(12) 特許協力条約に基づいて公開された国際出願

(19) 世界知的所有権機関
国際事務局

(43) 国際公開日
2012 年 5 月 3 日(03.05.2012)

PCT

(10) 国際公開番号
WO 2012/057170 A1

(51) 国際特許分類:
G06F 11/36 (2006.01)

(21) 国際出願番号: PCT/JP2011/074595

(22) 国際出願日: 2011 年 10 月 25 日(25.10.2011)

(25) 国際出願の言語: 日本語

(26) 国際公開の言語: 日本語

(30) 優先権データ:
特願 2010-240775 2010 年 10 月 27 日(27.10.2010) JP

(71) 出願人 (米国を除く全ての指定国について): 株式会社日立製作所 **(HITACHI, LTD.)** [JP/JP]; 〒1008280 東京都千代田区丸の内一丁目６番６号 Tokyo (JP).

(72) 発明者; および

(75) 発明者/出願人 (米国についてのみ): 近久　真章 **(CHIKAHISA Masaki)** [JP/JP]; 〒1858601 東京都国分寺市東恋ヶ窪一丁目２８０番地　株式会社日立製作所　中央研究所内 Tokyo (JP). 市井　誠 **(ICHII Makoto)** [JP/JP]; 〒1858601 東京都国分寺市東恋ヶ窪一丁目２８０番地　株式会社日立製作所　中央研究所内 Tokyo (JP). 野口　秀人 **(NOGUCHI Hideto)** [JP/JP]; 〒1858601 東京都国分寺市東恋ヶ窪一丁目２８０番地　株式会社日立製作所　中央研究所内 Tokyo (JP).

(74) 代理人: ポレール特許業務法人 **(POLAIRE I.P.C.)**; 〒1040032 東京都中央区八丁堀二丁目７番１号 Tokyo (JP).

(81) 指定国 (表示のない限り、全ての種類の国内保護が可能): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

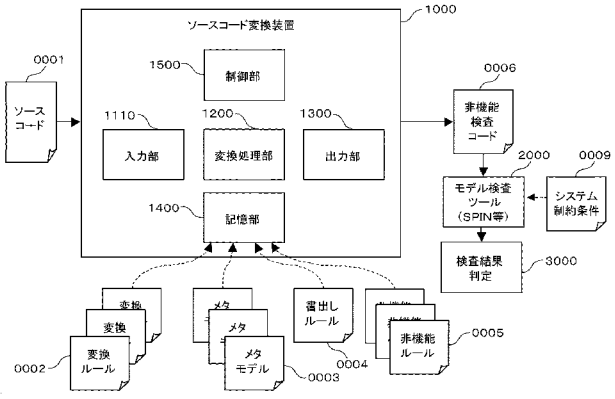
(84) 指定国 (表示のない限り、全ての種類の広域保護が可能): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), ユーラ

[続葉有]

(54) **Title**: METHOD OF CONVERTING SOURCE CODE AND SOURCE CODE CONVERSION PROGRAM

(54) 発明の名称 : ソースコード変換方法およびソースコード変換プログラム

図３



0001 SOURCE CODE
0002 CONVERSION RULE
0003 META MODEL
0004 WRITING-OUT RULE
0005 NONFUNCTIONAL RULE
0006 NONFUNCTIONAL INSPECTION CODE
0009 SYSTEM CONSTRAINT CONDITION
1000 SOURCE CODE CONVERSION DEVICE
1110 INPUT UNIT
1200 CONVERSION PROCESSING UNIT
1300 OUTPUT UNIT
1400 STORAGE UNIT
1500 CONTROL UNIT
2000 MODEL INSPECTION TOOL (SPIN, etc.)
3000 INSPECTION RESULT DETERMINATION

(57) **Abstract**: A method of converting a source code for converting a source code of software to an inspection code by using a computer, comprising: a step for inputting a source code of software; a step for inputting a plurality of different conversion rules; a step for inputting a nonfunctional rule that is a constraint relating to process performance; and a step for converting the source code to a nonfunctional inspection code written in an input language of a validation tool by means of the plurality of different conversion rules and the nonfunctional rule.

