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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 August 2007 (30.08.2007)

PCT

(10) International Publication Number
WO 2007/097881 A1

(51) International Patent Classification:
G06F 9/45 (2006.01) *G06F 17/00* (2006.01)

(21) International Application Number:
PCT/US2007/002320

(22) International Filing Date: 25 January 2007 (25.01.2007)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
11/276,362 27 February 2006 (27.02.2006) US

(71) Applicant *(for all designated States except US)*: **MICROSOFT CORPORATION** [US/US]; One Microsoft Way, Redmond, Washington 98052-6399 (US).

(72) Inventors: **MILLER, James S.**; One Microsoft Way, Redmond, wa 98052-6399 (US). **QUINN, Thomas E.**; 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, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, 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, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

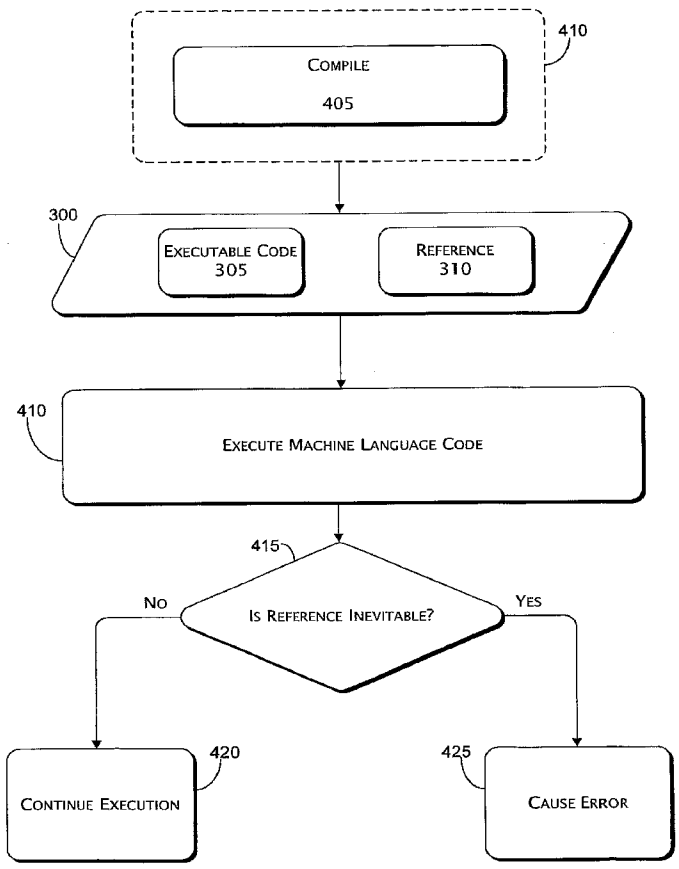
Declarations under Rule 4.17:

— *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:** ADAPTIVE COMPILED CODE



(57) **Abstract:** In a managed execution environment, an error may be deferred until execution of the application, program, function, or other assemblage of code reaches a point at which calling the reference to a module associated with a missing type or type member becomes inevitable.

WO 2007/097881 A1

Published:
— *with international search report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

ADAPTIVE COMPILED CODE

BACKGROUND

[0001] Applications, programs, functions, and other assemblages of programmable and executable code are typically written for third party (*i.e.,* "customer") usage. Therefore, effective code is written in such a manner that third party usage scenarios are enabled and meet third party expectations. However, such expectations may be difficult to meet when an application, program, function, or other assemblage of code that has been designed to run on an existing platform attempts to run on an older, or otherwise different, version of the platform.

SUMMARY

[0002] When an application, program, function, or other assemblage of code attempts to run on an older, or otherwise different, version of the platform on which it was generated, an error associated with a missing type or type member may be deferred until execution of the application, program, function, or other assemblage of code reaches a point at which calling a reference to the missing type or type member becomes inevitable.

DESCRIPTION OF THE DRAWINGS

[0003] The present description references the following figures.

[0004] FIG. 1 shows devices communicating over a network, with the devices implementing example technologies related to adaptive compiled code.

[0005] FIG. 2 shows an example of an execution environment for implementing example technologies related to adaptive compiled code.

[0006] FIG. 3 shows an example data structure for adaptive compiled code.

[0007] FIG. 4 shows an example data flow for implementing example technologies related to adaptive compiled code.

DETAILED DESCRIPTION

[0008] Tools, systems, and methodologies for compiling adaptive code and executing the adaptive compiled code are described herein. Further, the description pertaining to at least one of compiling and executing one or more applications, programs, functions, or other assemblage of code having at least one adaptive compiled module, may relate to tools, systems, processes, instructions, techniques, and routines that may be utilized to defer an error associated therewith. That is, the aforementioned tools, systems, processes, instructions, techniques, and routines may be utilized to defer one or more errors associated with a reference in the compiled code until a call to the reference is, at least, statistically inevitable. The aforementioned tools, systems, and processes may be implemented in one or more devices, or nodes, in a network environment.

[0009] "Module," as described herein, may refer to separate entities such as methods, classes, DLLs (dynamic link libraries), frameworks, *etc.*, that may utilize common physical and/or logical resources.

[0010] "Assemblage," as described herein, may refer to a unit of deployment for code.

