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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 October 2014 (30.10.2014)

WIPO | PCT

(10) International Publication Number
WO 2014/176137 A1

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

(21) International Application Number:
PCT/US2014/034739

(22) International Filing Date:
21 April 2014 (21.04.2014)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
13/867,143 22 April 2013 (22.04.2013) US

(71) Applicant: **MICROSOFT CORPORATION** [US/US]; One Microsoft Way, Redmond, WA 98052-6399 (US).

(72) Inventors: **TROFIN, Mircea**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (US). **DUSSUD, Patrick**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (US). **MARTIN, Rudi**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (US). **HAMBY, John Lawrence**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (US). **STREHOVSKY, Michal**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (US). **WRIGHTON, David Charles**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (US). **KANAMORI, Atsushi**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (US). **HANNA, Fadi M.**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (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, LT, 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, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

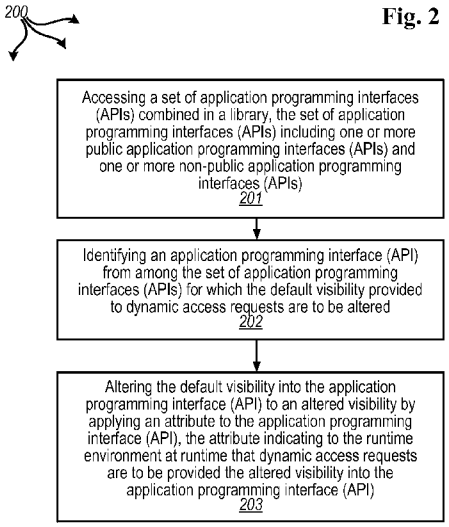
— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

[Continued on next page]

(54) Title: CONTROLLING RUNTIME ACCESS TO APPLICATION PROGRAMMING INTERFACES

Fig. 2



(57) Abstract: The present invention extends to methods, systems, and computer program products for controlling runtime access to application programming interfaces Embodiments of the invention allow library developers to more precisely and easily control which of their libraries' APIs can be called dynamically. Thus, their servicing and versioning burden can be more appropriately controlled. Further, application developers can control which such APIs to further exclude from dynamic calling scenarios, to minimize the runtime support overhead (e.g., preventing generation of metadata).

WO 2014/176137 A1

Published:

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

CONTROLLING RUNTIME ACCESS TO APPLICATION PROGRAMMING INTERFACES

BACKGROUND

[0001] **Background and Relevant Art**

[0002] Computer systems and related technology affect many aspects of society. Indeed, the computer system's ability to process information has transformed the way we live and work. Computer systems now commonly perform a host of tasks (e.g., word processing, scheduling, accounting, etc.) that prior to the advent of the computer system were performed manually. More recently, computer systems have been coupled to one another and to other electronic devices to form both wired and wireless computer networks over which the computer systems and other electronic devices can transfer electronic data. Accordingly, the performance of many computing tasks is distributed across a number of different computer systems and/or a number of different computing environments.

[0003] During code development, software developers often use Application Programing Interfaces (APIs) to facilitate communication between software components. An API can include a specification for routines, data structures, object classes, and variables associated with the API. As such, one developer can use an API specification to determine how to call an API written by another developer.

[0004] Often, one software developer (a library developer) develops code having a number of related APIs that are grouped into a library offering specified functionality. The software developer can make parts of the functionality available to other programs by exposing corresponding APIs within the library as public APIs. Thus, another developer (an application developer) can access the available parts of functionality from within their code through calls to the public APIs. The software developer can also maintain other parts of the functionality as private. The private functionality can be used internally between APIs within the library or to access other private APIs in other libraries. APIs providing the private functionality are not directly exposed to other programs.

[0005] However, many runtime environments permit code to dynamically call any API in a third party library (e.g., using reflection to access metadata). As such, an application program or library can identify and call private APIs within another library. When a software developer maintains an API is private, the software developer does not expect the API to be externally called. Unfortunately, servicing or versioning changes to internal implementation

details of such libraries, such as, for example, renames or removals of private methods, have the potential of causing breaking changes in applications using these libraries.

[0006] In addition, runtime support for dynamic calls carries a size and working set overhead (e.g., metadata). The size and working set overhead is maintained for APIs (private or public) whether or not the APIs are actually called dynamically. Maintaining size and working set overhead for uncalled APIs unnecessarily consumes computing resources.

BRIEF SUMMARY

[0007] The present invention extends to methods, systems, and computer program products for controlling runtime access to application programming interfaces. Embodiments of the invention include controlling runtime access to an application programming interfaces (API). A runtime environment provides dynamic access requests (e.g., through reflection or other dynamic calling techniques) with a default visibility into APIs based on API type. For example, the default visibility into non-public APIs may prevent dynamic access.

[0008] A set of APIs combined in a library is accessed. The set of APIs include one or more public APIs and one or more non-public APIs. An API is identified from among the set of APIs for which the default visibility provided to dynamic access requests is to be altered.

[0009] The default visibility into the API is altered by applying an attribute to the API. The attribute indicates to the runtime environment at runtime that dynamic access requests are to be provided the altered visibility into the API. Altering visibility can include permitting dynamic access to a non-public API where by default dynamic access is not permitted.

[0010] Other embodiments include reducing the default visibility into an accessible API. Application code for an application is accessed. The application code refers to one or more accessible APIs combined in a library. An accessible API referred to within the application code is identified. The accessible API is selected from among the one or more accessible APIs.

[0011] It is determined that the accessible API is not to be dynamically accessed at runtime. An attribute is applied to the accessible API to reduce the default visibility into the accessible API. The attribute indicates to a runtime environment at runtime that dynamic access requests (e.g., through reflection or other dynamic calling techniques) are to be provided with reduced visibility into the accessible API. Reducing visibility into an API can

correspondingly reduce metadata generation. In some embodiments, reducing visibility into an API includes preventing dynamic access to the API.

[0012] Further embodiments include providing a consumer with specified visibility into an API. Executable code is executed in a runtime environment. The executable code is derived from application code. During execution of the executable code, a dynamic call is received from a consumer to execute a portion of the executable code. The portion of the executable code is derived from a portion of the application code that refers to an API within a library. A runtime default visibility into the API is accessed based on the type of the API.

[0013] Any attributes applied to the API are accessed. Attributes applied to the API can be indicative of a desire by the author of API to alter the runtime default visibility into the API. Any attributes applied to the portion of the application code are accessed. Attributes applied to the portion of the application code can be indicative of a desire by the author of the application code to provide less visibility into the API than indicated by the runtime default visibility as altered by any attributes applied to the API.

[0014] A calculated visibility into the API is determined based on one or more of: the runtime default visibility into the API, any attributes applied to the API, and any attributes applied to the portion of the application code that refers to the API. The dynamic call is provided with the calculated visibility into the API.

[0015] This summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter.

[0016] Additional features and advantages of the invention will be set forth in the description which follows, and in part will be obvious from the description, or may be learned by the practice of the invention. The features and advantages of the invention may be realized and obtained by means of the instruments and combinations particularly pointed out in the appended claims. These and other features of the present invention will become more fully apparent from the following description and appended claims, or may be learned by the practice of the invention as set forth hereinafter.

BRIEF DESCRIPTION OF THE DRAWINGS

[0017] In order to describe the manner in which the above-recited and other advantages and features of the invention can be obtained, a more particular description of the invention briefly described above will be rendered by reference to specific embodiments thereof which are illustrated in the appended drawings. Understanding that these drawings depict only

typical embodiments of the invention and are not therefore to be considered to be limiting of its scope, the invention will be described and explained with additional specificity and detail through the use of the accompanying drawings in which:

[0018] Figures 1 illustrates an example computer architecture that facilitates controlling runtime access to an application programming interface.

[0019] Figure 2 illustrates a flow chart of an example method for controlling runtime access to an application programming interface.

[0020] Figure 3 illustrates an example computer architecture that facilitates reducing the default visibility into an accessible application programming interface.

[0021] Figure 4 illustrates a flow chart of an example method for reducing the default visibility into an accessible application programming interface.

[0022] Figure 5 illustrates an example computer architecture that facilitates providing a consumer with specified visibility into an application programming interface.

[0023] Figure 6 illustrates a flow chart of an example method for providing a consumer with specified visibility into an application programming interface.

