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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
27 October 2022 (27.10.2022)

WIPO | PCT

(10) International Publication Number
WO 2022/221927 A1

(51) International Patent Classification:
G06F 9/455 (2018.01) *G06F 8/40* (2018.01)
G06F 8/35 (2018.01) *G06N 3/00* (2006.01)

(21) International Application Number:
PCT/AU2022/050373

(22) International Filing Date:
22 April 2022 (22.04.2022)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
2021901195 22 April 2021 (22.04.2021) AU

(71) Applicant: E & K ESCOTT HOLDINGS PTY LTD [AU/AU]; C/- Michael Buck IP, PO Box 78, Red Hill, Queensland 4059 (AU).

(72) Inventor: ESCOTT, Eban Peter; C/- Michael Buck IP, PO Box 78, Red Hill, 4059 (AU).

(74) Agent: MICHAEL BUCK IP; PO Box 78, Red Hill, Queensland 4059 (AU).

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

(54) Title: A METHOD FOR IMPROVED CODE GENERATION BY ARTIFICIALLY INTELLIGENT AGENTS

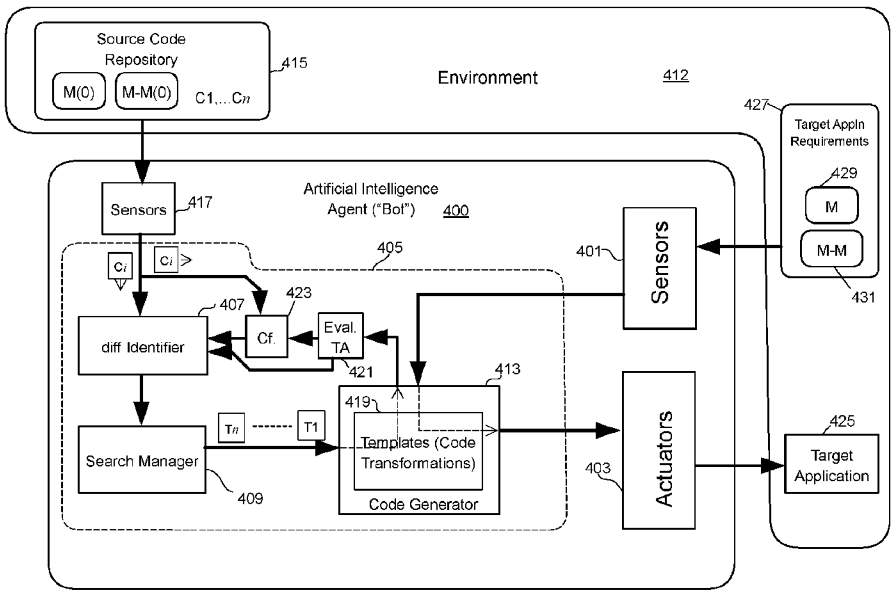


FIG. 4

(57) Abstract: A computer-implemented method for producing code generation templates. The method includes receiving a series of commits, each of the commits in the series of commits including a version of a reference application, each version of the reference application having a corresponding meta-model and model and identifying differences between time separated commits to thereby obtain one or more identified differences therebetween. The method searches to find transformations for transforming a previous version of the reference application to a current version of the reference application, adds the transformations to a current template, applies the current template, including the transformations, to a recent commit to thereby produce a current evaluation target application, and stores the current template for later use upon the current evaluation target application passing a comparison evaluation with a version of the reference application of the recent commit.

WO 2022/221927 A1

[Continued on next page]

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

Published:

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

A METHOD FOR IMPROVED CODE GENERATION BY ARTIFICIALLY INTELLIGENT AGENTS

TECHNICAL FIELD

The present disclosure relates to computer-implemented systems, methods, and apparatus for generating code for a specific functionality for example by use of an artificially intelligent (AI) agent or “bot”.

BACKGROUND

Any references to methods, apparatus or documents of the prior art are not to be taken as constituting any evidence or admission that they formed, or form part of the common general knowledge.

Software written for applications such as mobile applications, web-based systems, embedded systems, and enterprise applications, use software patterns to solve problems and deliver a solution. Traditionally, programming languages like Java, C++, Python and the like are used to write source code for a target application by a software developer. Writing these applications is a time intensive task and carries a high risk of error. The source code is ultimately compiled to executable code which can be read and acted upon by one or more microprocessors of a computer. Under control of the application the microprocessors process data from sensors such as keyboards, touch screen, mouse, camera and possibly industrial sensors such as temperature and pressure transducers and the like. The microprocessors act upon the received data from the sensors and operate various actuators in accordance with the application. The various actuators may include one or more display screens or almost any other kind of machine controllable actuator, such as solenoids or disk drive or solid-state drive storage controllers for example.

Human software engineers or “developers” can accomplish these tasks though much of what they do may be considered infrastructure in that it involves reusing prebuilt code. More recently, it has been known for software to be designed and built using computer implemented modelling environments whereby some of the source code for a target application can be automatically generated.

Model-based applications can be used to represent a software system either graphically, textually or with a hybrid approach. The models can be used to generate large portions of the software system, which leads to many tangible benefits.

Computer software can be created, updated and read using model-based representations. Unlike traditional source code, the model is a high-level representation of the source code and code generators, i.e. suitably programmed computers, can be used to write code from the model, instead of the code being written by a human. The models allow the modeller to express the requirements that the resultant target software application is required to meet. Graphical models, presented by way of a graphical user interface on a display screen of a computer, usually allow shapes to be rendered, moved and connected, with other elements to represent the software. Textual models like Domain-Specific Languages (DSLs) can look like natural language and are ideal for some problem scenarios. A hybrid approach uses both graphical and textual notations. A table or spreadsheet could be considered a hybrid approach and used as a model of the software application and a way for the modeller to express the intent of their software.

Model-Driven Engineering (MDE) is an advanced approach to software engineering that uses models in the software development life cycle. As an example, Figure 1 is a diagram showing a meta-model 10, model 12, XML representation of the model 14, templates 9, code library 7, code generator 16 and output application code or “Target Code” 18 which produces output 20, for example a display on a computer monitor. In the meta-model 10 entities 17 and associations 15 between entities 17 are defined. In the model layer a model 12 is stored that, in this example, includes two instances of an entity of the meta-model being Alpha 19 and Bravo 21. The two instances Alpha 19 and

Bravo 21 are related by a relationship 13 that is an instance of one of the associations 15 defined in the meta-model 10. The model 12 can be represented in a number of ways, for example it can be represented graphically as shown in Figure 1 or textually. The XML document 14 captures all of the instances of elements and associations of the model. The XML document 14 can be applied to a software application known as a code generator 16. The output from the code generator 16 comprises a target software application 18 for execution by a computer to generate an output 20. Consequently, it will be understood that modification of the model 12 and in some circumstances also of the meta-model 10 will result in modification of the target software application 18. Conversely, for a given target software application it is possible to define a corresponding model and also a corresponding meta-model.

It will therefore be realised that a model, such as model 12 of Figure 1, is a high-level representation of software application 18. The model 12 can undergo model-to-model (M2M) and model-to-text (M2T) transformations that results in some - not necessarily all - of the source code of the target application 18 being automatically generated. Some examples of well-known modelling environments include the Unified Modelling Language (UML), Business Process Modelling Notation (BPMN), and Business Process Execution Language (BPEL).

The meta-model defines what can be found in the model. For example, if the meta-model has an entity such as a class called “BirdClass”, then the modeller can add as many instances of “BirdClass” to the model as required and label those instances accordingly, e.g. “Seagull”, “Hawk”, “Bluebird” etc.

A well-known example of a model driven engineering layer hierarchy is the Meta-Object Facility (MOF) illustrated in Figure 2 in which four layers are defined as follows; real world 2, model 4, meta-model 6 and meta-meta-model 8.

Figure 3 illustrates a scenario that arises in the field of MDE software development. In this scenario a person with a need to have a software application ("target software") created, such as Enterprise Owner 29, engages a software Developer 30 to produce Enterprise Software, 31, depicted as versions 31.1,...,.31.n, that automates business processes 33, which arise in the running of the Enterprise Owner's business. The Enterprise Owner 29 and the Developer 31 have conversations and exchange emails and possibly meet face to face so that the Developer 31 forms an understanding of the requirements that are to be implemented in the Enterprise Software 31 in order to automate the business processes 33. The Developer 31 then devises an initial meta-model 35 and a corresponding initial model 37 based on the requirements from the Enterprise Owner 29 for the Enterprise Software 31.

Part of the work of the Developer 30 is to create templates 39 that are consistent with the meta-model 35 and model 37 and which are tailored to the Enterprise Owner's requirements. Once the initial meta-model 35, model 37 and templates 39 have been created they are then applied to the code generator 41 which produces an initial version V1 31.1 of the Enterprise Software. The Enterprise Software V1 31.1 is made available to the Enterprise Owner 29 for testing to see if it meets the Enterprise Owner's requirements. The Enterprise Owner 29 provides feedback to the Developer 30 and in response the Developer 30 will typically revise the templates 39. In some situations it may be that the Enterprise Owner 29 will introduce new requirements which involve incorporation of additional entities into the software specification that have not been previously discussed. In that case the Developer will typically also update the Meta-Model 35 and correspondingly the Model 37 to take into account feedback from the Enterprise Owner 29. One part of the cycle that is time consuming is the writing and updating of the templates 39.

The developer 30 may use a version management tool to store successive iterations or "commits" of models, meta-models, templates and versions of the Enterprise Software during the progressive development of the Enterprise Software.

Manually writing templates is time intensive and prone to human error. This creates a barrier to entry for enterprises seeking to use MDE. Some attempts have been made in the past to automate the updating of the templates. For example, in recent years artificial neural network (ANN) based artificial intelligence (AI) agents have been trained to generate code using Machine Learning (ML) based approaches such as that of DeepCoder, https://www.microsoft.com/en-us/research/publication/deepcoder-learning-write-programs/

There are a number of difficulties in using ML approaches. For example, to train a ML algorithm a large data set is required. To find large datasets, researchers have used open-source code repositories and the like that are available online. But these datasets have a large variation in consistency and quality making it difficult for the machine to learn. Furthermore, the code repositories usually do not have the original requirements stored alongside them so it is difficult to know the intention of the code base. Finally, ML approaches require a way to test its output as the algorithm learns. The open-source code repositories rarely have adequate tests making finding large datasets difficult. These approaches aim to solve the problem of coding with limited reference to the expressly stated models of the application owner. Consequently, they do not scale very well to larger applications.

Further, these approaches do not have explicability from the start. Decisions taken by the coding system are based in the state of the neuronal network as a whole and cannot be broken out. Deep learning explainability is a research area in its infancy.

There is a need for a method for improving code generation by artificially intelligent agents which overcomes or at least ameliorates one or more disadvantages of the prior art.

SUMMARY OF THE INVENTION

A computer-implemented method for producing code generation templates comprising:

receiving a series of commits (C_i=C_0,...,C_n), each of the commits C_1,...,C_n of the series of commits including a version of a reference application RA_1,...,RA_n, each version of the reference application RA_1,...,RA_n having a corresponding meta-model and model, commit C_0, comprising an initial model and meta model of a first version of the reference application RA_1;

for $\underline{i}$=1 to n, identifying differences between C_i and C_{i-1} to thereby obtain one or more identified differences therebetween;

searching to find one or more transformations for transforming a previous version of the reference application to a current version of the reference application,

adding the one or more transformations to a current template T_i;

applying the current template T_i, including the one or more transformations, to the last commit C_{i-1} to thereby produce a current evaluation target application ETA_i (i=1,...,n); and

storing the current template T_i for later use upon the current evaluation target application ETA_i passing a comparison evaluation with a version of the reference application of C_{i-1} ,

whereby a series of templates are ultimately stored.