[0011] FIG. 1 shows example network environment 100 in which example technologies may be implemented for compiling adaptive code and executing one or more applications, programs, functions, other code having at least one adaptive compiled module. However, such example technologies are not limited to network environments. Such technologies may include, but are not limited to, tools, methodologies (*e.g.*, techniques), and systems, associated with adaptive compiled code 120, as described herein. In FIG. 1, client device 105, server device 110, and "other" device 115 may be

communicatively coupled to one another via network 125; and, further, at least one of client device 105, server device 110, and "other" device 115 may be capable of implementing the aforementioned technologies.

[0012] Client device 105 may represent at least one of a variety of known computing devices, including a desktop personal computer (PC), workstation, mainframe computer, Internet appliance, set-top box, or gaming console, that is able to implement example technologies for at least one of producing and utilizing adaptive compiled code 120. Client device 105 may further represent at least one device that is capable of being associated with network 125 by a wired and/or wireless link, including a mobile (*i.e.*, cellular) telephone, personal digital assistant (PDA), laptop computer, *etc.* Further still, client device 105 may represent the client devices described above in various quantities and/or combinations thereof. "Other" device 115 may also be embodied by any of the above examples of client device 105.

[0013] Server device 110 may represent any device that is capable of providing any of a variety of data and/or functionality to client device 105 or "other" device 115 in accordance with at least one implementation for at least one of compiling and utilizing adaptive compiled code 120. The data may be publicly available or alternatively restricted, *e.g.*, restricted to only certain users or only if an appropriate subscription or licensing fee is paid. Server device 110 may be at least one of a network server, an application server, a blade server, or any combination thereof. Typically, server device 110 may represent any device that may be a content source, and client device 105 may represent any device that may receive such content either via network 125 or in an off-line manner. However, according to the example implementations described herein, client device 105 and server device 110 may interchangeably be a sending node or a receiving node in network environment 100. "Other" device 115 may also be embodied by any of the above examples of server device 110.

[0014] "Other" device 115 may represent any further device that is capable of compiling and/or utilizing adaptive compiled code 120 according to one or more of the example technologies described herein. That is, "other" device 115 may represent a device that is capable of compiling code or receiving compiled code for which one or more errors associated with a reference in the compiled code may be deferred until a call to the reference is, at least, statistically inevitable. Thus, "other" device 115 may be a computing or processing device having at least one of an operating system, an interpreter, converter, compiler, or runtime execution environment implemented thereon. These examples are not intended to be limiting in any way, and therefore should not be construed in that manner.

[0015] Network 125 may represent any of a variety of conventional network topologies and types, which may include wired and/or wireless networks. Network 125 may further utilize any of a variety of conventional network protocols, including public and/or proprietary protocols. Network 125 may include, for example, the Internet as well at least portions of one or more local area networks (also referred to, individually, as a "LAN"), such as an 802.11 system or, on a larger scale, a wide area network (*i.e.*, WAN"); or a personal area network (*i.e.*, PAN), such as Bluetooth.

[0016] Computer architecture in at least one of devices 105, 110, and 115 has typically defined computing platforms in terms of hardware and software. Software for computing devices has been categorized into groups, based on function, which may include: a hardware abstraction layer (alternatively referred to as a "HAL"), an operating system (alternatively referred to as "OS"), and applications.

[0017] A runtime execution environment may reside between an OS and an application, program, function, or other assemblage of code. The runtime execution environment may serve as a space in which the application, program, function, or other assemblage of code may execute specific tasks on any one or more of processing devices

105, 110, and 115. More particularly, a runtime execution environment may enhance the reliability of the execution of an application, program, function, or other assemblage of code on a growing range of processing devices 105, 110, and 105, including servers, desktop computers, laptop computers, and mobile processing/communication devices by providing a layer of abstraction and services for an application running on such devices, and by further providing the application, program, function, or other assemblage of code with capabilities including memory management and configuration thereof.

[0018] A runtime execution environment may serve as at least one of a programming and an execution platform. As a programming platform, a runtime execution environment may compile one or more targeted applications, programs, functions, or other assemblages of code, which may be written in one of multiple computing languages, into an intermediate language (hereafter "IL") or bytecode. IL is typically independent of the platform, and the central processing unit (hereafter "CPU") executes IL. In fact, IL is a higher level language than many CPU machine languages.

[0019] As an execution platform, a runtime execution environment may interpret compiled IL into native machine instructions. A runtime execution environment may utilize either an interpreter or a compiler (*e.g.*, "just-in-time," alternatively "JIT," compiler) to execute such instructions. Regardless, the native machine instructions may then be directly executed by the CPU. Since IL is CPU-independent, IL may execute on any CPU platform as long as the OS running on that CPU platform hosts an appropriate runtime execution environment.

[0020] Alternatively, at least portions of applications, programs, functions, or other assemblages of code may be precompiled and loaded as one or more native image files in the runtime execution environment, thus circumventing CPU consumption required for compilation. Effectively, the precompiled portions are software modules that are distributed in an IL format (*e.g.*, assemblies, methods, or types) rather than in a

native platform execution format. A source of such precompiled IL may be disposed in either of a non-managed execution environment or a separate implementation of a runtime execution environment on a same or separate one of devices 105, 110, and 115. The source may deploy the precompiled IL during or before install time for the application, program, method, function, or other assemblage of code to which the precompiled IL corresponds.

[0021] Regardless, examples of runtime environments, in which technologies for compiling and/or utilizing adaptive compiled code 120 may be implemented, include: Visual Basic runtime environment; Java® Virtual Machine runtime environment that is used to run, *e.g.*, Java® routines; or Common Language Runtime (CLR) to compile, *e.g.*, Microsoft .NET™ applications into machine language before executing a calling routine. However, this listing of runtime environments provides examples only. The example technologies described herein are not limited to just these managed execution environments. More particularly, the example implementations are not just limited to managed execution environments, for one or more examples may be implemented within testing environments and/or unmanaged execution environments.