DETAILED DESCRIPTION

[0024] The present invention extends to methods, systems, and computer program products for controlling runtime access to application programming interfaces. Embodiments of the invention include controlling runtime access to an application programming interfaces (API). A runtime environment provides dynamic access requests (e.g., through reflection or other dynamic calling techniques) with a default visibility into APIs based on API type. For example, the default visibility into non-public APIs may prevent dynamic access.

[0025] A set of APIs combined in a library is accessed. The set of APIs include one or more public APIs and one or more non-public APIs. An API is identified from among the set of APIs for which the default visibility provided to dynamic access requests is to be altered.

[0026] The default visibility into the API is altered by applying an attribute to the API. The attribute indicates to the runtime environment at runtime that dynamic access requests are to be provided the altered visibility into the API. Altering visibility can include permitting dynamic access to a non-public API where by default dynamic access is not permitted.

[0027] Other embodiments include reducing the default visibility into an accessible API. Application code for an application is accessed. The application code refers to one or more

accessible APIs combined in a library. An accessible API referred to within the application code is identified. The accessible API is selected from among the plurality of accessible APIs.

[0028] It is determined that the accessible API is not to be dynamically accessed at runtime. An attribute is applied to the accessible API to reduce the default visibility into the accessible API. The attribute indicates to a runtime environment at runtime that dynamic access requests (e.g., through reflection or other dynamic calling techniques) are to be provided with reduced visibility into the accessible API. Reducing visibility into an API can correspondingly reduce metadata generation. In some embodiments, reducing visibility into an API includes preventing dynamic access to the API.

[0029] Further embodiments include providing a consumer with specified visibility into an API. Executable code is executed in a runtime environment. The executable code is derived from application code. During execution of the executable code, a dynamic call is received from a consumer to execute a portion of the executable code. The portion of the executable code is derived from a portion of the application code that refers to an API within a library. A runtime default visibility into the API is accessed based on the type of the API.

[0030] Any attributes applied to the API are accessed. Attributes applied to the API can be indicative of a desire by the author of API to alter the runtime default visibility into the API. Any attributes applied to the portion of the application code are accessed. Attributes applied to the portion of the application code can be indicative of a desire by the author of the application code to provide less visibility into the API than indicated by the runtime default visibility as altered by any attributes applied to the API.

[0031] A calculated visibility into the API is determined based on one or more of: the runtime default visibility into the API, any attributes applied to the API, and any attributes applied to the portion of the application code that refers to the API. The dynamic call is provided with the calculated visibility into the API.

[0032] Embodiments of the present invention may comprise or utilize a special purpose or general-purpose computer including computer hardware, such as, for example, one or more processors and system memory, as discussed in greater detail below. Embodiments within the scope of the present invention also include physical and other computer-readable media for carrying or storing computer-executable instructions and/or data structures. Such computer-readable media can be any available media that can be accessed by a general purpose or special purpose computer system. Computer-readable media that store computer-executable instructions are computer storage media (devices). Computer-readable media

that carry computer-executable instructions are transmission media. Thus, by way of example, and not limitation, embodiments of the invention can comprise at least two distinctly different kinds of computer-readable media: computer storage media (devices) and transmission media.

[0033] Computer storage media (devices) includes RAM, ROM, EEPROM, CD-ROM, solid state drives ("SSDs") (e.g., based on RAM), Flash memory, phase-change memory ("PCM"), other types of memory, other optical disk storage, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store desired program code means in the form of computer-executable instructions or data structures and which can be accessed by a general purpose or special purpose computer.

[0034] A "network" is defined as one or more data links that enable the transport of electronic data between computer systems and/or modules and/or other electronic devices. When information is transferred or provided over a network or another communications connection (either hardwired, wireless, or a combination of hardwired or wireless) to a computer, the computer properly views the connection as a transmission medium. Transmissions media can include a network and/or data links which can be used to carry desired program code means in the form of computer-executable instructions or data structures and which can be accessed by a general purpose or special purpose computer. Combinations of the above should also be included within the scope of computer-readable media.

[0035] Further, upon reaching various computer system components, program code means in the form of computer-executable instructions or data structures can be transferred automatically from transmission media to computer storage media (devices) (or vice versa). For example, computer-executable instructions or data structures received over a network or data link can be buffered in RAM within a network interface module (e.g., a "NIC"), and then eventually transferred to computer system RAM and/or to less volatile computer storage media (devices) at a computer system. Thus, it should be understood that computer storage media (devices) can be included in computer system components that also (or even primarily) utilize transmission media.

[0036] Computer-executable instructions comprise, for example, instructions and data which, when executed at a processor, cause a general purpose computer, special purpose computer, or special purpose processing device to perform a certain function or group of functions. The computer executable instructions may be, for example, binaries, intermediate format instructions such as assembly language, or even source code. Although the subject

matter has been described in language specific to structural features and/or methodological acts, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the described features or acts described above. Rather, the described features and acts are disclosed as example forms of implementing the claims.

[0037] Those skilled in the art will appreciate that the invention may be practiced in network computing environments with many types of computer system configurations, including, personal computers, desktop computers, laptop computers, message processors, hand-held devices, multi-processor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers, mobile telephones, PDAs, tablets, pagers, routers, switches, and the like. The invention may also be practiced in distributed system environments where local and remote computer systems, which are linked (either by hardwired data links, wireless data links, or by a combination of hardwired and wireless data links) through a network, both perform tasks. In a distributed system environment, program modules may be located in both local and remote memory storage devices.

[0038] Embodiments of the invention can also be implemented in cloud computing environments. In this description and the following claims, "cloud computing" is defined as a model for enabling on-demand network access to a shared pool of configurable computing resources. For example, cloud computing can be employed in the marketplace to offer ubiquitous and convenient on-demand access to the shared pool of configurable computing resources. The shared pool of configurable computing resources can be rapidly provisioned via virtualization and released with low management effort or service provider interaction, and then scaled accordingly.

[0039] A cloud computing model can be composed of various characteristics such as, for example, on-demand self-service, broad network access, resource pooling, rapid elasticity, measured service, and so forth. A cloud computing model can also expose various service models, such as, for example, Software as a Service ("SaaS"), Platform as a Service ("PaaS"), and Infrastructure as a Service ("IaaS"). A cloud computing model can also be deployed using different deployment models such as private cloud, community cloud, public cloud, hybrid cloud, and so forth. In this description and in the claims, a "cloud computing environment" is an environment in which cloud computing is employed.

[0040] Embodiments of the invention allow library developers to more precisely and easily control which of their libraries' APIs can be called dynamically. Thus, their servicing and versioning burden can be more appropriately controlled. Further, application developers

can control which such APIs to further exclude from dynamic calling scenarios, to minimize the runtime support overhead.

[0041] Figures 1 illustrates an example computer architecture 100 that facilitates controlling access to an application programming interface (API). Referring to Figure 1, computer architecture 100 includes development environment 101 and runtime environment 102. development environment 101 and runtime environment 102 can be connected to one another over (or be part of) a network, such as, for example, a system bus, a Local Area Network ("LAN"), a Wide Area Network ("WAN"), and even the Internet. Accordingly, development environment 101 and runtime environment 102 as well as any other connected computer systems and their components, can create message related data and exchange message related data (e.g., Internet Protocol ("IP") datagrams and other higher layer protocols that utilize IP datagrams, such as, Transmission Control Protocol ("TCP"), Hypertext Transfer Protocol ("HTTP"), Simple Mail Transfer Protocol ("SMTP"), etc. or using other non-datagram protocols) over the network.

[0042] Development environment 101 can be a software application that provides facilities for software development, including but not limited to: a source code editor, build automation, a debugger, a version control system, a class browser, an object inspector, a class hierarchy diagram, etc. In some embodiments, development environment 101 includes or is included in an Integrated Development Environment (IDE). A library developer can use development environment 101 to apply an attribute to an API to change the visibility of the API to dynamic access requests.

[0043] Runtime environment 102 can be a software application that provides facilities for software execution. Runtime environment 102 can include a compiler (e.g., a just-in-time (JIT) compiler) and/or an interpreter for executing code developed in software development environment 101. In some embodiments, runtime environment 102 includes or is included in an Integrated Development Environment (IDE). Runtime environment 102 can include mechanisms (e.g., reflection) for dynamically requesting access to an API.

