



- (51) **International Patent Classification:**
G06F 9/54 (2006.01) **H04L 29/08** (2006.01)
- (21) **International Application Number:**
 PCT/US2014/065520
- (22) **International Filing Date:**
 13 November 2014 (13.11.2014)
- (25) **Filing Language:** English
- (26) **Publication Language:** English
- (30) **Priority Data:**
 62/023,162 10 July 2014 (10.07.2014) US
 14/525,575 28 October 2014 (28.10.2014) US
- (71) **Applicant: ORACLE INTERNATIONAL CORPORATION** [US/US]; 500 Oracle Parkway, Redwood Shores, CA 94065 (US).
- (72) **Inventor: CHOUHAN, Nirdosh;** P-413, Ittina Mahavir, Electronics City, Bangalore 560100 (IN).
- (74) **Agents: KULAS, Charles, J. et al.;** Trellis Intellectual Property Law Group, PC, 1900 Embarcadero Road, Suite 109, Palo Alto, CA 94303 (US).

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

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

Published:

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

(54) **Title:** SERVICE DISCOVERY AND/OR EFFORT ESTIMATION IN NETWORKED COMPUTING ENVIRONMENTS

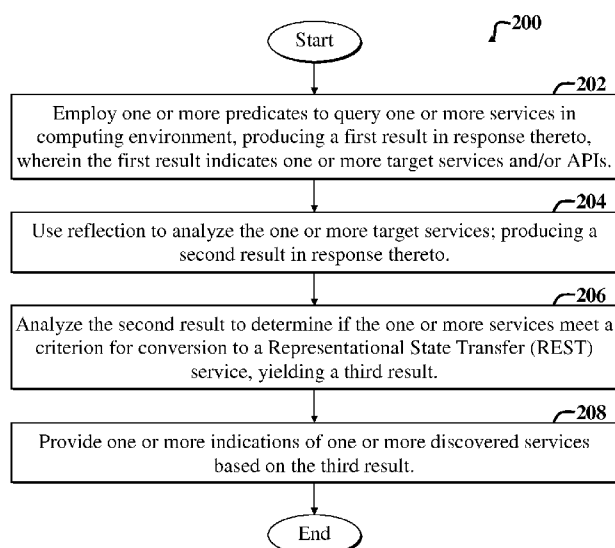


FIG. 3

(57) **Abstract:** A system and method for facilitating discovery of a software resource. The example method includes employing one or more predicates to query one or more services in a computing environment, producing a first result, wherein the first result indicates one or more target services; using reflection to analyze the one or more target services; producing a second result; analyzing the second result to determine if the one or more services meet a criterion for conversion to a Representational State Transfer (REST) service, yielding a third result; and providing one or more indications of one or more discovered services based on the third result.

PATENT APPLICATION

SERVICE DISCOVERY AND/OR EFFORT ESTIMATION IN NETWORKED COMPUTING ENVIRONMENTS

Cross References to Related Applications

This application claims priority from U.S. Provisional Patent Application Serial No. 62/023,162, entitled SERVICE DISCOVERY AND EFFORT ESTIMATE AUTOMATION, filed on July 10, 2014, which is hereby incorporated by reference as if set forth in full in this application for all purposes.

This application is related to the following application, U.S. Patent Application Serial No. 14/315,248 (Atty. Docket No. ORACP0118), entitled REST SERVICE SOURCE CODE GENERATION, filed on June 25, 2014, which is hereby incorporated by reference, as if set forth in full in this specification.

Background

[01] The present application relates to software and more specifically to systems and methods for facilitating discovering and analyzing software computing resources, such as web services or other remotely accessible software.

[02] Remotely accessible software is employed in various demanding applications, including enterprise cloud-based software accessible to distributed clients, such as mobile devices. In many enterprise computing environments, different software systems must exchange data, such as via web services or via other mechanisms for communicating data between software systems via the Internet. Such computing environments often demand efficient mechanisms for analyzing existing services and Application Programming Interfaces (APIs), such as to determine whether the services and/or APIs should be upgraded or otherwise modified to meet changing network demands.

[03] Systems and method for discovering and analyzing services and APIs of networked computing environments can be particularly important in cloud computing environments, where many computing resources may interact. For example, certain cloud computing environments may employ a certain type of Java service or API, which is to be replaced with or converted to a different type of service, e.g., a Representational State Transfer (REST) service. In a large networked computing environment, finding, i.e., discovering, the Java services and/or APIs for conversion to REST services, can be difficult.