The templates may be generated by an artificially intelligent agent ("Bot") in accordance with the above method so that the bot learns new versions of the reference application in an incremental fashion based on changes in consecutive versions of the reference application, which may include updates to model and meta-models corresponding to the versions of the reference application.

According to another aspect of the present invention there is provided an artificially intelligent agent ("the bot") hosted on a computer platform providing a computing environment for the bot, the bot comprising:

one or more sensors configured to read incrementally updated versions of a reference application stored in the computing environment;

one or more actuators configured to write files to the computing environment; and

a mapping arrangement responsive to the one or more sensors and coupled to the one or more actuators for control thereof,

the mapping arrangement including,

a difference Identifier subassembly configured to identify one or more differences in code of a current version of the reference application relative to an immediately earlier version of the reference application to thereby produce one or more identified differences,

a search manager subassembly responsive to the difference identifier for receiving the one or more identified differences and configured to determine transformations transforming the immediately earlier version of the reference application into the latest version of the reference application wherein the search manager encodes the transformations in a template;

a code generation subassembly arranged to process the template with reference to a model corresponding to the immediately earlier version of the reference application to thereby generate an evaluation target application; and

a comparison subassembly arranged to compare the evaluation target application to the latest version of the reference application;

wherein upon output from the comparison subassembly indicating that the evaluation target application passes comparison requirements, the template is retained for subsequent use by the code generator subassembly.

A computer-implemented method for producing code generation templates comprising:

receiving a series of commits, each of the commits in the series of commits including a version of a reference application, each version of the reference application having a corresponding meta-model and model;

identifying differences between time separated commits to thereby obtain one or more identified differences therebetween;

searching to find one or more transformations for transforming a previous version of the reference application to a current version of the reference application,

adding the one or more transformations to a current template,

applying the current template, including the one or more transformations, to a recent commit to thereby produce a current evaluation target application; and

storing the current template for later use upon the current evaluation target application passing a comparison evaluation with a version of the reference application of the recent commit,

whereby a series of templates are ultimately stored.

In an embodiment the series of commits comprises commits $C_i = C_0, \ldots, C_n$, wherein each of the commits $C_1, \ldots, C_n$ of the series of commits includes a version of a reference application $RA_1, \ldots, RA_n$.

In an embodiment each version of the reference application commit C_0, comprises an initial model and initial meta model of a first version of the reference application RA_1.

In an embodiment the method includes checking if a new commit of the series of commits has been received in a source code repository.

In an embodiment the method may include updating a current template with folder and file structure information based on the new commit.

In an embodiment, for each identified difference of the one or more identified differences, the computer implemented method includes the steps:

searching to find a transformation in respect of the identified difference,

adding the transformation to a current template, and

applying the current template, including the transformations for each of the identified differences, to a recent commit to thereby produce a current evaluation target application,

whereby a series of templates are ultimately stored containing incremental transformations.

In an embodiment the method includes making a number of candidate templates and selecting a best one by making a target application from each candidate template and then evaluating the target application based on how many lines of code in the target application match lines of code in the reference application wherein the candidate template that generated the target application that has the best match is deemed the template.

In an embodiment the method includes evaluating the current Evaluation Target Application relative to the current reference application of the current commit by checking if the Evaluation Target Application compiles.

In an embodiment evaluating the current Evaluation Target Application comprises determining if the number of lines of code in the current Evaluation Target Application match the number of lines in the current Reference Application.

In an embodiment evaluating the current Evaluation Target Application comprises determining if characters comprising the current Evaluation Target Application match characters comprising the current Reference Application.

In an embodiment evaluating the current Evaluation Target Application is syntactically correct with respect to the programming language in which it has been generated. For example if it is correct with respect to Python, Java, C# etc.

In an embodiment, if evaluating the current Evaluation Target Application results in a "fail" then the method includes identifying differences between the current Evaluation Target Application and the current Reference Application.

In an embodiment transforms for the differences that are identified are separately searched for and included in the current template.

In an embodiment the transforms for the differences that are identified are separately and consecutively searched for and included in the current template.

According to a further aspect of the present invention, there is provided a computer-implemented method for code generation, the method comprising:

receiving requirements in the form of a model for a desired subsequent target application;

obtaining one or more templates previously generated according to the previously described method; and

applying the one or more templates and the requirements to a code generator to thereby produce code for implementing the desired subsequent target application.

According to another aspect of the present invention, there is provided a computer-implemented method for producing code generation templates comprising:

receiving a series of commits ($C_i=C_0,\ldots,C_n$), each of the commits $C_1,\ldots,C_n$ of the series of commits including a version of a reference application $RA_1,\ldots,RAn$, each version of the reference application $RA_1,\ldots,RAn$ having a corresponding meta-model and model, commit C_0, comprising an initial model and meta model of a first version of the reference application RA_1;

for $\underline{i}=1$ to n, identifying differences between C_i and C_{i-1} to thereby obtain one or more identified differences therebetween;

searching to find one or more templates T_i for generating a target application that compares to a current version of the reference application, applying the one or more templates T_i, to the last commit C_{i-1} to thereby produce a current evaluation target application ETA_i (i=1,...,n); and

storing the current template T_i for later use upon the current evaluation target application ETA_i passing a comparison evaluation with a version of the reference application of C_{i-1} ,

whereby a series of templates are ultimately stored.

According to another aspect, there is provided a computer-implemented method for producing code generation templates comprising:

receiving a series of commits (C_i=C_0,...,C_n), each of the commits C_1,...,C_n of the series of commits including a version of a reference application RA_1,...,RA_n, each version of the reference application RA_1,...,RA_n having a corresponding meta-model and model, commit C_0, comprising an initial model and meta model of a first version of the reference application RA_1;

for i=1 to n, identifying differences between C_i and C_{i-1} to thereby obtain one or more identified differences therebetween;

searching to find one or more transformations for a template that when applied to C_{i-1} transforms a previous version of the reference application to a current version of the reference application,

adding the one or more transformations to a current template T_i;

applying the current template T_i, including the one or more transformations, to the last commit C_{i-1} to thereby produce a current evaluation target application ETA_i (i=1,...,n); and

storing the current template T_i for later use upon the current evaluation target application ETA_i passing a comparison evaluation with a version of the reference application of C_{i-1} ,

whereby a series of templates are ultimately stored.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred features, embodiments and variations of the invention may be discerned from the following Detailed Description which provides sufficient information for those skilled in the art to perform the invention. The Detailed Description is not to be regarded as limiting the scope of the preceding Summary of the Invention in any way. The Detailed Description will make reference to a number of drawings as follows:

Figure 1	is a diagram illustrating model driven engineering (MDE) software development.
Figure 2	is a diagram illustrating the levels of a Meta-Object Facility of the Unified Modelling Language (UML) used in MDE.
Figure 3	is a diagram illustrating a prior art, human developer-based approach to developing enterprise software using MDE to meet requirements of a person such as an enterprise owner.
Figure 4	is a block diagram of an artificially intelligent (AI) agent or “Bot” hosted on a computing platform, according to one embodiment of the present invention and shown interacting with a computational environment of the computing platform.
Figure 5	is a flowchart of a method according to an embodiment of the present invention.
Figure 5a	is a flowchart of a method according to another embodiment of the present invention.
Figure 5b	is a flowchart of a searching method according to an embodiment of the present invention.
Figure 6	is a timeline showing a series of “commits”, i.e. source code submitted by a human developer to a code repository along a timeline whilst perfecting a computer application, or “Reference Application” used by the Bot to learn code transformations that are encoded in a series of templates.
Figure 7	is a portion of code of a reference application for which a corresponding transformation is to be learned by the Bot.
Figure 8	is a template encoding the transformation for generating the code of Figure 7 with reference to a model.
Figure 9	is a further portion of code of a reference application for which a further corresponding transformation is to be learned by the Bot.

Figure 10 is a further template encoding the transformation for generating the code of Figure 9.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

Referring now to Figure 4, there is depicted a block diagram of an artificially intelligent agent or "Bot" 400 according to an embodiment of the present invention. It will be realised that the methods that are described herein are not necessarily limited to using a bot with the exact architecture of Figure 4 and that embodiments of the invention may be implemented in other ways, for example on a standalone computer or in a virtual computing environment.

The functional subassemblies making up Bot 400 will initially be described and then subsequently a method of operation will be discussed with reference to the flowchart of Figure 5 and timeline of Figure 6.

Bot 400 is hosted on a computer platform which provides a computational environment 412 for the bot. Bot 400 includes Sensors 401, 417 for receiving data from the computational environment 412 and actuators 403 for effecting changes in the computational environment 412. For example, actuators 403 may effect change by sending commands for storage devices to write files, create directories and send messages, for example to a human via Human Machine Interfaces (HMI) of the Environment 412 and/or to other computers/Bots via data network communication devices.

The Environment 412 includes a Source Code Repository 415 for a developer to store versions of a computer application or "commits", for example in similar fashion to the versions of Enterprise Software 31.1,...,31.n that have previously been discussed in relation to Figure 3. The Source Code Repository 415 may comprise a *git* repository (available from https://git-scm.com, retrieved 23/03/2021) which is a free and open-source distributed version control system. Other version control source code management systems are also available and may also be used. Each commit that a developer submits to the Source Code Repository 415 will typically also include a current model and meta-model corresponding to the current reference application. As will be discussed the sequence of commits stored in Source Code Repository 415 are used by the bot 400 to generate templates that encode transformations for transforming a reference application of a last commit C_{i-1}, to a reference application of a current commit C_i.

Bot 400 includes a mapping assembly 405 which incorporates a number of subassemblies as follows:

diff Identifier subassembly 407 is configured to identify differences in consecutive commits $C_1,\ldots,C_n$ that appear over time in the Source Code Repository 415 of the Environment 412. The diff Identifier subassembly 407 is configured to identify that changes have occurred and, in some embodiments, where the changes have occurred in consecutive commits $C_1,\ldots,C_n$ and/or the number of changes. For example, where Source Code Repository 415 comprises the *git* repository as previously discussed then the diff Identifier subassembly 407 applies a "git diff" command to two consecutive commits C_{i-1}, C_i in order to identify differences between them. The git diff command is documented at https://git-scm.com/docs/git-diff (retrieved 23/03/2021). It will be realised that the git repository and "git diff" command are simply one example of a mechanism for identifying differences between consecutive commits. The diff Identifier subassembly 407 is also responsive to output from Comparison subassembly ("Cf.") 423 and is arranged to compare an Evaluation Target Application 421, which will be discussed below, to a current reference application of a current commit C_i in order to identify differences therebetween.

Search manager subassembly 409 is configured to search for a transformation from the previous commit C_{i-1} to the current commit C_i. In some embodiments, Search manager subassembly 409 searches for a transformation for each of the differences that have been identified by the diff Identifier subassembly 407 to transform the previous commit C_{i-1} to the current commit C_i. In some other embodiments, Search manager subassembly 409 searches for one or more transformations for a template that when applied to the previous commit C_{i-1} transforms the previous commit C_{i-1} to the current commit C_i.

Search manager subassembly 409 is arranged to store the transformations as templates in **Templates storage assembly** 419 of Code Generator subassembly 413. The Templates storage assembly 419 is shown within the Code Generator 413 in the present embodiment but it will be realised that it could be located elsewhere provided the Code Generator subassembly 413 is able to access the templates. The templates T1,...,T*n* encode transformations for the code generator to ultimately be able to create a target application 425 based on requirements 427 expressed in a model 429 and meta-model 431.