[0044] Development environment 101 and execution environment 102 can be integrated into the same environment or can be resident in separate environments.

[0045] Runtime environment 102 includes visibility calculation module 109. Visibility calculation module is configured to calculate the visibility into an API based on API type (e.g., internal, private, public, etc.) and applied attributes. Default visibility rules 108 can define a default visibility (e.g., permit dynamic access or remove dynamic access) for each API type. Applied attributes can be used to alter or override a default visibility. As such,

applied attributes give a library developer more precise control over how individual APIs can be accessed dynamically.

[0046] In some embodiments, default visibility rules 108 define that dynamic access is removed for APIs indicated as private or internal. A library developer can use development environment 101 to apply an attribute to the private or internal API. The applied attribute can indicate that dynamic access is to be permitted for the private or internal API. Upon receiving a dynamic call to the private or internal API, visibility calculation module 109 can determine that the applied attribute overrides the default visibility (of removed dynamic access) for the private or internal API. As such, the dynamic call is permitted access to the private or internal API.

[0047] Figure 2 illustrates a flow chart of an example method 200 for controlling access to an application programming interface (API). Method 200 will be described with respect to the components and data of computer architecture 100.

[0048] Method 200 includes accessing a set of application programming interfaces (APIs) combined in a library, the set of application programming interfaces (APIs) including one or more public application programming interfaces (APIs) and one or more non-public application programming interfaces (APIs) (201). For example, development environment 101 can access library 103. Library 103 includes one or more public APIs including public API 104. Library 103 includes one or more nob-public (e.g., private or internal) APIs including non-public API 106.

[0049] Method 200 includes identifying an application programming interface (API) from among the set of application programming interfaces (APIs) for which the default visibility provided to dynamic access requests is to be altered (202). For example, development environment 101 (possibly in response to author input) can identify that the default visibility for dynamic access requests (e.g., using reflection or other dynamic calling techniques) into non-public API 106 is to be altered.

[0050] Method 200 includes altering the default visibility into the application programming interface (API) to an altered visibility by applying an attribute to the application programming interface (API), the attribute indicating to the runtime environment at runtime that dynamic access requests are to be provided the altered visibility into the application programming interface (API) (203). For example, author 113 can enter visibility input 112 at development environment 101. In response to visibility input 112, development environment 101 can apply attribute 107 to non-public API 106. Attribute 107

can indicate to runtime environment 102 that the default visibility for non-public APIs defined in default visibility rules 108 is to be altered for non-public API 106.

[0051] Subsequently, library 103 can be compiled along with other source code into executable code 111. During execution of executable code 111, visibility calculation module 109 can consider both default visibility rules 108 and attribute 107 when calculating visibility into non-public API 106. In some embodiments, attribute 107 indicates the dynamic access to non-public API 106 is allowed even though default visibility rules 108 indicate that dynamic access to non-public APIs is to be prevented.

[0052] Figure 3 illustrates an example computer architecture 100 that facilitates reducing the default visibility into an accessible application programming interface (API). Referring to Figure 3, computer architecture 300 includes development environment 301 and runtime environment 302. development environment 301 and runtime environment 302 can be connected to one another over (or be part of) a network, such as, for example, a system bus, a Local Area Network ("LAN"), a Wide Area Network ("WAN"), and even the Internet. Accordingly, development environment 301 and runtime environment 302 as well as any other connected computer systems and their components, can create message related data and exchange message related data (e.g., Internet Protocol ("IP") datagrams and other higher layer protocols that utilize IP datagrams, such as, Transmission Control Protocol ("TCP"), Hypertext Transfer Protocol ("HTTP"), Simple Mail Transfer Protocol ("SMTP"), etc. or using other non-datagram protocols) over the network.

[0053] Development environment 301 can be a software application that provides facilities for software development, including but not limited to: a source code editor, build automation, a debugger, a version control system, a class browser, an object inspector, a class hierarchy diagram, etc. In some embodiments, development environment 301 includes or is included in an Integrated Development Environment (IDE). An application developer can use development environment 301 to apply an attribute to code referencing an accessible API (e.g., included in a third party library). Attributes applied to code referencing an accessible API can be used to reduce default visibility into the accessible API.

[0054] Runtime environment 302 can be a software application that provides facilities for software execution. Runtime environment can include a compiler (e.g., a just-in-time (JIT) compiler) and/or an interpreter for executing code developed in software development environment 301. In some embodiments, runtime environment 302 includes or is included in an Integrated Development Environment (IDE). Runtime environment 302 can include mechanisms (e.g., reflection) for dynamically requesting access to an API.

[0055] Development environment 301 and execution environment 302 can be integrated into the same environment or can be resident in separate environments.

[0056] Runtime environment 302 includes visibility calculation module 309. Visibility calculation module 309 is configured to calculate the visibility into an accessible API (e.g., public API or non-public API attributed to permit dynamic access) based on attributes applied to code referencing the accessible API. Default visibility rules 308 can permit dynamic access to accessible APIs. As such, applied attributes give an application developer a mechanism to exclude otherwise accessible APIs (e.g., included in a third party library) from dynamic access. Excluding an otherwise accessible API from dynamic access minimizes runtime support overhead (e.g., metadata generation) and thereby conserves resources.

[0057] Figure 4 illustrates a flow chart of an example method 400 for reducing the default visibility into an accessible application programming interface (API). Method 400 will be described with respect to the components and data of computer architecture 300.

[0058] Method 400 includes accessing application code for an application, the application code referring to one or more accessible application programming interfaces (APIs) combined in a library (401). For example, development environment 301 can access application code 314. Development environment 310 can also access library 30. Application code 314 can refer to one or more APIs, such as, for example, APIs 304 and 306, included in library 303.

[0059] Method 400 includes identifying an accessible application programming interface (API) referred to within the application code, the accessible application programming interface (API) selected from among the one or more accessible application programming interfaces (APIs) (402). For example, development environment 301 can identify API 306 referred to by API reference 316.

[0060] Method 400 includes determining that the accessible application programming interface (API) is not to be dynamically accessed at runtime (403). For example, development environment 301 (possibly in response to author input) can determine that dynamic access to API 306 is to be prevented.

[0061] Method 400 includes reducing the default visibility into the accessible application programming interface (API) to a reduced visibility by applying an attribute to the portion of the application code referring to the accessible application programming interface (API), the attribute indicating to the runtime environment at runtime that dynamic access requests are to be provided the reduced visibility into the accessible application programming

interface (API) (404). For example, author 313 can enter visibility input 312 at development environment 301. In response to visibility input 312, development environment 301 can apply attribute 317 to API reference 316 (a reference to API 306). Attribute 317 can indicate to runtime environment 302 that the default visibility for API 306 is to be reduced.

[0062] Subsequently, application code 314 and library 303 can be compiled along (possibly with other source code) into executable code 311. During execution of executable code 311, visibility calculation module 309 can consider both default visibility rules 308 and attribute 317 when calculating visibility into API 306. In some embodiments, attribute 317 indicates the dynamic access to API 306 is prevented even though default visibility rules 308 indicate that dynamic access to accessible APIs is to be allowed.

[0063] Limited visibility into an API can include not generating metadata for the API. For example, as depicted in computer architecture 300, metadata is not generated for API 306. On the other hand, metadata 319 can be generated for API 304 (another accessible API referenced form application code 314).

[0064] Figure 5 illustrates an example runtime environment 500 that facilitates providing a consumer with specified visibility into an application programming interface (API). Referring to Figure 5, runtime environment 500 includes visibility calculation module 501 and consumer 531. Visibility calculation module 501 and consumer 531 can be connected to one another over (or be part of) a network, such as, for example, a system bus, a Local Area Network ("LAN"), a Wide Area Network ("WAN"), and even the Internet. Accordingly, visibility calculation module 501 and consumer 531as well as any other connected computer systems and their components, can create message related data and exchange message related data (e.g., Internet Protocol ("IP") datagrams and other higher layer protocols that utilize IP datagrams, such as, Transmission Control Protocol ("TCP"), Hypertext Transfer Protocol ("HTTP"), Simple Mail Transfer Protocol ("SMTP"), etc. or using other non-datagram protocols) over the network.