[0022] An application, program, function, or other assemblage of code compiled into IL may be referred to as "managed code," and that is why a runtime execution environment may be alternatively referred to as a "managed execution environment." It is noted that code that does not utilize a runtime execution environment to execute may be referred to as a native code application.

[0023] FIG. 2 shows an example of runtime execution environment 200 in which example technologies may be implemented for compiling adaptive code 120 (see FIG. 1) and executing one or more applications, programs, functions, other code having at least one adaptive compiled module.

[0024] In the description of the modules of FIG. 2, which may also be referred to by the descriptions of FIGS. 3 and 4, various operations may be described as being performed by different components of runtime execution environment 200. The operations that are described with respect to a particular component may be carried out by the particular components itself, by the particular component in cooperation with another of the components of runtime execution environment 200, or by the particular components in cooperation with a processing component from an unmanaged execution environment. Thus, the descriptions provided herein pertain to example implementations, and are not intended to be limiting in any manner.

[0025] Accordingly, runtime execution environment 200 may facilitate execution of managed code for either of an application programming or application execution platform. Managed code may be considered to be part of a core set of application-development technologies, and may further be regarded as code that is compiled for execution on runtime execution environment 200 to provide a corresponding service to the computing device platform. In addition, runtime execution environment 200 may translate managed code at an interpretive level into instructions that may be proxied and then executed by a processor. A framework for runtime execution environment 200 also provides class libraries, which may be regarded as software building blocks for managed applications.

[0026] According to a further example implementation, runtime execution environment 200 may provide at least partial functionality that may otherwise be expected from a kernel, which may or may not be lacking from a computing device platform depending upon resource constraints for the particular one of device 105, 110, and 115. Thus, at least one example of runtime execution environment 200 may implement the following: input/output (hereafter "I/O") routine management, memory management, compilation, and service routine execution. Thus, runtime execution

environment 200 may include I/O module 205, compiler 210, loader 215, memory management component 220 (alternatively referred to as a virtual machine), and service routine manager 225. These components are to be described in further detail below, and are provided only as examples, and may be implemented in examples of runtime execution environment 200 in various combinations and configurations thereof. The examples are not intended to be limiting to any particular implementation of a runtime execution environment, and no such inference should be made.

[0027] I/O module 205 of runtime execution environment 200 may provide asynchronous access to data sources (*i.e.*, processor and peripherals) associated with either of an application programming or application execution platform. More particularly, I/O module 205 may provide runtime execution environment 200 with robust system throughput and further streamline performance of code from which an I/O request originates.

[0028] Compiler 210 may refer to a module within runtime execution environment 200 that may interpret compiled IL into native machine instructions for execution in runtime execution environment 200 by, *e.g.*, execution module 230 or, alternatively, for execution by a CPU in a non-managed execution environment.

[0029] Loader 215 may refer to an assembly manager that may be invoked to locate and read assemblies as needed. Loader 215 may be disposed in execution environment 200, although at least one implementation of an unmanaged execution environment (*i.e.*, OS) may include loader 215 therein. Loader 215 may garner precompiled IL during deployment or install time, for loading into runtime execution environment 200. Thus, according to at least one alternative implementation of runtime execution environment 200, loader 215 may effectively serve as an entry point for precompiled IL into runtime execution environment 200.

[0030] Memory management component 220 may be regarded as a "garbage collector." Garbage collection may be regarded as a robust feature of managed code execution environments by which an object is automatically freed (*i.e.*, de-allocated) if an object is no longer used by any applications, upon a sweep or scan of a memory heap. In at least one example of memory management component 210, a sweep of free memory heap may be implemented as a linear search. Such implementation may be well-suited for an example of a computing device platform for which memory size is constrained and for which a delay in completion of a sweep may be perceived by a user of a corresponding device.

[0031] Further functions implemented by memory management component 220 may include: managing one or more contiguous blocks of finite volatile RAM (*i.e.*, memory heap) storage or a set of contiguous blocks of memory amongst the tasks running on the computing device platform; allocating memory to at least one application running on the computing device platform; freeing at least portions of memory on request by at least one of the applications; and preventing any of the applications from intrusively accessing memory space that has been allocated to any of the other applications.

[0032] Administrator 225 may refer to a module within runtime execution environment 200 that serves to receive, verify, and administer execution of at least a portion of an application, program, method, function, or other assemblage of code in runtime execution environment 200. In accordance with at least one example implementation of technologies associated with compiled adaptive code 120, administrator 225 may control the behavior of the application, program, method, function, or other assemblage code in runtime execution environment 220 without touching or affecting any executable portion thereof, at compile time, initial runtime, or at any time thereafter during execution of an application. More particularly,

administrator 225 may enforce type matching for code compiled by compiler 210 or loaded by loader 215. That is, administrator 225 may compare types or members of types requested by at least a portion of the code to the types or members of types used to satisfy those requests, in searching of satisfying matches. In the event that such a match does not exist for a respective request, administrator 225 may instruct execution module 230 to proceed with execution of the code compiled by compiler 210 or loaded by loader 215 until a request for the missing type or member of the missing type is at least statistically inevitable.

[0033] Execution module 230 may enable execution of managed code (*i.e.*, compiled native code) for the computing device platform. Execution module 230 may be regarded as a module in which execution of the code compiled by compiler 210 or loaded by loader 215 may be implemented in runtime execution environment 200, and in which runtime services (*e.g.*, device access and memory management) may be provided.