For example, the model 429 and meta-model 431 may correspond to the initial model 37 and meta-model 35 of Figure 3 which express requirements for enterprise software 31 to automate enterprise processes 33. Consequently, as the Bot 400 acquires more templates over time by searching for transformations with reference to the reference application versions of the commits that the developer submits to the Source Code Repository it effectively incrementally learns how to write a target application 425 for a given set of requirements 427 based on the reference application from which it originally learnt.

Code Generator subassembly 413 is configured to apply a template from Templates storage assembly 419 (and thus a transformation encoded in the template) to a previous commit in order to generate an evaluation target application 421. The Evaluation Target Application 421 may be stored, for example, in a branch of the Source Code Repository 415 in order that the diff Identifier 407 can apply a git diff command to compare the Evaluation Target Application 421 to the current reference application of the current commit.

Comparison subassembly 423 is configured to compare the evaluation target application ETA*i* 421 to the current reference application RA*i* according to a number of comparison criteria. The comparison subassembly 423 generates an output in the form of a pass or fail value depending on whether or not the evaluation target application passes the comparison criteria with reference to the reference application to the diff Identifier subassembly 407. The diff Identifier subassembly 407 is arranged so that where the output from the Comparison subassembly 423 is a “fail” value then the diff Identifier subassembly 407 compares the evaluation Target Application to the current reference application of the current commit in Source Code Repository 415 to identify areas of the current reference application that differ from the Evaluation Target application. The areas that differ can then be the subject of further searching for an improved transform to upgrade the current template. Alternatively, if the Comparison subassembly 423 returns a “fail” value, the Search manager subassembly 409 may conduct a new search for transformations for a new (or updated) template that when applied to the previous commit C_{i-1} transforms the previous commit C_{i-1} to the current commit $C_{i.}$

Operation of Bot 400 – Learning Mode

An example of the Bot 400 operating to generate the templates T1,...,Tn that are stored in Templates storage assembly 419 will now be described with reference to the flowchart of Figure 5 and timeline of Figure 6. Mapping assembly 405 of the Bot 400 is programmed to implement the various functional subassemblies that have been described, e.g. items 407, 409, 413 and 423 and to proceed through the various steps of the flowcharts of Figures 5, 5a and 5b that will be described.

Figure 6 graphically illustrates the development of a reference application 601 by a human developer over time and through a number of iterations (reference applications 601.1 to 601.*n*) from a first Commit 1 at time t1 to a final Commit *n* at time t*n*. The reference application 601.1 at Commit 1 is produced by the human developer corresponding to an initial Meta-Model(t0) and Model(t0) in similar fashion to Enterprise Software V1 31.1 of Figure 3. Here, “Model(t0)” and “Meta-Model(t0) means the initial model and meta-model, i.e. at time zero, and thus effectively Model(t0) and Meta-Model(t0) comprise Commit 0 of the development, as they express the requirements of the first iteration of the reference application 601.1.

At box 501 Last Commit is set to C0 which comprises the initial Model(t0) and Meta-Model(t0).

At box 502 of Figure 5, Bot 400 checks via sensors 417 if a New Commit has been deposited into Source Code Repository 415. If a new commit has not been deposited then control proceeds to box 503 which checks if any external process termination signal has been received. Assuming that no such signal has been received then control returns to box 502 to check again for a new commit. Upon a new commit, e.g., Commit 1 (C1), being detected in Source Code Repository 415, control proceeds to box 505.

At box 505 the diff Identifier 407 initialises a Current Template T1 and updates Current template T1 with folder and file structure information based on the new Commit. For example, if the reference application 601.1 includes two files *alpha.java* and *bravo.java* in a directory *src\com\project\entities* of the Environment 412, then, as an example, at box 505 the diff Identifier subassembly 407 may include folder and file structure information in template T1 specifying that entity and attribute files, for example *entity.egl* and *attribute.egl* are to be written to a corresponding location with reference to a templates root directory, e.g., *templates\com\project\entities*.

At box 507 the diff Identifier subassembly 407 identifies differences between New Commit and Last Commit. On the first iteration all of the code 602 in the Reference application 601.1 will be new and the corresponding Model(t1) and Meta-Model(t1) will be unchanged from the initial Model(t0) and Meta-Model(t0). For example, the reference application 601.1 may be a java object file as set out in Figure 7, which sets out methods for getting and setting attributes for "yellow", "green" and "blue" objects, which are defined in Model(t0) and are instances of a class specified in Meta-Model(t0). For the example Reference Application 601.1 of Figure 7, the diff Identifier subassembly 407 identifies a difference being eighteen new lines of code (numbered at left hand side of Figure 7) and including three instances of a class.

At boxes 509 to 513 in Figure 5, or box 514 in Figure 5a, the Search Manager 409 searches for a transformation that will, when run for the same three instances of the class, generate the same code as that of the reference application. After searching for an optimal transformation by trying different code combinations, for example using a hill climbing search strategy, Bot 400 settles on the transformation of Figure 8 which is encoded into template T1. Template T1, and any other templates described herein, may comprise a single template or a plurality of templates.

In order to find the template that matches the target application to the reference application, for example to find the template shown in Fig 8 to the reference application in Fig 7, different heuristics can be used to search the state space, that is, the space of possible templates, including templates that are unworkable.

For example, if each line in Fig 7 is represented as a color (lines 1-6 as yellow ("Y"), lines as 7-12 green ("G"), lines 13-18 as blue ("B")) then YYYYYYGGGGGGBBBBBB is the solution that is to be searched for. By taking the 6 lines from any color and copying them into the template, it is possible to replace the color name with the value of the attribute name variable as shown in Fig 8, execute the templates, and then compare the target application to determine which color guesses were correct. It is possible to evaluate the node based on how many matched colors (lines of code matched between the target and the reference).

For a computer-implemented search, such as that of the bot, to apply a search heuristic to find the correct color combination YYYYYYGGGGGGBBBBBB, it checks many others first. The simplest approach is to try random color combinations until a match is found. But there are ways to make the heuristic better than random. For instance, to start the search for this example, there are 18 lines with 3 colors. The first level of the search tree could guess something like:

YGBYGBYGBYGBYGBYGB

YYGGBBYYGGBBYYGGBB

YYYGGGBBBYYYGGGBBB
YYYYGGGGBBBBYYYYGG, and so on.

Each of these color combinations (nodes in the search tree) creates a template, execute the template, and then compare the target application to the reference application to see which node is the closest to the solution. This node is then chosen and another set of child nodes are created and evaluated.

This approach can be applied to other scenarios as well. For example, the code found in Fig 9 achieves the same functionality as Fig 8 but the developer has written the code differently. The solution color combination for this scenario would be YYYGGGBBBYYYGGGBBB. So, the search heuristic can be applied again to find the correct template shown in Fig 10.

The Inventors have found that simple random approaches work for implementing the search through to more advanced approaches that take advantage of typical practices that developers commonly use to organize their code.

Template T1 contains code for Code Generator subassembly 413 to generate an evaluation target application 421 at box 515, using current Template T1 and Model(t0) and Meta-Model(t0) of the last Commit as input.

The Search Manager subassembly 409 is preferably configured to operate using heuristics to assist it to identify typical human developer coding patterns in the Reference Application 601.1. In the example Reference Application code snippet of Figure 7 the optimal template of Figure 8 comprises a single *for* loop which encapsulates consecutive *get* and *set* methods to get and set attributes for each object in the Model because the human developer has consecutively coded get and set methods for each of the attributes "yellow", "green" and "blue".

Alternatively, in the example Reference Application code snippet of Figure 9, the corresponding optimal template of Figure 10 is comprised of two consecutive *for* loops being a first *for* loop to *get* attributes and a second *for* loop to *set* for attributes. The first *for* loop in the template of Figure 10 is contains instructions based on lines 1 to 9 of the Reference Application code snippet of Figure 9 and the second *for* loop in the template of Figure 10 contains code based on lines 10 to 18 of Figure 9.

As previously discussed, the examples of Figures 7 and 8 and Figures 9 and 10 illustrate that it is preferable to program bot 400 to configure Search Manager subassembly 409 to hunt for repeating patterns of code in the Reference Application to arrive at an optimal template. As developers commonly use patterns and architectures to create software applications the search heuristic can be programmed to use these to find an optimal solution within a reasonable amount of time. As opposed to an exhaustive or random search that would result in unreasonable running times to be practical.

At box 517 Bot 400 operates Comparison (Cf.) subassembly 423 to compare the current evaluation Target Application 421 to the current reference application of the current commit C*i*. The Comparison subassembly 423 is configured to make comparisons using a number of metrics such as:

1. Does the Evaluation Target Application 421 compile?
2. Do the number of lines in the current Evaluation Target Application match the number of lines in the current reference application?
3. Do the characters in the current Evaluation Target Application match the characters in the current reference application?
4. Is the coding of the current Evaluation Target Application 421 syntactically correct with respect to the programming language in which it has been generated?

Alternatively, if at box 519 the output from the Comparison subassembly 421 is a "fail" then, at box 521 the diff Identifier assembly 407 identifies differences between the current Evaluation Target Application 421 and the current Reference Application, e.g. Reference Application 601.1. The differences that are identified at box 521 are then searched for by Search Manager subassembly 409 according to boxes 509 to 513 in Figure 5 or box 514 in Figure 5a. It will be noted that the Search Manager subassembly 409 is configured to consecutively search for a transform in respect of each identified difference separately in the example illustrated in Figure 5. However, in the illustrated example shown in Figure 5a, the Search Manager subassembly 409 is configured to search for a transformation in respect of the entire reference application to create a template that is not based on an earlier template. With the second approach, the Inventor has found that learning the entire reference application results in a template that is more easily read and understood by a developer, which can be important to the process when a developer is required to intervene and manually edit the code.

Regardless of the approach used from either Figure 5 or Figure 5a, eventually, at box 519, the output from the Comparison subassembly 421 is a “pass”. The “pass” output indicates that transforms for all of the differences between the current Evaluation Target Reference and the current Reference Application have been found and meet the comparison criteria. Control then proceeds to box 523 where the next “Last Commit” is set to the previous “New Commit”, and then box 525, where the “Current Template” is updated to be the “Next Template”, prior to a new iteration of the previously described cycle commencing again at box 502.

In overview, in the next iteration at box 502, upon a new commit, e.g. Commit 2 (C2) being deposited by the human developer in Source Code Repository 415, the diff Identifier subassembly 407 identifies differences in the new iteration of the Reference Application 601.2 relative to the last iteration 601.1. Those differences are graphically illustrated as additional code snippets 603 and 605 in Figure 6.

Then, the Search Manager subassembly 409 searches for a transformation to update the template to learn and incorporate additional transformations for the most recently identified differences from the new commit.

In one example, through the loop of boxes 509 to 513, the Search Manager subassembly 409 firstly searches for a transformation for code snippet 603, being the first difference, and then code snippet 605, being the second difference. At box 511 the Search Manager subassembly 409 on respective first and second iterations of the *For* loop writes the transforms for the first code difference 603 and then the second code difference 605 into the Current Template (T2).

At box 515 the Code Generator subassembly 413 uses the Model(t1) and Meta-Model(t1) of the Last Commit, which is currently C1, and the Current Template (T2) to generate Evaluation Target Application 421. The Current Template is finalized, if it passes the test condition at box 519 implemented by Comparison subassembly 423, or is further improved, if it fails the comparison tests at box 519, by passing through box 521 and boxes 509 to 515 in Figure 5.

At the end of each cycle, i.e., upon reaching box 523, a new Template is added to the templates storage area 419 that is accessible to the Code Generator 413. Consequently, over time Bot 400 acquires templates T1,...,Tn which encode transformations for progressively transforming an initial model and meta model into a final target application. Accordingly, once the entire process is finished the bot 400 has learned or been "trained" how to generate a Target Application 425 from Target Application Requirements 427 specified in Model 429 and Meta-Model 431.

