Office de la Propriété Intellectuelle du Canada
Un organisme d'Industrie Canada

Canadian Intellectual Property Office
An agency of Industry Canada

CA 2956364 A1 2017/08/02

(21) 2 956 364

(12) DEMANDE DE BREVET CANADIEN
CANADIAN PATENT APPLICATION

(13) A1

(22) Date de dépôt/Filing Date: 2017/01/26

(41) Mise à la disp. pub./Open to Public Insp.: 2017/08/02

(30) Priorité/Priority: 2016/02/02 (US15/013,391)

(51) Cl.Int./Int.Cl. *G06F 11/36* (2006.01)

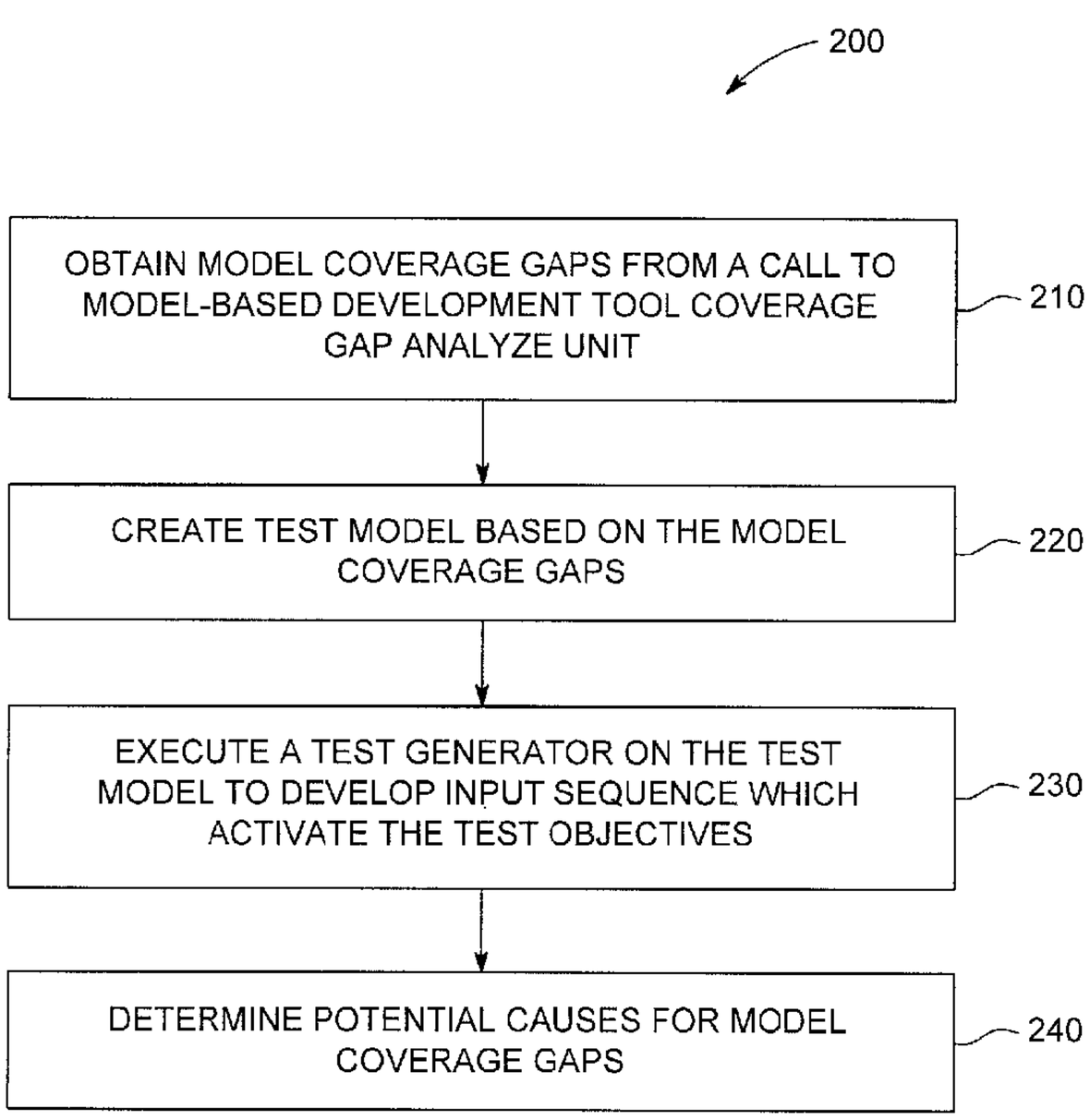
(71) Demandeur/Applicant:
GENERAL ELECTRIC COMPANY, US

(72) Inventeurs/Inventors:
LI, MENG, US;
DURLING, MICHAEL RICHARD, US;
DAI, JIAN, US;
STACEY, SCOTT ALAN, US

(74) Agent: CRAIG WILSON AND COMPANY

(54) Titre : SYSTEME ET METHODE D'AUGMENTATION DE TEST ELEMENTAIRE AUTOMATISE FONDE SUR LA COUVERTURE DESTINES A DES MODELES D'INGENIERIE

(54) Title: SYSTEM AND METHOD FOR COVERAGE-BASED AUTOMATED TEST CASE AUGMENTATION FOR DESIGN MODELS



(57) Abrégé/Abstract:
A method for automated test case augmentation includes receiving, at an automated test augmentation system, a design model and model coverage gap information from a model-based development tool, translating the model coverage gap information into

(21) 2 956 364

(13) A1

(57) Abrégé(suite)/Abstract(continued):
machine-readable mathematical test objective expressions, developing a set of test objective operators by translating the machine-readable mathematical test objective expressions, localizing target operators for the identified coverage gaps within the design model, attaching the test objective operators to target operators of the design model to create a test model, augmenting the test model by propagating test objectives at the target operators to a test node operator of the design model, and executing, by a test generator, the augmented test model to obtain the test cases to cover the coverage gaps and the causes for the model coverage gaps. A system for implementing the model-based design and a non-transitory computer readable medium are also disclosed.