(57) 要約:　計算機を活用してソフトウェアのソースコードを検査コードに変換するソースコード変換方法であって、ソフトウェアのソースコードを入力するステップと、異なる複数の変換ルールを入力するステップと、処理性能に関係する制約である非機能ルールを入力するステップと、前記ソースコードを、前記異なる複数の変換ルール及び前記非機能ルールにより、検証ツールの入力言語で記述された非機能検査コードに変換するステップとを有する。

WO 2012/057170 A1

シア (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), ヨーロッパ (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, ML, MR, NE, SN, TD, TG).

添付公開書類:

— 国際調査報告（条約第 21 条(3)）

明　細　書

発明の名称 ：

ソースコード変換方法およびソースコード変換プログラム

技術分野

[0001] 本発明は、ソースコード変換方法およびソースコード変換プログラムに係り、特に、ソフトウェアのモデル検査において、計算機を活用してソフトウェアのソースコードを検査コードに変換する場合に、システム（ソフトウェア）の非機能すなわち処理性能を検証できるようにする技術に関する。

背景技術

[0002] 近年、ソフトウェアシステムが一般社会に浸透し、ソフトウェアに求められる信頼性が非常に高くなってきている一方で、ソフトウェアは複雑化および大規模化の一途をたどっており、手作業でのレビューや、テストによる品質確保が非常に困難になってきている。

[0003] モデル検査技術は、たとえば非特許文献１に開示される方法であって、ソフトウェアの振舞いを、特定モデルチェッカの入力言語で記述し、特定モデルチェッカを実行することで、前記ソフトウェアが持つべき性質を満たしているかどうかを、前記ソフトウェアの取り得る状態を網羅的に検査する技術である。非特許文献１に開示される方法によると、ソフトウェアの振る舞いを、Promelaと呼ばれる入力言語で記述し、SPINと呼ばれるモデルチェッカに入力することで、ソフトウェアが満たすべき機能的性質に関して検査を実施する。

[0004] さらに非文献特許６に開示される方法によると、ソフトウェアの振る舞いと時間制約を時間オートマトンで表現し、UPPAALと呼ばれるモデルチェッカに入力することで、ソフトウェアが満たすべき処理時間に関して検査を実施する。

[0005] モデル検査技術は複雑化及び大規模化の一途をたどるソフトウェアの品質確保に有望な技術であるが、ソフトウェアの取り得る状態を網羅的に検査す

るため、規模の大きなソフトウェアでは、検査すべき状態数が膨大な分量となる状態爆発と呼ばれる現象が起き、処理に必要な時間計算量が現実的に許容不可能な大きさとなる現象と、もしくは、処理に必要な空間計算量が処理に用いる計算機に搭載された記憶領域を超える現象の、両方もしくは一方が起き、検査を完了することができないことがあり得る。

[0006] 状態爆発に対応するために、抽象化と呼ばれる処理を、ソースコードもしくは検査コードに対して行い、状態数を検査可能な範囲まで削減することがある。抽象化は、例えば、選択的実行文の分岐条件の簡略化がある。抽象化によって、本来存在しない実行パスが生じる、もしくは、存在する実行パスが消滅することがあり得るため、検査コードに対する検査結果が示すソフトウェアの性質と、本来のソフトウェアの性質に差異が生まれることがあり得る。そのため、ソフトウェアに対して検査すべき性質に鑑み、抽象化の水準を検討した上で、抽象化を適用することが望ましい。

