

COMPUTER-EXECUTABLE MODEL REVERSE ENGINEERING METHOD AND APPARATUS

BACKGROUND OF THE DISCLOSURE

1. Field

[0001] This disclosure relates to a computer-executable model reverse engineering method and apparatus and, more specifically, to a computer-executable model reverse engineering method and apparatus that defines a structure and a rule which converts a source code of a program into a model to automatically generate a desired type of model, thereby analyzing and redesigning a program structure with ease.

2. Description of Related Art

[0002] An existing computer program may be analyzed by making a user to directly read a source code or by visualizing the source code using an effect analysis tool. The former largely depends on a technical ability of an analyst, and the latter simply and mechanically visualizes a shape of the program as it is so that it costs expenses to process the program into a desired type of data. In both methods, a product generated is only reference data and cannot be reusable.

[0003] The product expressed by a model reverse engineering is largely classified into two types. One is a modeling language devised to be used generally, which corresponds to an UML (Unified Modeling Language). The other is a dedicated expression scheme that is used by an effect analysis tool corresponding to a Change Miner. Fig. 7a illustrates constructional elements of a general modeling language UML, and Fig. 7b illustrates a Change Miner that is an effect analysis tool.

[0004] In case of the former, while it may provide a general notation so as to generate a model for a variety of SW environments and programming languages, its volume is enormous. Also, while it may be possible to make a sketchy SW model,

source code information whose notation is not provided by the UML is lost. Accordingly, it is not possible to generate an executable source code again with the model generated by the reverse engineering. In case of the latter, the product of reverse engineering provided by the effect analysis tools has a high utilization in the viewpoint that a variety of source code information can be expressed. In the latter case, however, there is provided only a simple report with which a secondary utilization cannot be implemented, including using the product of the reverse engineering as a model and performing a design based on the model, resulting in a reduced utilization. Also, since the product shows the shape of the source code as it is, it contains lots of unnecessary information.

[0005] Accordingly, there is a need to provide an apparatus that can make a desired shape of model through a reverse engineering to be reused in designing, utilize all information of a source code and then produce a source code executable in a variety of programming languages. Using that, it is possible to enhance a convenience of SW development and a quality thereof.

[0006] Korean Patent Registration No.10-0463833 discloses a component automatic conversion system and method, which automatically analyzes a source code of a system and converts the source code into a component using the analyzed information in an object-oriented computing system.

[0007] Korean Patent Registration No.10-0965706 discloses a computer apparatus that is able to reinsert codes and a code reinserting method, which provides a reinsertion and conversion of a code unit through an extendable and constructional set of a code reinsertion unit that can be executed in various steps during development, arrangement and execution of the code unit.

SUMMARY OF THE DISCLOSURE

[0008] One embodiment of the present disclosure provides a computer-executable server model reverse engineering method and apparatus, capable of converting an existing source code into a desired type of model to perform an analysis, a structural understanding and a redesigning of a program with ease.

[0009] Another embodiment of the present disclosure provides a computer-executable model reverse engineering method and apparatus, capable of expressing all information of a source code in a model without any loss of information and confirming all contents of the source code using the model only.

[0010] Yet another embodiment of the present disclosure provides a computer-executable model reverse engineering method and apparatus, capable of converting a source code into an expression of an abstracted format to abstractly expressing all information without being restricted to the programming language with which a source code is prepared.

[0011] Yet another embodiment of the present disclosure provides a computer-executable model reverse engineering method and apparatus, capable of generating a program that performs the same function with a variety of programming language using a model.

[0012]

[0013] According to embodiments of the present disclosure, there is provided a computer-executable model reverse engineering method, including (a) receiving a source code; (b) receiving a conversion rule to convert the source code into a goal model; (c) parsing the source code using a parser related to the source code to store parsing data having a specific material structure in a memory; (d) converting the stored parsing data into a meta model; and (e) converting the meta model into a goal model according to the received conversion rule.

[0014] According to one embodiment, the method may further comprise (f) converting the converted goal model in a form adapted to a UML (Unified Modeling Language) standard again to be stored in a file.

[0015] According to one embodiment, the step (b) may comprise receiving a conversion rule configured of a source conversion rule, a model conversion rule, a naming rule and a relationship rule. Here, the step (e) may comprise designating and converting a specific element of the source code or a specific element of the goal model according to the conversion rule.

