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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
9 July 2009 (09.07.2009)

PCT

(10) International Publication Number
WO 2009/085521 A1

(51) International Patent Classification:
G06F 9/30 (2006.01) *G06F 11/08* (2006.01)
G06F 9/06 (2006.01) *G06F 9/44* (2006.01)

(21) International Application Number:
PCT/US2008/084994

(22) International Filing Date:
26 November 2008 (26.11.2008)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
11/963,346 21 December 2007 (21.12.2007) US

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

(72) Inventors: **BARNETT, Michael**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **FAHNDRICH, Manuel A.**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **GRUNKEMEYER, Brian M.**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **SCHULTE, Wolfram**; One Microsoft Way, Redmond, Washington 98052-6399 (US).

(74) Common Representative: **MICROSOFT CORPORATION**; Attention : Sharon Rydberg (sharonr), LCA, International Patent Department, One Microsoft Way, 8/2321, 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, BR, BW, BY, BZ, CA, CH, 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, IS, JP, KE, KG, KM, 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, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, 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, HR, HU, IE, IS, IT, LT, LU, LV, MC, MT, NL, NO, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: CONTRACT PROGRAMMING FOR CODE ERROR REDUCTION

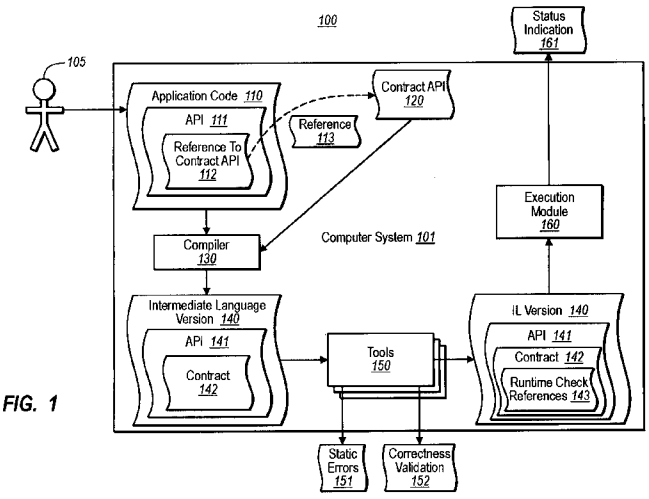

FIG. 1

(57) Abstract: In one embodiment, a computer system provides an application programming interface (API) for augmenting an application API. A computer system receives software code written in a second programming language indicating a user's intention to augment an application API with contracts from a contract API written in a first programming language. The software code includes a reference to the contract API. The contracts include assertions indicating appropriate use of the application API. The computer system accesses portions of the contract API according to the reference in the software code and compiles the received software code and the referenced portions of the contract API into an intermediate language (IL) version of the software code. The IL version is in an intermediate language common to both the first programming language and the second programming language. The IL version includes the assertions indicating appropriate use of the application API.

WO 2009/085521 A1

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))*

Published:

— *with international search report*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

CONTRACT PROGRAMMING FOR CODE ERROR REDUCTION

BACKGROUND

[0001] As computer use has proliferated in today's society, the number of methods and systems for developing software applications to run on these computers has also increased. Software applications may be used to perform a wide variety of tasks based on the design of the application being used. Software applications typically include a number of individual files designed to work together to create some type of desirable end result. In order to write such software applications, software developers typically decide on a programming language to use for that application. Many programming languages are in use today and, as a result, many software applications are written in different languages. These languages are often incompatible with each other, not a few of which use different syntax, different compiling methods and other, different code elements.

[0002] To simplify the integration and use of multiple programming languages in their various implementations, managed code has been introduced to provide a platform in which these various applications can interact. Managed code, among other things, allows application developers to specify attributes of a program so that other developers seeking to use some portion of the application will know how best to do so. For example, a developer may specify various elements corresponding to an application variable, a method or an entire process. The developer may, for instance, state in a contract or specification that variable X is to have a certain type, or is to be greater than a certain value. Such application contracts or specifications are often used to reduce the number of errors created in the code before the code is compiled.

[0003] Such application contracts, however, are very limited in what they can specify. For example, most application contracts are limited to specifying a variable type. This is useful in eliminating type errors; however, many other errors that may exist would not be prevented by using such a contract.

BRIEF SUMMARY

[0004] Embodiments described herein are directed to augmenting an application API. In one embodiment, a computer system provides an application programming

interface (API) for augmenting an application API. A computer system receives software code written in a second programming language indicating a user's intention to augment an application API with contracts from a contract API written in a first programming language. The software code includes a reference to the contract API. The contracts include assertions indicating appropriate use of the application API. The computer system accesses portions of the contract API according to the reference in the software code and compiles the received software code and the referenced portions of the contract API into an intermediate language (IL) version of the received software code. The IL version is in an intermediate language common to both the first programming language and the second programming language. The IL version includes the assertions indicating appropriate use of the application API.

[0005] In other embodiments, a computer system receives software code from a computer user indicating the user's intention to augment an application API with contracts from a contract API. The software code includes a reference to the contract API. The contracts include assertions indicating appropriate use of the application API. The computer system accesses portions of the contract API according to the reference in the software code, compiles the received software code and the referenced portions of the contract API into an intermediate language version of the received software code including the one or more assertions indicating appropriate use of the application API, and provides the compiled intermediate language version of the software code to the computer user.

[0006] 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.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] To further clarify the above and other advantages and features of embodiments of the present invention, a more particular description of embodiments of the present invention will be rendered by reference to the

appended drawings. It is appreciated that these drawings depict only typical embodiments of the invention and are therefore not to be considered 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:

[0008] Figure 1 illustrates a computer architecture in which embodiments of the present invention may operate including modifying and/or generating an API contract.

[0009] Figure 2 illustrates a flowchart of an example method for augmenting an application API.

[0010] Figure 3 illustrates a flowchart of an alternative example method for augmenting an application API.

[0011] Figure 4 illustrates an exemplary embodiment of a modified contract.

[0012] Figure 5 illustrates an exemplary embodiment of a generated contract.

