Otherwise, the request is sent to the server 140 to grant a temporary license for executing the

container. The temporary license may be restricted to a predefined time period, a number of executions of the same container, and/or a number of executions of the different containers. In an embodiment, a monetization method is associated with the temporary licenses. That is, a user granting such a license may be billed for the service.

[0053] In another embodiment, containers stored in the data warehouse 150 can be associated with computational algorithms included in printed text books, scientific, and/or academic publications. The association is achieved using, for example, a QR code minted in the printed publication. A user, by means of a device 130, can access the containers in the data warehouse 150 by simply scanning the QR code. Containers accessed through QR codes are retrieved and saved in the server 140 to allow execution by the user device 130. In an embodiment, all such containers are also listed in a user account, allowing the user to easily access and execute the containers at a later time.

[0054] In some implementations, the server 140 typically includes a processing system (not shown) connected to a memory (not shown). The memory contains a plurality of instructions that are executed by the processing system. Specifically, the memory may include machine-readable media for storing software. Software shall be construed broadly to mean any type of instructions, whether referred to as software, firmware, middleware, microcode, hardware description language, or otherwise. Instructions may include code (e.g., in source code format, binary code format, executable code format, or any other suitable format of code). The instructions, when executed by the one or more processors, cause the processing system to perform the various functions described herein.

[0055] The processing system may comprise or be a component of a larger processing system implemented with one or more processors. The one or more processors may be implemented with any combination of general-purpose microprocessors, microcontrollers, digital signal processors (DSPs), field programmable gate array (FPGAs), graphics processing units (GPUs), programmable logic devices (PLDs), controllers, state machines, gated logic, discrete hardware components, dedicated hardware finite state machines, or any

other suitable entities that can perform calculations or other manipulations of information.

[0056] In some implementations, the agents 132-1 and 132-2 can operate and be implemented as a stand-alone program or, alternatively, can communicate and be integrated with other programs or applications executed in the client device 130.

[0057] In certain configurations, either or each of the agents 132-1 and 132-2 can be realized as a hardware component, such as a general-purpose microprocessor, microcontroller, a DSP, a FPGA, a PLD, a controller, a state machine, gated logic, a discrete hardware component, a dedicated hardware finite state machine, or any other suitable entities that can perform calculations or other manipulations of information.

[0058] It should be understood that the embodiments disclosed herein are not limited to the specific architecture illustrated in Fig. 1 and other architectures may be equally used without departing from the scope of the disclosed embodiments. Specifically, the server 140 may reside in the cloud computing platform 110, a different cloud computing platform or datacenter. Moreover, in an embodiment, there may be a plurality of servers 140 operating as described hereinabove and configured to either have one as a standby, to share the load between them, or to split the functions between them.

[0059] Fig. 3 depicts an example flowchart 300 illustrating a method for generating a container for a computational algorithm according to one embodiment.

[0060] At S310, a textual description of a computational algorithm is received. The textual description may be in a format of a pseudo code, one or more mathematical equations, a written description, a code sample or samples, any combination thereof, and the like.

[0061] At S320, if available, the programming language of the received algorithm is identified. The identification may be based on the file extension name, reserved words in the received textual description, keywords, syntax characters (e.g., '{' '}'; indentations, etc.), and so on. The programming language is any high-level

programming language that may or may not require compiling by another program. Preferably, the programming language is a scripting language.

[0062] At S330, the received algorithm's textual description is analyzed to generate a dependency tree. At S340, based, in part, on the dependency tree, a minimal set of functions that should be included in the container is determined. In an embodiment, the output of S320 is utilized for the analysis at S340. The operation of S330 and S340 will now be described with reference to Fig. 4 that depicts a schematic diagram of a dependency tree 400.

[0063] The textual description (code) of the algorithm utilized to generate the example dependency tree 400 is:

```
#!/usr/bin/env python
import os
from sys import path
from shutil import rmtree
from wand.image import Image
from wand.display import display
from lib.images import *

os.chdir('/data/)

image = 'testing_image.jpg'
resized_image = get_resized_image(image, 100, 120)
display(resized_image)
```

[0064] The textual description is a Python script. Typically, execution of a Python script requires installation of a Python environment. Further, any Python environment typically requires native Python. The received code is scanned to determine which libraries and their functions (if any) are needed for the code's execution. Such libraries and functions are identified based on the *"from <> import <>"* syntax.