[0007] さらに、モデル検査技術は、検査対象のソフトウェアを、特定モデルチェッカの入力言語で記述する労力が大きいことが実用上の問題となることがあり得る。特許文献１に開示される方法では、ソースコードが、特定モデルチェッカの入力言語で書かれた検査コードへ、翻訳マップを用いて変換される。検査コードは、前記変換とは別に利用者により定義された環境モデルを用いて、前記特定モデルチェッカにより検査される。また、特許文献２に開示される方法によると、ソースコードから、中間表現に変換し、実時間制約を満足するパラメータを生成することでパラメータ付き時間オートマトンを生成している。

[0008] 図１５に、従来例による、計算機を活用したモデル検査技術の一例を示す。従来例のモデル検査システムでは、まず設計をモデル化し、次に実行環境に合わせた時間オートマトンを作成し、自動検査を行う。この自動検査により、検査対象のソフトウェアに対して、網羅的な性能確認が行われる。

[0009] また、ソフトウェアの開発技術のひとつとして、モデル駆動型開発がある。モデル駆動型開発は、ソフトウェアの設計情報をモデルとして記述し、そ

のモデルを変換操作により詳細化することで、ソフトウェア開発を進める技術である。例えば、モデル駆動型開発において、モデルのフォーマットや意味は非特許文献2に開示される方法であるMOFにより記述されたメタモデルにより規定され、非特許文献3に開示される方法であるQVTによりモデルを詳細化する変換ルールが記述され、非特許文献4に開示される方法であるOCLによりモデルの整合性や健全性に関する制約の記述および検証が行われ、非特許文献5に開示される方法であるMOFM2Tによりモデルからソースコードが生成される。

[0010] なお、モデル検査技術における「モデル」と、モデル駆動型開発における「モデル」は、互いに独立した概念であり、一般にデータ構造や意味などに関する共通性はない。

先行技術文献

特許文献

[0011] 特許文献1：特開2000-181750号公報

特許文献2：WO 2004/038620号公報

非特許文献

[0012] 非特許文献1：Gerard J. Holzmann, "The SPIN Model Checker: Primer and Reference Manual", Addison-Wesley Professional, 2003, ISBN: 978-0321228628

非特許文献2：The Object Management Group, "Meta Object Facility (MOF) Core Specification", formal/06-01-01, January 2006, http://www.omg.org/spec/MOF/2.0/PDF

非特許文献3：The Object Management Group , "Meta Object Facility (MOF) 2. 0 Query/View/Transformation Specification", formal/2008-04-03, April 2008, http: //www.omg.org/spec/QVT/1.0/PDF

非特許文献4：The Object Management Group、"Object Constraint Language", formal/2006-05-01, May 2006, http://www.omg.org/spec/OCL/2.0/PDF

非特許文献5：The Object Management Group, "MOF Model to Text Transfor

mation Language, v1.0", formal/2008-01-16, January 2008, http://www.omg.org/spec/MOFM2T/1.0/PDF

非特許文献6：Gerd Behrmann, Alexandre David and Kim G. Larsen, "A Tutorial on Uppaal", Formal Methods for the Design of Real-Time Systems Lecture Notes in Computer Science, 2004, Volume 3185/2004

発明の概要

発明が解決しようとする課題

[0013] モデル検査によるソフトウェアの信頼性確保を有効に実施するためには、ソースコードからモデルチェッカの入力言語で記述された検査用コードを自動生成するなどの方法により、検査用コードを得るための労力を低減させること、および、ソフトウェアの仕様および設計を、モデルチェッカによる網羅的検査が現実的な時間計算量および空間計算量で終了するように抽象化することが必要となる。

[0014] また処理性能に関しては、実際にはキャッシュやバスの利用状況／外部機構の物理的制約等の要因から少なからず設計値から外れてしまうため、選択した抽象レベルに従って実測データを測定し、検証結果の妥当性を確認する必要がある。

[0015] しかし、特許文献１に開示される方法によると、（１）抽象化の水準の変更が困難である、（２）ソフトウェア設計変更への追従コストが高い、（３）異なるモデルチェッカにて検査する際のコストが高い、（４）利用者によって選択された抽象化水準で処理時間を測定する際のコストが高い、という課題が存在する。