Alternatively, after identifying each additional code snippet, through box 514 in Figure 5a, the Search Manager subassembly 409 searches for a transformation or transformations for the entirety of the Reference Application 601.2 and writes the transforms for the Reference Application 601.2 into the Current Template (T2) or templates.

As an example, at box 514a of Figure 5b, the Search Manager subassembly 409 first sorts the reference application (or reference files within the reference application) into sets of files to prepare for searching. The reference files of the reference application are sorted based on commonality. For example, two files that are instances of an element (such as a class or enum) will be sorted into the same set.

Next, at box 514b, the Search Manager subassembly 409 initialises a solution for each of the reference files in each set and associates a solution with a reference file.

The Search Manager subassembly 409, at box 514c, then initiates a search to find a solution for each reference file. In searching for a solution, the Search Manager subassembly 409 traverses a search tree comprising a number of nodes which each represent a guess (or possible solution in the form of a transform) for lines of code from the reference file. This can be implemented using different searching techniques, such as a hill climber, A*, exhaustive, and other hybrid approaches. In some embodiments, the Search Manager subassembly 409 searches for patterns within the reference file. A pattern is a repeated or regular form that is found in the code. A pattern can take a number of forms. For example, a pattern may be a protected region of the reference file that the bot is not required to learn. Alternatively, the repeating pattern may be represented as a repeating pattern across a number of lines.

Once a solution has been found for each reference file, the Search Manager subassembly 409, at box 514d, attempts to resolve any discrepancies between solutions in a set. A discrepancy occurs when the bot is learning from multiple examples and it is not clear what the developers intention is. In such cases, the bot will ask the developer for some help and the developer will respond with an answer to help guide the bot.

Once any discrepancies are resolved, the Search Manager subassembly 409 computes a merged solution from the solutions associated with each reference file at box 514e and updates the current template with the merged solution.

Next, similar to the example illustrated in Figure 5 and described above, at box 515 the Code Generator subassembly 413 then uses the Model(t1) and Meta-Model(t1) of the Last Commit, which is currently C1, and the Current Template

(T2) to generate Evaluation Target Application 421. The Current Template is finalized, if it passes the test condition at box 519 implemented by Comparison subassembly 423, or is further improved, if it fails the comparison tests at box 519, by passing through box 521 and boxes 514 and 515.

At the end of each cycle, i.e., upon reaching box 523, a new Template is added to the templates storage area 419 that is accessible to the Code Generator 413. Consequently, over time Bot 400 acquires templates T1,...,Tn which encode transformations for progressively transforming an initial model and meta model into a final target application. Accordingly, once the entire process is finished the bot 400 has learned or been "trained" how to generate a Target Application 425 from Target Application Requirements 427 specified in Model 429 and Meta-Model 431.

As with the previous example, at the end of each cycle, i.e., upon reaching box 523, a new Template is added to the templates storage area 419 that is accessible to the Code Generator 413. Consequently, over time Bot 400 acquires templates T1,...,Tn which encode transformations for progressively transforming an initial model and meta model into a final target application. Accordingly, once the entire process is finished the bot 400 has learned or been "trained" how to generate a Target Application 425 from Target Application Requirements 427 specified in Model 429 and Meta-Model 431.

The use of the trained Bot 400 will now be described with reference to Figure 11. In Figure 11 Bot 400 receives requirements 427 for a desired Target Application that is wanted to automate processes such as business processes 33 of Figure 3. For example, the desired Target Application may be inventory control software for a warehousing operation of an enterprise. The Target Application may correspond to the final version of the Enterprise Application 31.n that has been previously discussed with reference to Figure 3. The requirements 427 are specified by means of a Meta-Model 429 and Model 431 which may respond to the initial metal-model 35 and model 37 of the example of Figure 3. The trained Bot 400 has been previously trained with iterations of a reference application for automating an inventory control system.

Trained Bot 400 initially applies template T1 to Model 429 and Meta-Model 431 to generate a Target Application V1. It then applies template T2 to the Target Application V1 and corresponding model and meta-model for Target Application V1 to generate Target Application V2. The process continues for all of the Templates up to Template Tn, being the final template, which results in the final Target Application 425, with its corresponding Meta-Model 433 and Model 435 being generated.

The final Target Application 425 can then be fine tuned by a human developer to arrive at the desired Target Application. By using the final Target Application 425 to produce the desired Target Application the human developer is saved much boilerplate work relating to coding commonly recurring patterns in software and consequently the development time is greatly reduced.

Embodiments of the invention described herein utilise search and optimization (or heuristic optimization) methods in Artificial Intelligence as opposed to machine learning methods.

The terms “transform” and “transformation” are to be understood as describing coding transformations stored in templates that are applied to a reference application of the last (most recent) commit before the current commit. These terms as used herein do not describe model-to-model transformations.

Implementations of the invention can be realized as one or more computer program products, i.e., one or more modules of computer program instructions encoded on a computer readable medium for execution by, or to control the operation of, data processing apparatus. The computer readable medium can be a machine-readable storage device, a machine-readable storage substrate, a memory device, a composition of matter affecting a machine-readable propagated signal, or a combination of one or more of them. The term "data processing apparatus" encompasses all apparatus, devices, and machines for processing data, including by way of example a programmable processor, a computer, or multiple processors or computers. The apparatus can include, in addition to hardware, code that creates an execution environment for the computer program in question, e.g., code that constitutes processor firmware, a protocol stack, a database management system, an operating system, or a combination of one or more of them.

A computer program (also known as a program, software, software application, script, or code) can be written in any form of programming language, including compiled or interpreted languages, and it can be deployed in any form, including as a stand-alone program or as a module, component, subroutine, or other unit suitable for use in a computing environment. A computer program does not necessarily correspond to a file in a file system. A program can be stored in a portion of a file that holds other programs or data (e.g., one or more scripts stored in a markup language document), in a single file dedicated to the program in question, or in multiple coordinated files (e.g., files that store one or more modules, sub programs, or portions of code). A computer program can be deployed to be executed on one computer or on multiple computers that are located at one site or distributed across multiple sites and interconnected by a communication network.

The processes and logic flows described in this disclosure can be performed by one or more programmable processors executing one or more computer programs to perform functions by operating on input data and generating output. The processes and logic flows can also be performed by, and apparatus can also be implemented as, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application specific integrated circuit).

Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors, and any one or more processors of any kind of digital computer. Generally, a processor will receive instructions and data from a read only memory or a random access memory or both. The essential elements of a computer are a processor for performing instructions and one or more memory devices for storing instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto optical disks, or optical disks. However, a computer need not have such devices. Moreover, a computer can be embedded in another device, e.g., a mobile telephone, a personal digital assistant (PDA), a mobile audio player, a Global Positioning System (GPS) receiver, to name just a few. Computer readable media suitable for storing computer program instructions and data include all forms of non-volatile memory, media and memory devices, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto optical disks; and CD ROM and DVD-ROM disks. The processor and the memory can be supplemented by, or incorporated in, special purpose logic circuitry.

To provide for interaction with a user, implementations of the invention can be implemented on a computer having a display device, e.g., a CRT (cathode ray tube) or LCD (liquid crystal display) monitor, for displaying information to the user and a keyboard and a pointing device, e.g., a mouse or a trackball, by which the user can provide input to the computer. Other kinds of devices can be used to provide for interaction with a user as well; for example, feedback provided to the user can be any form of sensory feedback, e.g., visual feedback, auditory feedback, or tactile feedback; and input from the user can be received in any form, including acoustic, speech, or tactile input.

Implementations of the present disclosure can be realized in a computing system that includes a back-end component, e.g., as a data server, or that includes a middleware component, e.g., an application server, or that includes a front end component, e.g., a client computer having a graphical user interface or a Web browser through which a user can interact with an implementation of the present disclosure, or any combination of one or more such back end, middleware, or front end components. The components of the system can be interconnected by any form or medium of digital data communication, e.g., a communication network. Examples of communication networks include a local area network ("LAN") and a wide area network ("WAN"), e.g., the Internet.

The computing system can include clients and servers. A client and server are generally remote from each other and typically interact through a communication network. The relationship of client and server arises by virtue of computer programs running on the respective computers and having a client-server relationship to each other.

While this disclosure contains many specifics, these should not be construed as limitations on the scope of the disclosure or of what may be claimed, but rather as descriptions of features specific to particular implementations of the disclosure. Certain features that are described in this disclosure in the context of separate implementations can also be provided in combination in a single implementation. Conversely, various features that are described in the context of a single implementation can also be provided in multiple implementations separately or in any suitable sub-combination. Moreover, although features may be described above as acting in certain combinations and even initially claimed as such, one or more features from a claimed combination can in some cases be excised from the combination, and the claimed combination may be directed to a sub-combination or variation of a sub-combination.

Similarly, while operations are depicted in the drawings in a particular order, this should not be understood as requiring that such operations be performed in the particular order shown or in sequential order, or that all illustrated operations be performed, to achieve desirable results. In certain circumstances, multitasking and parallel processing may be advantageous. Moreover, the separation of various system components in the implementations described above should not be understood as requiring such separation in all implementations, and it should be understood that the described program components and systems can generally be integrated together in a single software product or packaged into multiple software products.

Thus, particular implementations of the present disclosure have been described. Other implementations are within the scope of the following claims. For example, the actions recited in the claims can be performed in a different order and still achieve desirable results.

In compliance with the statute, the invention has been described in language more or less specific to structural or methodical features. The term “comprises” and its variations, such as “comprising” and “comprised of” is used throughout in an inclusive sense and not to the exclusion of any additional features. It is to be understood that the invention is not limited to specific features shown or described since the means herein described comprises preferred forms of putting the invention into effect.

The invention is, therefore, claimed in any of its forms or modifications within the proper scope of the appended claims appropriately interpreted by those skilled in the art.

Throughout the specification and claims (if present), unless the context requires otherwise, the term "substantially" or "about" will be understood to not be limited to the value for the range qualified by the terms.

Any embodiment of the invention is meant to be illustrative only and is not meant to be limiting to the invention. Therefore, it should be appreciated that various other changes and modifications can be made to any embodiment described without departing from the spirit and scope of the invention.

CLAIMS:

1. A computer-implemented method for producing code generation templates comprising:

receiving a series of commits, each of the commits in the series of commits including a version of a reference application, each version of the reference application having a corresponding meta-model and model;

identifying differences between time separated commits to thereby obtain one or more identified differences therebetween;

searching to find one or more transformations for transforming a previous version of the reference application to a current version of the reference application,

adding the one or more transformations to a current template,

applying the current template, including the one or more transformations, to the recent commit to thereby produce a current evaluation target application; and

storing the current template for later use upon the current evaluation target application passing a comparison evaluation with a version of the reference application of the recent commit,

whereby a series of templates are ultimately stored.

2. The method of claim 1, wherein the series of commits comprises commits $C_i=C_0,\ldots,C_n$, wherein each of the commits $C_1,\ldots,C_n$ of the series of commits includes a version of a reference application $RA_1,\ldots,RA_n$,

3. The method of claim 2, wherein each version of the reference application commit C_0, comprises an initial model and initial meta model of a first version of the reference application RA_1.

4. The method of any one of the preceding claims, including checking if a new commit of the series of commits has been received in a source code repository.

5. The method of claim 4, including updating a current template with folder and file structure information based on the new commit.

6. The method of any one of the preceding claims, wherein the comparison evaluation includes checking if the evaluation target application can be compiled by a compiler.

7. The method of any one of the preceding claims, wherein the comparison evaluation comprises determining if the number of lines of code in the current evaluation target application match the number of lines in the current Reference Application.