[0065] Runtime environment 500 can be a software application that provides facilities for software execution. Runtime environment can include a compiler (e.g., a just-in-time (JIT) compiler) and/or an interpreter for executing code developed in software development environment 500. In some embodiments, runtime environment 500 includes or is included in an Integrated Development Environment (IDE). Runtime environment 500 can include mechanisms (e.g., reflection) for dynamically requesting access to an API.

[0066] Runtime environment 500 includes visibility calculation module 501. Visibility calculation module 501 is configured to calculate the visibility into an each of a plurality of

APIs grouped together in a library. Visibility can be calculated for an API based on an API type (e.g., internal, private, public), attributes applied (e.g., a library author) to the API, and attributes applied to application that references the API. Applied attributes can be used to alter, override, reduce, etc. a default visibility.

[0067] In some embodiments, default visibility rules 502 define that dynamic access is removed for non-public (e.g., private or internal) APIs and that dynamic access is permitted for public APIs. However, the author of a non-public API (e.g., a library author) can apply an attribute to the non-public API to override the default visibility and permit dynamic access to the non-public API. Likewise, the author of a public API (e.g., a library author) can apply an attribute to the public API to override the default visibility and deny dynamic access to the public API. Other default visibility rules are also possible.

[0068] For any dynamically accessible APIs (whether dynamically accessible by default or dynamically accessible by an applied attribute), a third party author (e.g., an application author) can apply an attribute to code referencing the dynamically accessible API to remove dynamic access from the API. Thus, an application author can minimize runtime support overhead (e.g., metadata generation) and thereby conserves resources.

[0069] Figure 6 illustrates a flow chart of an example method 600 for providing a consumer with specified visibility into an application programming interface (API). Method 600 will be described with respect to the components and data of runtime environment 500.

[0070] Method 600 incudes executing executable code in the runtime environment, the executable code derived from application code (601). For example, executable code can be executed in runtime environment 500. Executable code 503 can be derived from application code that includes references to APIs contained in a library. For example, API references 511 and 517 can reference APIs contained in a library. The APIs contained in the library can include APIs 513 and 518.

[0071] Method 600 includes during execution of the executable code, method 600 includes receiving a dynamic call from a consumer to execute a portion of the executable code, the portion of the executable code derived from a portion of the application code that refers to an application program interface (API) within a library (602). For example, dynamic call 521 can be received from consumer 531. Dynamic call 521 can be a call to execute a portion of executable code 503.

[0072] In one embodiment, dynamic call 521 is a call to execute executable code that includes API reference 511. API reference 511 can reference either API 513 or API 518. In

another embodiment, dynamic call 521 is a call to execute executable code that includes API reference 517. API reference 517 can reference either API 513 or API 518.

[0073] Method 600 incudes accessing a runtime default visibility into the application program interface (API) based on the type of the application program interface (API) (603). As indicated by type 514, API 513 is a non-public API. Thus, when an API reference references API 513, a runtime default visibility for (e.g., preventing dynamic access to) non-public APIs can be accessed from default visibility rules 502 (for API 513). As indicated by type 519, API 518 is a public API. Thus, when an API reference references API 518, a runtime default visibility for (e.g., allowing dynamic access to) public APIs can be accessed from default visibility rules 502 (for API 518).

[0074] Method 600 incudes accessing any attributes applied to the application program interface (API), attributes applied to the application program interface (API) indicative of a desire by the author of application program interface (API) to alter the runtime default visibility into the application program interface (API) (604). For example, when an API reference references API 513, attribute 516 can be accessed. Attribute 516 can indicate a desire by the library author to alter the runtime default visibility (as defined in default visibility rules 502) into API 513. For example, by default, dynamic access to non-public APIs can be prevented. However, attribute 513 can indicate that dynamic access to API 513 is to be permitted.

[0075] Method 600 incudes accessing any attributes applied to the portion of the application code that refers to the application program interface (API), attributes applied to the portion of the application code indicative of a desire by the author of the application code to provide visibility into the application program interface (API) to a lesser extent than indicated by the runtime default visibility as altered by any attributes applied to the application program interface (API) (605). For example, when dynamic call 521 is a call to execute executable code that includes API reference 511, attribute 512 can be accessed. Attribute 512 can indicate a desire by an application author to reduce visibility into a reference API (e.g., API 513 or API 518). For example, attribute 512 can indicate that dynamic access to the references API (e.g., API 513 or API 518) is to be prevented. As such, even if the library author otherwise permits dynamic access to an API, the application developer can apply attribute 512 to prevent dynamic access to the API.

[0076] Method 600 incudes determining a calculated visibility into the application program interface (API) based on one or more of: the runtime default visibility into the application program interface (API), any attributes applied to the an application program

interface (API), and any attributes applied to the portion of the application code that refers to the application program interface (API) (606). Thus, visibility calculation module 501 can determine a calculated visibility 522 into an API based on a default visibility for the API's type (as defined in default visibility rules 502), any attributes applied to the API, and any attributes applied to an API reference that references the API.

[0077] Method 600 incudes providing the dynamic call with visibility into the application program interface (API) in accordance with the calculated visibility (607). For example, consumer 531 can be provided with visibility 522 into a dynamically called API in executable code 503. Results 524 of dynamic call 521 can also be returned to consumer 531. Results 524 may indicate that the dynamic access to the dynamically called API is not permitted.

[0078] Within executable code 503 various different combinations of referring code and APIs are possible. In one embodiment, dynamical call 521 uses API reference 511 to call API 513. In this embodiment, visibility 522 is determined from a default visibility for API type 514 (e.g., as defined in default visibility rules 502), attribute 512, and attribute 516. In another embodiment, dynamical call 521 uses API reference 511 to call API 518. In this embodiment, visibility 522 is determined from a default visibility for API type 519 (e.g., as defined in default visibility rules 502) and attribute 512.

[0079] In a further embodiment, dynamical call 521 uses API reference 517 to call API 513. In this further embodiment, visibility 522 is determined from a default visibility for API type 514 (e.g., as defined in default visibility rules 502) and attribute 516. In an additional embodiment, dynamical call 521 uses API reference 517 to call API 518. In this further embodiment, visibility 522 is determined from a default visibility for API type 519 (e.g., as defined in default visibility rules 502).

[0080] The present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

CLAIMS

1. At a computer system, the computer system including a development environment for developing executable code that includes application programming interfaces (APIs), the executable code for running in a runtime environment that can dynamically requests access to the application program interfaces (APIs), the runtime environment providing dynamic access requests with a default visibility into application program interfaces (APIs) based on application program interface (API) type, a method for controlling runtime access to the application programming interfaces (APIs), the method comprising:

accessing a set of application programming interfaces (APIs) combined in a library, the set of application programming interfaces (APIs) including one or more public application programming interfaces (APIs) and one or more non-public application programming interfaces (APIs);

identifying an application programming interface (API) from among the set of application programming interfaces (APIs) for which the default visibility provided to dynamic access requests is to be altered; and

altering the default visibility into the application programming interface (API) to an altered visibility by applying an attribute to the application programming interface (API), the attribute indicating to the runtime environment at runtime that dynamic access requests are to be provided the altered visibility into the application programming interface (API).

2. The method of claim 1, wherein identifying an application programming interface (API) comprises exposing the application programming interface (API) to the author of the application programming interface (API); and

wherein altering the default visibility of the application programming interface (API) comprises the author of the application programming interface (API) altering the default visibility of the application programming interface.

3. The method of claim 1, wherein identifying an application programming interface (API) comprises identifying a non-public application programming interface (API), the default visibility for the non-public application programming interface (API) preventing dynamic access to the non-public application programming interface (API).

4. The method of claim 3, wherein altering the default visibility of the application programming interface (API) comprises applying at attribute to the non-public application programming interface (API), the attribute altering the default visibility for the non-public application programming interface (API) to allow dynamic access to the non-public application programming interface (API).

5. The method of claim 1, wherein identifying an application programming interface (API) comprises identifying a public application programming interface (API), the default visibility for the public application programming interface (API) allowing dynamic access to the public application programming interface (API).

6. The method of claim 5, wherein altering the default visibility of the application programming interface (API) comprises applying at attribute to the public application programming interface (API), the attribute altering the default visibility for the public application programming interface (API) to prevent dynamic access to the public application programming interface (API).