[0016] According to one embodiment, the step (c) may comprise parsing the source

code into data having different material structures depending on programming language types of the source code.

[0017] According to one embodiment, the step (d) may comprise converting the parsing data into a meta model showing all elements of the source code in a specific material structure regardless of the source code types and the programming language types.

[0018] According to one embodiment, the method may further comprise (g) excluding a source code that is not concerned so as not to be converted according to the conversion rule.

[0019] According to one embodiment, the method may further comprise (h) receiving a separate conversion rule for source code information that cannot be expressed in the UML. In another embodiment, the step (h) may further comprise generating a message in at least one of a message lifeline, an exception lifeline, a validation lifeline or a code lifeline, with respect to the source code information that cannot be expressed in the UML.

[0020] According to one embodiment, the method may further comprise verifying whether all information of the source code is included in the goal model.

[0021] According to one embodiment, the step (e) may comprise (e-1) converting the meta model into a source model according to the received conversion rule, (e-2) receiving a model-to-model conversion rule to convert the source model into the goal model, and (e-3) converting the source model into the goal model according to the model-to-model conversion rule.

[0022] According to one embodiment, the method may further comprise (i) receiving the goal model to perform a configuration to generate a source code, (j) reading elements of the goal model to confirm whether the elements are used to generate the source code, (k) validating the model elements used to generate the source code according to a validation rule that the elements have, and (l) generating the source code according to a result of the validation.

[0023] According to one embodiment, the step (i) may comprise

[0024] registering elements of the goal model and conversion rules for respective elements and designating an execution condition of the conversion rule to perform the configuration.

[0025] According to embodiments of the present disclosure, there is provided a model reverse engineering apparatus, comprising a memory configured to store a model reverse engineering available program therein, and a processor configured to execute the program, wherein the processor parses a received source code into parsing data having a specific material structure using a parser, converts the parsed data into a meta model, converts the meta model into a goal model according to a received conversion rule and generates a source code using the converted goal model.

[0026] According to one embodiment, the processor may receive a source code and a conversion rule to convert the source code into a goal model, and stores parsing data having a parsed specific material structure in a memory. In another embodiment, the processor may parse the received source code into data having different types of material structures depending on programming language types of the source code.

[0027] According to one embodiment, the processor converts the parsing data into the meta model showing all elements of the source code in a specific material structure regardless of the source code types and the programming language types. In another embodiment, the processor may receive a separate conversion rule for source code information that cannot be expressed in the UML.

[0028] According to one embodiment, the processor may convert the source model into the goal model according to a model-to-model conversion rule which converts the source model received into the goal model. In another embodiment, the processor may receive a model-to-model conversion rule that is configured of a qualified name of the source model and a qualified name of the goal model.

[0029] According to one embodiment, the processor may receive the goal model to perform a configuration to generate a source code, read elements of the goal model to confirm whether the elements are source code generation target elements, verify the generation target model elements according to a validation rule that the elements

have, and generate a source code according to a result of the validation verification. In another embodiment, the processor may register elements of the goal model and conversion rules for respective elements and designates an execution condition of the conversion rule to perform the configuration.

[0030] According to embodiments of the present disclosure, there is provided a computer-readable storage medium in which a program for executing a model reverse engineering method in a computer is stored, the method comprising (a) receiving a source code, (b) receiving a conversion rule to convert the source code into a goal model, (c) parsing the source code using a parser related to the source code to store parsing data having a specific material structure in a memory, (d) converting the stored parsing data into a meta model, and (e) converting the meta model into a goal model according to the received conversion rule.

[0031]

[0032] The disclosed technology may have following effects. However, since it does not mean that a specific embodiment should include all or only the effects, it should not be understood that the scope of the disclosed technology is limited thereto.

[0033] According to one embodiment of the present disclosure, a computer-executable server model reverse engineering method and apparatus is capable of converting an existing source code into a desired type of model to perform an analysis, a structural understanding and a redesigning of a program with ease.

[0034] According to another embodiment of the present disclosure, a computer-executable model reverse engineering method and apparatus is capable of expressing all information of a source code in a model without any loss of information and confirming all contents of the source code using the model only.

[0035] According to yet another embodiment of the present disclosure, a computer-executable model reverse engineering method and apparatus is capable of converting a source code into an expression of an abstracted format to abstractly expressing all information without being restricted to the programming language with which a source code is prepared.