[0013] Figure 6 illustrates an exemplary system for generating an executable binary using a contract's API.

DETAILED DESCRIPTION

[0014] Embodiments described herein are directed to augmenting an application API. In one embodiment, a computer system provides an application programming interface (API) for augmenting an application API. A computer system receives software code written in a second programming language indicating a user's intention to augment an application API with contracts from a contract API written in a first programming language. The software code includes a reference to the contract API. The contracts include assertions indicating appropriate use of the application API. The computer system accesses portions of the contract API according to the reference in the software code and compiles the received software code and the referenced portions of the contract API into an intermediate language (IL) version of the received software code. The IL version is in an intermediate language common to both the first programming language and the second programming language. The IL version includes the assertions indicating appropriate use of the application API.

[0015] In other embodiments, a computer system receives software code from a computer user indicating the user's intention to augment an application API with contracts from a contract API. The software code includes a reference to the contract API. The contracts include assertions indicating appropriate use of the application API. The computer system accesses portions of the contract API according to the reference in the software code, compiles the received software code and the referenced portions of the contract API into an intermediate language version of the received software code including the one or more assertions indicating appropriate use of the application API, and provides the compiled intermediate language version of the software code to the computer user.

[0016] In other embodiments, a computer system determines that a software application has an API configured to include at least one contract written in a first programming language. The computer system receives software code written in a second programming language from a computer user indicating the user's intention to generate an executable contract for the software application using the software code written in the second programming language.

[0017] The computer system sends the software code to a managed framework component configured to receive software code written in the second programming language and output a version of the software code in a runtime language common to both the first programming language and the second programming language. The computer system receives an intermediate language version of the software code written in the second programming language and generates an executable contract for the software application based on the received intermediate language version of the software code.

[0018] Embodiments of the present invention may comprise or utilize a special purpose or general-purpose computer including computer hardware, 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 physical storage media. 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: physical storage media and transmission media.

[0019] Physical storage media includes RAM, ROM, EEPROM, CD-ROM or 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.

[0020] 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. Transmission media can include a network and/or data links which can be used to carry or transport 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.

[0021] However, it should be understood, that 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 physical storage media. For example, computer-executable instructions or data structures received over a network or data link can be buffered in RAM within a network interface card, and then eventually transferred to computer system RAM and/or to less volatile physical storage media at a computer system. Thus, it should be understood that physical storage media can be included in computer system components that also (or even primarily) utilize transmission media.

[0022] Computer-executable instructions comprise, for example, instructions and data which 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.

[0023] 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, 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.

[0024] Figure 1 illustrates a computer architecture 100 in which the principles of the present invention may be employed. Computer architecture 100 includes computer system 101. In some embodiments, computer system 101 includes application code 110. Application code 110 may be any type of software application code of any size or complexity, designed for any type of use. Application code 110 may be part of a software application or may, in itself, comprise a software application. Application code 110 may include any type of code written in any programming language. Code 110 may include methods, processes, linked libraries, individual lines of code or any other type of information that can be used to interact with a computer system.

[0025] In some embodiments, application code 110 also includes application programming interface (API) 111. An API may be any type of interface that allows a user or developer to access functionality of a software application. For example, application developers typically write API's to allow third party developers to write software applications that use features of the developer's application. In some cases, as will be explained in greater detail below, it may be advantageous to provide assertions in the API's contract (e.g., contract 142) indicating how the application's API is to be used. Using the information in the contract, a developer may better understand how to interact with the software application's API, and, as a result, draft higher quality, bug-free code.

[0026] Contract API 120 in computer system 101 may be configured to receive reference 113, the reference indicating that the contract API is to be used to modify or generate an API contract. For example, API 111 may include reference to contract API 112, indicating that contract API 120 is to be used to modify one of API 111's contracts. Contracts may include assertions indicating how application API 111 is to be used. Assertions may define boundaries within which application API 111 is best suited to operate. Examples of assertions include preconditions, postconditions, specific assertions, invariants, lock declarations and checked exceptions. Assertions will be explained in greater detail below.

[0027] User 105 may be any type of computer user including a software developer, a client/end-user, a system administrator or any other type of computer user. In some cases, user 105 may send application code 110 to compiler 130. Compiler 130 may be any type of software application configured to compile some portion of software code. Compiler 130 may output code in an intermediate form, in some type of intermediate language (IL), or may output executable, machine-readable code. In some embodiments, compiler 130 may be part of a managed framework, and may be referred to as a managed framework component.

[0028] As used herein, a managed framework may be a means for allowing code files written in multiple different programming languages to be reduced (or compiled) to intermediate language instructions that can be operated in a uniform manner by a managed framework (such as the Microsoft Common Language

Runtime, a Java Virtual Machine, or a list processing language (LISP) interpreter). This allows for interoperability among multiple programming languages. Thus, each managed framework has languages that are compatible with that framework and languages that are not.

[0029] In some embodiments, computer system 101 may be configured to determine whether application code 110 is written in a language compatible with a given managed framework or compiler (e.g., compiler 130). Moreover, software code stored in computer system 101 and accessed by compiler 130 may be processed to determine its language compatibility. Thus, at least in some embodiments, it may be determined that application code 110 is compatible with compiler 130, regardless of whether the code was received from user 105 or was accessed in a data store on computer system 101.

[0030] Compiler 130 may be configured to process application code 110 and sends a intermediate language (IL) version of the software code 140, including API 141 and API contract 142 to tools 150. In some embodiments, tools 150 may include a variety of software tools configured to perform a variety of functions. Tools 150 may include tools for determining whether static errors exist in IL version 140, for performing correctness validations on the IL version or inserting references to runtime checks into the contract such that runtime checks can be injected into application code according to the corresponding references. Many other tools may be included in tools 150. Tools 150 may be configured to send IL version 140 with API 141 and contract 142 with runtime check references 143 to execution module 160. Execution module 160 may be any type of software method, process or application configured to execute IL version 140. Execution module 160 may output status indication 161 indicating that the execution has produced a failure or contract violation, or, alternatively, that no failures or contract violations occurred. This process will be explained in greater detail in connection with Figures 2 & 3.