7. At a computer system, the computer system including a development environment for developing executable code that includes application programming interfaces (APIs), the executable code for running in a runtime environment that can dynamically request access to application program interfaces (APIs), the runtime environment having default visibility into accessible application program interfaces (APIs), a method for reducing the default visibility into an accessible application programming interfaces (API), the method comprising:

accessing application code for an application, the application code referring to one or more accessible application programming interfaces (APIs) combined in a library;

identifying an accessible application programming interface (API) referred to within the application code, the accessible application programming interface (API) selected from among the one or more accessible application programming interfaces (APIs);

determining that the accessible application programming interface (API) is not to be dynamically accessed at runtime;

reducing the default visibility into the accessible application programming interface (API) to a reduced visibility by applying an attribute to the portion of the application code referring to the accessible application programming interface (API), the attribute indicating to the runtime environment at runtime that dynamic access requests are to be provided the reduced visibility into the accessible application programming interface (API).

8. The method of claim 7, wherein applying an attribute to a portion of the application code referring to the accessible application programming interface (API) comprises the author of the application code applying an attribute to a portion of the application code referring to the accessible application programming interface (API).

9. The method of claim 7, wherein applying an attribute to a portion of the application code referring to the accessible application programming interface (API) comprises applying an attribute to one of: a portion of the application code referring to a public API or a portion of the application code referring to a non-public API.

10. At a computer system, the computer system including a runtime environment that can request dynamic access to application program interfaces (APIs), the runtime environment providing dynamic access requests with a default visibility into application program interfaces (APIs) based on application program interface (API) type, a method providing a consumer with specified visibility into an application programming interface (API), the method comprising:

executing executable code in the runtime environment, the executable code derived from application code;

during execution of the executable code:

receiving a dynamic call from a consumer to execute a portion of the executable code, the portion of the executable code derived from a portion of the application code that refers to an application program interface (API) within a library;

accessing a runtime default visibility into the application program interface (API) based on the type of the application program interface (API);

accessing any attributes applied to the application program interface (API), attributes applied to the application program interface (API)

indicative of a desire by the author of application program interface (API) to alter the runtime default visibility into the application program interface (API);

accessing any attributes applied to the portion of the application code that refers to the application program interface (API), attributes applied to the portion of the application code indicative of a desire by the author of the application code to provide visibility into the application program interface (API) to a lesser extent than indicated by the runtime default visibility as altered by any attributes applied to the application program interface (API);

determining a calculated visibility into the application program interface (API) based on one or more of: the runtime default visibility into the application program interface (API), any attributes applied to the an application program interface (API), and any attributes applied to the portion of the application code that refers to the application program interface (API); and

providing the dynamic call with visibility into the application program interface (API) in accordance with the calculated visibility.

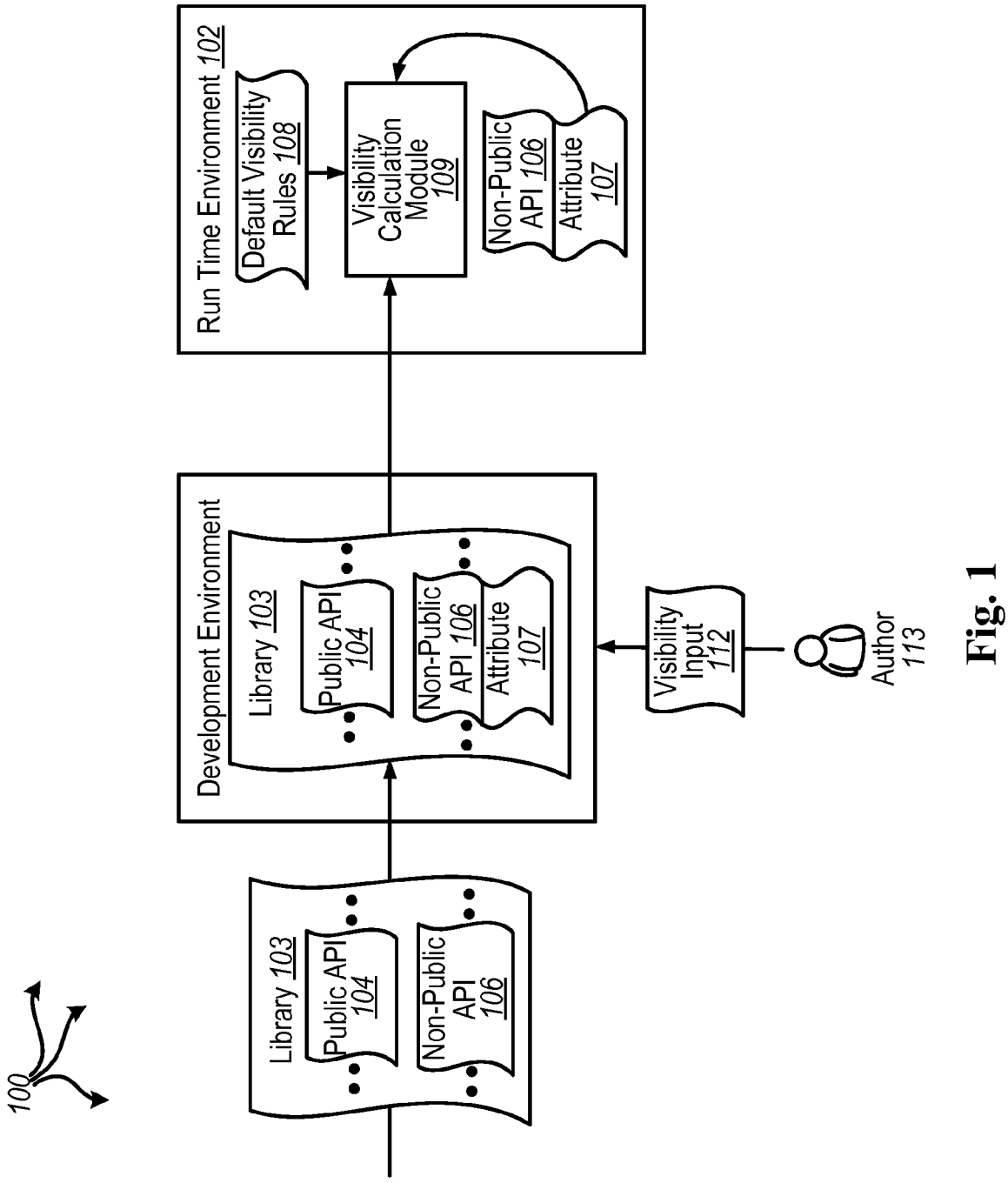

100
Library 103
Public API 104
Non-Public API 106
Development Environment
Library 103
Public API 104
Non-Public API 106
Attribute 107
Visibility Input 112
Author 113
Run Time Environment 102
Default Visibility Rules 108
Visibility Calculation Module 109
Non-Public API 106
Attribute 107