[04] Conventionally, the task of discovering Java APIs and services to be converted to REST services involves manually analyzing each service and method to determine if it can be converted to a REST service, and then listing all DTOs (Data Transaction Objects) to be converted. However, identifying convertible Java APIs and identifying DTOs using such conventional methods can be error prone; can result in not discovering important services; and may involve inefficient duplicative effort.

Summary

[05] An example method for facilitating discovery of a software resource, such as a Java service or API, includes employing one or more predicates to query one or more services in a computing environment, producing a first result in response thereto, wherein the first result indicates one or more target services; using reflection to analyze the one or more target services; producing a second result in response thereto; analyzing the second result to determine if the one or more services meet a criterion for conversion to a Representational State Transfer (REST) service, yielding a third result; and providing one or more indications of one or more discovered services based on the third result.

[06] In a more specific embodiment, the example method employs computer code in a JAR (Java ARchive) file to implement the method, which further includes employing a predicate Java Application Programming Interface (API) to facilitate implementing the

step of employing. A reflection Java API facilitates implementing the step of using reflection.

[07] The one or more indications of one or more discovered services includes a list that indicates one or more packages, classes, and/or methods that meet predetermined criteria for conversion of the one or more services to one or more REST services. The criteria may include criteria specified via the one or more predicates.

[08] The example method further includes analyzing a package of the one or more services to determine if one or more duplicate Data Transaction Objects (DTOs) occur in the package. Any duplication of DTOs that may result from conversion of a Java service to a REST service is/are flagged.

[09] The example method may further include employing a Java API to measure function point complexity for a particular candidate Java service to be converted to a REST service. The function point complexity measurement may be based on a count of function points of a Java service being analyzed via the Java API. The function point complexity may be employed to calculate an effort estimate, which can be a metric indicating a measurement of effort required to convert a discovered service to a REST service.

[10] In summary, certain embodiments disclosed herein provide a tool, which may be implemented via a Java API, which facilitates automating finding services that can be converted to REST services or APIs. Additional analysis, such as complexity analysis, REST conversion effort estimates, and so on may further facilitate efficient service discovery, analysis, and conversion.

[11] Software that implements certain embodiments discussed herein may provide an output file that includes service analysis results, including how many services may be converted to REST services, what services may be converted to REST services, and which packages should be converted to work with REST services, and so on.

[12] A further understanding of the nature and the advantages of particular embodiments disclosed herein may be realized by reference of the remaining portions of the specification and the attached drawings.

Brief Description of the Drawings

[13] Fig. 1 is a diagram illustrating a first example system enabling automatic discovery of Java services and APIs in a networked computing environment.

[14] Fig. 2 is a flow diagram of an example process adapted for use with the embodiments of Fig. 1.

[15] Fig. 3 is a flow diagram of an example generalized method adapted for use with the embodiments of Figs. 1-2.

Detailed Description of Embodiments

[16] For the purposes of the present discussion, a computing environment may be any collection of computing resources used to perform one or more tasks involving computer processing. An enterprise computing environment may be any computing environment used for a business or organization. An example enterprise computing environment includes various computing resources distributed across a network and may further include private and shared content on intranet web servers, databases, files on local hard discs or file servers, email systems, document management systems, portals, and so on.

[17] A computing resource, also simply called resource herein, may be any aspect or feature of a computing system or environment, such as a computer network, that can be identified, e.g., via a URI or other mechanism. Similarly, a software resource may be any collection of computer code that is implemented via a computing system. Certain software resources may represent web resources.

[18] For the purposes of the present discussion, a web resource may be anything that can be named and addressed via a computer network, including computing language

classes (e.g., Java classes), objects, web services, and so on. In general, anything that is referenced via a Uniform Resource Identifier (URI) or web address, such as a URL, can be considered a web resource. A URI may be any string of characters specifying a network address.

[19] Enterprise software may be any set of computer code that is adapted to facilitate implementing any enterprise-related process or operation, such as managing enterprise resources, managing customer relations, and so on. The terms “enterprise software” and “enterprise application” may be employed interchangeably herein. However, an enterprise application may include one or more enterprise software modules or components, such as user interface (UI) software modules or components.

[20] For clarity, certain well-known components, such as hard drives, processors, operating systems, power supplies, Internet Service Providers (ISPs), integration repositories, runtime system, service loaders, and so on, are not specifically called out in the figures. However, those skilled in the art with access to the present teachings will know which components to implement and how to implement them to meet the needs of a given implementation.

[21] Fig. 1 is a diagram illustrating a first example system 10 enabling automatic discovery of Java services and APIs in a networked computing environment. The example system 10 includes one or more JEE (Java Platform Enterprise Edition) systems 16 accessible to a developer system 30, e.g., via an example interface 32.