[0031] Figure 2 illustrates a flowchart of a method 200 for augmenting an application API. The method 200 will now be described with frequent reference to the components and data of Figures 1, 4 and 6.

[0032] Method 200 includes an act of receiving software code written in the second programming language indicating the user's intention to augment the application API with contracts from the contract API written in the first programming language, the software code including a reference to the contract API, the contracts including one or more assertions indicating appropriate use of the application API (act 210). For example, compiler 130 may receive application code 110 (the terms software code and application code are used synonymously herein) written in a second programming language (e.g., Visual Basic (VB), C#, etc.) indicating user 105's intention to augment application API 111 with contracts (e.g., contract 142) from contract API 120 written in a first programming language (e.g., VB, C#, etc.). Application code 110 includes reference to contract API 112, and in some cases, may communicate with or refer to contract API 120 using reference 113. Contracts (e.g., contract 142) include assertions indicating appropriate use of application API 111. Due to the general nature of contract API 120, contracts may be fully extensible to include any type of code, methods, processes, etc. Furthermore, the first programming language may be any general use programming language. Contract extensibility may allow for new methods to be added to the contract API, thus enabling new kinds of contracts to be written. Clients may be able to use such new methods written in the second programming language in the contracts.

[0033] In some cases, assertions may be used by a developer to write an application that uses that API. For example, if a third party developer wanted to write Application A and use functionality provided in Application B, the developer would use Application B's API to access Application B's functionality. In some embodiments, a contract in Application B's API may include assertions such as preconditions and postconditions that are to be satisfied when using Application B's API. Assertions may define boundaries within which Application B is best suited to operate.

[0034] Assertions may include, but are not limited to the following: preconditions, postconditions, specific assertions, invariants, lock declarations and checked exceptions. Preconditions, as used herein, indicate requirements that the caller of

the function is to satisfy. Preconditions are required to hold true upon entry to the function. Postconditions, as used herein, indicate guarantees that the implementation of a function will fulfill on completion (potentially both successful & exceptional completion). Postconditions must hold true upon exit from the function. Both preconditions and postconditions may be included as part of the declaration of a method.

[0035] Specific assertions, as used herein, are included as part of the code of a program and serve as sanity checks or clarifications in the program logic. They may also serve as a help to static verifiers. Specific assertions must be true at the appropriate point in the control flow of the program where they are included. Invariants, as used herein, indicate conditions which hold true across all operations on a type. In some embodiments, invariants are included as part of the declaration of a type and may act as assertions on data. Invariants may be temporarily invalidated during some windows of execution of the program, as long as external clients (human or programs) cannot observe any of the invalid periods. Lock declarations, as used herein, are statements used to support analysis related to behavior of the program under concurrency. Lock declarations are designed to specify which locks protect which data. Preconditions and postconditions may be used to express which locks are to be held or not held. Checked exceptions, as used herein, constrain the set of exceptions which can be thrown from a method. Checked exceptions are similar to postconditions, but cover exceptional exits. Checked exceptions may be included as part of the declaration of a method.

[0036] In general, assertions are designed to perform a similar function: to indicate that a program is incorrect if the stated condition is not true at a specified point in execution. Thus, assertions may be used for documentation of the state or behavior of the program at a specific point, used as a computable expression which can be used to help analyze or detect errors in a program, used as a contract between distinct parts of a program (a requirement that code leading up to the assertion point must fulfill, or an assurance that code following the assertion point can rely on), or in any other of a number of uses (such as code generation via a compiler, assuming the assertions are trusted or proved correct). In some cases,

assertions in a contract may be used by static program verifiers to check if the code conforms against the contract or by bug finding tools to find possible errors in the code.

[0037] Thus, application code 110 may include API 111 which may have existing contracts, or may not have any contracts. In some cases, API 141 may include multiple contracts, each capable of having multiple assertions corresponding to different methods. In some cases, API contract 142 may include one or more preconditions specifying those items that are to be satisfied before a caller is permitted to call a method. Thus, in order to use a method having such a precondition, the precondition must be satisfied before the method may be called. In other words, if the precondition is not satisfied, that method may not be called (or if the method is called, the method may fail). For example, it may be advantageous to prevent calling a method when a precondition is violated via checks performed in the build process. Furthermore, the preconditions may be evaluated at runtime as a potential backup for when a user fails to use the complete build process. In cases where a precondition is not met, running the method may fail by throwing an exception or performing some other type of behavior (such as triggering an escalation policy).

[0038] Similarly, API contract 142 may include one or more postconditions specifying one or more guarantees that hold upon return of a method. Thus, at least in some cases, if the postcondition is not satisfied upon return of the method, the contract will no longer allow use of the API. API contract 142 may also include one or more object invariants that indicate conditions which hold true across all operations on a type. It should be noted that any of the assertions in contract 142 may be declarative. Thus, declarative assertions may describe the function of the assertion without describing each of the implementation details.

[0039] Method 200 includes an act of accessing portions of the contract API according to the reference in the software code (act 220). For example, compiler 130 may access portions of contract API 120 according to reference 112 in application code 110. Thus, in some cases, compiler 130 may use reference 112 to

access portions of contract API 120 for use in compiling intermediate language version 140.

[0040] Method 200 also includes an act of compiling the received software code and the referenced portions of the contract API into an intermediate language version of the received software code, the intermediate language version being in an intermediate language common to both the first programming language and the second programming language, the intermediate language version including the one or more assertions indicating appropriate use of the application API (act 230). For example, compiler 130 may compile application code 110 and the referenced portions of contract API 120 into intermediate language version 140 of application code 110. Intermediate language version 140 may be in an intermediate language (IL) common to both the first programming language (e.g., VB, C#, etc.) and the second programming language (e.g., VB, C#, etc.). The IL version 140 may include assertions indicating appropriate use of application API 141.

[0041] In some embodiments, for example, application code 110 may be written in Visual Basic and contract API may be written in C#, compiler 130 may be configured to output an IL version of the Visual Basic code that is compatible with the contract API 120 written in C#. In some cases, both languages may be part of a managed framework. Because, in such cases, both languages are part of a common managed framework, code written in one language may be used to modify code written in another language. This may be advantageous in situations where contract API 120 is written in a programming language unfamiliar to user 105, who nonetheless desires to modify an application API's contract, possibly by adding various assertions.