[0065] As shown in Fig. 4, the required libraries are: '*path*' 411 from '*sys*' 410 library, '*display*' 421 and 'image' 422 from the '*wand*' 420 library, '*rmtree*' 431 from the '*shutil*' 430 library, as well '*os*' 440. The required functions are '*display*' 421-1 from the 'display' library, '*get_resized_image*' 422-1 from the 'image' library, and '*chdir*' 441 from the '*os*' library.

[0066] The analysis of the code (using, e.g., the identified programming language) indicates that the libraries '*sys*' 410 and '*os*' 440 are part of the native Python. Therefore, the minimal set of functions (in addition to native Python) that should be installed (or included) in the container are the 'display' 422-1 and '*get_resized_image*' 423-1. The '*chdir*' is part of the native Python that must be included in a container for executing the above example code.

[0067] It should be noted that the analysis of the code and the generation of the dependency tree ensure generation of compact containers. That is, for example, the '*wand*' library 420 includes many sub-libraries, and each sub-library includes many functions. Thus, instead of generating a container that contains the entire '*wand*' library 420, only the '*get_resized_image*' 422-1 would be included in the container.

[0068] Referring back to Fig. 3, where at S350, a container descriptor is created based on the analysis performed at S340. The container descriptor lists, at least, a minimal set of resources and their dependencies that should be included in the container. Referring to the example described herein above with respect to Fig. 4, the container descriptor would list a Python native and the functions '*display*' and '*get_resized_image*'.

[0069] In an embodiment, the container descriptor is created in a format that is supported by the type of container to be generated. For example, if such container type is Docker, the container descriptor format is a Dockerfile. A Dockerfile (container descriptor) is a script composed of various instructions and arguments that, when successively executed, perform actions on a container descriptor in order to generate a new container.

[0070] At S360, a container for executing the received algorithm is generated using the container descriptor. As a non-limiting example, such container may be a

Docker container. In an embodiment, the container is generated from a "base" container descriptor which defines mandatory resources for the proper execution of the algorithm. Referring back to the above example, the base container descriptor may define native Python and the new container would include the additional functions '*display*' and '*get_resized_image*'. In an embodiment, the user can edit or modify the container descriptor or any container generated thereof.

[0071] The various embodiments have been discussed herein with a reference to a software container. The software container can be replaced with a virtual machine, a light virtual machine, or any applicable virtualization technologies without departing from the scope of the disclosure.

[0072] The various embodiments disclosed herein can be implemented as hardware, firmware, software, or any combination thereof. Moreover, the software is preferably implemented as an application program tangibly embodied on a program storage unit or computer readable medium consisting of parts, or of certain devices and/or a combination of devices. The application program may be uploaded to, and executed by, a machine comprising any suitable architecture. Preferably, the machine is implemented on a computer platform having hardware such as one or more central processing units ("CPUs"), a memory, and input/output interfaces. The computer platform may also include an operating system and microinstruction code. The various processes and functions described herein may be either part of the microinstruction code or part of the application program, or any combination thereof, which may be executed by a CPU, whether or not such a computer or processor is explicitly shown. In addition, various other peripheral units may be connected to the computer platform such as an additional data storage unit and a printing unit. Furthermore, a non-transitory computer readable medium is any computer readable medium except for a transitory propagating signal.

[0073] It should be understood that any reference to an element herein using a designation such as "first," "second," and so forth does not generally limit the quantity or order of those elements. Rather, these designations are generally

used herein as a convenient method of distinguishing between two or more elements or instances of an element. Thus, a reference to first and second elements does not mean that only two elements may be employed there or that the first element must precede the second element in some manner. Also, unless stated otherwise a set of elements comprises one or more elements. In addition, terminology of the form "at least one of A, B, or C" or "one or more of A, B, or C" or "at least one of the group consisting of A, B, and C" or "at least one of A, B, and C" used in the description or the claims means "A or B or C or any combination of these elements." For example, this terminology may include A, or B, or C, or A and B, or A and C, or A and B and C, or 2A, or 2B, or 2C, and so on.