[0036] According to yet another embodiment of the present disclosure, a computer-

executable model reverse engineering method and apparatus is capable of generating a program that performs the same function with a variety of programming language using a model.

[0037]

BRIEF DESCRIPTION OF THE DRAWINGS

[0038] FIG. 1 is a block diagram illustrating a configuration of a model reverse engineering apparatus.

[0039] FIG. 2 is a block diagram illustrating a configuration of a model reverse engineering program executor in FIG. 1.

[0040] FIG. 3 is a view illustrating a configuration and operation of a source code model converter in FIG. 2.

[0041] FIG. 4 is a view illustrating a reverse rule received by a conversion rule receiver in FIG. 3.

[0042] FIG. 5 is a view illustrating a configuration and operation of a model-to-model converter in FIG. 2.

[0043] FIG. 6 is a view illustrating a code series expression reverse engineering method according to a source code model converter in FIG. 2.

[0044] FIG. 7 is a view illustrating a code generator configuration and a method for generating code in FIG. 2.

[0045] FIG. 8 is a view illustrating elements of a general modeling language UML and a Change Miner that is an effect analysis tool.

DETAILED DESCRIPTION OF THE DISCLOSURE

[0046] Explanation of the present disclosure is merely an embodiment for structural or functional explanation, so the scope of the present disclosure should not be

construed to be limited to the embodiments explained in the embodiment. That is, since the embodiments may be implemented in several forms without departing from the characteristics thereof, it should also be understood that the described embodiments are not limited by any of the details of the foregoing description, unless otherwise specified, but rather should be construed broadly within its scope as defined in the appended claims. Therefore, various changes and modifications that fall within the scope of the claims, or equivalents of such scope are therefore intended to be embraced by the appended claims.

[0047] Meanwhile, terms described in the present disclosure may be understood as follows.

[0048] While terms such as “first” and “second,” etc., may be used to describe various components, such components must not be understood as being limited to the above terms. The above terms are used to distinguish one component from another. For example, a first component may be referred to as a second component without departing from the scope of rights of the present disclosure, and likewise a second component may be referred to as a first component.

[0049] It will be understood that when an element is referred to as being “connected to” another element, it can be directly connected to the other element or intervening elements may also be present. In contrast, when an element is referred to as being “directly connected to” another element, no intervening elements are present. In addition, unless explicitly described to the contrary, the word “comprise” and variations such as “comprises” or “comprising,” will be understood to imply the inclusion of stated elements but not the exclusion of any other elements. Meanwhile, other expressions describing relationships between components such as “between”, “immediately between” or “adjacent to” and “directly adjacent to” may be construed similarly.

[0050] Singular forms “a”, “an” and “the” in the present disclosure are intended to include the plural forms as well, unless the context clearly indicates otherwise. It will be further understood that terms such as “including” or “having,” etc., are intended to indicate the existence of the features, numbers, operations, actions, components, parts, or combinations thereof disclosed in the specification, and are not intended to

preclude the possibility that one or more other features, numbers, operations, actions, components, parts, or combinations thereof may exist or may be added.

[0051] Identification letters (e.g., a, b, c, etc.) in respective steps are used for the sake of explanation and do not describe order of respective steps. The respective steps may be changed from a mentioned order unless specifically mentioned in context. Namely, respective steps may be performed in the same order as described, may be substantially simultaneously performed, or may be performed in reverse order.

[0052] The present disclosure may be embodied in a computer readable recording medium and in computer readable codes, and the computer readable recording medium may include any kind of recording device in which computer system readable data are stored. Examples of the computer readable recording medium may be ROM, RAM, CD-ROM, magnetic tape, floppy disk, optical data storage device, etc., including a medium embodied in a form of carrier wave (for example, transmission over Internet). Further, the computer readable recording medium may be distributed in computer systems connected together over network, so that computer readable codes are stored and implemented in a distribution scheme.

[0053] The terms used in the present application are merely used to describe particular embodiments, and are not intended to limit the present disclosure. Unless otherwise defined, all terms used herein, including technical or scientific terms, have the same meanings as those generally understood by those with ordinary knowledge in the field of art to which the present disclosure belongs. Such terms as those defined in a generally used dictionary are to be interpreted to have the meanings equal to the contextual meanings in the relevant field of art, and are not to be interpreted to have ideal or excessively formal meanings unless clearly defined in the present application.