[0034] FIG. 3 shows example data structure 300 in accordance with one or more example technologies associated with adaptive compiled code 120 (see FIG. 1). More particularly, data structure 300 may represent at least a portion of compiled code corresponding to an application, program, or function that includes a reference to a type or member of a type that is missing. According to implementations of adaptive compiled code 120, an error associated with the missing type, or missing type member, may be deferred until a call to the reference is, at least, statistically inevitable. This implementation scenario may occur, typically though by no means exclusively, when the application, program, function, or other assemblage of code to which data structure 300 corresponds attempts to run on an older, or otherwise different, version of the platform on which it is generated, and is subjected to a verification process at compile time, initial runtime, or at any time thereafter during execution of the application, program, function, or assemblage of code to which data structure 300 corresponds.

[0035] Module 305 may refer to one or more modules of executable instructions corresponding to an application, program, function, or other assemblage of code being executable in accordance with, *e.g.,* execution component 230 in runtime execution environment 200 (see FIG. 2). More particularly, module 305 may refer to an entity such as methods, classes, DLLs (dynamic link libraries), frameworks, *etc.* Module 305 may be compiled by compiler 210 or loaded into runtime execution environment 200 by loader 215. That is, module 305 may be native code (*i.e.,* machine-readable code).

[0036] However, module 305 is not limited to the examples of native code only. Rather, alternative implementations of the technologies described herein may be application to code that is not compiled, and therefore be relevant to intermediate language code or other code written in any one of a variety of known languages for which at least one of multiple syntactic characteristics and construct properties may be sampled.

[0037] Module 310 may refer to a reference that is associated with module 305. More particularly, the reference of module 310 may be to a type or a member of a type that is not included in the application, program, function, or assemblage of code to which data structure 300 corresponds. However, in keeping with the example scenario in which implementations of adaptive compiled code 120 are presently described, it may be assembled for descriptive purposes only that the type, or at least one member of the type, to which the reference of module 310 corresponds is missing. Further, the referenced types, or members thereof, may be, but are by no means limited to, an integer, a floating point, a string, logical, or binary.

[0038] FIG. 4 shows example data flow 400 for producing and utilizing at least one example implementation of compiling and/or utilizing adaptive compiled code 120 (see FIG. 1).

[0039] In the following description, various operations will be described as being performed on or for one or more modules of data structure 300 (see FIG. 3) by components associated with runtime execution environment 200 (see FIG. 2). The operations that are described with respect to any particular one of these components may be carried out by the component itself, in combination with other components associated with the tool, or by the particular component in cooperation with one or more components of runtime execution environment 200. In addition, the operations may be executed by a processor or processors and implemented as hardware, firmware, or software, either singularly or in various combinations together. Further still, the operations may occur, typically though by no means exclusively, when the application, program, function, or other assemblage of code to which data structure 300 corresponds attempts to run on a different version of the platform on which it is generated, and is subjected to a verification process at compile time, initial runtime, or at any time thereafter during execution of the application, program, function, or assemblage of code to which data structure 300 corresponds.

[0040] Block 405 may refer to the compiling of code that results in data structure 300.

[0041] Environment 410 may be a managed execution environment, and block 405 may refer to a compiler interpreting IL into native machine instructions for execution in the managed execution environment or, alternatively, for execution by a CPU in a non-managed execution environment.

[0042] Environment 410 may, alternatively, refer to block 405 may refer to the compiling of at least a portion of an application, program, function, or other assemblage of code that is then loaded as one or more native image files in the aforementioned managed execution environment, thus circumventing CPU consumption required for compilation.

[0043] Data structure 300, as described above, may represent at least a portion of compiled code corresponding to an application, program, or function that includes a reference to a type or member of a type that is missing.

[0044] Module 305 may refer to one or more modules of executable instructions corresponding to an application, program, function, or other assemblage of code being executable in accordance with a managed execution environment. Module 305 may refer to an entity such as methods, classes, DLLs (dynamic link libraries), frameworks, *etc.*

[0045] Module 310 may refer to a reference to a type or a member of a type that is not included in the application, program, function, or assemblage of code to which data structure 300 corresponds.

[0046] Block 410 may refer to execution module 230 of runtime (*i.e.*, managed) execution environment 200 executing executable code 305 corresponding to data structure 300.

[0047] Decision 415 may refer to administrator 225 implementing a verification process with regard to the application, program, function, or assemblage of code to which data structure 300 corresponds. More particularly, administrator 225 may compare the types and members of types that may be requested during execution of executable code 305 to determine whether such requests are to be satisfied by the types and members of types found in the application, program, function, or assemblage of code to which data structure 300 corresponds.

[0048] The verification process implemented by administrator 225 may take place at compile time, initial runtime, or at any time thereafter during execution of the application, program, function, or assemblage of code to which data structure 300 corresponds. Therefore, at least portions of the functionality of administrator may occur at any point during data flow 400, and thus data flow 400, as depicted in FIG. 4 and

described herein, is provided as an example only. That is, alternative implementations of data flow 400 are not beholden to the order shown and described here.

[0049] Regardless, as the verification process reveals that a type or a member of a type that is subject to reference 310 is not included in the application, program, function, or assemblage of code to which data structure 300 corresponds, reference 310 may be flagged or otherwise noted without requiring an interruption to the execution of the application, program, function, or assemblage of code to which data structure 300 corresponds. Further, according to at least one implementation of data flow 400, administrator 225 may statistically determine a likelihood or probability that reference 310 to the missing type or type member may be called as other modules of executable code are executed.