[0074] All examples and conditional language recited herein are intended for pedagogical purposes to aid the reader in understanding the principles of the disclosed embodiments and the concepts contributed by the inventor to furthering the art, and are to be construed as being without limitation to such specifically recited examples and conditions. Moreover, all statements herein reciting principles, aspects, and embodiments, as well as specific examples thereof, are intended to encompass both structural and functional equivalents thereof. Additionally, it is intended that such equivalents include both currently known equivalents as well as equivalents developed in the future, i.e., any elements developed that perform the same function, regardless of structure.

CLAIMS

What is claimed is:

1. A method for evaluating computational algorithms, comprising:

receiving a textual description of a computational algorithm;

generating a software container based on the received textual description;

instantiating a computing environment on a computing device, wherein the computing environment includes computing resources configured to support execution of the software container;

executing the software container in the computing environment; and

displaying, on a user device, results that are output in response to execution of the software container.

2. The method of claim 1, further comprising:

saving the software container in a data warehouse.

3. The method of claim 2, further comprising:

generating metadata indicating at least the contents of the software container; and

indexing the software container stored in the data warehouse using the generated metadata, thereby providing searchable software containers.

4. The method of claim 1, wherein the textual description may be in a format of at least one of: pseudo code, a mathematical equation, a written description, and a code sample.

5. The method of claim 1, wherein the textual description is extracted from at least one printed publication, wherein the printed publication is any of: a text book, a scientific publication, and an academic publication.

6. The method of claim 1, wherein the textual description is uploaded by a user.

7. The method of claim 1, further comprising:

benchmarking the computational algorithm based on the execution of the software container.

8. The method of claim 7, further comprising:

providing a unified input for the execution of the software container, wherein the unified input is the same for all software containers being benchmarked; and

measuring performance parameters during the execution of the software container, wherein each of the performance parameters is any of: agnostic to functions performed by the computational algorithm, and specific to at least one function performed by the computational algorithm.

9. The method of claim 1, wherein generating the software container further comprises:

analyzing the received textual description to generate a dependency tree;

determining based at least on the generated dependency tree, a minimal set of functions and resources to be included in the software container;

generating container descriptor based on the determined minimal set of functions and resources; and

generating the software container using the container descriptor.

10. The method of claim 9, further comprising:

identifying a programming language for the computational algorithm; and

determining the minimal set of functions and resources based on the identified programming language.

11. The method of claim 10, wherein the computing environment includes resources selected based on the identified programming language.

12. The method of claim 1, wherein instantiating the computing environment further comprises:

instantiating the computing environment on-demand.

13. The method of claim 1, further comprising:

caching resources produced during execution of the software container, wherein the resources include at least one of compilation files, installation files, and the results.

14. The method of claim 13, wherein the cached resources are shared among a plurality of computing environments.

15. A non-transitory computer readable medium having stored thereon instructions for causing one or more processing units to execute a method for evaluating computational algorithms comprising the steps of:

receiving a textual description of a computational algorithm;

generating a software container based on the received textual description;

instantiating a computing environment on a computing device, wherein the computing environment includes computing resources configured to support execution of the software container;

executing the software container in the computing environment; and

displaying, on a user device, results that are output in response to execution of the software container.

16. A system for evaluating computational algorithms, comprising:

a processing system; and

a memory, the memory containing instructions that, when executed by the processing system, configure the system to:

receive a textual description of a computational algorithm;

generate a software container based on the received textual description;

instantiate a computing environment on a computing device, wherein the computing environment includes computing resources configured to support execution of the software container;

execute the software container in the computing environment; and

display, on a user device, results that are output in response to execution of the software container.

17. The system of claim 16, wherein the system is further configured to:

save the software container in a data warehouse.

18. The system of claim 16, wherein the system is further configured to:

generate metadata indicating at least the contents of the software container; and

index the software container stored in the data warehouse using the generated metadata, thereby providing searchable software containers.

19. The system of claim 16, wherein the textual description may be in a format of at least one of: pseudo code, a mathematical equation, a written description, and a code sample.