[0054]

[0055] FIG. 1 is a block diagram illustrating a configuration of a model reverse engineering apparatus.

[0056] Referring to FIG. 1, a model reverse engineering apparatus 10 according to

an embodiment of the disclosure may include a processor 110, a non-volatile memory 120, a non-volatile memory 130, a network interface 140, an input device 150 and an output device 160.

[0057] The processor 110 The processor 210 may include a model reverse engineering program executor 112 and a memory manager 114. The model reverse engineering program executor 112 executes a model reverse engineering program to execute a model generation and conversion procedure through a source code reverse engineering. The memory manager 214 manages memory areas in the non-volatile memory 120 and volatile memory 130 that are read and written by the model reverse engineering program executor 112.

[0058] The non-volatile memory 120 may be configured of a solid state disk SSD or a hard disk drive HDD, and used to store a model reverse engineering program and data needed to execute the model reverse engineering program. The volatile memory 130 is used to temporarily store the model reverse engineering program stored in the non-volatile memory 120.

[0059] The network interface 140 may include an environment to connect the model reverse engineering apparatus 10 to an external server, for example, an adaptor for a local area network LAN communication.

[0060] The input device 150 may include an environment to receive a user input, including a mouse, a track ball, a touch pad, a graphic tablet, a scanner, a touch screen, a keyboard or a pointing device, for example. The output device 160 may include an environment to output to a user specific information (for example, a meta model, a goal model or a source code), which may include a monitor or a touch screen, for example. In an embodiment, the input device 150 and the output device 160 may be connected each other through a remote connection.

[0061]

[0062] FIG. 2 is a block diagram illustrating a configuration of a model reverse engineering program executor in FIG. 1.

[0063] Referring to FIG. 2, a model reverse engineering program executor 112 according to an embodiment of the present disclosure includes a source code model

converter 210, a model-to-model converter 220 and a code generator 230. The model reverse engineering program executor 112 is an apparatus that can generate and convert a model through a source code reverse engineering, capable of receiving a source code or a source model to generate and convert a model of a user's desired form, and designating flexible and diverse verification rules when preparing and applying conversion rules. Further, the model reverse engineering program executor 112 stores all information of a source code in a model to be converted into a programming language. The source code model converter 210, the model-to-model converter 220 and the code generator 230 will be described below with reference to FIGS. 3 to 7.

[0064]

[0065] FIG. 3 is a view illustrating a configuration and operation of a source code model converter in FIG. 2.

[0066] Referring to FIG. 3, a model converter 210 according to an embodiment of the present disclosure includes a conversion rule receiving unit 212, a parser 214, a meta model converting unit 216 and a model converting unit 218. When the source code is inputted, the conversion rule receiving unit 212 may receive a conversion rule, the parser 214 may parse the source code to store the data in a memory, and the meta model converting unit 216 may fetch the data stored in the memory to store source code information in a meta model which is independent from the program language. The model converting unit 218 may receive the conversion rule and convert the meta model into a goal model, calculating a model which meets a UML standard. The source code is a code to be converted into a UML model, which may be one of sources that are prepared of diverse programming languages, for example, Java, C and COBOL.

[0067] The conversion rule receiving unit 212 may receive a conversion rule to convert the received source code into a goal model. When there is a naming convention in the source code, the conversion rule receiving unit 212 may understand the convention and receive a conversion rule including a configuration of a conversion goal model. The conversion rule receiving unit 212 may receive the conversion rule including information on the inputted source code, information on

goal model to be converted, information on the naming convention and information on a relationship. The conversion rule will be described below with reference to FIG. 4.

[0068] The parser 214 may parse the source code to store the parsing data having a specific material structure in a memory. That is, the parser may parse the source code to upload the data having a material structure of a tree format on the memory. The parser 214 may have different forms depending on what programming language the source code has. The parsing is a step of a procedure where a compiler or a translator translates a source code into a machine language in a computer, which is a procedure to analyze a grammatical construction or syntax of each sentence. That is, it is a task to receive a string of token appearing in a source program to configure it as a parse tree, which is correct grammatically. Accordingly, the parser 214 may have different forms depending on the sort of the source code, and parse the source code with data having different material structure depending on a sort of a programming language of the source code.