[0016] 上記（１）の課題に関して、特許文献１に開示される方法によると、翻訳マップの変更は、ソースコードの改訂など、新しいタイプの命令がソースコード内に導入された場合のみに限定される。そのため、利用者が抽象化の水準を変更する方法は、変換前の検査対象のソースコードに修正を加える方法と、変換後の特定モデルチェッカの入力言語で書かれた検査コードに修正を加える方法と、環境モデルに修正を加える方法とに限定され、いずれの方法

においても、利用者が大きな手間を費やすことになる。

[0017] 上記（２）の課題に関して、特許文献１に開示される方法によると、使用するライブラリの変更などの変更が生じた際には、翻訳マップの修正と、環境モデルの修正とを行う必要がある。しかし、翻訳マップは、前記ソースコードから、前記検査コードへと直接変換するマップ要素で構成されていることと、環境モデルが特定モデルチェッカの入力言語で書かれていることとに起因して、設計変更に追従するように整合性を保ちながら修正することは困難である。

[0018] 上記（３）の課題に関して、特許文献１に開示される方法によると、異なるモデルチェッカにて検査するためには、翻訳マップの修正と、環境モデルの修正とを行う必要がある。しかし、翻訳マップは、前記ソースコードから前記検査コードへと直接変換するマップ要素で構成されていることや、環境モデルが特定モデルチェッカの入力言語で書かれていることとに起因して、翻訳マップと環境モデルのすべてを修正する必要があり、大きなコストを要する。

[0019] 上記（４）課題に関して、特許文献２に開示される方法によると、利用者が選択した抽象度に従って観測用のコードを挿入することは出来ず、別途手動でソースコードに挿入する必要があり、大きなコストを要する。

[0020] また、利用者は機能要件の検査しか実施できない為、非機能要件の検証を行うためには利用者が検査したい水準で検査コードを手動で作成する必要があり、コストが高く、再利用性も低い、等の課題が存在する。

[0021] また、検査水準と状態数のトレードオフを、利用者側で管理したいというニーズがある。すなわち、複雑なシステムの検査では、状態爆発が容易に発生し、検査を完了することができない。このようなとき、何も検査できないよりも、水準を若干落としてでも、検査を完了できることが望ましい場合がある。例えば、繰り返し実行時のみに発生する特定の不具合がある場合、その繰返しを除去することで、特定の不具合の検出はできなくなるものの、大幅に状態数を削減できる、という場合もある。

[0022] また、図１５に示したモデル検査技術を利用した自動的・網羅的な性能検証技術では、理論値が実測値からかけ離れる場合が多くなる。すなわち、実装方法による影響度が大きいため、理論値と実測値との乖離が大きくなり易い。そのため、実測値を計測する手段が必要となる。

[0023] 本発明の目的は、従来の技術の課題を踏まえ、利用者が検査したい水準で非機能要件の検証を行う検査コードを容易に生成可能とし、コスト低減及び再利用性の向上を実現するソースコード変換方法およびソースコード変換プログラムを提供することにある。

[0024] 本発明の他の目的は、抽象化の水準等に柔軟に対応でき、抽象化水準に従った測定観測用のコードをソースコードに挿入でき、理論値と実測値との乖離を測定し、検査水準と状態数のトレードオフを容易にするソースコード変換方法およびソースコード変換プログラムを提供することにある。

課題を解決するための手段

[0025] 本発明の代表的なものを示せば、次の通りである。本発明のソースコード変換装置によるソースコード変換方法は、ソースコード変換装置によるソースコード変換方法であって、ソフトウェアのソースコードを入力するステップと、異なる複数の変換ルールを入力するステップと、処理性能に関係する制約である非機能ルールを入力するステップと、前記ソースコードを、前記異なる複数の変換ルール及び前記非機能ルールにより、検証ツールの入力言語で記述された非機能検査コードに変換するステップとを有することを特徴とする。