20. The system of claim 16, wherein the textual description is extracted from at least one printed publication, wherein the printed publication is any of: a text book, a scientific publication, and an academic publication.

21. The system of claim 16, wherein the textual description is uploaded by a user.

22. The system of claim 16, wherein the system is further configured to:

benchmark the computational algorithm based on the execution of the software container.

23. The system of claim 22, wherein the system is further configured to:

provide a unified input for the execution of the software container, wherein the unified input is the same for all software containers being benchmarked; and

measure performance parameters during the execution of the software container, wherein each of the performance parameters is any of: agnostic to functions performed by the computational algorithm, and specific to at least one function performed by the computational algorithm.

24. The system of claim 17, wherein the system is further configured to:

analyze the received textual description to generate a dependency tree;

determine based at least on the generated dependency tree, a minimal set of functions and resources to be included in the software container;

generate container descriptor based on the determined minimal set of functions and resources; and

generate the software container using the container descriptor.

25. The system of claim 24, wherein the system is further configured to:

identify a programming language for the computational algorithm; and

determine the minimal set of functions and resources based on the identified programming language.

26. The system of claim 25, wherein the computing environment includes resources selected based on the identified programming language.

27. The system of claim 25, wherein the system is further configured to:

instantiate the computing environment on-demand.

28. The system of claim 16, wherein the system is further configured to:

cache resources produced during execution of the software container, wherein the resources include at least one of compilation files, installation files, and the results.

29. The system of claim 28, wherein the cached resources are shared among a plurality of computing environments.

30. The system of claim 14, wherein the system is configured to host the computing environment.

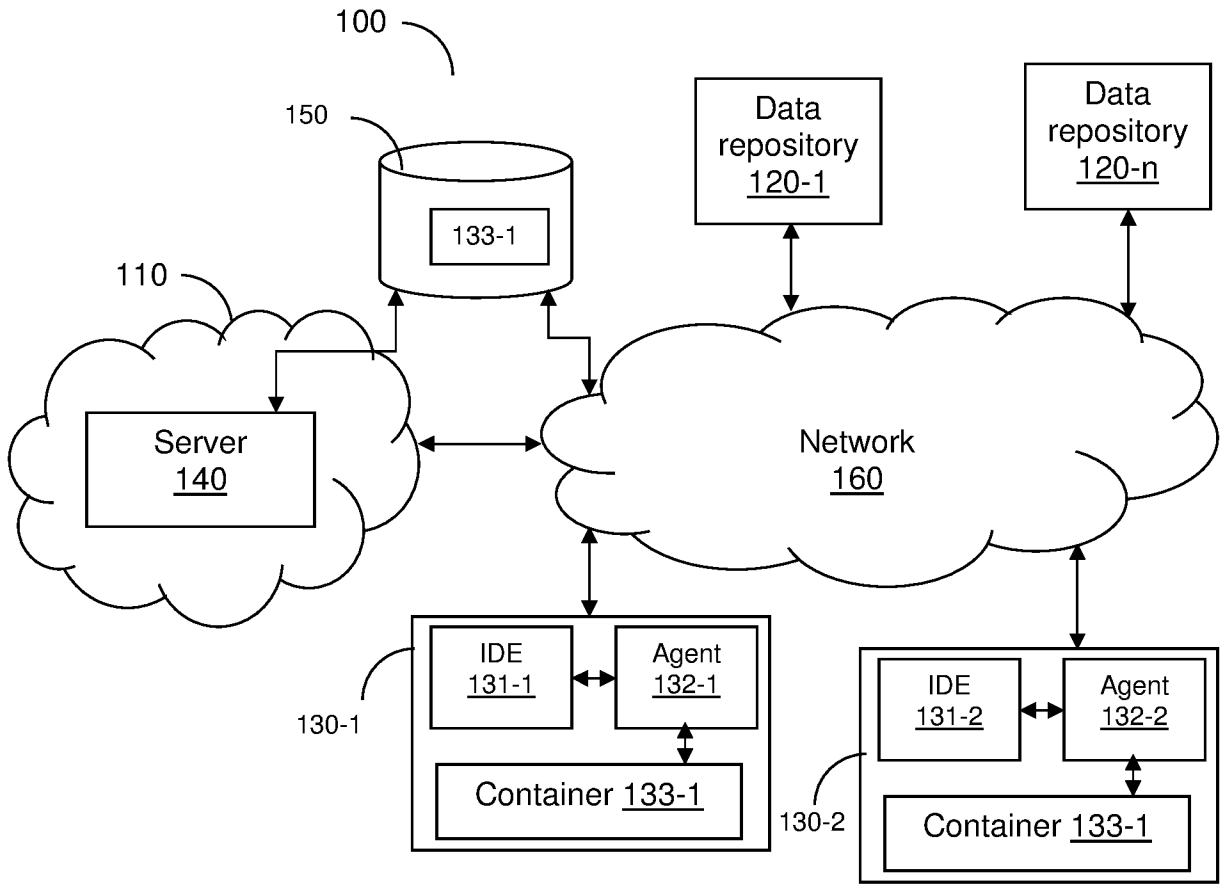
100
150
133-1
Data repository 120-1
Data repository 120-n
110
Server 140
Network 160
IDE 131-1
Agent 132-1
130-1
Container 133-1
IDE 131-2
Agent 132-2
130-2
Container 133-1