[0069] The meta model converting unit 216 may convert the stored parsing data into a meta model. The meta model converting unit 216 may convert the data prepared by a variety of source codes into a meta model material structure. The meta model converting unit 216 enables the model reverse engineering apparatus 10 to generate the UML model having the same quality and form when changing the parser 214 regardless of what programming language the source code has. Accordingly, the meta model converting unit 216 may convert the data into a meta model showing all elements of the source code as a specific material structure regardless of sorts of the source code and programming languages.

[0070] The model converting unit 218 may convert the meta model converted by the meta model converting unit 216 into a goal model according to the conversion rule received by the conversion rule receiving unit 212. The model converting unit 218 may convert the converted goal model into a form meeting the UML standard again to write it on a file. The model converting unit 218 may receive a separate conversion rule for a syntax of a simple expression series which is not expressed as a UML model, to convert it into a UML model. The model converted through the

model converting unit 218 may be stored in an UML model form, and it may be confirmed through a UML editing software such as the IBM Rational Software Architect. The model converting unit 218 may exclude the source code that is not concerned from objects to be converted according to the conversion rule, and designate specific elements of the source code or specific elements of the goal model according to the conversion rule, to be converted. The specific elements of the source code are object items to be converted into the goal model among components of the source code, which may correspond to a Package, a Class, an Attribute and an Operations of the source code, and specific elements of the goal model are conversion result items of the converted goal model, which may correspond to a Package, an Interface, a Class, an Attribute, an Operation, a Collaboration, a Sequence Diagram, a Class Diagram, a Code Lifeline, a Message Lifeline, a Validation lifeline, an Exception Lifeline. Here, the source code that is not concerned is elements that are not designated as specific elements, which may be selected by a user. For example, when a user selects a comment or uncalled information as the source code that is not concerned, the related information may be discarded. Further, the model converting unit 218 may verify whether all information of the source code is included in the goal model.

[0071]

[0072] FIG. 4 is a view illustrating a reverse rule received by a conversion rule receiver in FIG. 3.

[0073] In FIG. 4, a conversion rule may be configured of information on input source code (source conversion rule) 111, information on converted goal model (model conversion rule) 112, information on a naming convention of converted model element 113, and information on relationship among model elements.

[0074] The source conversion rule 40 may designate a source to be converted into a model using a regular expression. The source may include the regular expression including a qualified name, an attribute, a function and a source sort. The qualified name is a regular expression designating a reverse engineering target source. The source code matched to the regular expression defined in the items among the class name including packages or folders only is converted into a model. When the

package or the class name has the naming convention, the model converting unit 218 may classify the source and convert it into a model using the items, and exclude the source code that is not concerned from the reverse engineering target. The attribute is a regular expression indicating a function included in the class or file. The model converting unit 218 may convert into a model a class matched to the qualified name or all or specific functions among functions included in the file only. When the source is not written in java but in c or other languages, and the file name is identified with a filename extension only, the model converting unit 218 may put a value corresponding to the filename extension in the item to convert a specific sort of source only into a model.

[0075] The model conversion rule 420 may show constructional information for the goal model. An ID only is a value to identify an element to be converted into the model, which may be given by a user. The model element is an item to designate the element of the model to be converted, designating a Package, an Interface, a Class, an Attribute, an Operation, a Collaboration, a Sequence Diagram, a Class Diagram, a Code Lifeline, a Message Lifeline, a Validation Lifetime, and an Exception Lifetime. The source element is an item to designate the element of the source to be converted into the model, which may designate a Package, a Class, an Attribute, and an Operation. When the source element is not designated, elements that are not to be in the source and to be in the goal model may be inserted. A Stereotype may have a stereo written so as to be designated in the converted model element. An Upper Element ID is used to designate an upper element ID in which the corresponding element is included.

[0076] The naming convention 430 may indicate information on a naming convention to name the model element to be converted. A Source Prefix is a prefix of the source to be removed when converting into a model. A Model Prefix is a prefix to be included in a model when converting into a model. A Source Prefix is a prefix of a source to be removed when converting into a model. A Model Prefix is a prefix to be included in the model element name when converting into a model. A Naming Convention is a naming convention to be applied when converting a source name corresponding to a model element into a model naming. Items such as None, Korean, Custom may be designated, wherein the None uses the source name as a

model name, the Korean converts an English name of the source into a Korean name using Korean name-English name dictionary inputted previously to use it as a model name, and the Custom prepares a source program to designate in case that it is not a simple English-Korean conversion. The Naming Class indicates Java class information used when determining a model name in case that an item is designated as the Custom in the naming convention.