[0042] In some embodiments, computer system 101 includes tools 150 which may include a variety of different software tools. For instance, tools 150 may include a tool configured to access compiled intermediate language version 140 and insert one or more runtime check references 143 into contract 142. Runtime check references 143 may be used as placeholders to identify places in application code where runtime checks may be inserted. Such a tool may be further configured to execute IL version 140 including contract 142 with runtime check references, and

determine that at least one of the assertions in contract 142 was violated during the execution of the IL version. Furthermore, based on such a determination, the tool may be configured to provide an indication that at least one of the assertions was violated (e.g., in status indication 161). In other cases, one of tools 150 may be configured to execute IL version 140 including contract 142 with inserted runtime check references, and determine that none of the assertions in contract 142 were violated during the execution of the IL version. And, based on such a determination, the tool may be configured to provide an indication that none of the assertions were violated (e.g., in status indication 161).

[0043] In some embodiments, tools 150 may include a tool configured to perform a static analysis on IL version 140 to determine whether the assertions of contract 142 were satisfied. In some cases, if the static analysis determines that the assertions of the contract were satisfied, the user may be notified of such (e.g., in correctness validation 151). Similarly, if one or more of the assertions failed to be satisfied, a notification (e.g., static errors 151) may be configured to include such information. In some cases, the user may simply be notified that the contract was modified.

[0044] In an exemplary embodiment described in Figure 4, a contract may be augmented with assertions in the manner described above. Application code 406 in its untouched, original form (405), includes API 408 and contract 409 with multiple assertions 1, 2 and 3, which may correspond to various assertions such as preconditions, postconditions, invariants, or other elements. User 105 may input application code 406 that, after being compiled with portions of contract API 120, is used to augment contract 409. The resultant modified version 425 of application code 426 includes API 428 and contract 429 with augmented (or new) assertions 1A, 3 & 4. Assertion 1A is a modified version of original assertion 1, assertion 3 is the same, and assertion 4 has been added. Thus, contract augmentation may include altering existing assertions, removing assertions (such as element 2) and/or adding assertions to a contract.

[0045] Figure 3 illustrates a flowchart of an alternative method 300 for augmenting an application API. The method 300 will now be described with frequent reference to the components and data of Figures 1, 5 and 6.

[0046] Method 300 includes an act of receiving software code from a computer user indicating the user's intention to augment an application API with contracts from a contract API, the software code including a reference to the contract API, the contracts including one or more assertions indicating appropriate use of the application API (act 310). For example, application code 110 may be received from computer user 105 indicating user 105's intention to augment application API 111 with contracts 142 from contract API 120. Application code 110 includes reference to contract API 112. Contract 142 includes one or more assertions indicating appropriate use of application API 141. As indicated above, contract 142 may include one or more preconditions specifying those items that are to be satisfied before a caller is permitted to call a method. Additionally or alternatively, contract 142 may include one or more postconditions specifying one or more guarantees that hold upon return of a method. Invariants, specific assertions, lock declarations, checked exceptions and other assertions may also be individually or collectively included in API contract 142.

[0047] Method 300 also includes an act of accessing portions of the contract API according to the reference in the software code (act 320). For example, compiler 130 may access portions of contract API 120 according to reference 112 in application code 110.

[0048] Method 300 includes an act of compiling the received software code and the referenced portions of the contract API into an intermediate language version of the received software code including the one or more assertions indicating appropriate use of the application API (act 330). For example, compiler 130 may compile application code 110 and the referenced portions of contract API 120 into IL version 140 including API 141 and contract 142 with various assertions indicating appropriate use of application API 141. In some cases, compiler 130 may be part of a managed framework configured to receive software code and output a intermediate language (IL) version of the application code 140 that is

common to multiple different programming languages included in the managed framework. For example, if software code 110 is written in C# and contract 142 is written in Visual Basic, compiler 130 may be configured to output an IL version of the C# code that is compatible with the contract 142 written in Visual Basic.

[0049] Method 300 includes an act of providing the compiled intermediate language version of the software code to the computer user (act 340). For example, computer system 101 may provide compiled IL version 140 to computer user 105. In some cases, the IL version may be provided to user 105 for subsequent application to tools 150. In some cases, computer system 101 may provide IL version 140 directly to tools 150. As indicated above, many different tools may be used to perform anything from error checking (runtime or static), documentation generation, logging, or other features. Documentation, for example, may be used to inform such users how to use the contract with the application API. Such documentation may be generated automatically by tools 150.

[0050] Contract 142 may represent an augmented version of an existing contract already a part of API 111. Additionally or alternatively, contract 142 may represent a new contract generated using contract API 120. In an exemplary embodiment described in Figure 5, contract 529 for API 528 may be generated in a manner similar to contract augmentation described above. In other words, contract generation may be equivalent to augmenting a contract that contains no information (i.e., no assertions). In this example, application code 506 in its untouched, original form (505) includes API 508. Application code 110, after being converted to an IL version, may be used to produce an application API with a generated contract (525). The resultant generated application code 526 includes API 528 and generated contract 529 with assertions 1, 2 and 3, which may correspond to various assertions such as preconditions, postconditions, invariants, or other elements. Thus, API 528 may be configured to include and use generated contract 529, including any or all of the various assertions included in the contract.

[0051] Some embodiments may include a system for generating an executable binary that includes application code, one or more contracts, library references and other elements. For example, as described in Figure 6, compiler 615 may be

configured to receive software application code including API 605 with contract 606 identifying one or more assertions 607 (some of which may be conditional) that specify how to interact with the software application code. Contract 606 is extensible and may include one or more references indicating portions of application code that are to be replaced with a value. Furthermore, contract 606 may be written in any general purpose programming language.

[0052] Compiler 615 may be configured to integrate the one or more assertions, along with any received binaries 610, into corresponding methods in the software application code. Rewriter 625 may be configured to receive compiled binary 620 binary from compiler 615, where the binary includes software application code 621, contract 622 and library references 623, and rewrite the received binary such that the portions of application code indicated by the references in the contract are replaced with values in the intermediate (runtime) language.