[0050] Negative decision 420 may result in the continued execution of executable code 305 and other executable code modules associated with the application, program, function, or assemblage of code to which data structure 300 corresponds. That is, even though reference 310 is to a type or member of a type that is not present in the application, program, function, or assemblage of code to which data structure 300 corresponds, reference 310 does not cause an error according to implementations of technologies associated with adaptive compiled code 120.

[0051] Positive decision 425 may result as a statistical inevitability, as determined by administrator 225, during the execution of the application, program, function, or assemblage of code to which data structure 300 corresponds, or simply as a result of reference 310 to a missing type or type member being called.

[0052] That is, if the execution of modules of executable code for the application, program, function, or assemblage of code to which data structure 300, and therefore reference 310, corresponds reaches a point of statistical inevitability that reference 310 to a missing type or type member is to be called, the managed execution

environment may cause an error. Alternatively, the managed execution environment may cause an error simply when reference 310 to a missing type or type member is called. A non-limiting example of such error may be throwing an exception.

[0053] Thus, by the description above, pertaining to FIGS. 1 - 4, an error associated with a reference to a missing type or type member may be deferred until runtime, when a reference to a missing type or type member may actually be called.

[0054] However, the example implementations described herein are not limited to just the environment of FIG. 1, components of FIG. 2, modules of FIG. 3, or the process of FIG. 4. Technologies (*e.g.,* tools, methodologies, and systems) associated with adaptive compiled code 120 (see FIG. 1) may be implemented by various combinations of the features described with reference to FIGS. 2 - 4.

[0055] Further, the computer environment for any of the examples and implementations described above may include a computing device having, for example, one or more processors or processing units, a system memory, and a system bus to couple various system components.

[0056] The computing device may include a variety of computer readable media, including both volatile and non-volatile media, removable and non-removable media. The system memory may include computer readable media in the form of volatile memory, such as random access memory (RAM); and/or non-volatile memory, such as read only memory (ROM) or flash RAM. It is appreciated that other types of computer readable media which can store data that is accessible by a computer, such as magnetic cassettes or other magnetic storage devices, flash memory cards, CD-ROM, digital versatile disks (DVD) or other optical storage, random access memories (RAM), read only memories (ROM), electric erasable programmable read-only memory (EEPROM), and the like, can also be utilized to implement the example computing system and environment.

[0057] Reference has been made throughout this specification to "an example," "alternative examples," "at least one example," "an implementation," or "an example implementation" meaning that a particular described feature, structure, or characteristic is included in at least one implementation of the present invention. Thus, usage of such phrases may refer to more than just one implementation. Furthermore, the described features, structures, or characteristics may be combined in any suitable manner in one or more implementations.

[0058] One skilled in the relevant art may recognize, however, that code module initialization may be implemented without one or more of the specific details, or with other methods, resources, materials, *etc.* In other instances, well known structures, resources, or operations have not been shown or described in detail merely to avoid obscuring aspects of the invention.

[0059] While example implementations and applications of the code module initialization have been illustrated and described, it is to be understood that the invention is not limited to the precise configuration and resources described above. Various modifications, changes, and variations apparent to those skilled in the art may be made in the arrangement, operation, and details of the methods and systems of the present invention disclosed herein without departing from the scope of the invention, as both described above and claimed below.

WE CLAIM

1. A computer-readable medium having a verified data structure thereon, the data structure comprising:

an executable module; and

a reference module that makes reference to a module associated with a type that is missing from an assemblage of code to which the compiled data structure is associated.

2. A computer-readable medium according to Claim 1, wherein the executable module is a method.

3. A computer-readable medium according to Claim 1, wherein the module associated with a type that is missing is a type.

4. A computer-readable medium according to Claim 1, wherein the module associated with a type that is missing is a member of a type.

5. A computer-readable medium according to Claim 1, wherein a call to the reference module throws an exception.

6. A computer-readable medium according to Claim 1, wherein the verified data structure is compiled in a runtime execution environment.

7. A computer-readable medium according to Claim 1, wherein the verified data structure is received, by a runtime execution environment, in compiled form.

8. A method, comprising:

verifying source code;

compiling the verified source code into a machine language;

calling a method associated with the machine language; and

causing an error when the calling includes a reference to a module associated with a type that is missing from the compiled source code.

9. A method according to Claim 8, wherein the method is executed in a managed execution environment.

10. A method according to Claim 8, wherein the compiling is executed external to a managed execution environment and the calling and the causing are executed in the managed execution environment.

11. A method according to Claim 8, wherein the reference to the module associated with the type that is missing from the compiled source code is included in the compiled source code.

12. A method according to Claim 8, wherein the reference to the module associated with the type that is missing from the compiled source code is a reference to a type.

13. A method according to Claim 8, wherein the reference to the module associated with the type that is missing from the compiled source code is a reference to a member of a type.

14. At least one computer-readable medium having one or more executable instructions thereon that, when read, cause one or more processors to:

verify source code;

compile the verified source code into a native machine code;

execute native machine code that includes a reference to a module that is missing from the compiled code; and

cause an error when execution of the native machine code is determined to include an inevitable call to the reference.

15. At least one computer-readable medium according to Claim 14, wherein the one or more executable instructions to compile the verified source code are executed in a managed execution environment.

16. At least one computer-readable medium according to Claim 14, wherein the one or more executable instructions to compile the verified source code are executed by a just-in-time compiler.