Fig. 1

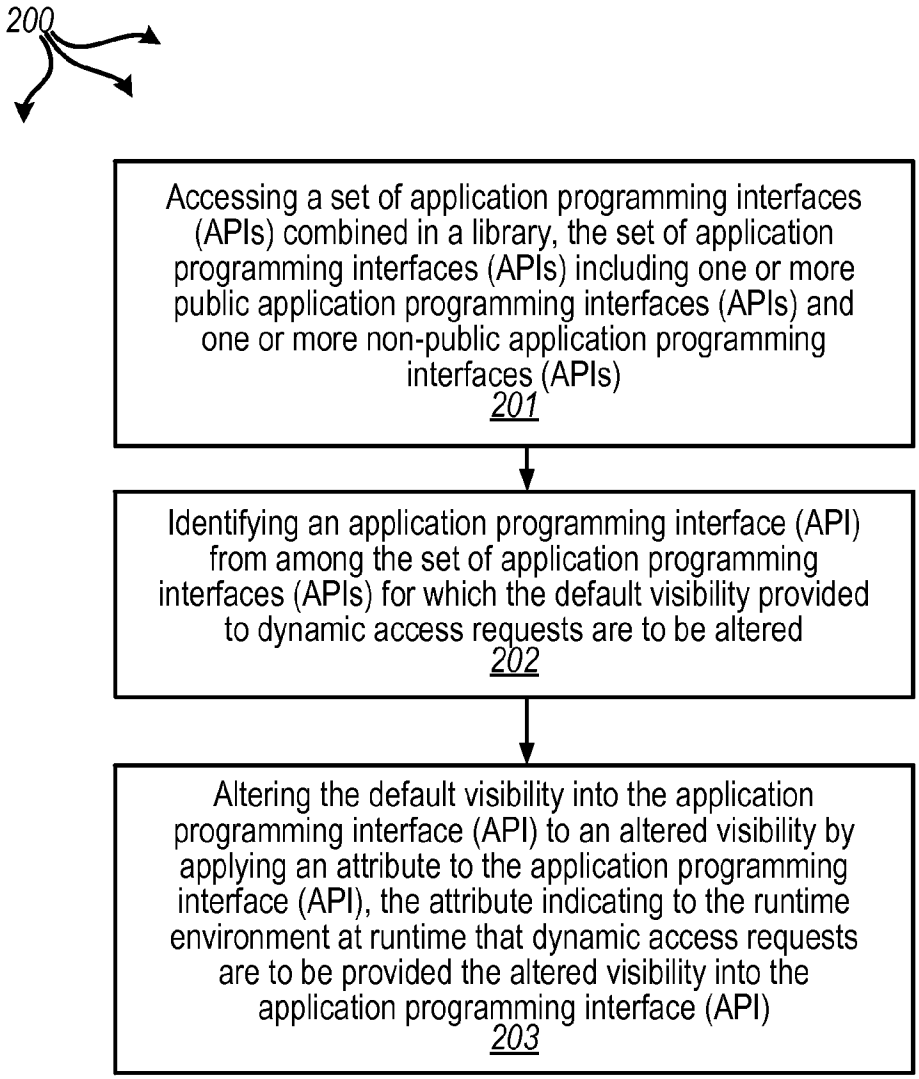

200
Accessing a set of application programming interfaces (APIs) combined in a library, the set of application programming interfaces (APIs) including one or more public application programming interfaces (APIs) and one or more non-public application programming interfaces (APIs)
201
Identifying an application programming interface (API) from among the set of application programming interfaces (APIs) for which the default visibility provided to dynamic access requests are to be altered
202
Altering the default visibility into the application programming interface (API) to an altered visibility by applying an attribute to the application programming interface (API), the attribute indicating to the runtime environment at runtime that dynamic access requests are to be provided the altered visibility into the application programming interface (API)
203

Fig. 2

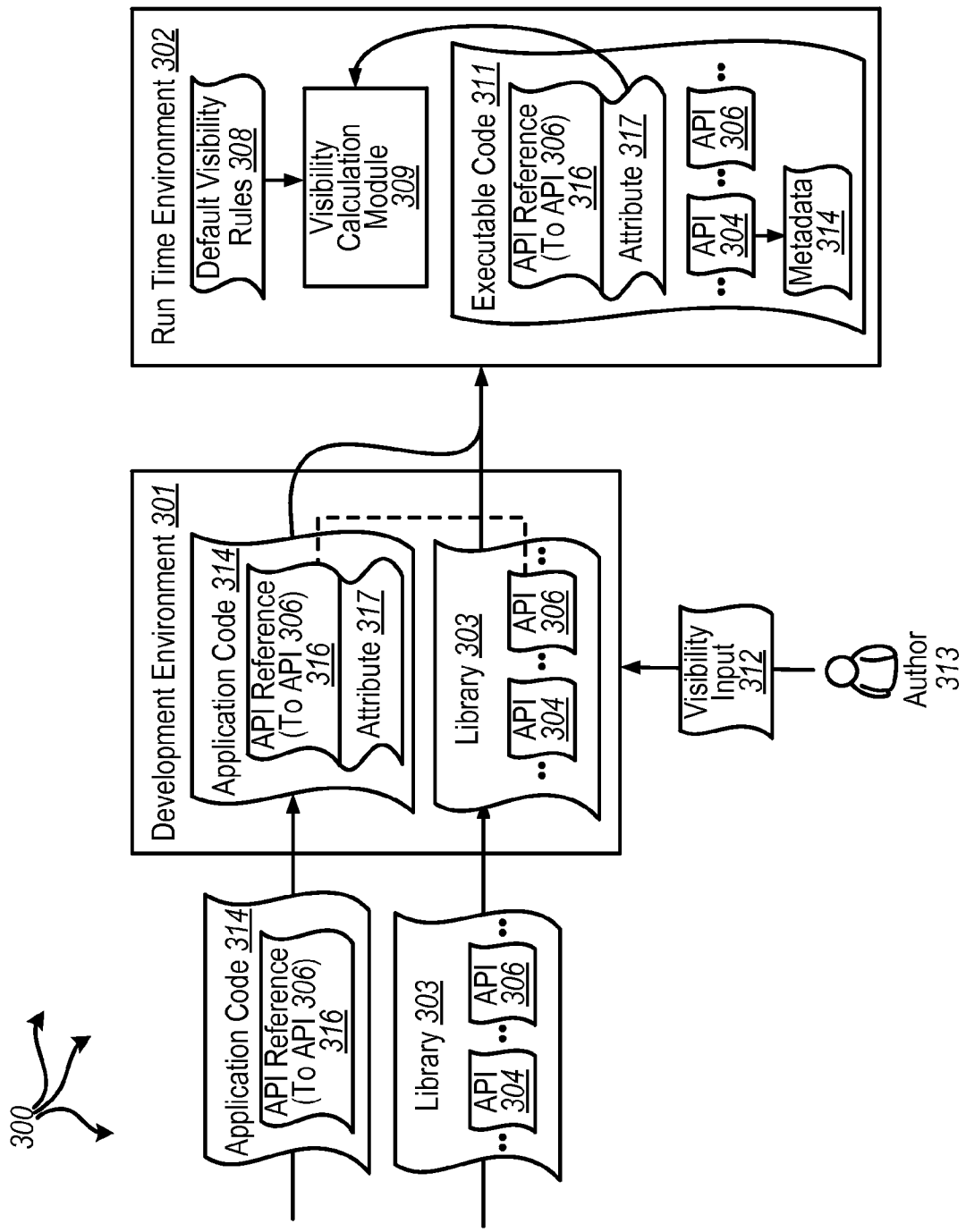

300
Application Code 314
API Reference (To API 306) 316
Library 303
API 304
API 306
Development Environment 301
Application Code 314
API Reference (To API 306) 316
Attribute 317
Library 303
API 304
API 306
Visibility Input 312
Author 313
Run Time Environment 302
Default Visibility Rules 308
Visibility Calculation Module 309
Executable Code 311
API Reference (To API 306) 316
Attribute 317
API 304
API 306
Metadata 314