[0053] In some cases, the binary may be rewritten for both runtime checking as well as subsequent reading. Furthermore, this new (rewritten) binary may be used as input for compiling a new piece of software that uses the rewritten software. The rewritten binary may be output as executable binary 630. Rewriter 625 may optionally have access to one or more libraries 626 for use in rewriting. In some cases, the software code and the contract may be compiled separately and then later combined by rewriter 615. As indicated above, the assertions may include preconditions, postconditions, object invariants, lock declarations, checked exceptions, usage protocols, or any other type of assertion.

[0054] Thus, according to some embodiments, a user may be able to write software code in one language that may be used to augment a contract of an application API written in another language. The contract may include multiple assertions including preconditions and postconditions, which are to be satisfied in order to properly use the application API associated with the contract, which may lead to fewer programming errors.

[0055] 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

We claim:

1. A computer program product comprising one or more computer-readable media having thereon a contract application programming interface (API) written in a first programming language for augmenting an application API written in a second programming language, the application API including one or more contracts, the computer program product comprising computer-executable instructions that, when executed by one or more processors of the computing system, cause the computing system to perform a method, the method comprising:

an act of receiving software code written in the second programming language indicating the user's intention to augment the application API with contracts from the contract API written in the first programming language, the software code including a reference to the contract API, the contracts including one or more assertions indicating appropriate use of the application API;

an act of accessing portions of the contract API according to the reference in the software code; and

an act of compiling the received software code and the referenced portions of the contract API into an intermediate language version of the received software code, the intermediate language version being in an intermediate language common to both the first programming language and the second programming language, the intermediate language version including one or more assertions indicating appropriate use of the application API.

2. The API configured to perform the method of claim 1, further comprising:

an act of accessing the compiled intermediate language version with one or more software tools; and

an act of inserting one or more references to one or more runtime checks into at least one of the contracts such that runtime checks can be injected into the application code according to the corresponding references.

3. The API configured to perform the method of claim 2, further comprising:

an act of executing the intermediate language version including the inserted runtime checks;

an act of determining that at least one of the assertions in the contracts was violated during the execution of the intermediate language version; and

based on the determination, an act of providing an indication that at least one of the assertions was violated.

4. The API configured to perform the method of claim 2, further comprising:

an act of executing the intermediate language version including the inserted runtime checks;

an act of determining that none of the assertions in the contracts were violated during the execution of the intermediate language version;

based on the determination, an act of providing an indication that none of the assertions were violated.

5. The API configured to perform the method of claim 1, further comprising an act of statically checking at least one of the application API's contracts to determine whether any of the contract's assertions were violated.

6. The API configured to perform the method of claim 1, wherein the assertions include one or more preconditions specifying those items that are to be satisfied before a caller is permitted to call a method.

7. The API configured to perform the method of claim 1, wherein the assertions include one or more postconditions specifying one or more guarantees that hold upon return of a method upon at least one of normal termination of the method and exceptional termination of the method.

8. The API configured to perform the method of claim 1, wherein the assertions include at least one of lock declarations.

9. The API configured to perform the method of claim 1, wherein the assertions include at least one checked exception.

10. The API configured to perform the method of claim 1, wherein the assertions include at least one of object invariants.

11. The API configured to perform the method of claim 1, wherein the contracts are extensible as a result of the contract API, the extensibility allowing for new methods to be added to the contract API to enable new kinds of contracts to be

written such that one or more clients can use the new methods written in the second programming language in the contracts.

12. The API configured to perform the method of claim 1, wherein at least one of the assertions in the contracts is declarative.

13. The API configured to perform the method of claim 1, further comprising an act of automatically generating API documentation for the software application based on the augmented contract.

14. The method of claim 1, wherein contracts written using the contract API are persisted for a plurality of downstream tools.

15. At a computer system, a method for augmenting an application API, the application API including one or more contracts:

an act of receiving software code from a computer user indicating the user's intention to augment an application API with contracts from a contract API, the software code including a reference to the contract API, the contracts including one or more assertions indicating appropriate use of the application API;

an act of accessing portions of the contract API according to the reference in the software code;

an act of compiling the received software code and the referenced portions of the contract API into an intermediate language version of the received software code including one or more assertions indicating appropriate use of the application API; and

an act of providing the compiled intermediate language version of the software code to the computer user.

16. The method of claim 9, wherein the assertions include one or more of the following: preconditions, postconditions, lock declarations, checked exceptions, usage protocols and object invariants.

17. The method of claim 9, wherein the contract is extensible such that the contracts can be written in a fully general programming language.

18. A computer system for modifying an API contract, the system comprising:

a compiler configured to perform the following:

receive software application code including an API with at least one contract identifying one or more assertions that specify how to interact with the software application code, the contract being extensible to include one or more references indicating portions of application code that are to be replaced with a value, the contract being written in a general programming language; and

integrate the one or more assertions into corresponding methods in the software application code; and

a rewriter configured to perform the following:

receive a binary from the compiler, the binary including at least one of software application code and the at least one contract;

rewrite the received binary such that the portions of application code indicated by the references in the contract are replaced with values; and

output the rewritten binary as an executable binary.

19. The system of claim 18, wherein the compiler is configured to receive one or more binaries.

20. The system of claim 18, wherein the assertions are at least one of a precondition, a postcondition, a lock declaration, a checked exception, a usage protocol, and an object invariant.