[22] Note that in general, various modules of the system 10 may be grouped or arranged differently than shown, without departing from the scope of the present teachings. Similarly, interconnections between modules may vary depending upon the needs of a given implementation.

[23] For the purposes of the present discussion, a web service may be any software function or operation activated or implemented via software running on a server in response to a signal, e.g., a request message, sent from a client to the server via a

network, such as the World Wide Web. The software that implements a web service operation may also be called a web service, software service, or simply service herein.

[24] The example JEE system(s) supports various Java services 18, 20, a converted REST service 28 (that has been converted from a Java service), a Business Prosecution Language (BPEL) composite 26, and a Java API 24. In the present example embodiment, the JEE systems 16 and accompanying components 18-28 provide software functionality to facilitate interaction between client devices 12 and enterprise applications 14.

[25] Note that the term "Java service" as used herein refers to any Java service that is not a REST service. The developer system 30 and interface 32 are adapted to employ reflection and predicate APIs 38, 40 to discover which of the services and APIs 18-26 are suitable for conversion to REST services, such as the REST service 28, as discussed more fully below.

[26] In the present example embodiment, the example REST service 28 represents a RESTful web service implemented in accordance with the Jersey RESTful Web Services framework. The Jersey RESTful Web Services framework is an open source framework facilitating implementation of web services in Java in accordance with JAX-RS (Java API for RESTful Web Services) APIs, consistent with Java Specification Requests (JSRs) 311 and 339. A Jersey implementation of intercommunicating web resources provides a library that facilitates implementing RESTful web services in Java servlet containers.

[27] The example developer system 30 includes a User Interface (UI) display 34 in communication with discovery system GUI software 36, which is adapted to generate instructions for rendering, via the display 34, service discovery results retrieved from a discovery/analysis tool 42 of the interface 32.

[28] The example interface 32 is adapted to run the discovery/analysis tool 42, which includes computer code for using one or more Java reflection APIs 38 and Java

predicate APIs 40 to analyze the Java-based entities 18-28; particularly the Java services 18, 20, to determine if they are suitable for conversion to REST services.

[29] Exact details as to what constitutes a Java service that is suitable for conversion to a REST service are implementation specific, and may vary. For example, certain organizations may decide to invest more heavily in converting Java services to REST services and may, consequently, be willing to spend more effort converting some services than other organizations might. Those skilled in the art with access to the present teachings may readily determine appropriate criteria for selection of Java services for conversion to REST services and apply the criteria to search the JEE computing system(s) 16 for suitable services. The criterion or criteria for discovering candidate services for analysis may be provided via use of predicates, as discussed more fully below.

[30] In the present example embodiment, the discovery/analysis tool 42 includes computer code for implementing function point complexity analysis 44 and effort estimation 46 based on the function point complexity analysis results. Computer code for implementing the discovery/analysis tool 42 may be maintained in a Java ARchive (JAR) file.

[31] Note that while the function point complexity module 44 and effort estimation module 46 are shown running on an interface 32, that embodiments are not limited thereto. For example, the analysis modules 44, 46 or portions thereof may be implemented via computer code running on the developer system 30 as part of the discovery system GUI software 36.

[32] While the discovery/analysis tool 42 is shown only including the function point complexity module 44 and the effort estimation module 46, that in practice, additional functionality and resources may be employed, such as data storage for storing package information; computer code for performing duplicate Data Transaction Object (DTO) (also called Data Transfer Object) analysis; computer code for marshalling data from a data store into eXtensible Markup Language (XML) and/or HyperText Markup

Language (HTML) files, and so on, may be implemented via or in concert with the discover/analysis tool 42.

[33] The resulting HTML file contains discovery and analysis results, which are consumable and/or readable and displayable via the developer system and GUI 30. Note that the developer system 30 may be implemented as one of the client devices 12, without departing from the scope of the present teachings.

[34] In general, services and associated information indicated in XML or HTML files retrieved by the developer system 30 represent REST convertible Java services. In the present example embodiment, the discovered services are maintained in a JAR file that is accessible to a Java Virtual Machine (JVM) at runtime.

[35] Fig. 2 is a flow diagram of an example process 60 adapted for use with the embodiments of Figs. 1 and 2. With reference to Figs. 1 and 2, the method 60 may be implemented substantially via computer code running on the interface 32 and/or developer system 30 Fig. 1.

[36] The process 60 includes initiating package filtering by obtaining one or more package predicates 62 to be applied to a Java package 64 associated with Java services to be analyzed for conversion to REST services.