17. At least one computer-readable medium according to Claim 14, wherein the one or more executable instructions to compile the verified source code are executed in an unmanaged execution environment, and the native machine code is then received by a managed execution environment.

18. At least one computer-readable medium according to Claim 14, wherein the reference to a module that is missing from the compiled code is a reference to a type.

19. At least one computer-readable medium according to Claim 14, wherein the reference to a module that is missing from the compiled code is a reference to a member of a type.

20. At least one computer-readable medium according to Claim 14, wherein the one or more instructions to cause an error when execution of the native machine code is determined to include an inevitable call to the reference cause the one or more processors to statistically determine the inevitability of the call to the reference.

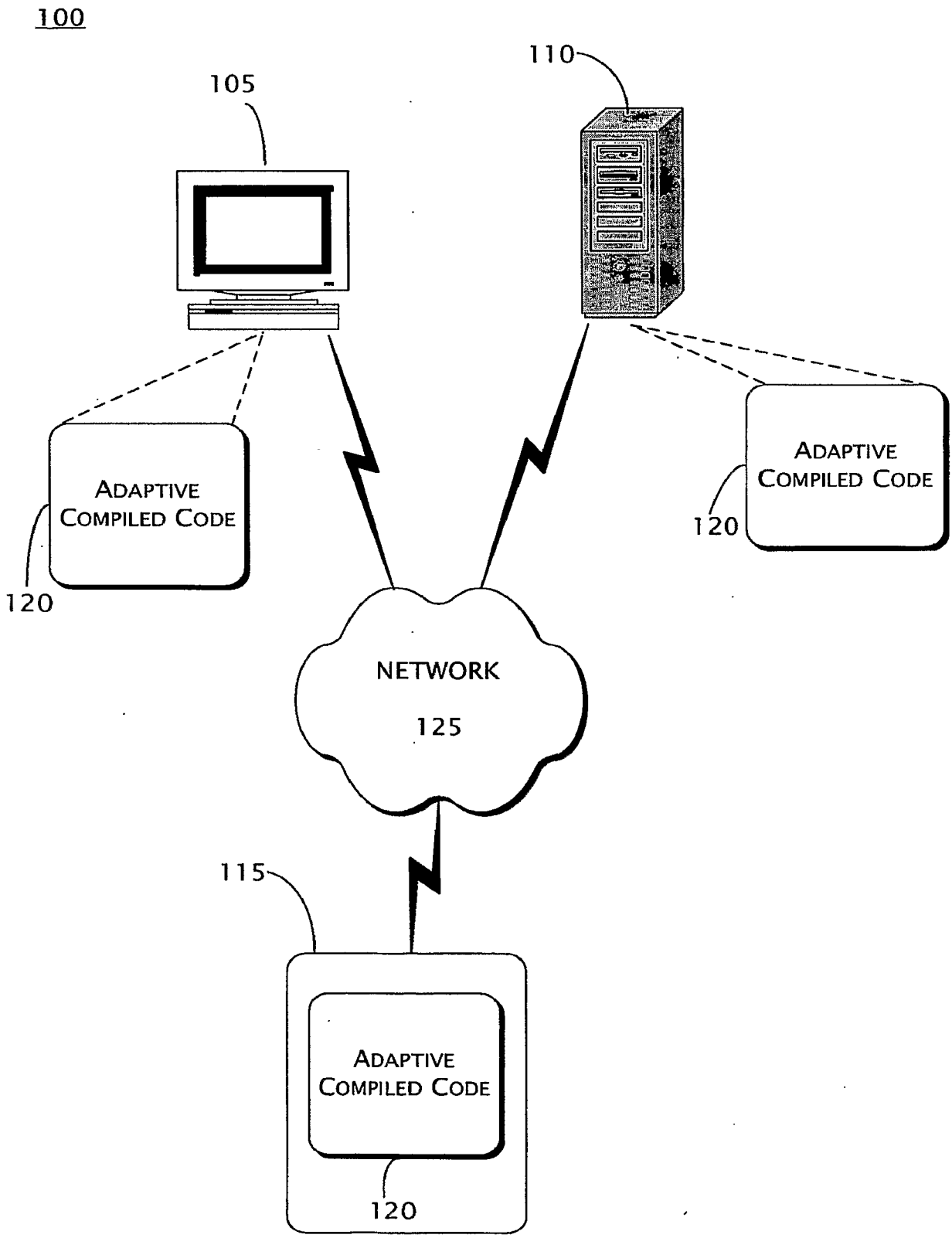

100
105
110
ADAPTIVE COMPILED CODE
120
ADAPTIVE COMPILED CODE
120
NETWORK
125
115
ADAPTIVE COMPILED CODE
120