280257

ABSTRACT

A method for automated test case augmentation includes receiving, at an automated test augmentation system, a design model and model coverage gap information from a model-based development tool, translating the model coverage gap information into machine-readable mathematical test objective expressions, developing a set of test objective operators by translating the machine-readable mathematical test objective expressions, localizing target operators for the identified coverage gaps within the design model, attaching the test objective operators to target operators of the design model to create a test model, augmenting the test model by propagating test objectives at the target operators to a test node operator of the design model, and executing, by a test generator, the augmented test model to obtain the test cases to cover the coverage gaps and the causes for the model coverage gaps. A system for implementing the model-based design and a non-transitory computer readable medium are also disclosed.

280257

SYSTEM AND METHOD FOR COVERAGE-BASED AUTOMATED TEST CASE AUGMENTATION FOR DESIGN MODELS

BACKGROUND

[0001] The present disclosure relates to systems and methods for coverage-based automated test case augmentation for design models.

[0002] Available model-based development tools can show un-reachability of certain model elements. Some tools can generate test inputs that satisfy standard coverage objectives as well as user-defined test objectives and requirements. These test inputs can also be combined with tests defined using measured data so that simulations are testing against model coverage, requirements, and real-world scenarios.

[0003] However, some important coverage criteria (e.g., mask modified condition/decision coverage (MC/DC)) are not supported by available model-based development tools and redundant test cases are generated to satisfy certain coverage criteria. Further, some conventional development tools do not provide the capability of augmenting test cases to satisfy coverage criteria.

[0004] Software certification standards (e.g., DO-178C) require safety-critical software, such as aviation software, to be tested with strict test coverage (including MC/DC). These standards can require that each condition that could independently affect the decision be tested. Manual inspection of the model/code to identify the inputs sequences that drive an internal variable to a particular value is hard and time-consuming, especially when the aviation software system is large and complex.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] FIG. 1 depicts automated test case augmentation system in accordance with embodiments;

[0006] FIG. 2 depicts a process flowchart for automated design model test case augmentation in accordance with embodiments;

[0007] FIG. 3 depicts a process flowchart for test model creation based on model coverage gaps in accordance with embodiments;

[0008] FIG. 4 depicts a user interface for the system of FIG. 1 in accordance with embodiments; and

[0009] FIG. 5 depicts a design model and coverage analysis report in accordance with embodiments.

DESCRIPTION

[0010] In accordance with embodiments, systems and methods provide support to the development of safety-critical software in a model-based development environment. Model test coverage gaps can be identified by model coverage analysis tools of the model-based development environment after the high-level requirements-based test cases are executed against the design model. The coverage gaps indicate the test criteria that have not been exercised by the high-level requirements-based test cases. Embodying systems and methods can automatically augment the test cases to cover the coverage gaps. Embodying systems and methods translate coverage gap information to machine-readable mathematical test objectives so that coverage gap information can be attached to a design model for automated test case augmentation, where the design model(s) is written in model-based development languages.

[0011] Embodying systems and methods employ a test generator, which can perform model-checking, constraint solving, and/ or reachability resolution technologies on the design model(s) with test objectives attached, to automatically identify and fill test coverage gaps for the design model(s) of the safety-critical software. Additionally, embodying systems and methods can also identify model deficiencies — such as dead code, unintended functions, deactivated functions, etc. Embodying systems and methods can

implement a coverage gap conversion rule that can convert the coverage gaps to test objectives. If an input sequence can activate the test objective, the input sequence can fill the coverage gap. The test objectives along with the design model are analyzed in test generators to generate test cases. Several coverage gap patterns are identified and their conversion rules are defined. Model test coverage gaps for design models developed in various conventional model-based development tools can be filled.

[0012] Safety-critical software, such as aviation software, are required by software certification standards (e.g. DO-178C) to be tested with strict test coverage, such as Modified Condition/Decision Coverage (MC/DC) which requires each condition to independently affect the decision. Manual inspection of the model/code to identify the inputs sequences that drive an internal variable to a particular value is hard and time-consuming, especially when the aviation software size is large and the complexity is growing.

[0013] Different coverage criteria are supported (e.g., Statement Coverage, Decision Coverage, MC/DC, masking MC/DC, State Coverage, Transition Coverage, etc.) with one or more conversion rules being implemented to accommodate the translation of the different criteria from coverage gap information to machine-readable mathematical test objectives. Embodying systems and methods can recognize the criteria, locate target design/code, perform test objective translation and attachment, propagate test objectives through the design architecture, and generate test cases based on the test objectives.

[0014] Figure 1 depicts automated test case augmentation system 100 for design model in accordance with embodiments. System 100 includes control processor 110 which executes computer instructions to control the operation of the system and its components. Control processor 110 can be located in a computer, or a server, and interconnected to the various components via communication link 120. The communication link can be an internal bus, an electronic communication network, or the like.

[0015] System 100 can generate augmented test cases to satisfy model coverage based on received design model 130 of the safety-critical software, along with received model coverage gaps 135. The design models can be written in conventional, model-based development languages, such as Simulink/Stateflow, SCADE, etc.

[0016] Gap converter unit 140 is structured to convert the model coverage gaps to test objectives 152 based on coverage gap conversion rules 158, which can be stored in data store 150. The test objectives define and/or set the goals for test generator unit 160. The test objectives are then attached to the corresponding operators in the design model to create test model 156. The test generator applies the test model as input from which it develops a set of test cases 154, which achieve the test objectives.