[0077] A relationship conversion rule 440 may set a relationship between the converted model elements when it is needed even though not existing in the source code. A Type indicates a relationship type such as an association, a generalization, a realization and a dependency. The relationship element ID indicates a model element ID to set a relationship.

[0078]

[0079] FIG. 5 is a view illustrating a configuration and operation of a model-to-model converter in FIG. 2.

[0080] In FIG. 5a, a model-to-model converter 220 includes a model-to-model conversion rule receiving unit 222 and a model-to-model converting unit 224.

[0081] In one embodiment, the source code model converter 210 may convert an inputted source code into a source model of a UML form according to a received conversion rule. The model-to-model converter 220 may receive a model-to-model conversion rule and convert the source model into a final goal model according to a model-to-model conversion rule. The model-to-model conversion rule is described in detail in FIG. 5b.

[0082] In more detail, the model-to-model converter 220 may move and reconstruct positions of a package and a class of the source model according to the model-to-model conversion rule. That is, the model-to-model converter 220 may separate the package in case of many classes of the source model package, and the class in case of many operations of the class. In the same manner, the model-to-model converter 220 may integrate the package in case of little classes of the source model package and the class in case of little operations of the class. In other word, when it is needed to integrate and reorganize the package or class of the converted source

model in the meta model according to the conversion rule, the model-to-model converter 220 may convert the source model into the goal model according to the model-to-model conversion rule.

[0083] Conversion rule receiving unit

[0084] FIG. 5b is a view illustrating a model-to-model conversion rule.

[0085] In FIG. 5b, a model-to-model conversion rule is configured of a source model Qualifiedname 510 and a goal model Qualifiedname 520. The source model Qualifiedname 510 is a qualified name of the source model element to convert into a goal model, and all low model elements of the source model element indicated by it are copied onto model elements indicated by a goal model Qualifiedname 520. The source model elements that are not same in the source model Qualifiedname 510 and same in the goal model Qualifiedname 520 are integrated as one goal model element when being copied onto the goal model. On the contrary, the source model elements that are same in the high element name of the source model Qualifiedname 510 and not same in the high element name of the model element Qualifiedname 520 are separately copied. The goal model Qualifiedname 520 indicates the Qualifiedname 520 of the goal model element onto which an element indicated by the source model Qualifiedname 510 is copied.

[0086]

[0087]

[0088] FIG. 6 is a view illustrating a code series expression reverse engineering method according to a source code model converter in FIG. 2.

[0089] In FIG. 6a, a source code model converter 210 may first receive a conversion rule of a regular expression type to convert the expression, and select and parse the source code to be converted to store parsed data in a memory. Next, the source code model converter 210 may fetch data stored in the memory to store source code information in a model independent from a program language through the meta model converter 216. The model converter 218 may receive the received conversion rule to convert the data converted by the meta model converting unit 216 according to the conversion rule. The converted data may be converted in a form

that meets the UML standard again and stored in in a file.

[0090] When converting the meta model into the UML model according to the conversion rule, a syntax of a simple expression series that is not expressed in the UML model may be converted by receiving a separate conversion rule. A detailed description for the conversion rule is given as follows.

[0091] 1) Common things: A logical expression that is not expressed as a function call or a combined fragment among internal logics of a function may generate a synchronous message in a code series lifeline suitable for purpose of use, such as a code, a validation, an exception, and a message and may be expressed a logical code in a document column.

[0092] 2) Message: A syntax generating a message may generate a message in a message lifeline. Whether to generate a message in the message lifeline is determined depending on whether there is a function matched to a regular expression that is propositioned by a specific function generating a message.

[0093] 3) Exception message: A syntax causing an error message may generate a message in an exception lifeline. Whether to generate a message in the exception lifeline is determined depending on whether there is a function matched to a regular expression that is propositioned by a specific function generating an error.

[0094] 4) Validation message: A syntax causing an error while validating an input parameter such as an essential input value checking, may generate a message in a validation lifeline. Whether to generate a message in the validation lifeline is determined depending on whether there is a function matched to a regular expression that is propositioned by a function and a syntax pattern corresponding to an input value validation.