Fig. 3

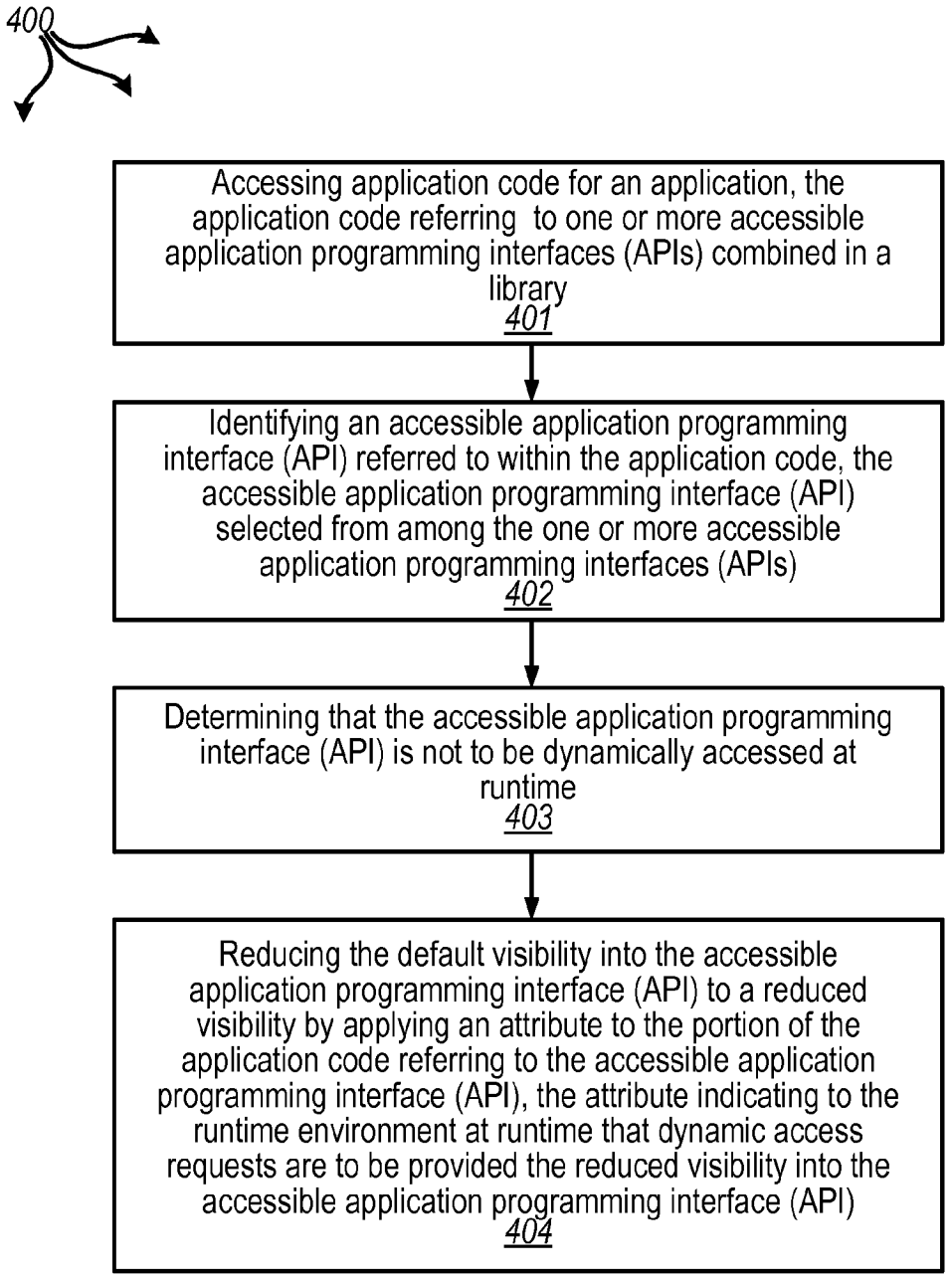

400
Accessing application code for an application, the application code referring to one or more accessible application programming interfaces (APIs) combined in a library
401
Identifying an accessible application programming interface (API) referred to within the application code, the accessible application programming interface (API) selected from among the one or more accessible application programming interfaces (APIs)
402
Determining that the accessible application programming interface (API) is not to be dynamically accessed at runtime
403
Reducing the default visibility into the accessible application programming interface (API) to a reduced visibility by applying an attribute to the portion of the application code referring to the accessible application programming interface (API), the attribute indicating to the runtime environment at runtime that dynamic access requests are to be provided the reduced visibility into the accessible application programming interface (API)
404

Fig. 4

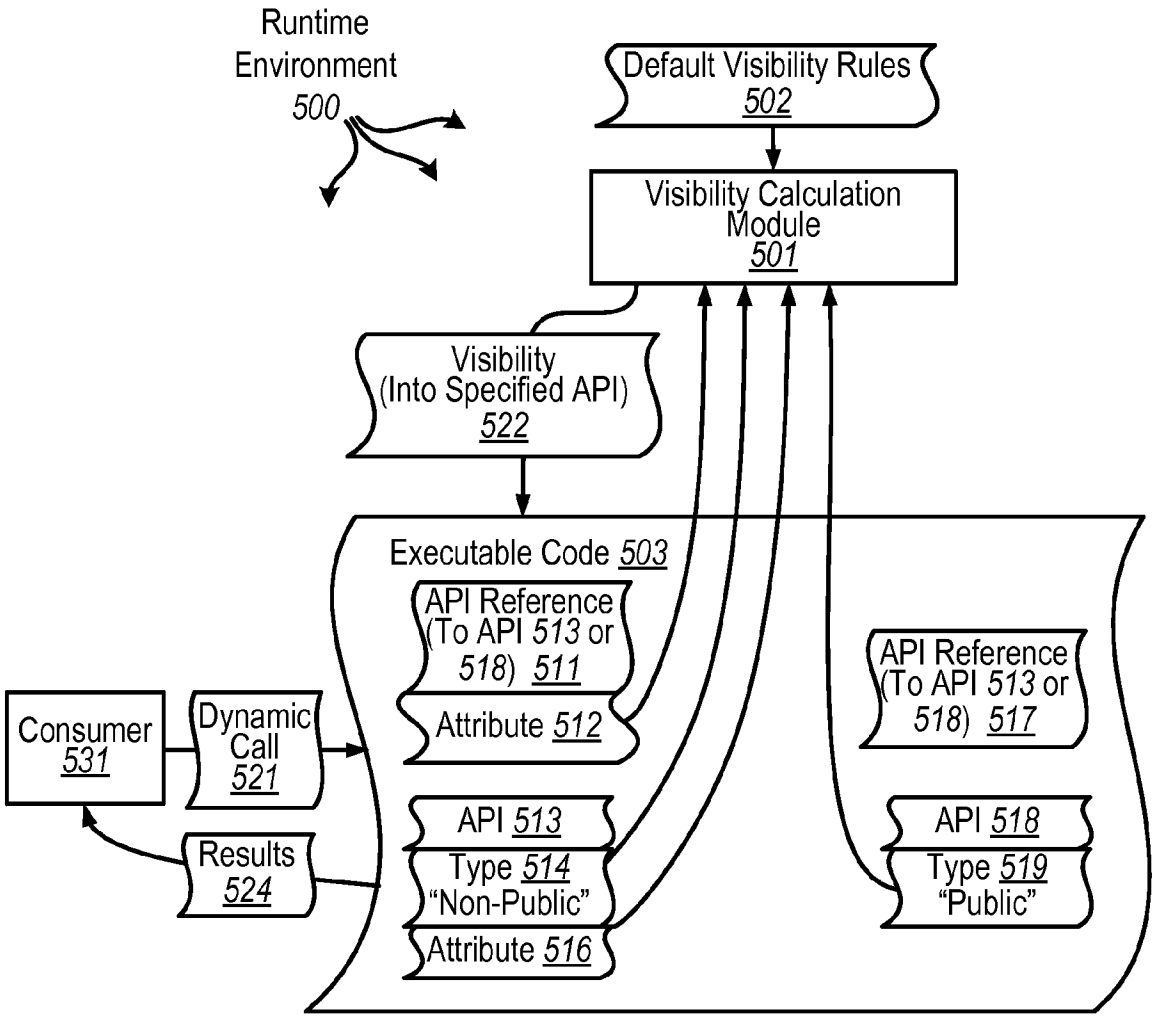

Runtime Environment 500
Default Visibility Rules 502
Visibility Calculation Module 501
Visibility (Into Specified API) 522
Executable Code 503
API Reference (To API 513 or 518) 511
Attribute 512
API 513
Type 514 "Non-Public"
Attribute 516
API Reference (To API 513 or 518) 517
API 518
Type 519 "Public"
Consumer 531
Dynamic Call 521
Results 524