8. The method of any one of the preceding claims, wherein the comparison evaluation comprises determining if characters comprising the current evaluation target application match characters comprising the current reference application.

9. The method of any one of the preceding claims, wherein the comparison evaluation comprises checking if the current evaluation target application is syntactically correct with respect to the programming language in which it has been generated.

10. The method of any one of the preceding claims, wherein if the comparison evaluation results in a “fail” then the method includes identifying differences between the current evaluation target application and the current reference application.

11. The method of any one of the preceding claims, wherein for each identified difference of the one or more identified differences, the computer implemented method includes the steps:

searching to find the one or more transformations in respect of the identified difference,

adding the one or more transformations to the current template, and applying the current template, including the one or more transformations for each of the identified differences, to a recent commit to thereby produce a current evaluation target application.

12. The method of any one of the preceding claims, wherein transformations for the identified differences are separately searched for and included in the current template.

13. A computer-implemented method for producing code generation templates comprising:

receiving a series of commits (C_i=C_0,...,C_n), each of the commits C_1,...,C_n of the series of commits including a version of a reference application RA_1,...,RA_n, each version of the reference application RA_1,...,RA_n having a corresponding meta-model and model, commit C_0, comprising an initial model and meta model of a first version of the reference application RA_1;

for i=1 to n, identifying differences between C_i and C_{i-1} to thereby obtain one or more identified differences therebetween;

searching to find one or more transformations for transforming a previous version of the reference application to a current version of the reference application,

adding the one or more transformations to a current template T_i;

applying the current template T_i, including the one or more transformations, to the last commit C_{i-1} to thereby produce a current evaluation target application ETA_i (i=1,...,n); and

storing the current template T_i for later use upon the current evaluation target application ETA_i passing a comparison evaluation with a version of the reference application of C_{i-1} ,

whereby a series of templates are ultimately stored.

14. The computer-implemented method of claim 1 or claim 13, wherein the templates are generated by an artificially intelligent agent ("the bot") in accordance with the above method wherein the bot learns new versions of the reference application in an incremental fashion based on differences in consecutive versions of the reference application.

15. The method of claim 14, wherein the differences in consecutive versions of the reference application include updates to models and meta-models corresponding to the versions of the reference application.

16. An artificially intelligent agent ("the bot") hosted on a computer platform providing a computing environment for the bot, the bot comprising:

one or more sensors configured to read incrementally updated versions of a reference application stored in the computing environment;

one or more actuators configured to write files to the computing environment; and

a mapping arrangement responsive to the one or more sensors and coupled to the one or more actuators for control thereof,

the mapping arrangement including,

a difference identifier subassembly configured to identify one or more differences in code of a current version of the reference application relative to an immediately earlier version of the reference application to thereby produce one or more identified differences,

a search manager subassembly responsive to the difference identifier for receiving the one or more identified differences and configured to determine transformations transforming the immediately earlier version of the reference application into the latest version of the reference application wherein the search manager encodes the transformations in a template;

a code generation subassembly arranged to process the template with reference to a model corresponding to the immediately earlier version of the reference application to thereby generate an evaluation target application; and

a comparison subassembly arranged to compare the evaluation target application to the latest version of the reference application;

wherein upon output from the comparison subassembly indicating that the evaluation target application passes comparison requirements, the template is retained for subsequent use by the code generator subassembly.

17. A computer-implemented software application writing method comprising:

receiving requirements in the form of a model for a desired subsequent target application;

obtaining one or more templates previously generated according to any one of claims 1 to 15; and

applying the one or more templates and the requirements to a code generator to thereby produce code for implementing the desired subsequent target application.

18. A computer-implemented method for producing code generation templates comprising:

receiving a series of commits (C_i=C_0,...,C_n), each of the commits C_1,...,C_n of the series of commits including a version of a reference application RA_1,...,RAn, each version of the reference application RA_1,...,RAn having a corresponding meta-model and model, commit C_0, comprising an initial model and meta model of a first version of the reference application RA_1;

for $\underline{i}$=1 to n, identifying differences between C_i and C_{i-1} to thereby obtain one or more identified differences therebetween;

searching to find one or more templates T_i for generating a target application that compares to a current version of the reference application,

applying the one or more templates T_i, to the last commit C_{i-1} to thereby produce a current evaluation target application ETA_i (i=1,...,n); and

storing the current template T_i for later use upon the current evaluation target application ETA_i passing a comparison evaluation with a version of the reference application of C_{i-1} ,

whereby a series of templates are ultimately stored.

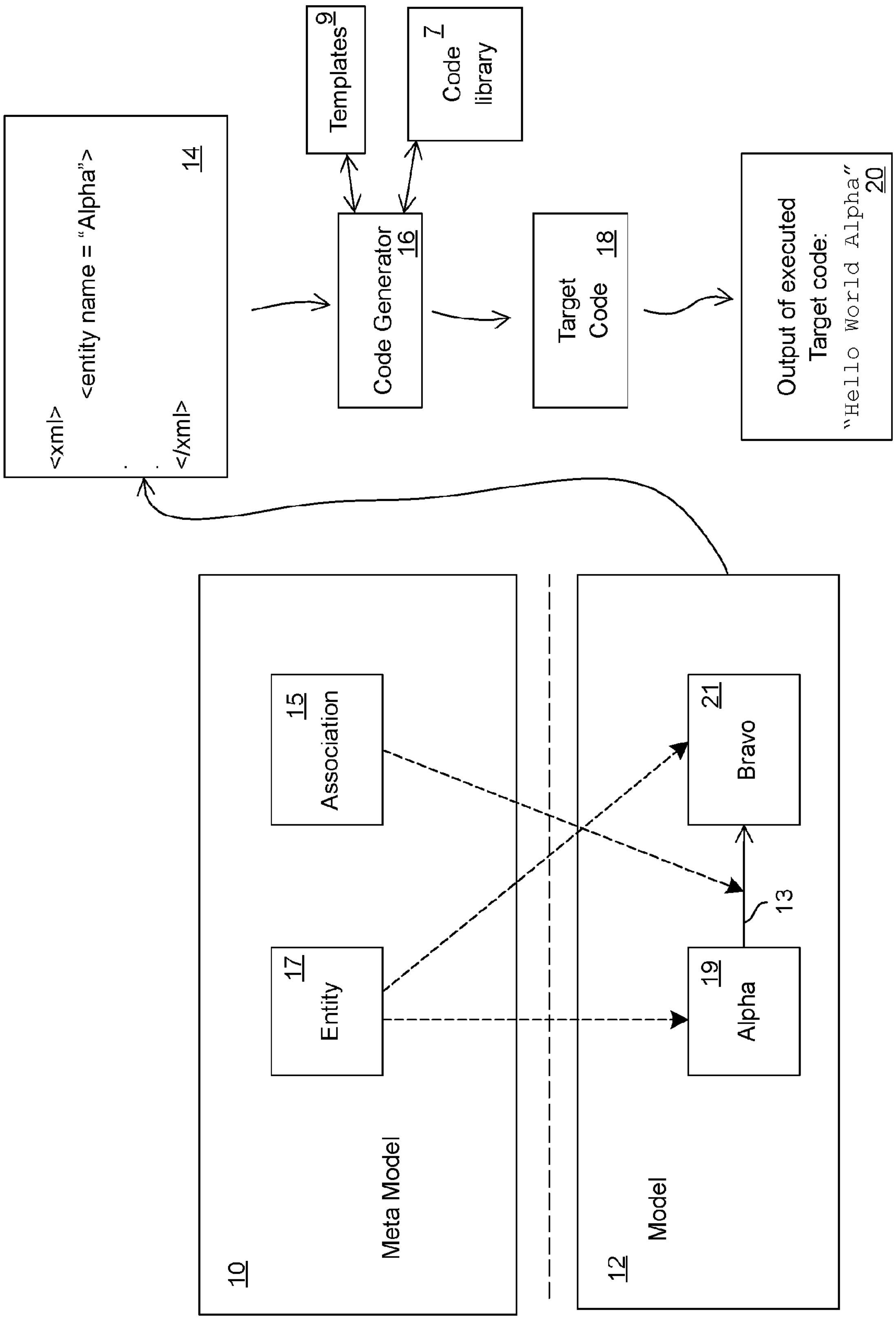

Meta Model
10
Entity
17
Association
15
Model
12
Alpha
19
Bravo
21
13
<xml>
<entity name = "Alpha">
. .
</xml>
14
Code Generator
16
Templates
9
Code
library
7
Target
Code
18
Output of executed
Target code:
"Hello World Alpha"
20

FIG. 1

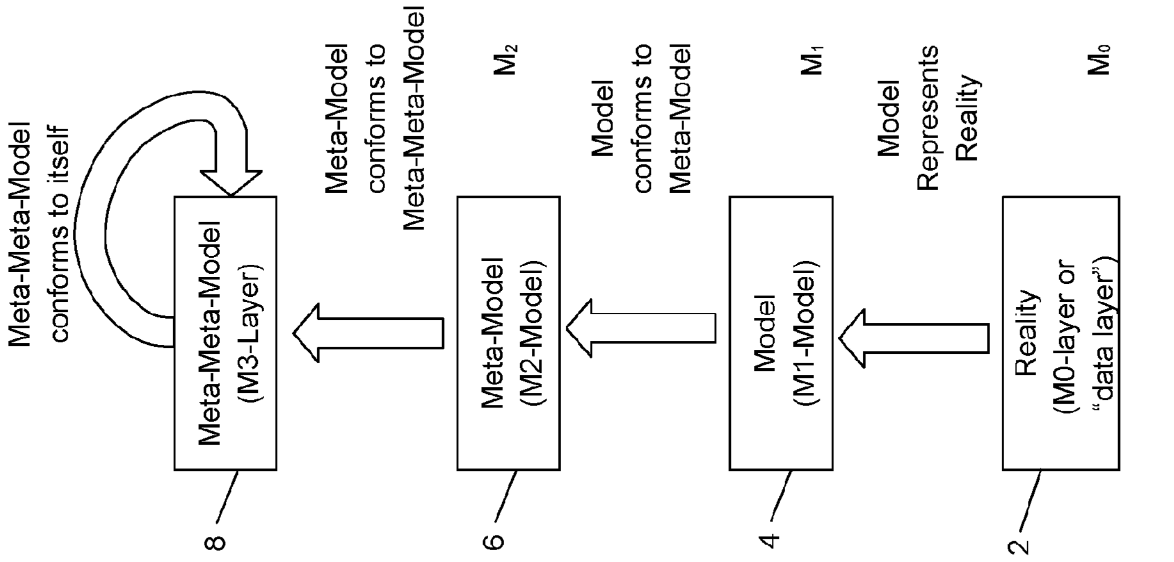

Meta-Meta-Model conforms to itself
Meta-Model conforms to Meta-Meta-Model
M_2
Model conforms to Meta-Model
M_1
Model Represents Reality
M_0
Meta-Meta-Model (M3-Layer)
Meta-Model (M2-Model)
Model (M1-Model)
Reality (M0-layer or "data layer")
8
6
4
2