[0095] 5) Code message: Other logics and comments that are not converted into an operation call, a combined fragment may generate a message in a code lifeline. The internal logic before the operation expresses a call as a synchronous message or generate a combined fragment, is wholly expressed in a code lifeline. An operation call syntax of the source code and an original syntax corresponding to the combined fragment are also expressed in a code lifeline immediately before the

modeling element appears (expressing the original source code wholly in order to validate a model conversion matching).

[0096] 6) Parameter declaration and initialization: A parameter name may be written in Korean using a prior information inputted in a form of English word name-Korean word name and changed in the form to be matched to a logic expression. As to Type name, a basic type converts a Type name according to a mapping table produced by mapping a source type and a model type, and a type other than the basic type converts the Type name into a type name converted into a model according to an arrangement notation defined in a logic expression according to whether it is arranged. The parameter initialization means to convert the syntax as it is.

[0097] 7) Conversion of function: The syntax is converted into the regular expression according to a regular expression rule defined in advance. Example) `strcpy\((parameter name),` in case of meeting the left-hand rule of `(parameter name) \)` $\rightarrow \$\{1\} = \$\{2\}$, the syntax is converted into a right-hand expression. Accordingly, the syntax such as `strcpy(a,b)` is converted into `a = b` syntax.

[0098] FIG. 6b illustrates how to express an expression of a simple sequence series that cannot be indicated as the UML expression in a sequence diagram. The source code model converter 210 generates an arbitrary lifeline for a code series expression such as a code, a validation, an exception, and a message in order to indicate a code series expression in a sequence diagram, draws a message with the lifeline and stores document contents of the message in the expression.

[0099]

[0100] FIG. 7 is a view illustrating a code generator configuration and a method for generating code in FIG. 2.

[0101] In FIG. 7a, a code generator 230 may include a Code Generator Configuration 232, a Model Scanner 234, a Model Validation 236 and a Code Maker 238. The code generator 230 is a module to generate a source code of a variety of programming languages based on a model. The code generator 230 may generate a program source code based on a conversion rule that is mapped for each of the

model element and logic expression. The code generator 230 may generate source codes of a variety of languages using a model.

[0102] In FIG. 7b, the code generator configuration 232 may perform a configuration to prepare a conversion rule to describe a desired source code (S710). First, one conversion rule project is generated to register a model element to be converted in the project and a conversion rule is generated and registered for each element. When registering the conversion rule, a conversion conduct of the conversion rule is described and an execution condition of the conversion rule is designated. When the conversion rule is completely registered for all elements, a structure of the source code to be prepared is defined.

[0103] When it is desired to generate a code for a model prepared after a software model is completely prepared or while the software model is being prepared, what is needed is to select a conversion rule project for a desired model and then perform a source code generation function. Regardless of the entire model or a partial model, it is possible to generate the source code.

[0104] When performing the source code generation function, a Model Scanner 234 is executed in the code generator 230. The model scanner 234 may continuously read elements from the most top element in a top-down scheme to confirm whether a relevant element is a source code generation target element. In case of the source code generation target element, the model scanner 234 transmits the relevant element to the model validation 236 (S720).

[0105] The model validation 236 may receive the source code generation target element and perform a validation according to the validation rule for the relevant element (S730). When there is an error after completing the validation, the model validation 236 may transfer the error to the model scanner 234 and the model scanner 234 may store the validation result.

[0106] When all model elements are completely validated and there is an error, the model scanner 234 may output the result of all validation on a screen (S740). When there is not an error as a result of the validation, a Code Maker 238 may generate a source code according to the conversion rule (S750).

[0107]

[0108] Although this document provides descriptions of preferred embodiments of the present disclosure, it would be understood by those skilled in the art that the present disclosure can be modified or changed in various ways without departing from the technical principles and scope defined by the appended claims.

[0109] According to the present disclosure, there is provided a computer-executable model reverse engineering method and apparatus that defines a structure and a rule which converts a source code of a program into a model to automatically generate a desired type of model, thereby analyzing and redesigning a program structure with ease.