[37] Java services represented by classes of the Java service package 64 of the package store 74 are obtained from a class path. In general, existing services are provided as input to the process 60 as compiled code that is configured in the class path. A class path may be information informing a JVM where to look for certain classes and packages.

[38] For the purposes of the present discussion, a Java package may be a mechanism for grouping or organizing Java classes used to implement Java applications or services. Java packages may be used group or store similarly categorized classes to facilitate organization of the classes.

[39] A package predicate 62, which may be specified by a developer or automatically, is employed to filter package(s) 64 based on packages obtained from a package store 74. Packages matching criterion or criteria specified via the package predicate 62 are indicated in a filtered package store 80.

[40] After Java packages are filtered based on any package predicate 62, class filtering begins. Class filtering includes applying a class predicate 66 to classes 68 of filtered packages of the filtered package store 80, which are maintained via a class store 76. All filtered classes that match a condition specified by the class predicate 66 are maintained in a filtered class store 82.

[41] Next, filtering begins, whereby methods of a method store 78 representing methods 72 from filtered classes of the filtered class store 82 are checked for matching a condition of a specified method predicate 70. Methods matching the criterion or criteria are maintained in a filtered method store 84.

[42] Subsequently, Input/Output (I/O) method parameters are analyzed in an initial analysis step 90, as part of complexity analysis. The results of the analysis step 90 may or include metrics, such as function point complexity.

[43] Duplicate Data Transaction Object (DTO) analysis 92 is then performed. Note that the DTO analysis 92 may be performed in parallel with the analysis step 90, without departing from the scope of the present teachings. The DTO analysis may include specifying all DTO objects that will be duplicative upon conversion to REST services if an adjustment is not made to remove the duplicity. Results of the initial analysis 90 and DTO analysis 92 may be stored in an analyzed DTO store 86.

[44] Filtered data and/or analysis results from the various stores 80-86 are then combined into a composite store 88, which maintains information pertaining to packages, methods, parameters, and so on, of discovered services. This data is then selectively marshalled, e.g., via an XML marshal 94, into eXtensible Stylesheet Language (XSL), and/or HTML or other markup language 96 readable in a displaying

step 98. The displaying step 98 may employ a browser to read output from the marshalling step 94.

[45] Fig. 3 is a flow diagram of an example generalized method 200 adapted for use with the embodiments of Figs. 1-2. The example method 200 includes a first step 202, which involves employing one or more predicates to query one or more services in computing environment, producing a first result in response thereto, wherein the first result indicates one or more target services.

[46] A second step 204 includes using reflection to analyze the one or more target services; producing a second result in response thereto.

[47] A third step 206 includes analyzing the second result to determine if the one or more services meet a criterion for conversion to a Representational State Transfer (REST) service, yielding a third result.

[48] A fourth step 208 includes providing one or more indications of one or more discovered services based on the third result.

[49] Note that various steps of the method 200 may be modified, omitted, augmented, or rearranged, without departing from the scope of the present teachings. For example, the method 200 may further include employs computer code in a JAR (Java ARchive) file to implement the method, and employing a predicate Java API to facilitate implementing the step of employing. A reflection Java API may facilitate implementing the step of using reflection.

[50] The one or more indications of one or more discovered services may include a list that indicates one or more packages, classes, and/or methods that meet predetermined criteria for conversion of the one or more services to one or more REST services. The criteria may include criteria specified via the one or more predicates.

[51] The method 200 further include analyzing a package of the one or more services to determine if one or more duplicate DTOs occur in the package. Any duplication of

DTOs that may result from conversion of a Java service to a REST service may then be flagged.

[52] The example method 200 may further include employing a Java API to measure function point complexity for a particular candidate Java service to be converted to a REST service. The function point complexity measurement may be based on a count of function points of a Java service being analyzed via the Java API. The function point complexity may be employed to calculate an effort estimate, which can be a metric indicating a measurement of effort required to convert a discovered service to a REST service.

[53] In summary, certain embodiments disclosed herein provide a tool, which may be implemented via a Java API, which automates finding services that can be converted to REST services or APIs. The example tool provides function point complexity metrics, which can be directly used for effort estimation. The example tool can be consumed by virtually any Java program to obtain various program details.

[54] The complexity analysis enabled via certain embodiments discussed herein may provide various metrics and indicators to enable developers and administrators to decide whether to proceed with conversion of a service to a REST service. Additional mechanisms facilitate automatically analyzing service packages and flagging or indicating any duplicate Data Transaction Objects (DTOs).