[0017] Figure 2 depicts process 200 for automated design model test case augmentation in accordance with embodiments. Model coverage gaps can be obtained, step 210, from a call to a model coverage gap analyzer unit 180 of the model-based development tool. A test model is created, step 220, based on design model 130 and model coverage gaps 135. A test generator is executed, step 230, on the test model driving the inputs to cover the model coverage gaps. Based on the results of test generator execution, the potential causes of the model coverage gaps are determined, step 240.

[0018] Figure 3 depicts process 300 for test model creation based on model coverage gaps (FIG. 2, step 220) in accordance with embodiments. The test models are created based on design model 130 and model coverage gaps 135. Coverage gap analyzer unit 180 identifies, step 305, coverage gaps in the design model. The coverage gaps are categorized and based on the categorization translated, step 310, into machine-readable, mathematical, test objective expressions. In accordance with implementations, the conversion can be achieved using a criteria conversion table, which are pre-created for each type of coverage criteria. Table I is an example of conversions from coverage gaps to test objective expressions in accordance with embodiments.

280257

Missing Coverage	Operator	Criteria	Test Objective Expression
1	Pwlinear::Counter/$ifthenelse1	True	IF#1 condition=true
2	RollCommandValidity::RollCommandValidity/$ifthenelse10	False	IF#10 condition=false
3	RollCommandValidity::RollCommandValidity/$ifthenelse8	True	IF#8 condition=true
4	RollCommandValidity::RollCommandValidity/IfBlock1:else:else:else:	Activated	Reachability(IfBlock1:ELSE:ELSE:ELSE)=true

Table I

[0019] Criteria conversion table categorizes coverage criteria patterns and define test objectives for each of the criteria patterns. The coverage criteria conversion can support different coverage criteria, such as Statement Coverage, Decision Coverage, Modified Condition/Decision Coverage (MC/DC), masking MC/DC, State Coverage, Transition Coverage, etc.

[0020] The translated mathematical test objective expressions are then translated, step 315, into a set of test objective operators which can be attached to the design model.

[0021] The test objective operator(s) are attached to target operator(s) in the design model. The target operator(s) are localized, step 320, based on the identified coverage gaps within the design model. In accordance with implementations, to achieve localization the coverage gap information can be parsed to obtain the coverage gap target operator. The

coverage gap information can include where the gap occurs, and what criteria is not satisfied.

[0022] Further, the test objective operators are attached, step 325, to connect the test objective corresponding signals in the target operator(s) to create the test model. The test objectives are also propagated, step 330, all the way to a test node operator which may be at a higher level in the model hierarchy, so that the test objective can be visible at the test node operator level. The test generator recognizes the test objectives of the test model and finds input sequences at the test node operator level to activate the test objective and cover the corresponding model coverage gaps.

[0023] Embodying systems and processes can determine the potential causes of the coverage gaps based on the test generator results. Determination can be made for the following conditions along with some design information:

[0024] 1) If a test objective is identified as reachable (i.e., an input sequence can be found to activate the test objective) and the source of the corresponding coverage gap does not trace back to a high level requirement or derived requirement, then the corresponding coverage gap is caused by insufficient high level or derived requirements;

[0025] 2) If a test objective is reachable and it traces to a high level requirement, then the corresponding coverage is missing because of the inadequate high level requirements-based tests;

[0026] 3) If a test objective is reachable and it traces to a derived requirement specified by the software designer, then the corresponding coverage is missing because of the derived requirements.

[0027] 4) If a test objective is unreachable (i.e., mathematically proven that no possible input sequences can be found to activate the test objective), then the test generator can execute again by including design model parameters as inputs to the design model. In this

case, test generator can generate updated design model parameters that include changes to close the coverage gap associated with the input design model parameter.

[0028] 4a) If a test objective is still identified as unreachable, then the coverage gap is unintended functions or dead code;

[0029] 4b) If the coverage gap is identified as reachable, then the coverage gap is a deactivated function. Design model parameter values can be generated that make the coverage gap reachable.

[0030] Figure 4 depicts user interface 400 of system 100 in accordance with embodiments. The user interface can be an interactive graphical interface with multiple panes. Users can select the report including model coverage gaps by clicking "OPEN MTC REPORT" button. Missing items pane 410 displays a listing of design model coverage gaps in the selected report. Users can select a subset of coverage gaps to generate test cases. Test case pane 420 displays the generated test cases as the result of system 100 to cover the selected coverage gaps in pane 410. If the corresponding coverage gap is reachable, then the directory of the generated test case is displayed. If the corresponding coverage gap is unreachable, then "UNREACHABLE" is displayed. Status pane 430 displays status reports from system 100 as it performs its automated design model test case augmentation process.

[0031] Embodying systems and methods receive coverage gap analyzer unit reports from model-based development tools and convert identified model test coverage gaps in the report to machine-readable test objective expressions. These test objective expressions are then automatically converted into test objective operators which are attached to appropriate places in the design model itself to create a test model. A test generator is applied on the test model to generate test cases to cover the coverage gaps and identify design deficiencies.

[0032] Figure 5 depicts an example of design model and coverage analysis report 500 in accordance with embodiments. Included in the analysis report is a logic flow diagram of the design model. The logic flow diagram is annotated by automated test case augmentation system 100 to indicate test coverage gap locations 510, 520, 530 where true, or false, values were not tested for the indicated logic flow. The analysis report is also annotated as a result of the test case augmentation system operation to indicate that conditional branches of the logic flow were not tested — for example, "else" block 540 is indicated as not tested. In accordance with embodiments, a coverage analysis report can be in tabular form as shown in Table II:

Missing Coverage	Operator	Criteria
1	Pwlinear::Counter/$ifthenelse1	True
2	RollCommandValidity::RollCommandValidity/$ifthenelse10	False
3	RollCommandValidity::RollCommand Validity/$ifthenelse8	True
4	RollCommandValidity::RollCommandValidity/IfBlock1:else: else:else:	Activated

Table II

[0033] In accordance with embodiments, in masking MC/DC rules can be followed when the criterion is a path starting from an operator and ending at an operator through a sequence of operators. For example, local variables can connect paths from the end of one path to the beginning of another path; and an output variable can connect paths from the end of one path to the beginning of another path by using a textual expression. In accordance with embodiments, test generator 160 can receive a trap condition (negation of test objective) and attempt to find a counterexample for the trap condition. If a counterexample is found, the counterexample is the test case that satisfies the test objective.