FIG. 2

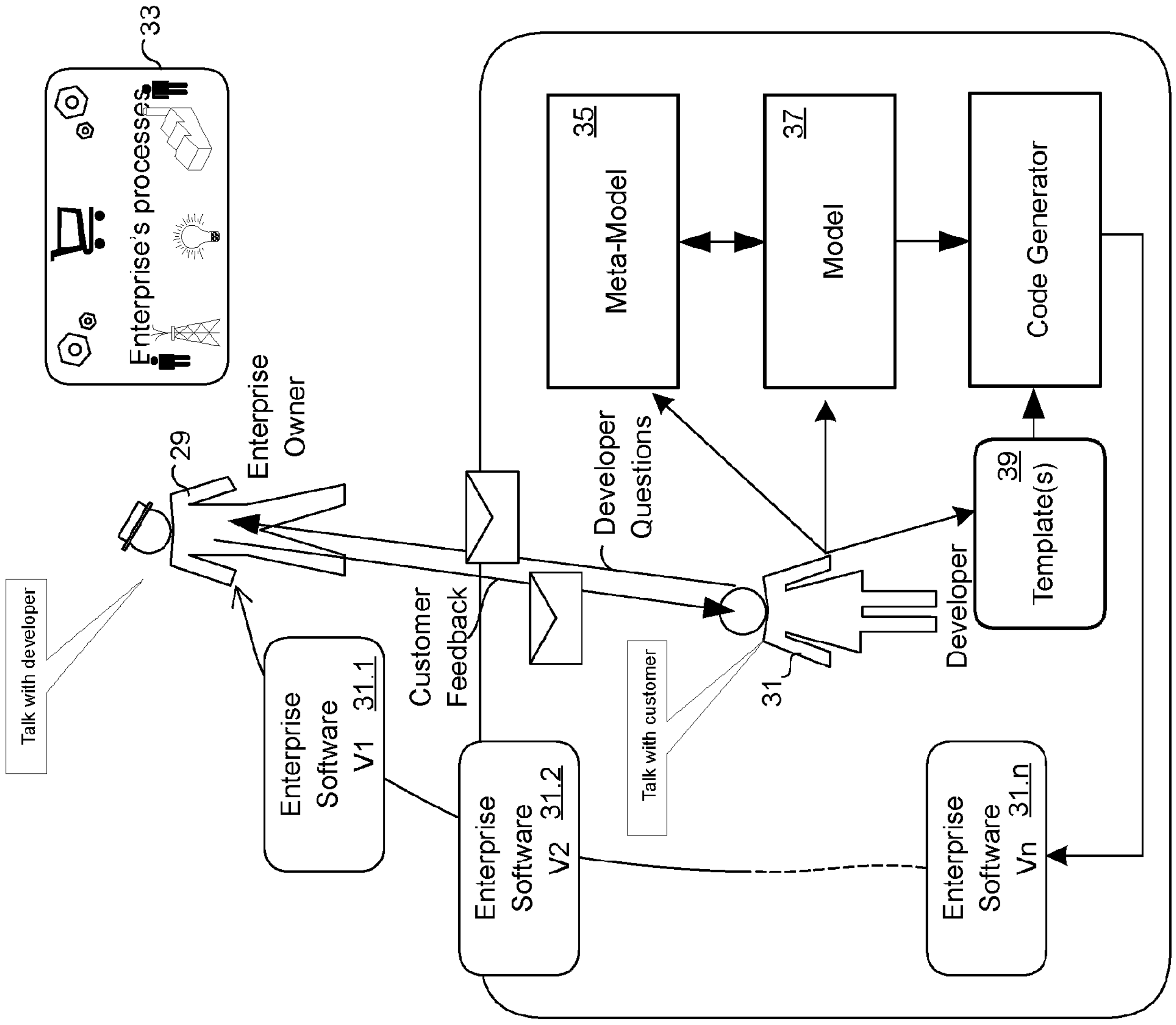

33
Enterprise's processes
29
Enterprise Owner
Talk with developer
Enterprise Software V1 31.1
Customer Feedback
Developer Questions
35
Meta-Model
37
Model
Code Generator
39
Template(s)
Developer
31
Talk with customer
Enterprise Software V2 31.2
Enterprise Software Vn 31.n

FIG. 3

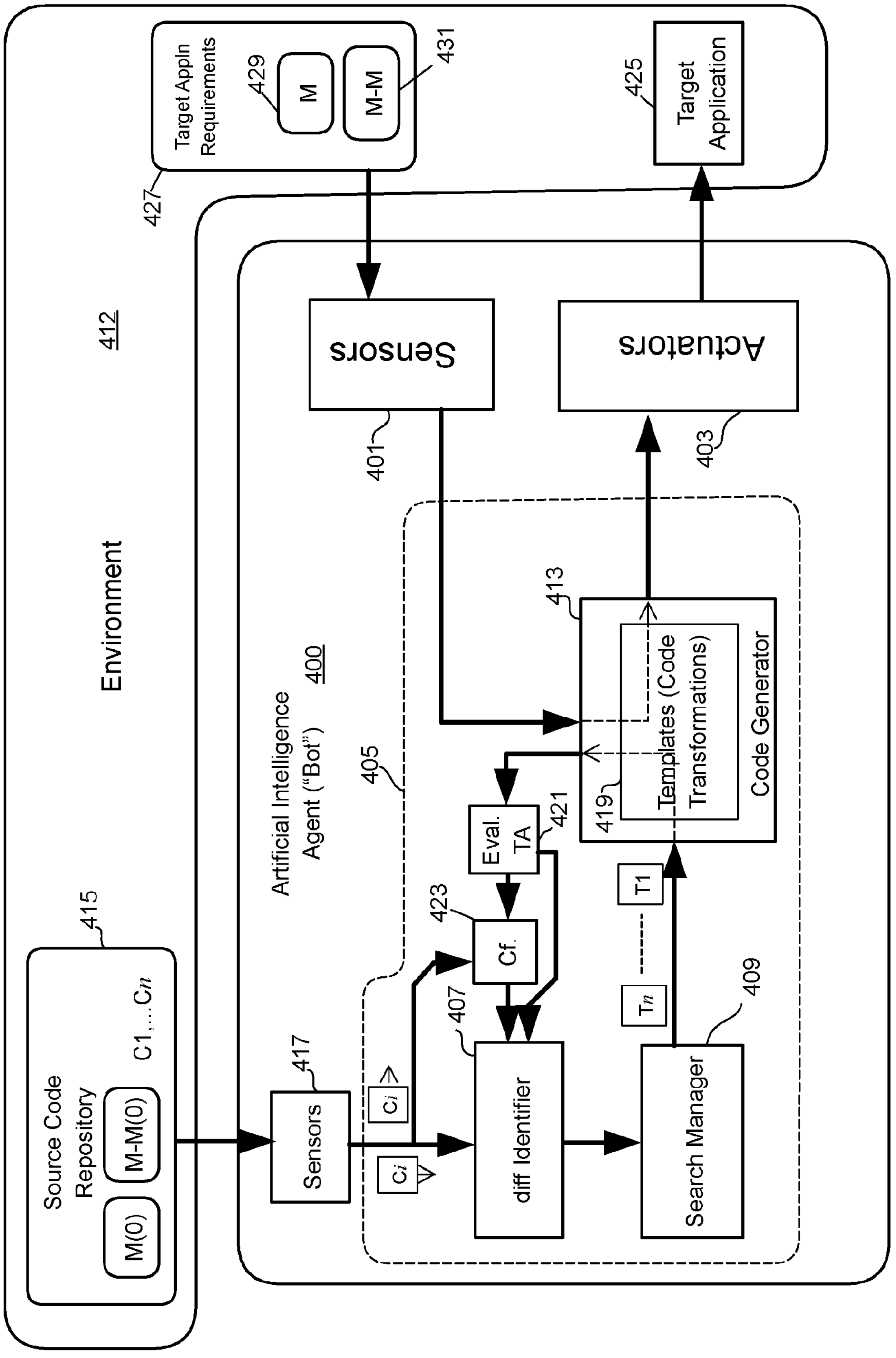

Environment
412
Source Code Repository
M(0)
M-M(0)
C1,...Cn
415
Target Appln Requirements
M
M-M
429
431
427
Target Application
425
Artificial Intelligence Agent ("Bot")
400
Sensors
401
Actuators
403
405
Sensors
417
Ci
diff Identifier
407
Cf
423
Eval. TA
421
Search Manager
409
Tn
T1
Templates (Code Transformations)
419
Code Generator
413

FIG. 4

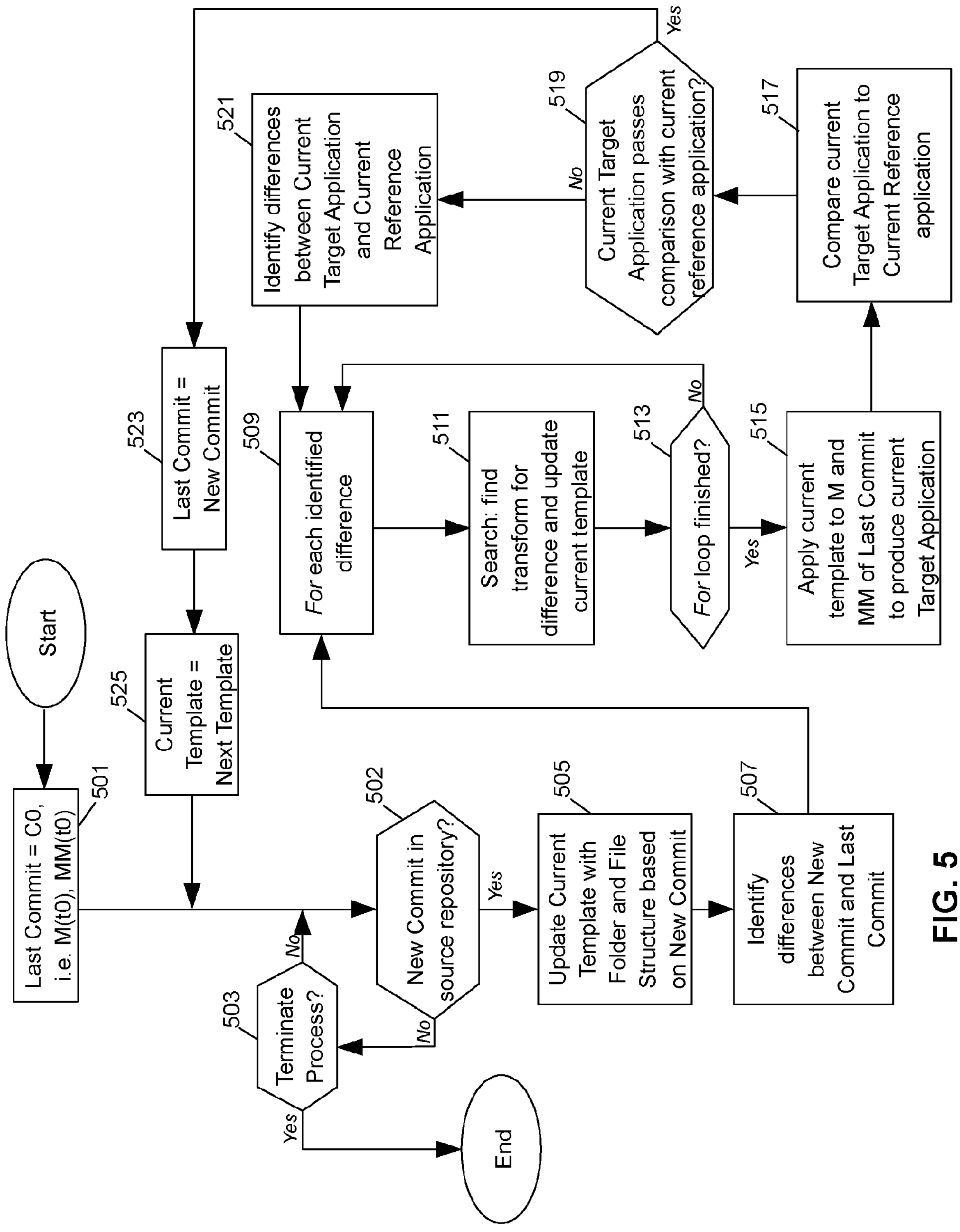
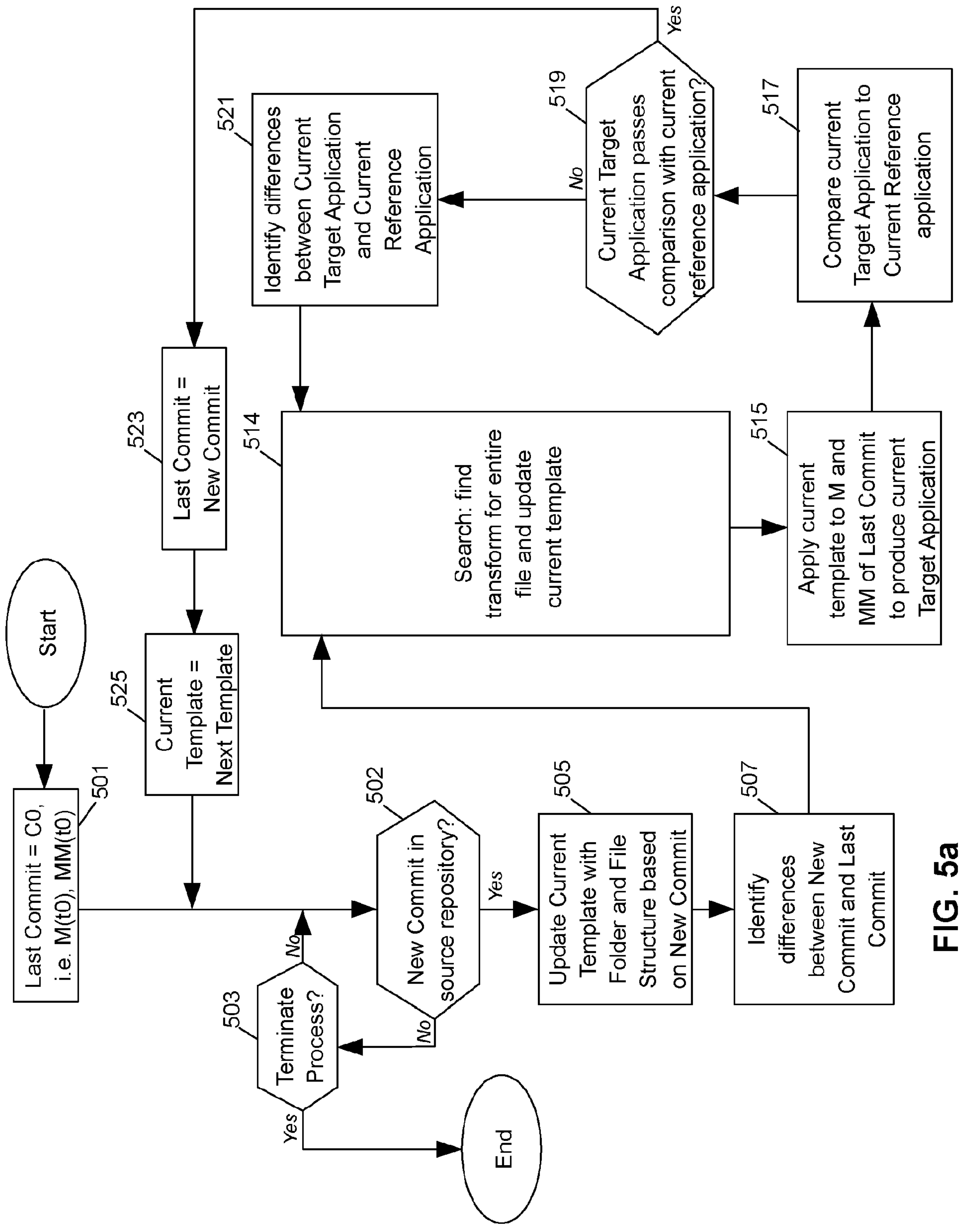