[55] Software that implements certain embodiments discussed herein may provide an output file that includes service analysis results, including how many services may be converted to REST services, what services may be converted to REST services, and which packages should be converted to work with REST services, and so on.

[56] Although the description has been described with respect to particular embodiments thereof, these particular embodiments are merely illustrative, and not restrictive. For example, while certain embodiments discuss analysis of Java services or APIs to facilitate determining suitability of the services or APIs for conversion to Jersey REST services, implementations are not limited thereto. Certain embodiment discussed

herein may be readily adapted to facilitate determining suitability of different types of services for conversion to other types of services, such as RestEasy or WildFly service implementations.

[57] Any suitable programming language can be used to implement the routines of particular embodiments including C, C++, Java, assembly language, etc. Different programming techniques can be employed such as procedural or object oriented. The routines can execute on a single processing device or multiple processors. Although the steps, operations, or computations may be presented in a specific order, this order may be changed in different particular embodiments. In some particular embodiments, multiple steps shown as sequential in this specification can be performed at the same time.

[58] Particular embodiments may be implemented in a computer-readable storage medium for use by or in connection with the instruction execution system, apparatus, system, or device. Particular embodiments can be implemented in the form of control logic in software or hardware or a combination of both. The control logic, when executed by one or more processors, may be operable to perform that which is described in particular embodiments.

[59] Particular embodiments may be implemented by using a programmed general purpose digital computer, by using application specific integrated circuits, programmable logic devices, field programmable gate arrays, optical, chemical, biological, quantum or nanoengineered systems, components and mechanisms may be used. In general, the functions of particular embodiments can be achieved by any means as is known in the art. Distributed, networked systems, components, and/or circuits can be used. Communication, or transfer, of data may be wired, wireless, or by any other means.

[60] It will also be appreciated that one or more of the elements depicted in the drawings/figures can also be implemented in a more separated or integrated manner, or even removed or rendered as inoperable in certain cases, as is useful in accordance with

a particular application. It is also within the spirit and scope to implement a program or code that can be stored in a machine-readable medium to permit a computer to perform any of the methods described above.

[61] As used in the description herein and throughout the claims that follow, “a”, “an”, and “the” includes plural references unless the context clearly dictates otherwise. Also, as used in the description herein and throughout the claims that follow, the meaning of “in” includes “in” and “on” unless the context clearly dictates otherwise.

[62] Thus, while particular embodiments have been described herein, latitudes of modification, various changes, and substitutions are intended in the foregoing disclosures, and it will be appreciated that in some instances some features of particular embodiments will be employed without a corresponding use of other features without departing from the scope and spirit as set forth. Therefore, many modifications may be made to adapt a particular situation or material to the essential scope and spirit.

Claims

We claim:

1. A method for facilitating discovery of a software resource in a computing environment that includes a client device operating in a system including one or more client devices in communication with a server system, wherein the server system executes webpage software, wherein the webpage software provides webpage information accessible to the client device, the client device executing steps of the method, the method comprising:

employing one or more predicates to query one or more services in computing environment, producing a first result in response thereto, wherein the first result indicates one or more target services;

using reflection to analyze the one or more target services; producing a second result in response thereto;

analyzing the second result to determine if the one or more services meet a criterion for conversion to a Representational State Transfer (REST) service, yielding a third result in response thereto; and

providing one or more indications of one or more discovered services based on the third result in response thereto.

2. The method of claim 1, further including implementing the method of claim 1, using computer code in a JAR (Java ARchive) file.

3. The method of claim 1, further including employing a predicate Java Application Programming Interface (API) to facilitate implementing the step of employing.

4. The method of claim 3, further including employing a reflection Java API to facilitate implementing the step of using reflection.

5. The method of claim 1, wherein the one or more indications include a list that includes one or more indications of one or more packages, classes, or methods that meet a criterion for inclusion in a process to convert the one or more services to one or more REST services.

6. The method of claim 1, further including specifying the criterion via the one or more predicates.

7. The method of claim 6, further including analyzing a package of the one or more services to determine if one or more duplicate Data Transaction Objects (DTOs) occur in the package.

8. The method of claim 1, further including employing a Java Application Programming Interface (API) to implement the steps of employing, using, and analyzing.

9. The method of claim 8, wherein the one or more services include one or more Java services, which are adapted to provide output indicating one or more characteristics of the one or more Java services in response to a signal from the API.

10. The method of claim 9, wherein the one or more predicates include one or more package predicates.

11. The method of claim 10, wherein the one or more package predicates includes a wildcard predicate.

12. The method of claim 9, wherein the one or more predicates include one or more class predicates.

13. The method of claim 12, wherein the one or more predicates is adapted to indicate whether a name of the one or more services meets a name criterion.