If no counterexample is found, the test objective is unreachable meaning no test cases can cover the corresponding coverage gap. In such an instance, the coverage criteria can be converted into test objective expressions, which are translated into trap condition blocks attached in the model.

[0034] In accordance with some embodiments, a computer program application stored in non-volatile memory, computer-readable medium (e.g., register memory, processor cache, RAM, ROM, hard drive, flash memory, CD ROM, magnetic media, etc.), and/or external memory 155 may include code or executable instructions that when executed may instruct and/or cause a controller or processor to perform methods discussed herein such as a method for the development of safety-critical software in a model-based development environment, as described above.

[0035] The computer-readable medium may be a non-transitory computer-readable media including all forms and types of memory and all computer-readable media except for a transitory, propagating signal. In one implementation, the non-volatile memory or computer-readable medium may be external memory.

[0036] While there have been described herein what are considered to be preferred and exemplary embodiments of the present invention, other modifications of these embodiments falling within the scope of the invention described herein shall be apparent to those skilled in the art.

280257

WHAT IS CLAIMED IS:

1. A system for automated test case augmentation, the system comprising:

a computer including a control processor, the control processor configured to execute program instruction;

a gap converter unit linked to the control processor, the gap converter unit configured to convert one or more model coverage gaps of a design model to mathematical test objective expressions;

a test generator unit linked to the control processor, the test generator unit configured to develop a set of test cases from the test objectives within a test model;

the gap converter unit further configured to translate the mathematical test objective expressions to test objective operators, and to create a test model by attaching the test objective operators to corresponding operators in the design model.

2. The system of claim 1, configured to receive the model coverage gaps from a model-based development tool.

3. The system of claim 1, the test objectives defining one or more testing goals.

4. The system of claim 1, the test objective operators being test objective operators that can be attached to the design model.

5. The system of claim 1, including a data store for storing test objectives, test models, and coverage gap conversion rules.

6. The system of claim 1, the control processor further configured to execute program instructions that cause the control processor to:

augment the test model by propagating test objectives at the target operators to the test node operator of the design model; and

execute the test generator on the augmented test model to obtain the test cases to cover the coverage gaps and the causes for the model coverage gaps.

7. The system of claim 6, the control processor further configured to execute program instructions that cause the control processor to translate coverage criteria translation by using a criteria conversion table.

8. The system of claim 7, the control processor further configured to execute program instructions that cause the control processor to obtain from the model-based development tool the criteria conversion table.

9. The system of claim 6, the control processor further configured to execute program instructions that cause the control processor to:

at least one of categorize and identify the coverage criteria; and

generate one or more conversion rules for the coverage criteria.