FIG. 1

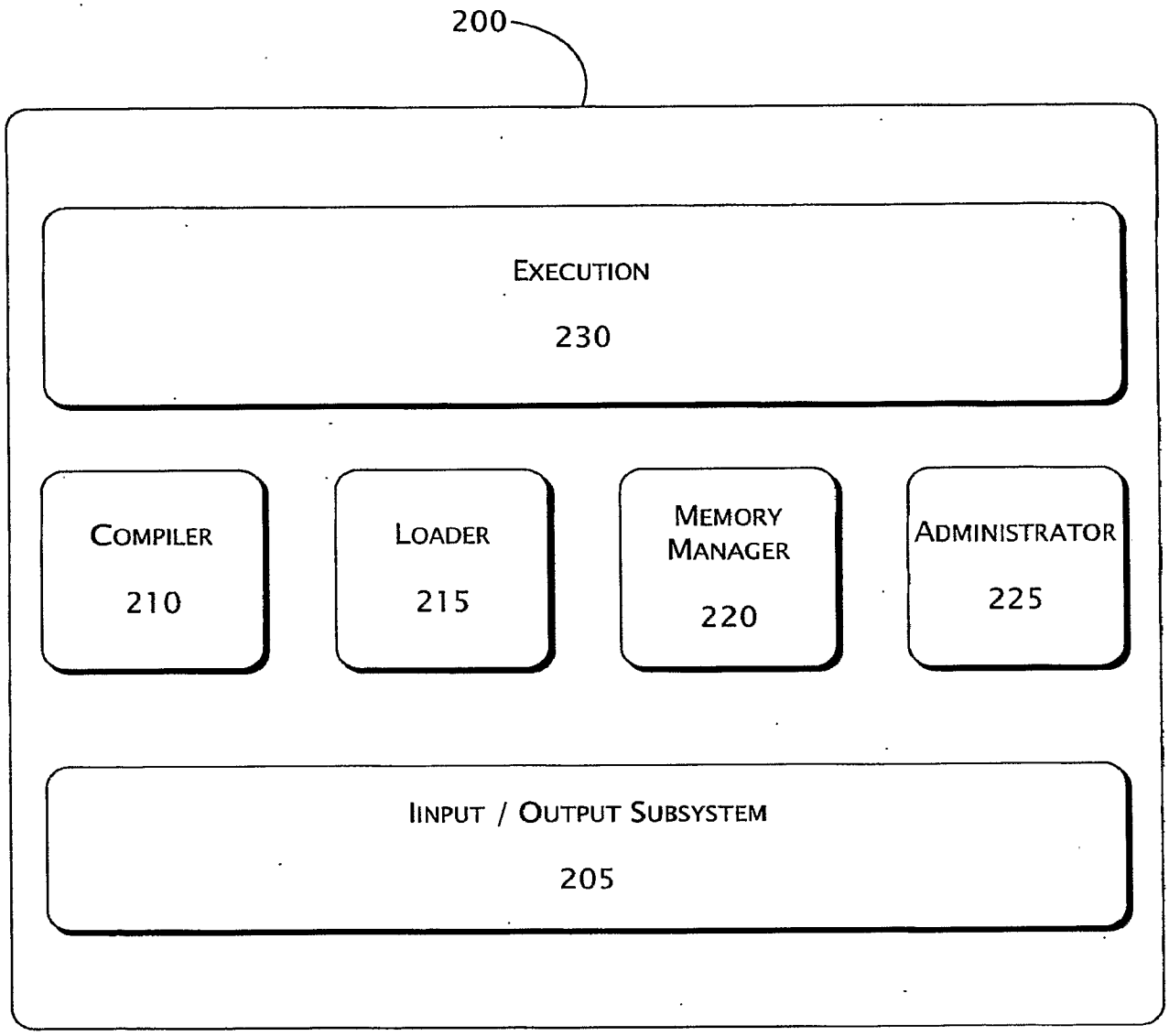

200
EXECUTION
230
COMPILER
210
LOADER
215
MEMORY MANAGER
220
ADMINISTRATOR
225
IINPUT / OUTPUT SUBSYSTEM
205

FIG. 2

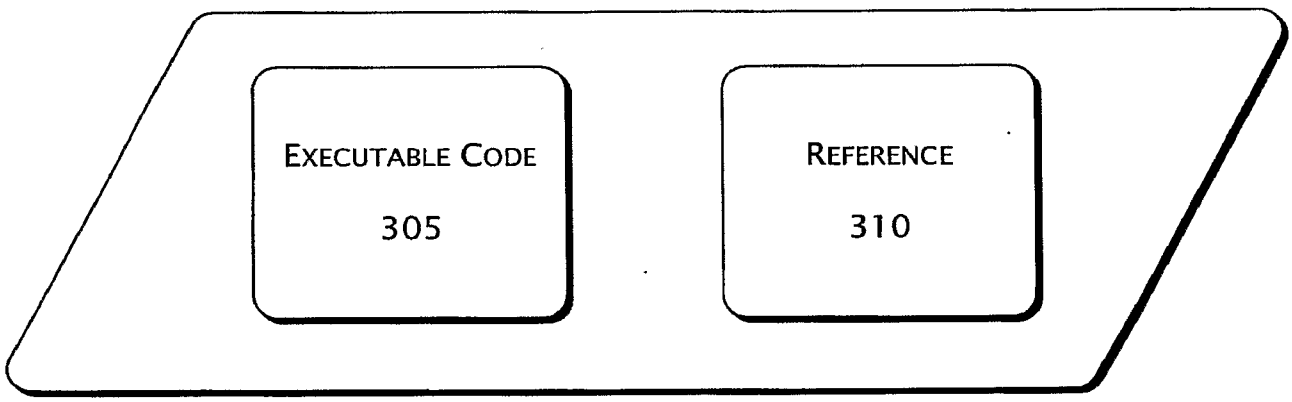


FIG. 3

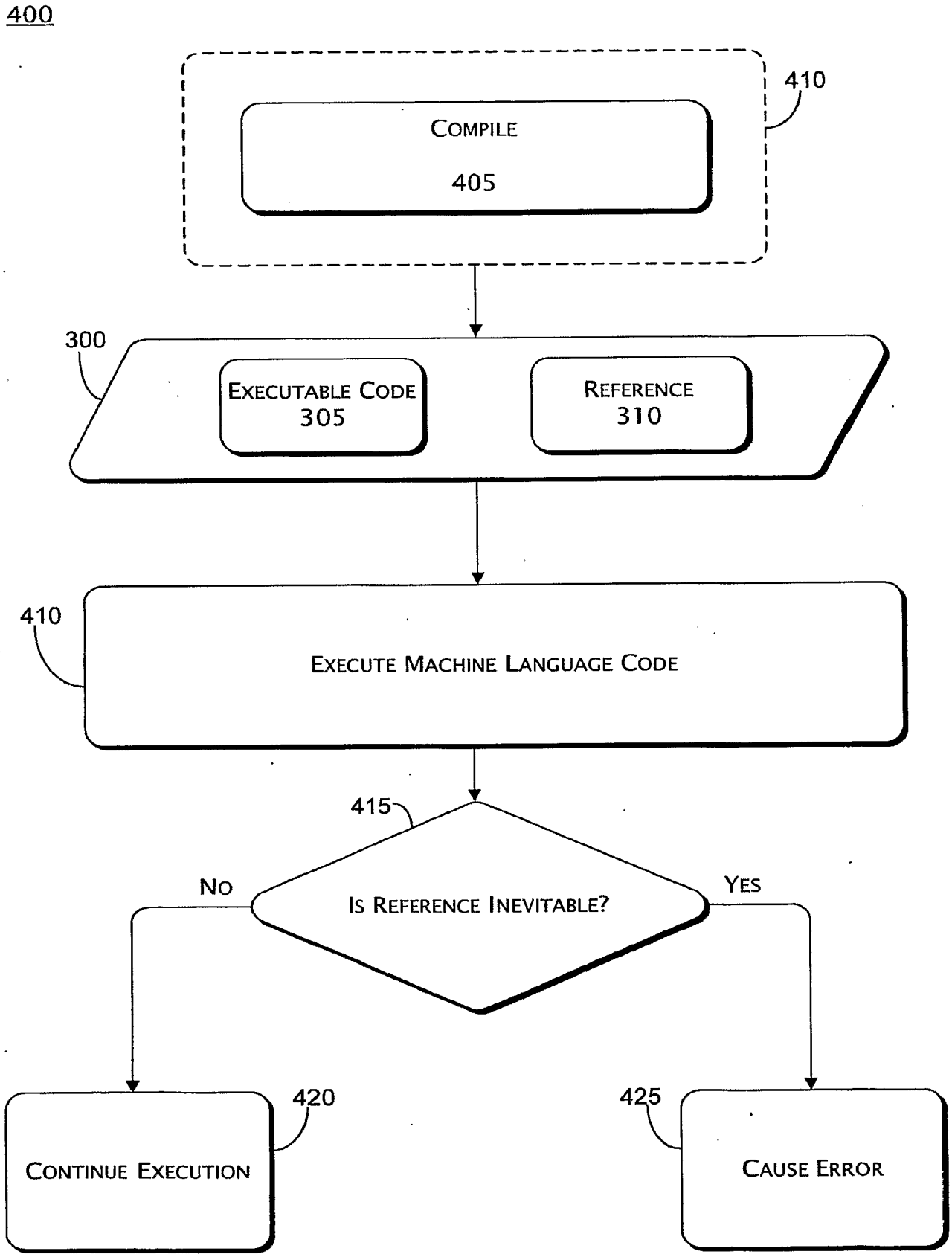

400
410
COMPILE
405
300
EXECUTABLE CODE
305
REFERENCE
310
410
EXECUTE MACHINE LANGUAGE CODE
415
IS REFERENCE INEVITABLE?
NO
YES
420
CONTINUE EXECUTION
425
CAUSE ERROR

FIG. 4

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2007/002320

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/45(2006.01)i, G06F 17/00(2006.01)i

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)

IPC 8: G06F

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

Korean utility models and applications for utility models since 1975

Japanese utility models and applications for utility models since 1975

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

eKIPASS(Kipo Internal), Google, YesKisti

keywords: install, operating system, OS , image, setting,| configuration

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2005/0172286 A1 (BRUMME C.W. et. al) 04 AUGUST 2005 See figure 1 and its description.	1-20
A	US 2002/0169999 A1 (BHANSALI S. et. al) 14 NOVEMBER 2002 See summary, especially [20], [21],[24],[25].	1-20
A	EP 1598739 A1 (ACCESS CO., LTD.) 23 NOVEMBER 2005 See claims 1,2.	1-20
A	US 2004/0268095 (SHPEISMAN T.) 30 DECEMBER 2004 See figures 3,5 and their descriptions.	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 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
22 MAY 2007 (22.05.2007)	**22 MAY 2007 (22.05.2007)**

Name and mailing address of the ISA/KR

Korean Intellectual Property Office
920 Dunsan-dong, Seo-gu, Daejeon 302-701,
Republic of Korea

Facsimile No. 82-42-472-7140

Authorized officer

YOON, Hye Sook

Telephone No. 82-42-481-8370

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2007/002320

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2005/0172286 A1	04.08.2005	NONE	
US 2002/0169999 A1	14.11.2002	EP 01258805A2	20.11.2002
EP 01598739 A1	23.11.2005	CN 1751291 A	22.03.2006
		JP 17502703	17.02.2004
		US 2006/174235 AA	03.08.2006
		WO 2004/075048 A1	02.09.2004
US 2004/0268095 A1	30.12.2004	NONE	