Start
Last Commit = C0, i.e. M(t0), MM(t0)
501
502
New Commit in source repository?
Yes
No
503
Terminate Process?
No
Yes
End
505
Update Current Template with Folder and File Structure based on New Commit
507
Identify differences between New Commit and Last Commit
509
For each identified difference
511
Search: find transform for difference and update current template
513
For loop finished?
No
Yes
515
Apply current template to M and MM of Last Commit to produce current Target Application
517
Compare current Target Application to Current Reference application
519
Current Target Application passes comparison with current reference application?
No
Yes
521
Identify differences between Current Target Application and Current Reference Application
523
Last Commit = New Commit
525
Current Template = Next Template

FIG. 5



Start
Last Commit = C0, i.e. M(t0), MM(t0)
501
New Commit in source repository?
502
Yes
No
Terminate Process?
503
Yes
No
End
Update Current Template with Folder and File Structure based on New Commit
505
Identify differences between New Commit and Last Commit
507
Search: find transform for entire file and update current template
514
Apply current template to M and MM of Last Commit to produce current Target Application
515
Compare current Target Application to Current Reference application
517
Current Target Application passes comparison with current reference application?
519
Yes
No
Identify differences between Current Target Application and Current Reference Application
521
Last Commit = New Commit
523
Current Template = Next Template
525

FIG. 5a

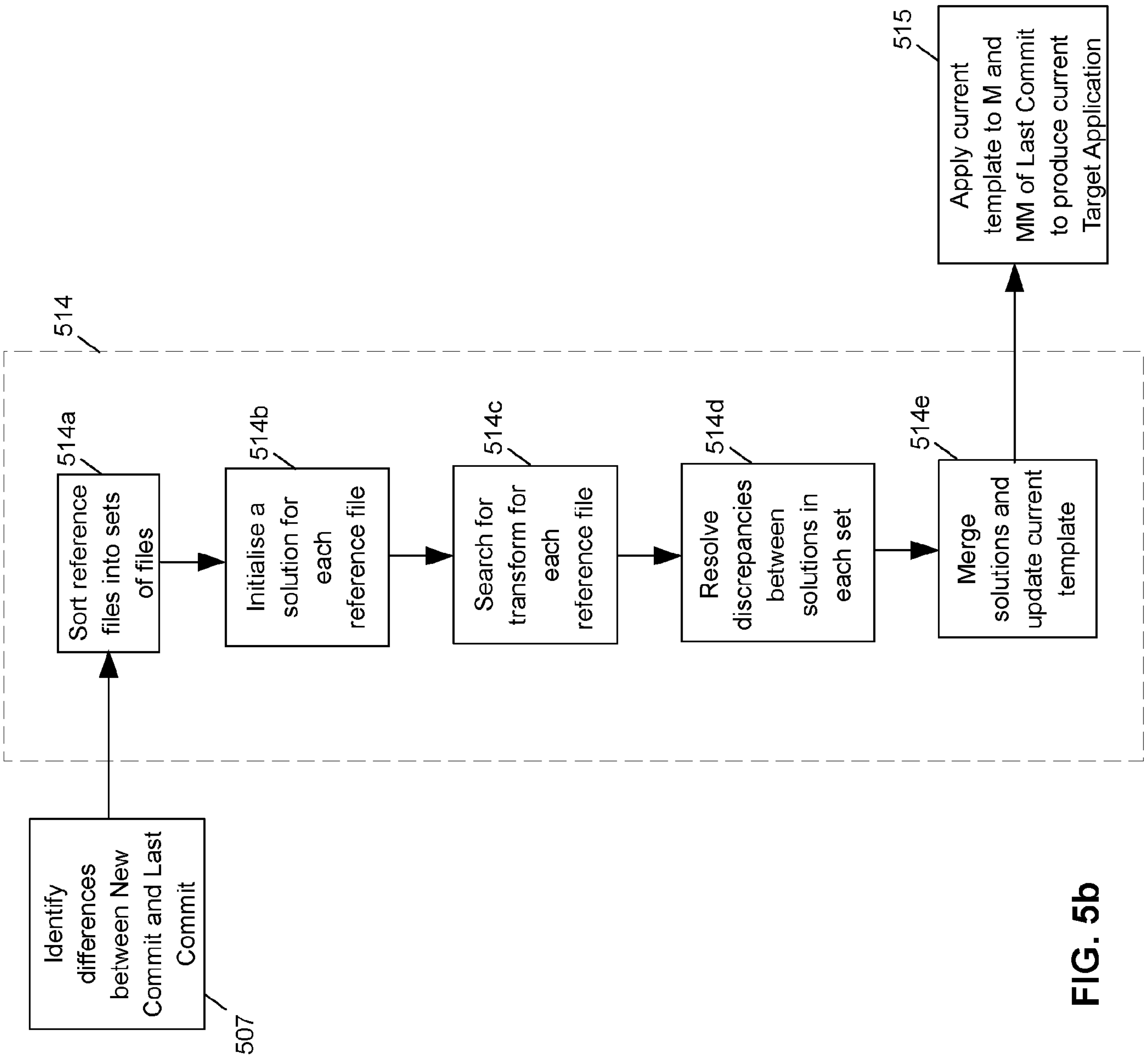

Identify differences between New Commit and Last Commit
507
514
Sort reference files into sets of files
514a
Initialise a solution for each reference file
514b
Search for transform for each reference file
514c
Resolve discrepancies between solutions in each set
514d
Merge solutions and update current template
514e
Apply current template to M and MM of Last Commit to produce current Target Application
515