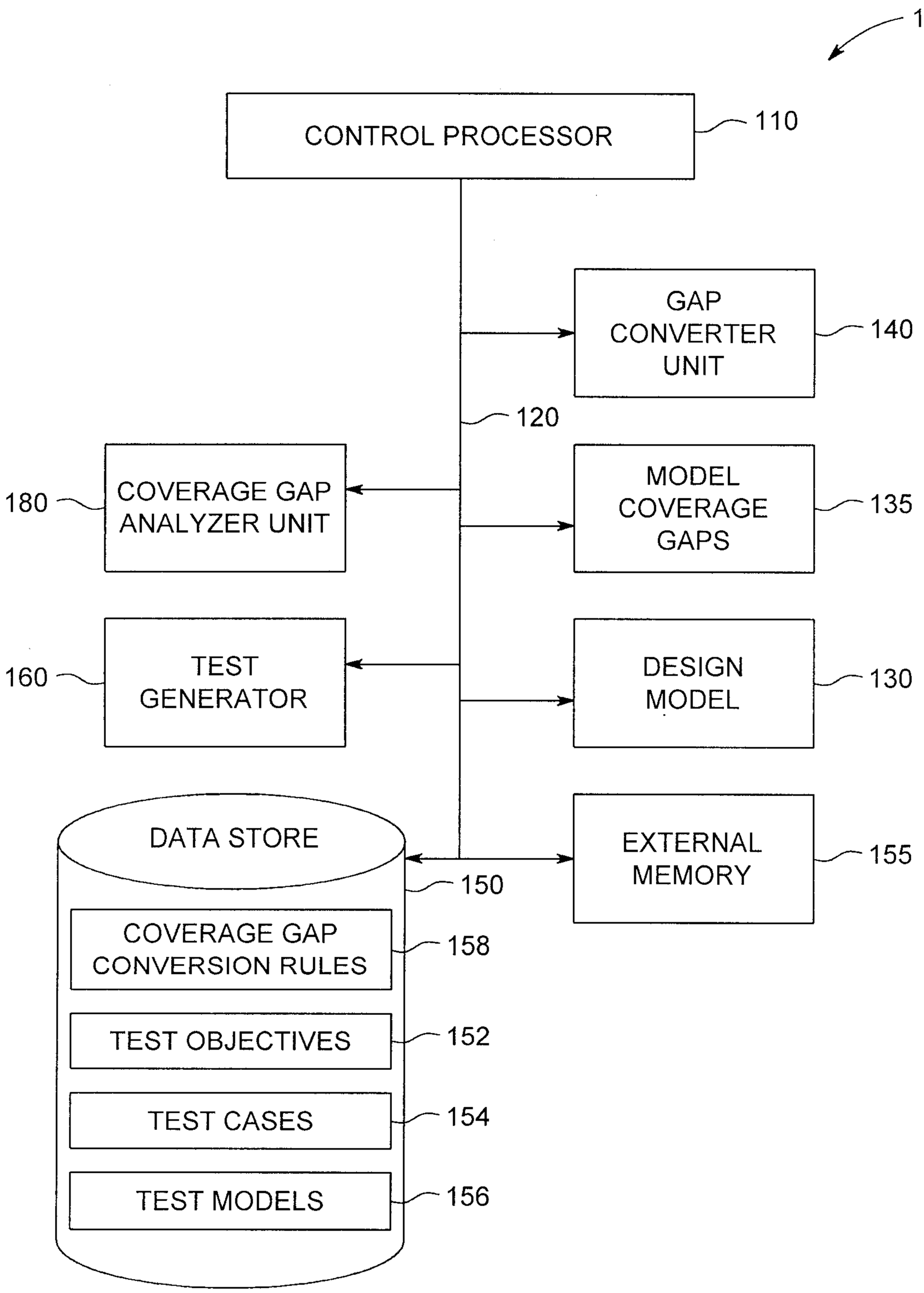

100
CONTROL PROCESSOR
110
GAP CONVERTER UNIT
140
120
COVERAGE GAP ANALYZER UNIT
180
MODEL COVERAGE GAPS
135
TEST GENERATOR
160
DESIGN MODEL
130
DATA STORE
150
EXTERNAL MEMORY
155
COVERAGE GAP CONVERSION RULES
158
TEST OBJECTIVES
152
TEST CASES
154
TEST MODELS
156

FIG. 1

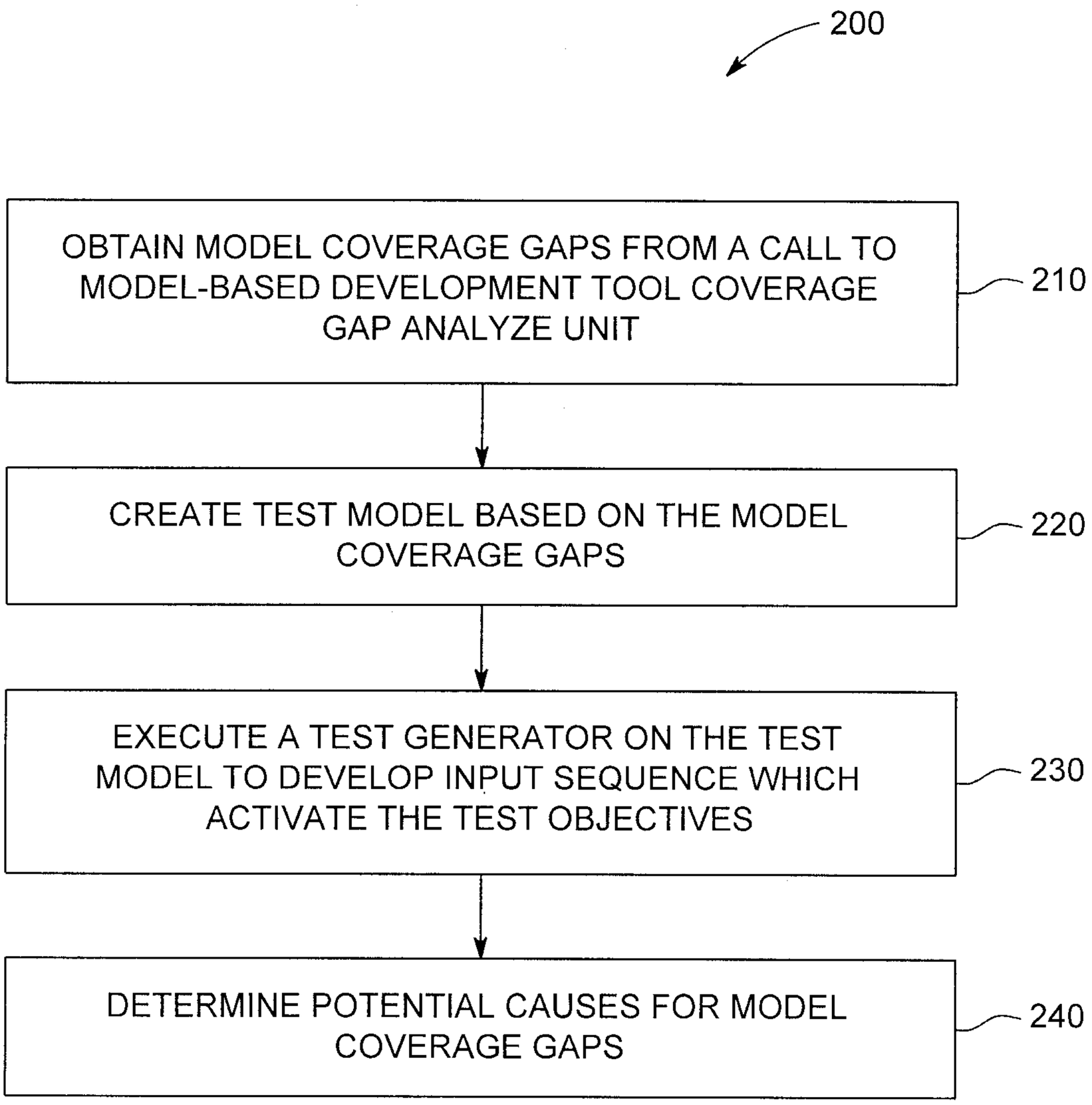

200
OBTAIN MODEL COVERAGE GAPS FROM A CALL TO MODEL-BASED DEVELOPMENT TOOL COVERAGE GAP ANALYZE UNIT
210
CREATE TEST MODEL BASED ON THE MODEL COVERAGE GAPS
220
EXECUTE A TEST GENERATOR ON THE TEST MODEL TO DEVELOP INPUT SEQUENCE WHICH ACTIVATE THE TEST OBJECTIVES
230
DETERMINE POTENTIAL CAUSES FOR MODEL COVERAGE GAPS
240