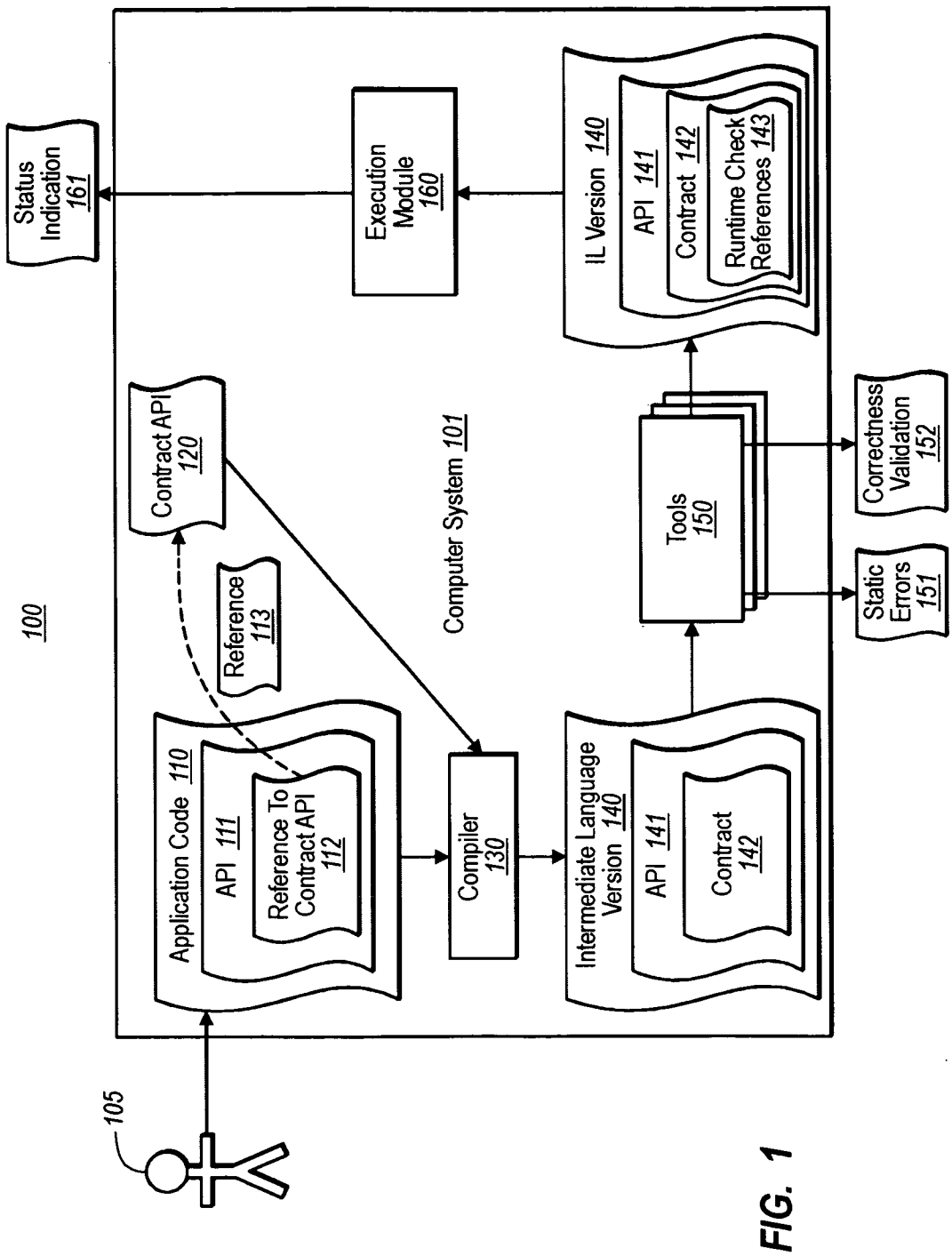
100
105
Application Code 110
API 111
Reference To Contract API 112
Reference 113
Contract API 120
Compiler 130
Intermediate Language Version 140
API 141
Contract 142
Computer System 101
Tools 150
IL Version 140
API 141
Contract 142
Runtime Check References 143
Execution Module 160
Status Indication 161
Static Errors 151
Correctness Validation 152

FIG. 1

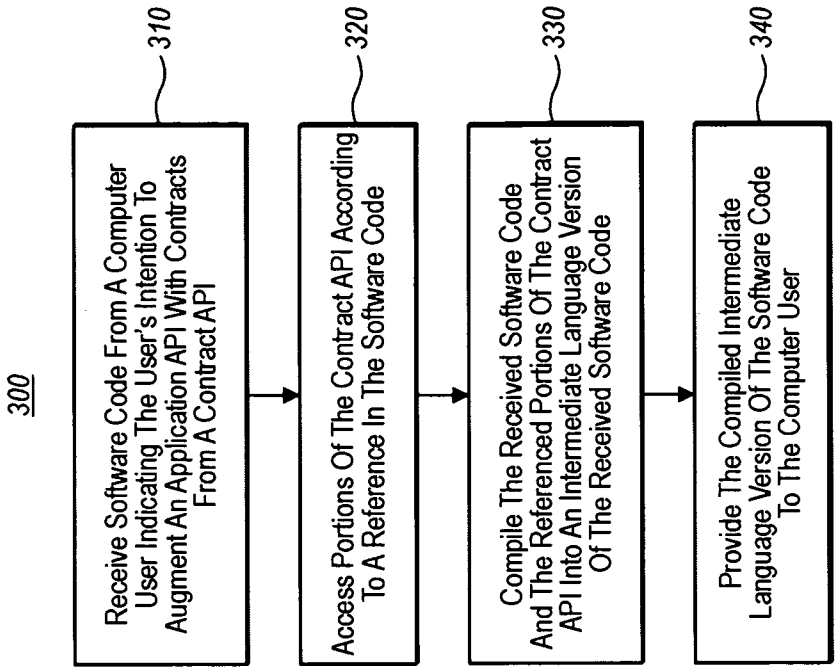
300
Receive Software Code From A Computer User Indicating The User's Intention To Augment An Application API With Contracts From A Contract API
310
Access Portions Of The Contract API According To A Reference In The Software Code
320
Compile The Received Software Code And The Referenced Portions Of The Contract API Into An Intermediate Language Version Of The Received Software Code
330
Provide The Compiled Intermediate Language Version Of The Software Code To The Computer User
340