FIG. 1

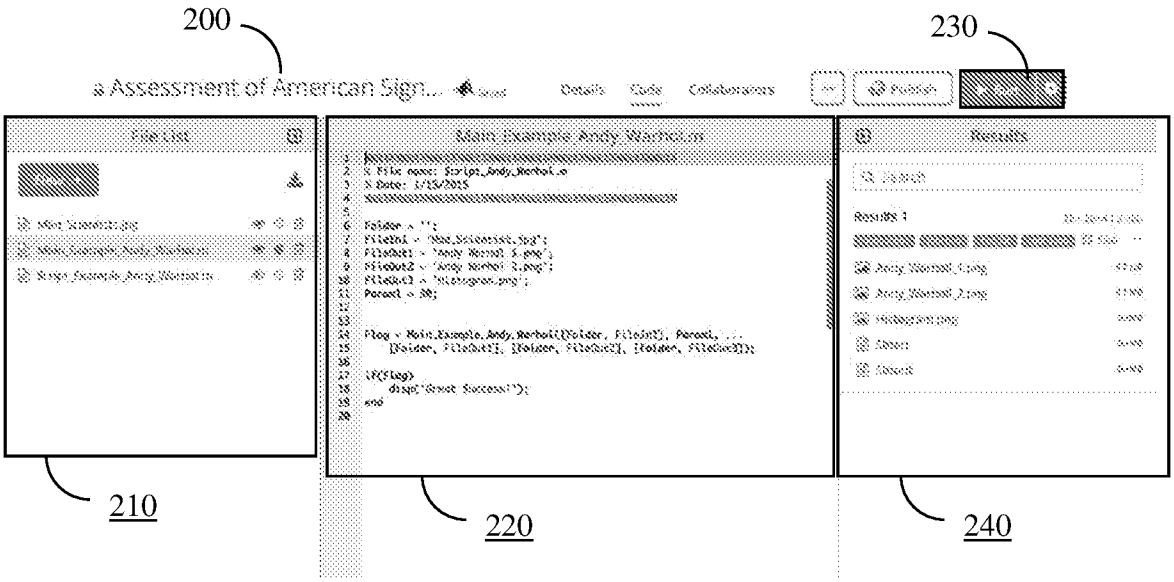
200
230
Assessment of American Sign...
Details
Code
Collaborators
Publish
File List
Main Example Andy Warhol.m
Results
210
220
240