FIG. 2

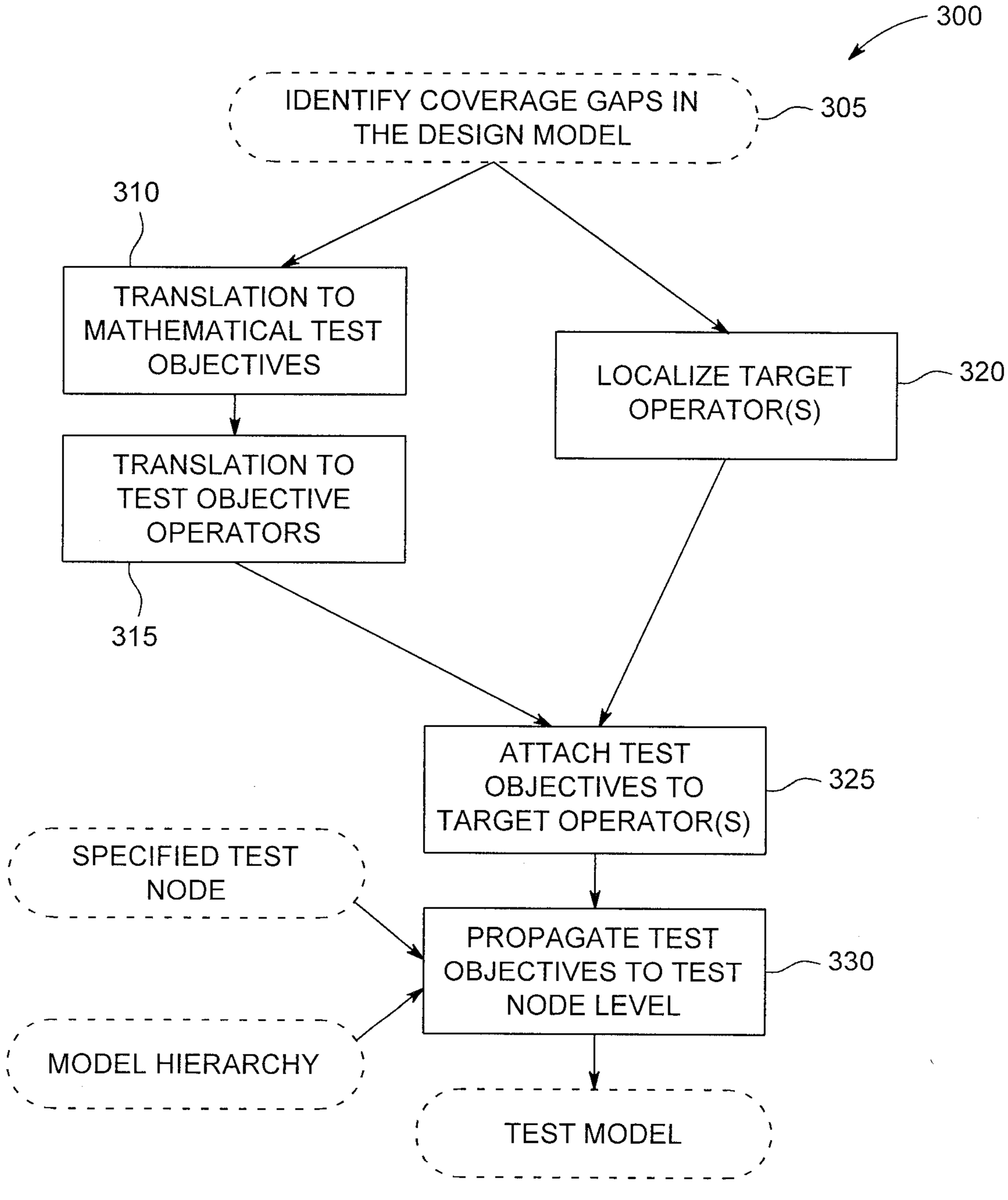

300
IDENTIFY COVERAGE GAPS IN THE DESIGN MODEL
305
310
TRANSLATION TO MATHEMATICAL TEST OBJECTIVES
TRANSLATION TO TEST OBJECTIVE OPERATORS
315
LOCALIZE TARGET OPERATOR(S)
320
ATTACH TEST OBJECTIVES TO TARGET OPERATOR(S)
325
SPECIFIED TEST NODE
MODEL HIERARCHY
PROPAGATE TEST OBJECTIVES TO TEST NODE LEVEL
330
TEST MODEL