FIG. 5b

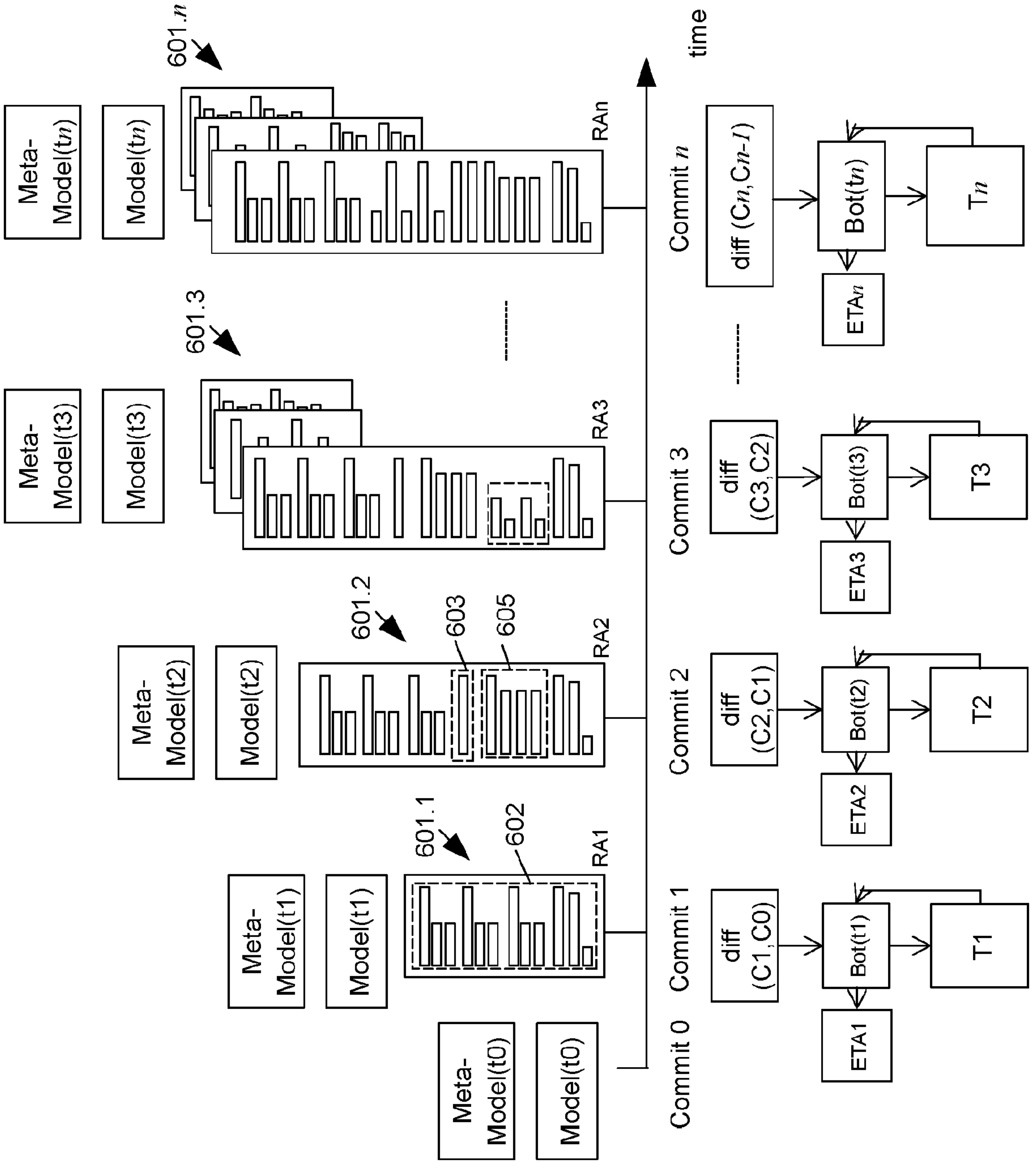

Meta-Model(t0)
Model(t0)
Commit 0
Meta-Model(t1)
Model(t1)
601.1
602
RA1
Commit 1
diff (C1,C0)
Bot(t1)
ETA1
T1
Meta-Model(t2)
Model(t2)
601.2
603
605
RA2
Commit 2
diff (C2,C1)
Bot(t2)
ETA2
T2
Meta-Model(t3)
Model(t3)
601.3
RA3
Commit 3
diff (C3,C2)
Bot(t3)
ETA3
T3
Meta-Model(tn)
Model(tn)
601.n
RAn
Commit n
diff (Cn,Cn-1)
Bot(tn)
ETAn
Tn
time

FIG. 6

```
public String getYellow() {
    return yellow;
}
public void setYellow(String yellow) {
    this.yellow = yellow;
}
public String getGreen() {
    return green;
}
public void setGreen(String green) {
    this.green = green;
}
public String getBlue() {
    return blue;
}
public void setBlue(String blue) {
    this.blue = blue;
}
```

FIG. 7

```
[% for (attribute in c.feature) {%]
    public String get[%=attribute.name%]() {
        return [%=attribute.name%];
    }
    public void set[%=attribute.name%]([String new[%=attribute.name%]) {
        this.[%=attribute.name%] = new[%=attribute.name%];
    }
[%}%]
}
```

FIG. 8

```
public String getYellow() {
    return yellow;
}
public String getGreen() {
    return green;
}
public String getBlue() {
    return blue;
}
public void setYellow(String yellow) {
    this.yellow = yellow;
}
public void setGreen(String green) {
    this.green = green;
}
public void setBlue(String blue) {
    this.blue = blue;
}
```

FIG. 9

```
[% for (attribute in c.feature) {%]
    public String get[%=attribute.name%]() {
        return [%=attribute.name%];
    }%]
[% for (attribute in c.feature) {%]
    public void set[%=attribute.name%]([String new[%=attribute.name%]) {
        this.[%=attribute.name%] = new[%=attribute.name%];
    }
```

FIG. 10

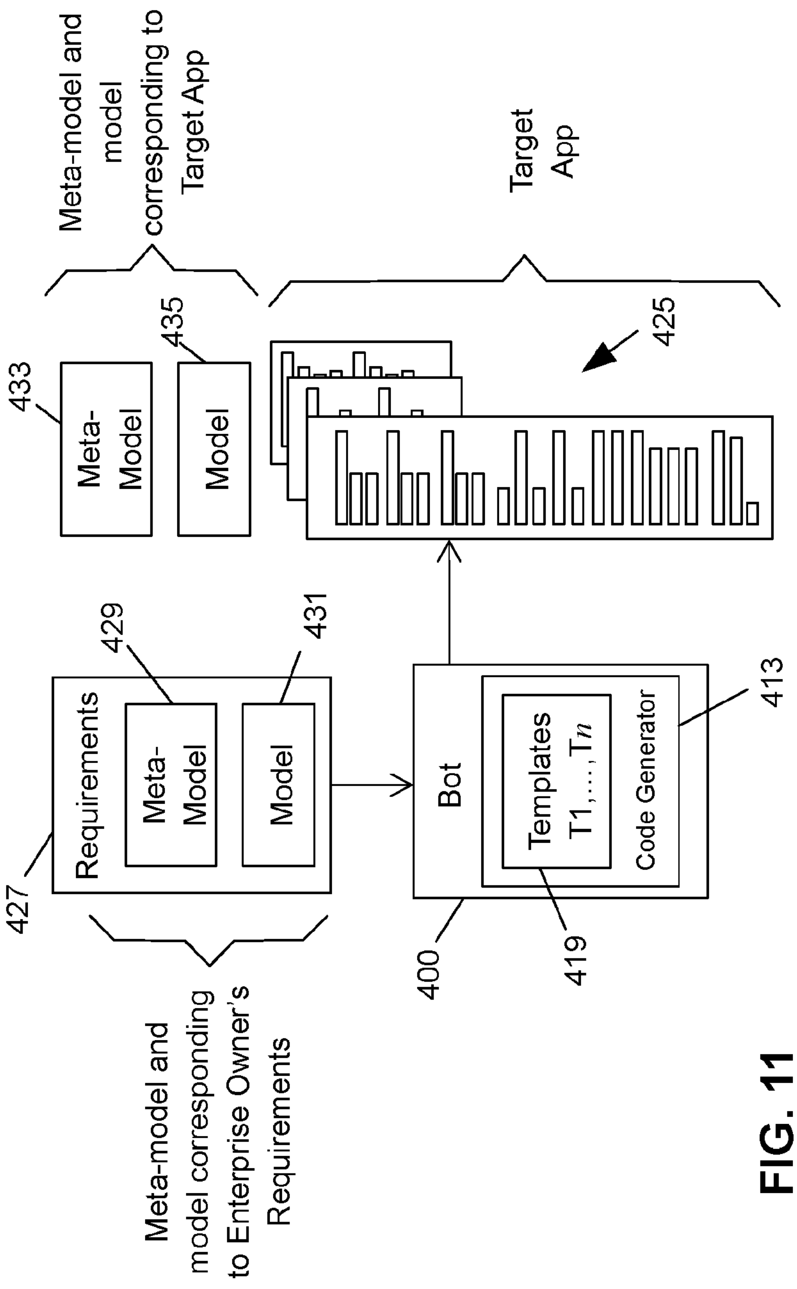

Meta-model and model corresponding to Target App
433
Meta-Model
435
Model
Target App
425
427
Requirements
429
Meta-Model
431
Model
Bot
Templates T1,…,Tn
Code Generator
413
400
419
Meta-model and model corresponding to Enterprise Owner's Requirements

FIG. 11

INTERNATIONAL SEARCH REPORT

International application No.
PCT/AU2022/050373

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/455 (2018.01) G06F 8/35 (2018.01) G06F 8/40 (2018.01) G06N 3/00 (2006.01)

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)

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

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

EPOQUE PATENW: IPC/CPC - G06F8/35, G06F8/40, G06F9/455, G06N3/006; Keywords - agent, application, build, change, code, commit, compare, compile, create, current, difference, find, generate, identify, intelligent, meta meta, meta model, model, previous, revision, search, target, template, template, variance, version, virtual & similar terms.
Google Patents & Scholar: Keywords - G06F8/35, G06F8/355, G06F8/40, G06N3/006, bot, changes, code, commit, difference, generation, map, meta, model, template, transform, version, "code generation", "intelligent agent"
Applicant & Inventor Name Searches: Performed in Espacenet, Google & IP Australia's internal databases.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
	Documents are listed in the continuation of Box C	

[X] Further documents are listed in the continuation of Box C [X] 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	"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
"D"	document cited by the applicant in the international application		
"E"	earlier application or patent but published on or after the international filing date	"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
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"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
"O"	document referring to an oral disclosure, use, exhibition or other means	"&"	document member of the same patent family
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search	Date of mailing of the international search report
5 July 2022	05 July 2022

Name and mailing address of the ISA/AU	**Authorised officer**
AUSTRALIAN PATENT OFFICE PO BOX 200, WODEN ACT 2606, AUSTRALIA Email address: pct@ipaustralia.gov.au	Ravi McCosker AUSTRALIAN PATENT OFFICE (ISO 9001 Quality Certified Service) Telephone No. +61262832933

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
	US 20150149979 A1 (AMAZON TECHNOLOGIES INC) 28 May 2015	
X	Entire document, especially: Para. 23-24, 29, 32-33 ,36, 44, 46, 49-50.	1-15, 17-18
Y	Entire document, especially: Para. 23-24, 29, 31, 44, 46, 50.	16
	US 20200160187 A1 (E & K ESCOTT HOLDINGS PTY LTD) 21 May 2020	
Y	Entire document, especially: Para. 1, 4, 14, 22, 28	16
A		
A	US 20100287528 A1 (LOCHMANN) 11 November 2010	
A	US 20200401382 A1 (THE ULTIMATE SOFTWARE GROUP INC) 24 December 2020	
A	US 20180074804 A1 (SMARTSHIFT TECHNOLOGIES INC) 15 March 2018	
A	US 20210073110 A1 (SAUCE LABS INC) 11 March 2021	
A	US 2020/0257613 A1 (FUJITSU LIMITED) 13 August 2020 Entire document	
A	GASCUENA, J.M. et al. "Model-driven engineering techniques for the development of multi-agent systems", Engineering Applications of Artificial Intelligence 25, No. 1 (2012): pp 159-173. Entire document	

INTERNATIONAL SEARCH REPORT Information on patent family members	International application No. **PCT/AU2022/050373**

This Annex lists known patent family members relating to the patent documents cited in the above-mentioned international search report. The Australian Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

Patent Document/s Cited in Search Report		**Patent Family Member/s**	
Publication Number	**Publication Date**	**Publication Number**	**Publication Date**
US 20150149979 A1	28 May 2015	US 2015149979 A1	28 May 2015
		US 9946517 B2	17 Apr 2018
		US 8949772 B1	03 Feb 2015
US 20200160187 A1	21 May 2020	US 2020160187 A1	21 May 2020
		AU 2018280354 A1	01 Aug 2019
		AU 2018280354 B2	19 Sep 2019
		EP 3635570 A1	15 Apr 2020
		WO 2018223196 A1	13 Dec 2018
US 20100287528 A1	11 November 2010	US 2010287528 A1	11 Nov 2010
		US 8448132 B2	21 May 2013
		EP 2249249 A1	10 Nov 2010
		EP 2249249 B1	15 Feb 2012
US 20200401382 A1	24 December 2020	US 2020401382 A1	24 Dec 2020
		US 2019266076 A1	29 Aug 2019
		US 10769056 B2	08 Sep 2020
US 20180074804 A1	15 March 2018	US 2018074804 A1	15 Mar 2018
		US 10481884 B2	19 Nov 2019
		US 2017090892 A1	30 Mar 2017
		US 9811325 B2	07 Nov 2017
US 20210073110 A1	11 March 2021	US 2021073110 A1	11 Mar 2021
		US 11042472 B2	22 Jun 2021
US 2020/0257613 A1	13 August 2020	US 2020257613 A1	13 Aug 2020
		US 10761962 B1	01 Sep 2020
		JP 2020129372 A	27 Aug 2020

End of Annex

Due to data integration issues this family listing may not include 10 digit Australian applications filed since May 2001.