FIG. 2

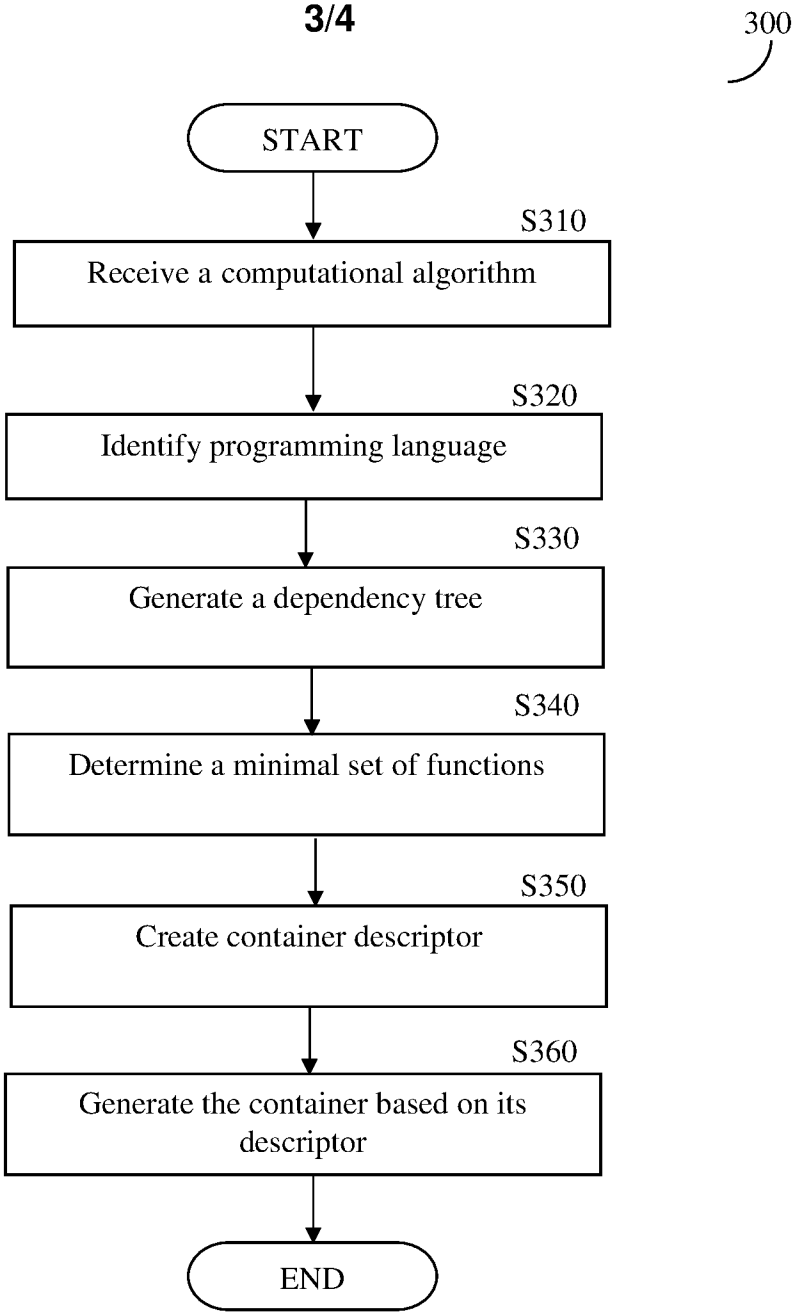
300
START
S310
Receive a computational algorithm
S320
Identify programming language
S330
Generate a dependency tree
S340
Determine a minimal set of functions
S350
Create container descriptor
S360
Generate the container based on its descriptor
END