FIG. 3

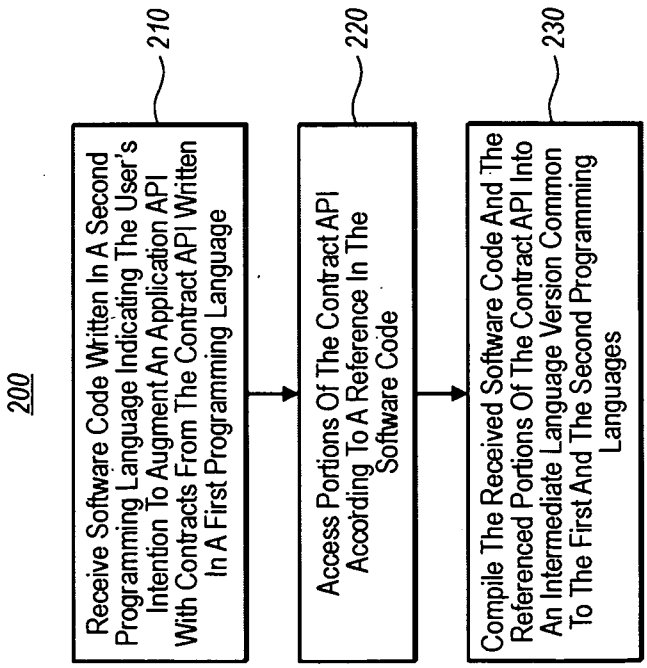
200
Receive Software Code Written In A Second Programming Language Indicating The User's Intention To Augment An Application API With Contracts From The Contract API Written In A First Programming Language
210
Access Portions Of The Contract API According To A Reference In The Software Code
220
Compile The Received Software Code And The Referenced Portions Of The Contract API Into An Intermediate Language Version Common To The First And The Second Programming Languages
230

FIG. 2

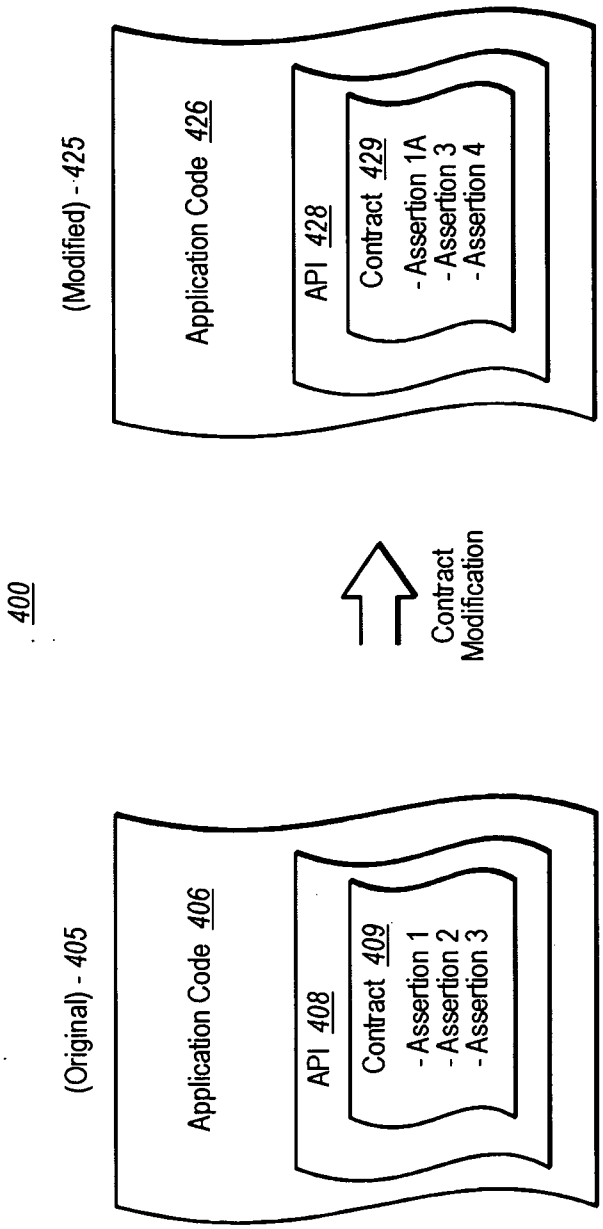
400
(Original) - 405
Application Code 406
API 408
Contract 409
- Assertion 1
- Assertion 2
- Assertion 3
Contract Modification
(Modified) - 425
Application Code 426
API 428
Contract 429
- Assertion 1A
- Assertion 3
- Assertion 4

FIG. 4

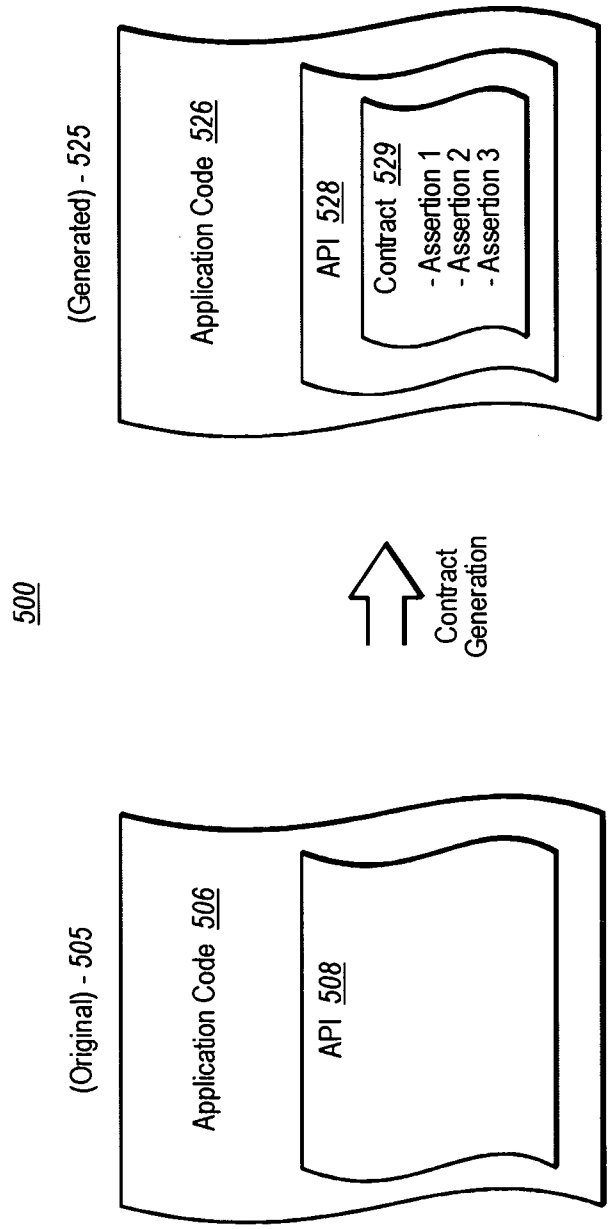
500
(Original) - 505
Application Code 506
API 508
Contract Generation
(Generated) - 525
Application Code 526
API 528
Contract 529
- Assertion 1
- Assertion 2
- Assertion 3