FIG. 3

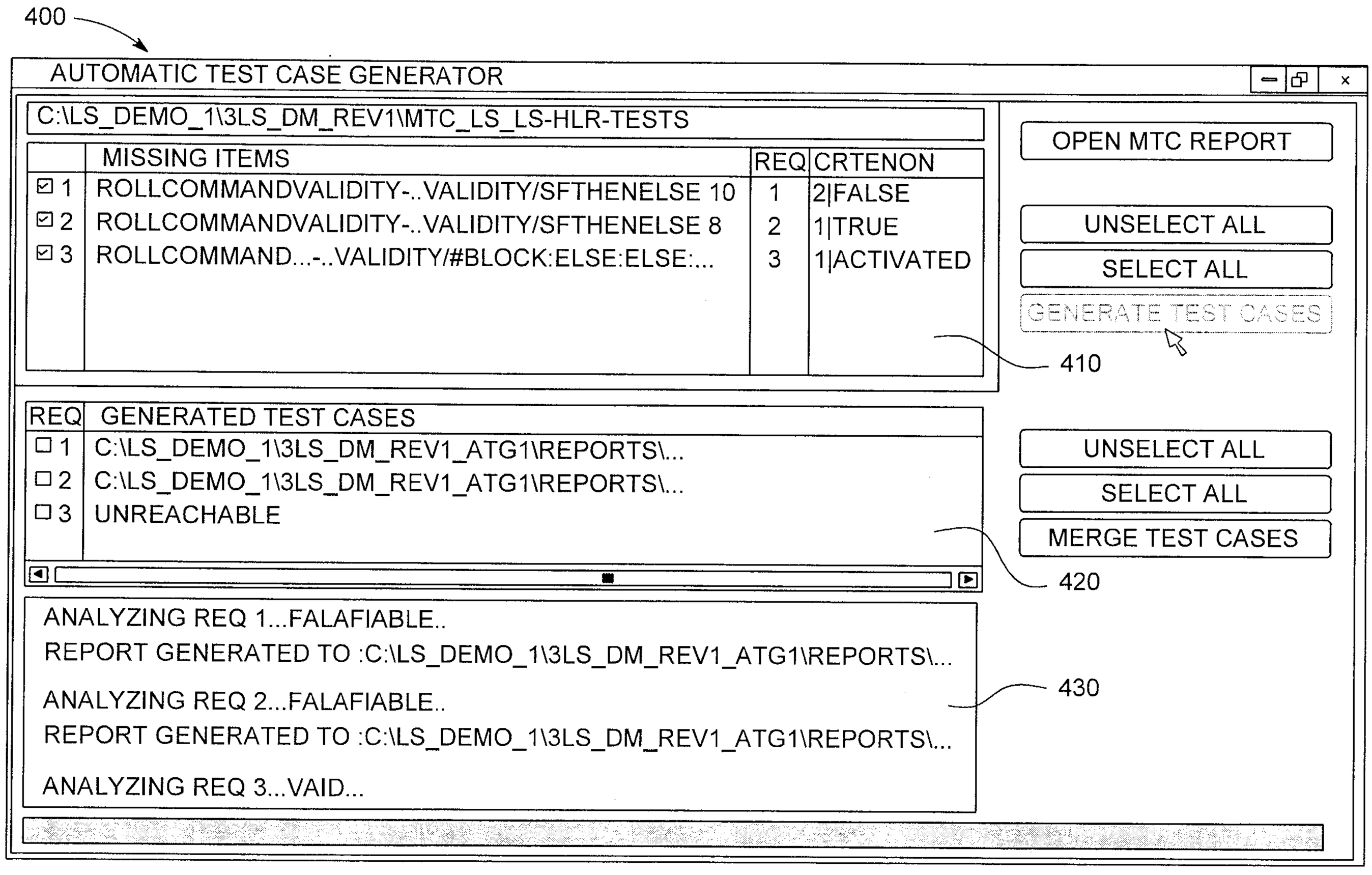


FIG. 4

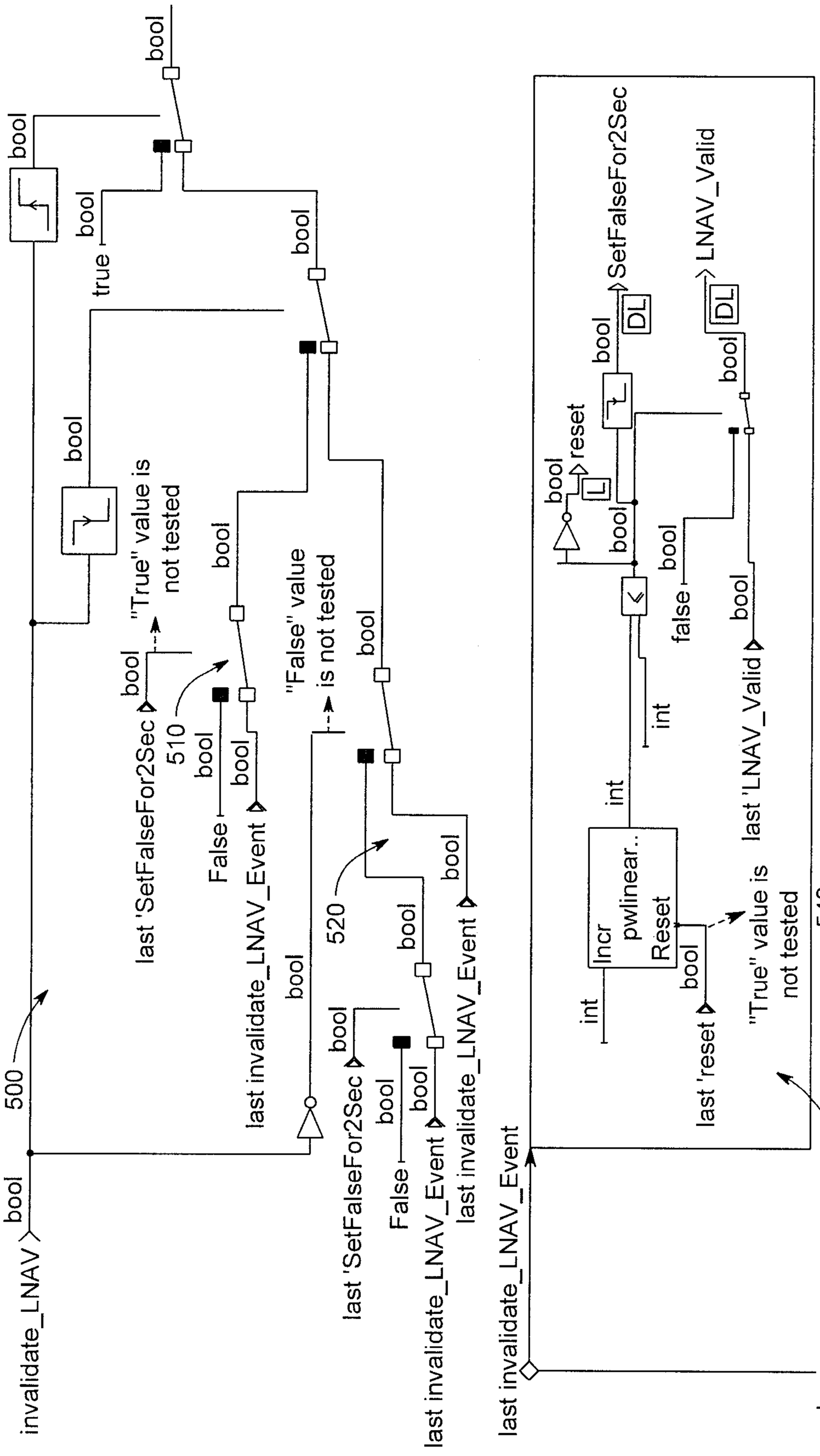
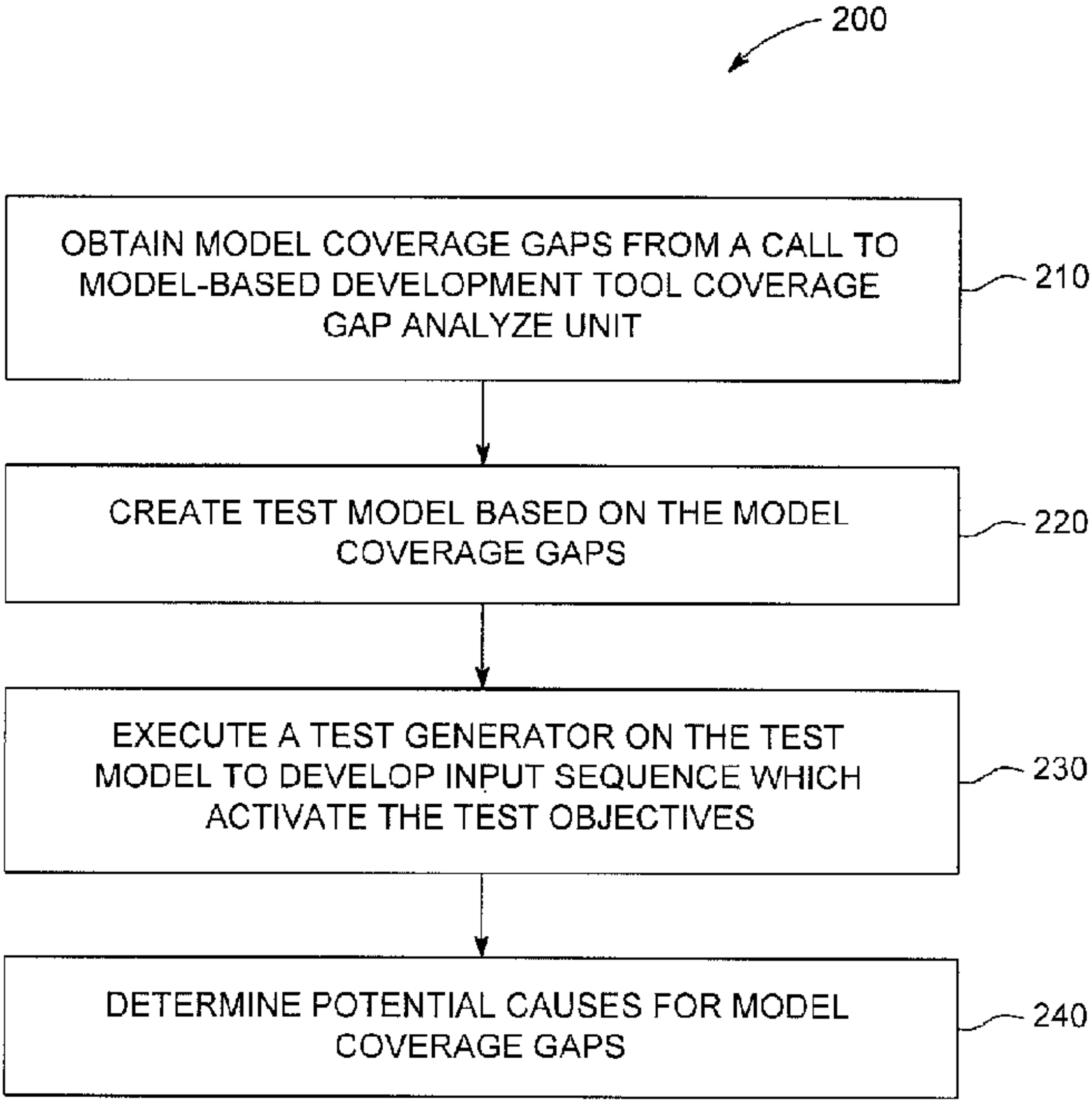

invalidate_LNAV
bool
500
last 'SetFalseFor2Sec
bool
False
last invalidate_LNAV_Event
bool
last 'SetFalseFor2Sec
bool
510
bool
bool
"True" value is not tested
bool
bool
"False" value is not tested
bool
520
bool
bool
last invalidate_LNAV_Event
bool
last invalidate_LNAV_Event
bool
reset
L
bool
DL
SetFalseFor2Sec
int
Incr
pwlinear..
Reset
bool
last 'reset
bool
false
int
bool
DL
LNAV_Valid
last 'LNAV_Valid
bool
"True" value is not tested
530
bool
False
"else" block is not tested
540
bool
DL
LNAV_Valid

FIG. 5


200
OBTAIN MODEL COVERAGE GAPS FROM A CALL TO MODEL-BASED DEVELOPMENT TOOL COVERAGE GAP ANALYZE UNIT
210
CREATE TEST MODEL BASED ON THE MODEL COVERAGE GAPS
220
EXECUTE A TEST GENERATOR ON THE TEST MODEL TO DEVELOP INPUT SEQUENCE WHICH ACTIVATE THE TEST OBJECTIVES
230
DETERMINE POTENTIAL CAUSES FOR MODEL COVERAGE GAPS
240