14. The method of claim 9, wherein the one or more predicates include one or more method predicates, and wherein the one or more predicates includes a modifier predicate, which is adapted to provide an indication as to whether a method exhibits a characteristic specified via the predicate.

15. The method of claim 8, further including using the Java API to measure function point complexity.

16. The method of claim 15, further including employing a function point complexity measurement to calculate an effort estimation, wherein the effort estimation represents a metric indicating effort to convert a discovered service to a REST service, and wherein function point complexity measurement includes a count of function points of a Java service being analyzed via the API.

17. The method of claim 16, further including automatically detecting any duplication of Data Transaction Objects (DTOs) that may result from conversion of a Java service to a REST service.

18. The method of claim 16, further including providing one or more statistics or metrics pertaining to the one or more services, wherein the one or more statistics include a first metric indicating a number of services included among the one or more services, which are to be converted to REST services, and wherein the one more statistics or metrics include one or more indications of one or more program packages to be modified for use with one or more REST services.

19. An apparatus comprising:

a digital processor coupled to a display and to a processor-readable storage device, wherein the processor-readable storage device includes one or more instructions executable by the digital processor to perform the following acts:

employing one or more predicates to query one or more services in computing environment, producing a first result in response thereto, wherein the first result indicates one or more target services;

using reflection to analyze the one or more target services; producing a second result in response thereto;

analyzing the second result to determine if the one or more services meet a criterion for conversion to a Representational State Transfer (REST) service, yielding a third result in response thereto; and

providing one or more indications of one or more discovered services based on the third result in response thereto.

20. A processor-readable storage device including instructions executable by a digital processor, the processor-readable storage device including one or more instructions for:

employing one or more predicates to query one or more services in computing environment, producing a first result in response thereto, wherein the first result indicates one or more target services;

using reflection to analyze the one or more target services; producing a second result in response thereto;

analyzing the second result to determine if the one or more services meet a criterion for conversion to a Representational State Transfer (REST) service, yielding a third result in response thereto; and

providing one or more indications of one or more discovered services based on the third result in response thereto.