FIG. 5

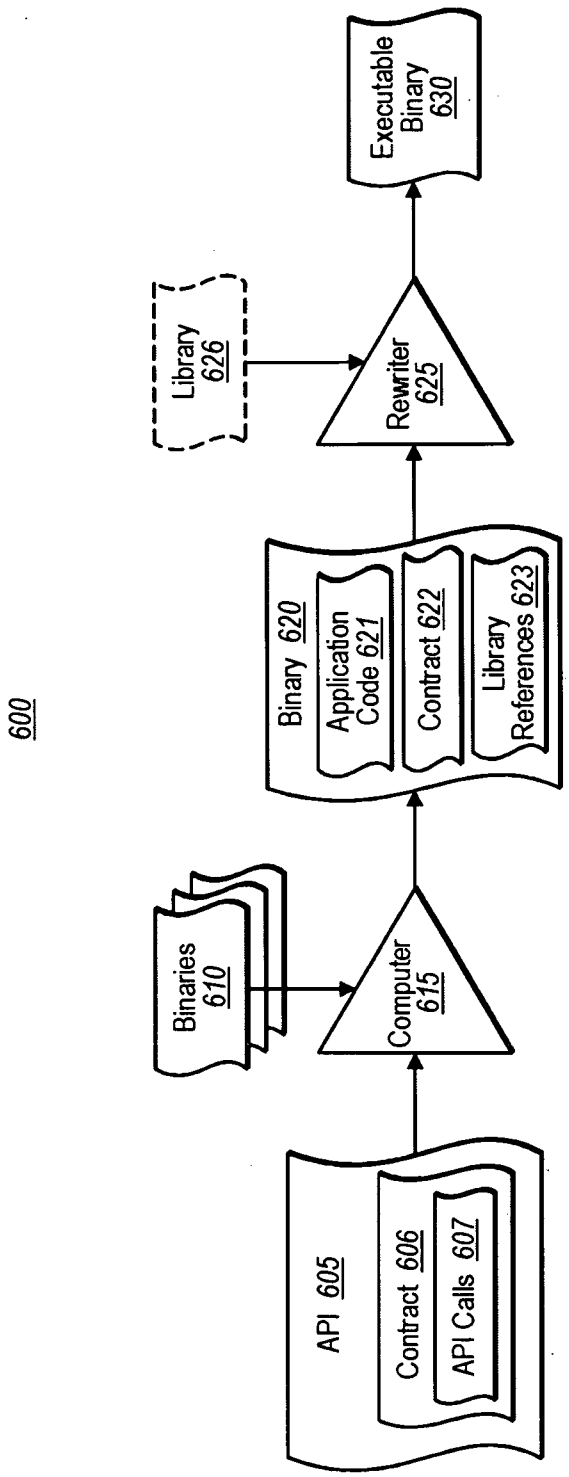
600
API 605
Contract 606
API Calls 607
Binaries 610
Computer 615
Binary 620
Application Code 621
Contract 622
Library References 623
Library 626
Rewriter 625
Executable Binary 630

FIG. 6

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2008/084994

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/30(2006.01)i, G06F 9/06(2006.01)i, G06F 11/08(2006.01)i, G06F 9/44(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 : 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), NDSL, Google

keywords: verif*, debug*, error, code, application, contract, API,library, compil*, interpret*, IL, intermediate, language

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US2007/0039010 A1 (GADRE, M.A.) 15 FEBRUARY 2007 See figures 4,5; paragraphs [3][5][14][27][31]~[33]	1-20
X	TRAN, N. et al., 'Design and Implementation of Assertions for the Common Language Infrastructure' In: Proceedings of the IEE Software Engineering, pp.329-336, 27 October 2007. Retrived from the Internet: <http://messagelab.monash.edu.au/Publications/Publication?action=download&upname=managed.pdf> See abstract; pages 1,2.	1-20
A	TRAN, N. et al., 'Managed Assertions for Components Contracts' In: Proceedings of the Integrated Design and Process Technology(IDPT-2003), June 2003. Retrived from the Internet:<http://messagelab.monash.edu.au/Publications/Publication?action=download&upname=ipdt2003.pdf> See abstract; figures 1 & 2; section IV(page 5).	1-20
A	US2005/0102656 A1 (VIEHLAND, R.E. et al.) 12 MAY 2005 See figures 2,5 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
15 JUNE 2009 (15.06.2009)	**16 JUNE 2009 (16.06.2009)**

Name and mailing address of the ISA/KR

Korean Intellectual Property Office
Government Complex-Daejeon, 139 Seonsa-ro, 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

International application No.
PCT/US2008/084994

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2004/0261065 A1 (ABRAMS, B. M. et al.) 23 DECEMBER 2004 See figures 2,3; paragraphs [8][30][34].	1-20
A	US 2004/0255268 A1 (MEIJER, E. et al.) 16 DECEMBER 2004 See figures 2,4,8; paragraphs [11][85][86]	1-20
PA	US 2008/0209395 A1 (ERNST, B.) 28 AUGUST 2008 See figure 5; paragraph [45]; claim 1.	1-20

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2008/084994

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2007-0039010 A1	15.02.2007	NONE	
US 2005-0102656 A1	12.05.2005	NONE	
US 2004-0261065 A1	23.12.2004	NONE	
US 2004/0255268 A1	16.12.2004	NONE	
US 2008/0209395 A1	28.08.2008	NONE	