Fig. 5

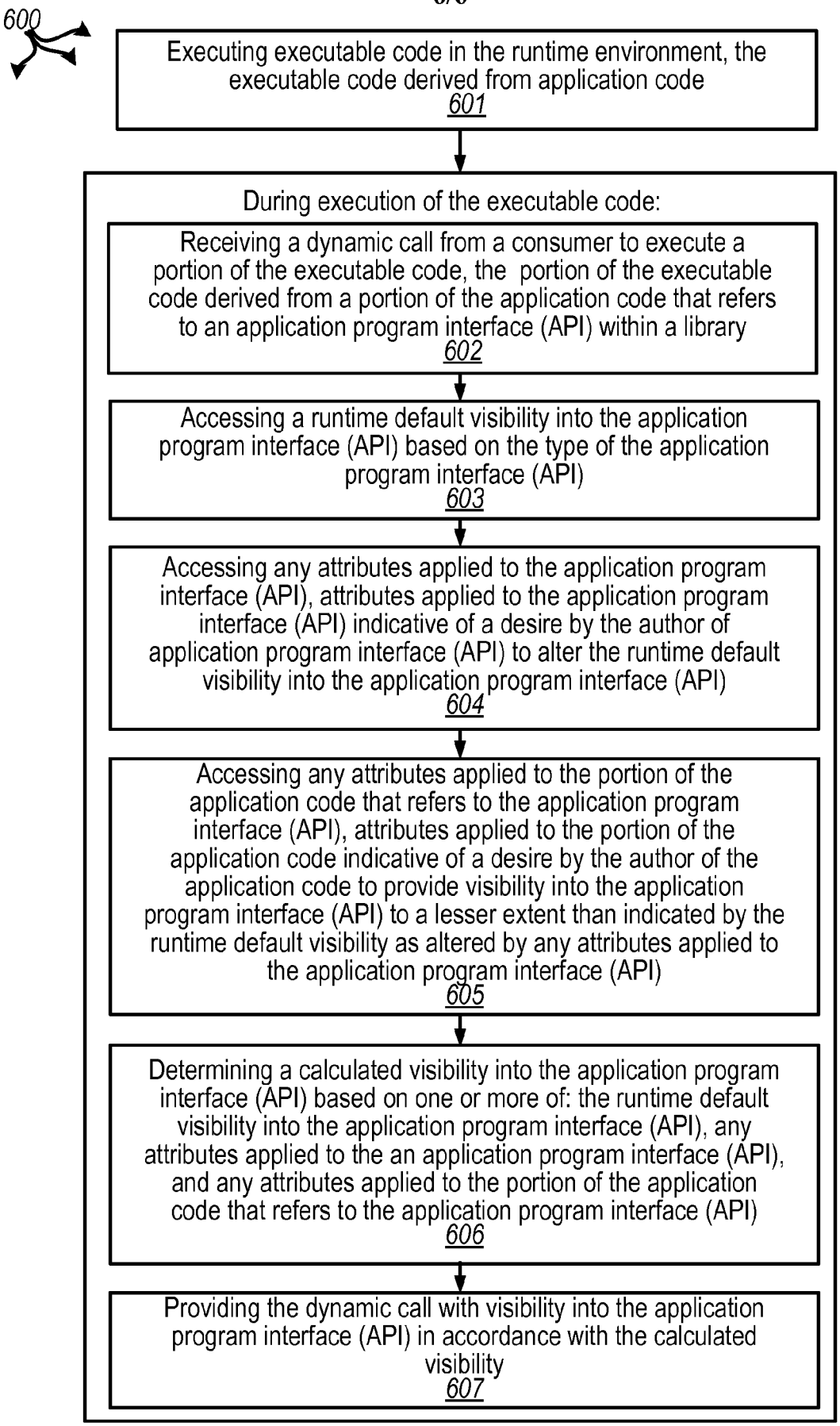

600
Executing executable code in the runtime environment, the executable code derived from application code
601
During execution of the executable code:
Receiving a dynamic call from a consumer to execute a portion of the executable code, the portion of the executable code derived from a portion of the application code that refers to an application program interface (API) within a library
602
Accessing a runtime default visibility into the application program interface (API) based on the type of the application program interface (API)
603
Accessing any attributes applied to the application program interface (API), attributes applied to the application program interface (API) indicative of a desire by the author of application program interface (API) to alter the runtime default visibility into the application program interface (API)
604
Accessing any attributes applied to the portion of the application code that refers to the application program interface (API), attributes applied to the portion of the application code indicative of a desire by the author of the application code to provide visibility into the application program interface (API) to a lesser extent than indicated by the runtime default visibility as altered by any attributes applied to the application program interface (API)
605
Determining a calculated visibility into the application program interface (API) based on one or more of: the runtime default visibility into the application program interface (API), any attributes applied to the an application program interface (API), and any attributes applied to the portion of the application code that refers to the application program interface (API)
606
Providing the dynamic call with visibility into the application program interface (API) in accordance with the calculated visibility
607

Fig. 6

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2014/034739

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/44
ADD.

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

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

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

EPO-Internal, 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	GOSLING J ET AL: "The Java Language Specification - Second Edition", THE JAVA LANGUAGE SPECIFICATION - SECOND EDITION, ADDISON-WESLEY, US, June 2000 (2000-06), pages 1-532, XP007921265, ISBN: 0-201-31008-2 page 1 page 81 page 229 page 251 section 13.4.3 at page 258 ----- -/--	1-6

[X] Further documents are listed in the continuation of Box C.

[ ] See patent family annex.

* Special categories of cited documents :

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

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

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

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

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

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

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

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

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
14 July 2014	22/07/2014

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Del Castillo, G

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2014/034739

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	M. Biczó ET AL: "Runtime access control in C# 3.0 using extension methods", Annales Univ. Sci. Budapest., Sect. Comp., 2009, pages 41-59, XP055034082, Retrieved from the Internet: URL:http://ac.inf.elte.hu/Vol_030_2009/041.pdf [retrieved on 2012-07-27] abstract sections 4 and 5	1-6

X	Stuart Dabbs Halloway: "Component Development for the Java Platform (Chapter 3: Type Information and Reflection)", , 14 December 2001 (2001-12-14), pages 57-103, XP055127743, ISBN: 0201753065 Retrieved from the Internet: URL:http://www.pearsonhighered.com/samplechapter/0201753065.pdf [retrieved on 2014-07-09] sections 3.3.3 and 3.3.3.1 at pages 76-78, especially the first paragraph of section 3.3.3	1-6
L	& "Extract from publisher's Web page www.pearsonhighered.com, containing bibliographic information on the cited book", , XP055127751, Retrieved from the Internet: URL:http://www.pearsonhighered.com/educator/product/Component-Development-for-the-Java-Platform/9780201753066.page [retrieved on 2014-07-09] (provides evidence of the publication date of the cited book)	

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2014/034739

Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☒ Claims Nos.: 7-10
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:
see FURTHER INFORMATION sheet PCT/ISA/210

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box No. III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fees, this Authority did not invite payment of additional fees.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest and, where applicable, the payment of a protest fee.

☐ The additional search fees were accompanied by the applicant's protest but the applicable protest fee was not paid within the time limit specified in the invitation.

☐ No protest accompanied the payment of additional search fees.

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

Continuation of Box II.2

Claims Nos.: 7-10

Claims 7-10 do not comply with the provisions of clarity of Article 6 PCT to such an extent that no meaningful search of the claims can be carried out for these claims, as the subject-matter for which protection is sought cannot be established with sufficient certainty (even taking into account the content of the description and drawings).

In particular,
claim 7 is not clear for the following reasons:
a. the meaning of the
step of " determining that the accessible application programming interface (API) is not to be dynamically accessed at runtime" is not clear: in particular, it is not clear who or what decides that an API is "not to be dynamically accessed" and according to which criteria;
b. the
meaning of the term "portion" is not clear: in particular, it is not clear what is the extent of the "portion of the application code" to which the attribute is applied (note that such a portion could be anything, from an individual instruction referring to the API to the whole program containing that instruction);
c. more generally, the
content of the claimed method as a whole is quite confusing for a skilled person, in particular because the visibility of an API is changed where the API is referred to, which seems to defy the purpose of defining such a visibility (so that the skilled person cannot make sense of the claimed method).

Claim 10 is not clear for the same reason as indicated at
item b above with respect to claim 7 and, furthermore, the meaning of the step of "determining a calculated visibility" is not clear (in particular, it is not clear who or what determines the "calculated visibility", how it is determined, or even what the possible values of such a "calculated visibility" can be).

Dependent claims 8 and 9 are
not clear as a result of the lack of clarity of claim 7, on which they depend.

The applicant's attention is drawn to the fact that claims relating to inventions in respect of which no international search report has been established need not be the subject of an international preliminary examination (Rule 66.1(e) PCT). The applicant is advised that the EPO policy when acting as an International Preliminary Examining Authority is normally not to carry out a preliminary examination on matter which has not been searched. This is the case irrespective of whether or not the claims are amended following receipt of the search report or during any Chapter II procedure. If the application proceeds into the regional phase before the EPO, the applicant is reminded that a search may be carried out during examination before the EPO (see EPO Guidelines C-IV, 7.2), should the problems which led to the Article 17(2) declaration be

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

overcome.