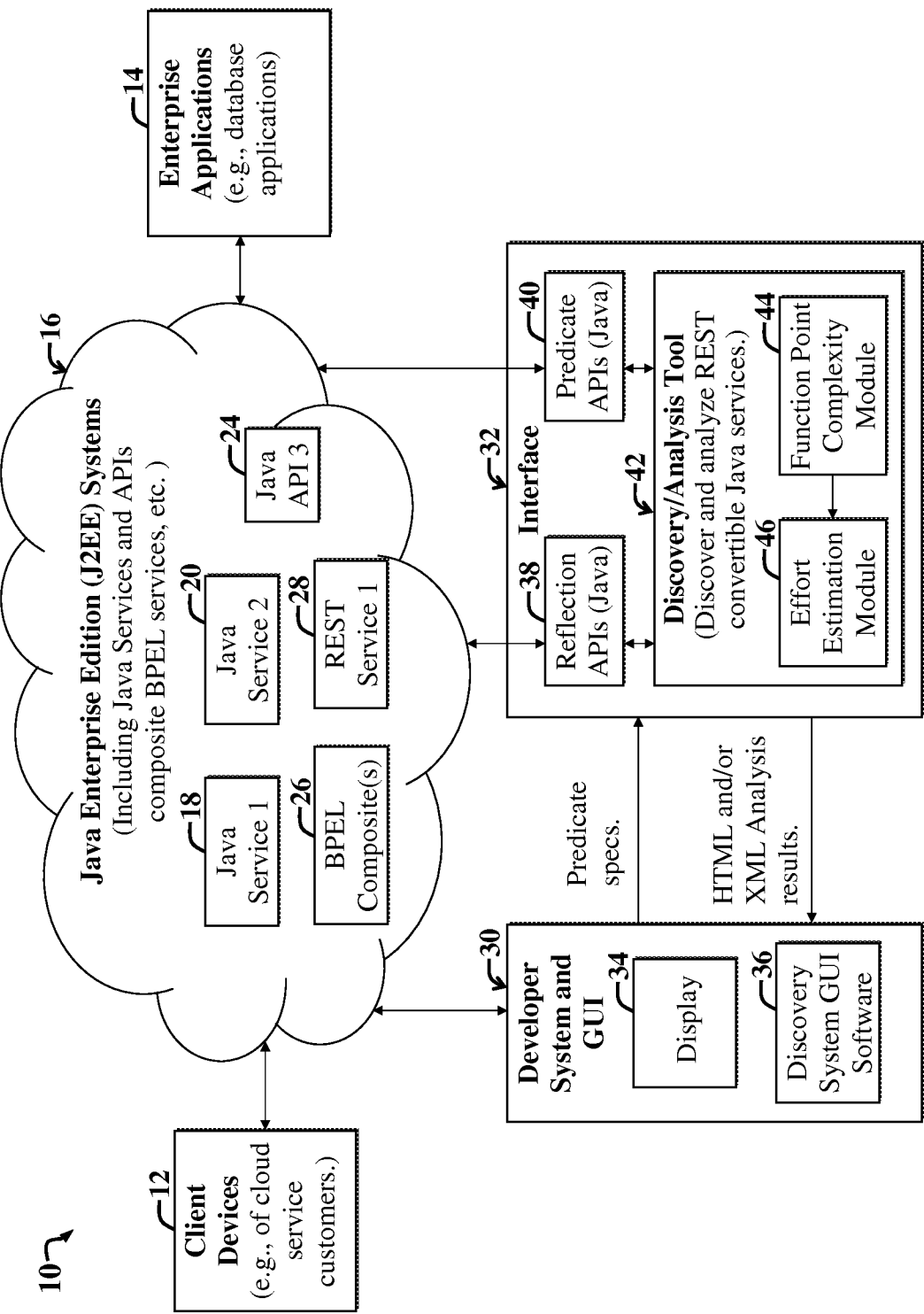


FIG. 1

2/3

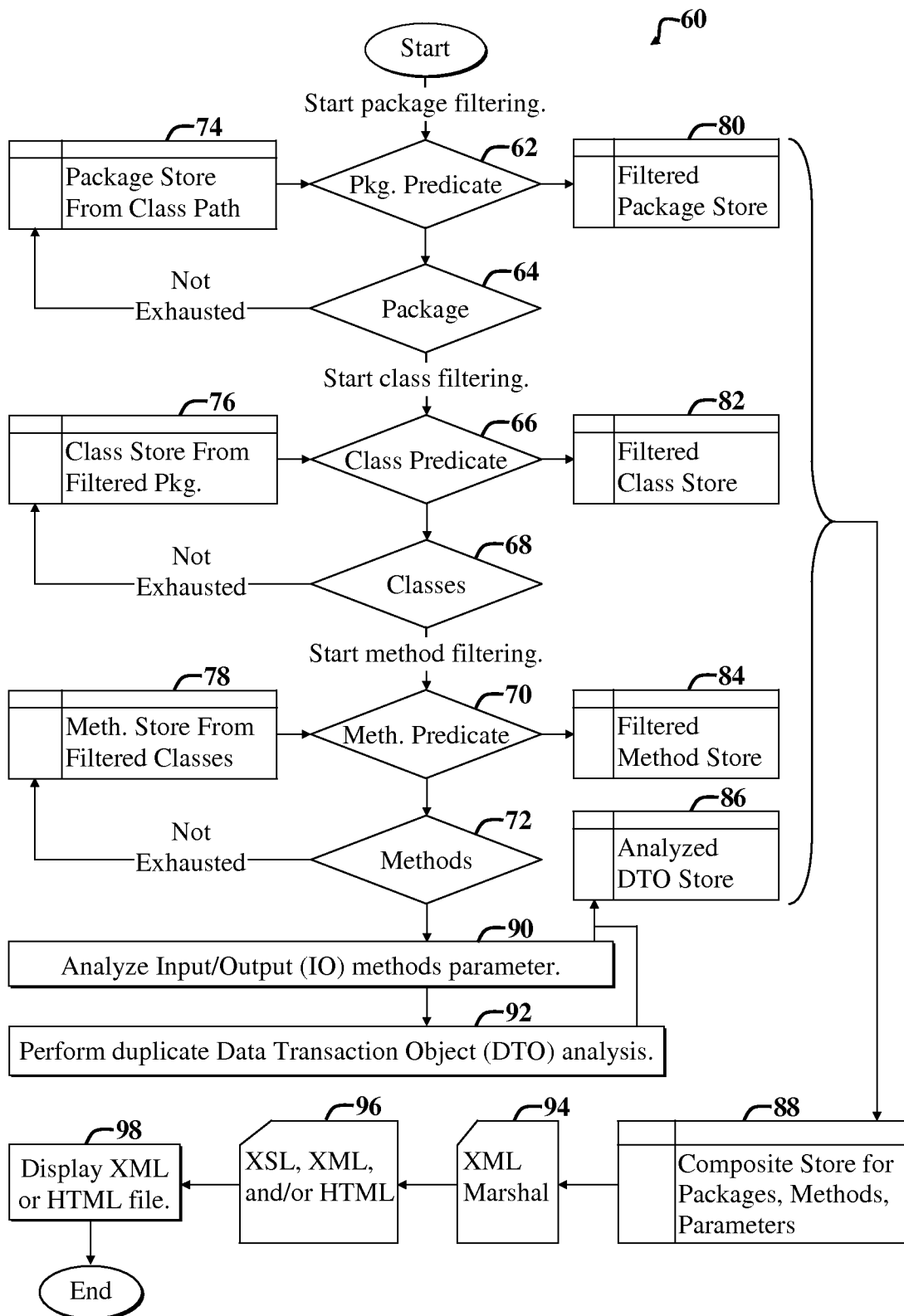
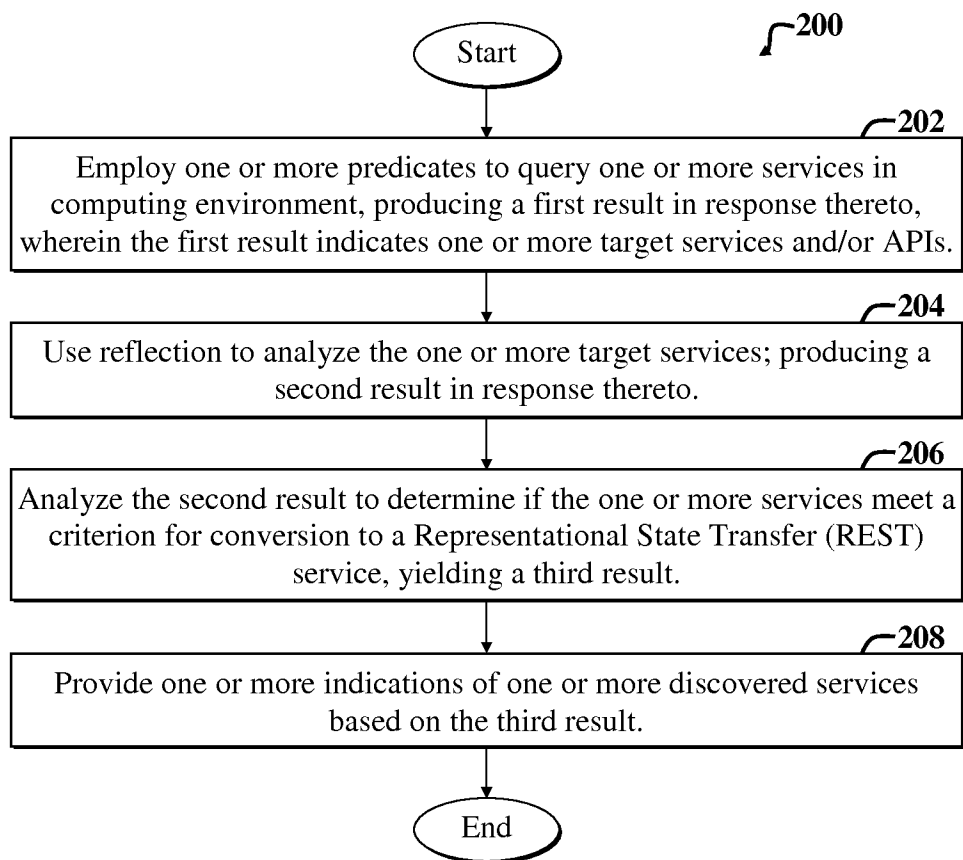


FIG. 2

3/3

**FIG. 3**

INTERNATIONAL SEARCH REPORT

International application No

PCT/US2014/065520

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/54 H04L29/08
ADD.

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F H04L

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

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

EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	A. KOZLENKOV ET AL: "A framework for architecture-driven service discovery", PROCEEDINGS OF THE 2006 INTERNATIONAL WORKSHOP ON SERVICE-ORIENTED SOFTWARE ENGINEERING , SOSE '06, May 2006 (2006-05), page 67, XP055173588, New York, New York, USA DOI: 10.1145/1138486.1138501 page 67, right column, third paragraph page 70, left column, third paragraph page 70, right column, first paragraph -----	1-20



Further documents are listed in the continuation of Box C.



See patent family annex.

* Special categories of cited documents :

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

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

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

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

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

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

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

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

"&" document member of the same patent family

Date of the actual completion of the international search

9 March 2015

Date of mailing of the international search report

18/03/2015

Name and mailing address of the ISA/

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

Authorized officer

Del Castillo, G