WHAT IS CLAIMED IS:

1. A computer-executable model reverse engineering method, comprising:
 - (a) receiving a source code;
 - (b) receiving a conversion rule to convert the source code into a goal model;
 - (c) parsing the source code using a parser related to the source code to store parsing data having a specific material structure in a memory;
 - (d) converting the stored parsing data into a meta model; and
 - (e) converting the meta model into a goal model according to the received conversion rule.
2. The method of claim 1, further comprising (f) converting the converted goal model in a form adapted to a UML(Unified Modeling Language) standard again to be stored in a file.
3. The method of claim 1, wherein the step (b) comprises receiving a conversion rule configured of a source conversion rule, a model conversion rule, a naming rule and a relationship rule.
4. The method of claim 3, wherein the step (e) comprises designating and converting a specific element of the source code or a specific element of the goal model according to the conversion rule.
5. The method of claim 1, wherein the step (c) comprises parsing the source code into data having different material structures depending on programming language types of the source code.

6. The method of claim 1, wherein the step (d) comprises converting the parsing data into a meta model showing all elements of the source code in a specific material structure regardless of the source code types and the programming language types.

7. The method of claim 1, further comprising (g) excluding a source code that is not concerned so as not to be converted according to the conversion rule.

8. The method of claim 1, further comprising (h) receiving a separate conversion rule for source code information that cannot be expressed in the UML.

9. The method of claim 8, wherein the step (h) further comprises generating a message in at least one of a message lifeline, an exception lifeline, a validation lifeline or a code lifeline, with respect to the source code information that cannot be expressed in the UML.

10. The method of claim 8, wherein the step (h) further comprises generating a message in at least one of a message lifeline, an exception lifeline, a validation lifeline or a code lifeline, with respect to the source code information that cannot be expressed in the UML.

11. The method of claim 1, wherein the step (e) comprises:

(e-1) converting the meta model into a source model according to the received conversion rule;

(e-2) receiving a model-to-model conversion rule to convert the source model into the goal model; and

(e-3) converting the source model into the goal model according to the

model-to-model conversion rule.

12. The method of claim 1, further comprising:

(i) receiving the goal model to perform a configuration to generate a source code;

(j) reading elements of the goal model to confirm whether the elements are used to generate the source code;

(k) validating the model elements used to generate the source code according to a validation rule that the elements have; and

(l) generating the source code according to a result of the validation.

13. The method of claim 12, wherein the step (i) comprises registering elements of the goal model and conversion rules for respective elements and designating an execution condition of the conversion rule to perform the configuration.

14. A model reverse engineering apparatus, comprising:

a memory configured to store a model reverse engineering available program therein; and

a processor configured to execute the program,

wherein the processor parses a received source code into parsing data having a specific material structure using a parser, converts the parsed data into a meta model, converts the meta model into a goal model according to a received conversion rule and generates a source code using the converted goal model.

15. The apparatus of claim 14, wherein the processor receives a source code

and a conversion rule to convert the source code into a goal model, and stores parsing data having a parsed specific material structure in a memory.

16. The apparatus of claim 15, wherein the processor parses the received source code into data having different types of material structures depending on programming language types of the source code.

17. The apparatus of claim 15, wherein the processor converts the parsing data into the meta model showing all elements of the source code in a specific material structure regardless of the source code types and the programming language types.

18. The apparatus of claim 15, wherein the processor receives a separate conversion rule for source code information that cannot be expressed in the UML.

19. The apparatus of claim 14, wherein the processor converts the source model into the goal model according to a model-to-model conversion rule which converts the source model received into the goal model.

20. The apparatus of claim 19, wherein the processor receives a model-to-model conversion rule that is configured of a qualified name of the source model and a qualified name of the goal model.

21. The apparatus of claim 14, wherein the processor receives the goal model to perform a configuration to generate a source code, reads elements of the goal model to confirm whether the elements are source code generation target elements, verifies the generation target model elements according to a validation

rule that the elements have, and generates a source code according to a result of the validation verification.

22. The apparatus of claim 21, wherein the processor registers elements of the goal model and conversion rules for respective elements and designates an execution condition of the conversion rule to perform the configuration.

23. A computer-readable storage medium in which a program for executing a model reverse engineering method in a computer is stored, the method comprising:

- (a) receiving a source code;
- (b) receiving a conversion rule to convert the source code into a goal model;
- (c) parsing the source code using a parser related to the source code to store parsing data having a specific material structure in a memory;
- (d) converting the stored parsing data into a meta model; and
- (e) converting the meta model into a goal model according to the received conversion rule.