FIG. 3

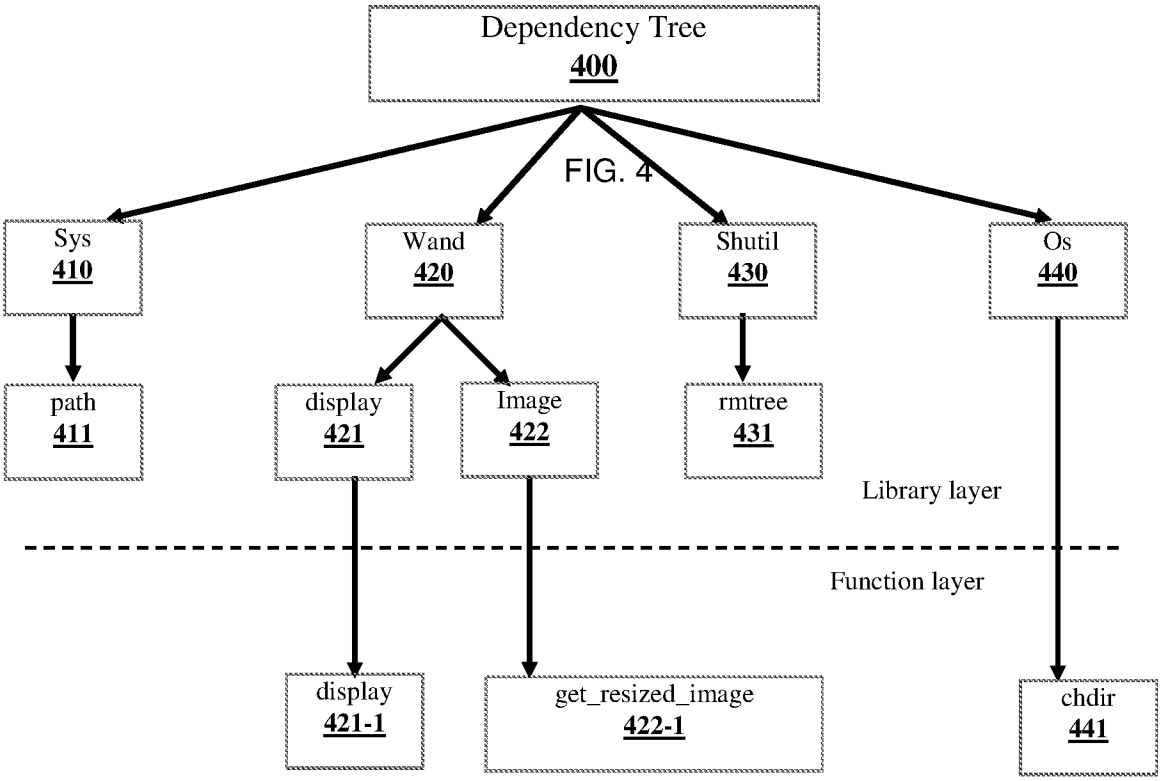
Dependency Tree
400
Sys
410
Wand
420
Shutil
430
Os
440
path
411
display
421
Image
422
rmtree
431
Library layer
Function layer
display
421-1
get_resized_image
422-1
chdir
441

FIG. 4

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US 2016/038050

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/44 *(2006.01)*

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 1/00, 9/00, 9/06, 9/44-9/54, 13/00, 13/14, 15/00, 15/16-15/177, 17/00, 21/00, H04L 29/00, 29/02, 29/06

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)

PatSearch (RUPTO internal), USPTO, PAJ, K-PION, Esp@cenet, Information Retrieval System of FIPS

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2012/0054716 A1 (THALES) 01.03.2012, abstract, paragraphs [0011], [0062], [0070], [0073], [0075]-[0078], [0085], [0096], [0112], [0115], [0145], [0151], [0157]-[0160], [0165], [0169]-[0170], [0198]-[0199], [0205], fig. 3	1, 4, 7, 12-16, 19, 22, 28, 29
Y		2, 3, 5, 6, 17, 18, 20, 21, 30
A		8-11, 23-27
Y	WO 2012/054016 A1 (HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P.) 26.04.2012, paragraphs [1012]-[1013], [1034], [1049]-[1051], [1054]-[1055], [1067], [1078]	2, 3, 6, 17, 18, 21, 30
Y	US 2015/0128105 A1 (SAP AG), 07.05.2015, paragraphs [0052], [0066]	3, 18
Y	US 2009/0108057 A1 (HONG MU et al.) 30.04.2009, paragraphs [0018], [0029], claims 1, 5	5, 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 document but published on or after the international filing date

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

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

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

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

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

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

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
04 October 2016 (04.10.2016)	13 October 2016 (13.10.2016)
Name and mailing address of the ISA/RU: Federal Institute of Industrial Property, Berezhkovskaya nab., 30-1, Moscow, G-59, GSP-3, Russia, 125993 Facsimile No: (8-495) 531-63-18, (8-499) 243-33-37	Authorized officer M. Demchenko Telephone No. (499) 240-25-91