発明の効果

[0026] 本発明によれば、細粒度に分割された複数の変換ルールや非機能ルールを入力するインタフェースを所有する。そのため、利用者による抽象化の水準の変更や、非機能要件の検証が、検査対象のソースコードに対応した、異なる複数の変換ルールを選択して入力する操作により、容易に実現される。これにより、複雑で大規模なソフトウェアであっても、入力したソースコードの機能要件と非機能要件の検証を利用者が選択した水準で容易に行うことが

出来、検査用コード等のソフトウェア開発コストを大幅に低減することができる。

図面の簡単な説明

[0027] [図1]本発明の基本概念を説明するための図。

[図2]本発明のソースコード変換処理における、変換ルールの入力インタフェースについて説明する図。

[図3]本発明の第一の実施例になるソースコード変換システムの構成例を示す図。

[図4]図３の変換システムにおけるソースコード変換装置の構成例を示す図。

[図5]第一の実施例の処理フローの例を示す図。

[図6]第一の実施例のソースコード変換装置の動作を説明する図。

[図7]第一の実施例のソースコード変換の手順をより詳細に説明する図。

[図8A]第一の実施例のソースコード変換により得られるオートマトンを説明する図。

[図8B]第一の実施例におけるソースコード変換技術の全体的な概要を示す図。

[図9]本発明の第２の実施例になるソースコード変換装置を含むソースコード変換システムの構成例を示す図。

[図10]第２の実施例におけるソースコード変換装置の構成例を示す図。

[図11]本発明の第二の実施例になるソースコード変換装置の処理フローの例を示す図。

[図12]第二の実施例のソースコード変換装置の動作を説明する図である。

[図13]第二の実施例のソースコードー実装モデル変換対応表（Ｔａｂｌｅ－Ａ～Ｃ）の構成例及び機能について説明する図。

[図14]第二の実施例のソースコード変換の手順を説明する図である。

[図15]従来例のソースコード変換装置の一例を示す図。

発明を実施するための形態

[0028] 本発明では、ソフトウェアの非機能すなわち処理性能を検証できるように

するために、異なる複数の変換ルールを用いて、検査対象のソースコードをモデルチェッカの入力言語で記述された非機能付きの検査コードへと変換する。前記異なる複数の変換ルールは、検査対象のソースコードをモデルチェッカの入力言語で記述された非機能付きの検査コードへと変換し、抽象化する一連の処理を細粒度に分割したものであり、複数の変換ルールを組み合わせて用いることにより、前記ソースコードから前記非機能付きの検査コードへの変換が実現される。

[0029] 本発明において、検査対象のソースコードを非機能付きの検査コードへと変換する一連の処理を、抽象化の処理も含めて、細粒度に分割し、分割されたそれぞれの処理を、「変換ルール」と呼ぶ。また、非機能付きの検査コードとは、検査コードが、細粒度に分割されたそれぞれの処理に関して、非機能要件と機能要件の検証を可能にする非機能の制約が付加された検査コード（以下、単に非機能検査コード）を意味する。そして、「非機能ルール」は、非機能（処理性能に関係する制約）を検査コードにするための変換ルールを定義したものである。

[0030] 本発明により実現されるソースコード変換装置は、ソースコードを検査コードへ変換する時、異なる複数の変換ルールや「非機能ルール」が、利用者によって、選択され、入力されるためのインタフェースを所有する。前記変換ルールや「非機能ルール」は、事前にソースコード変換装置内部に蓄積された複数の変換ルールや「非機能ルール」の中からの選択と、利用者の記述と、のいずれかの手段により入力される。

[0031] また、本発明において、変換ルールは、ソースコードを、前記ソースコードの記述言語に依存しない、汎化されたプログラム情報をもつ形式（汎化モデル）へと変換する実装ー汎化変換ルールと、汎化モデルを抽象化する抽象化変換ルールと、汎化モデルをモデルチェッカの記述言語へと変換する汎化ー検査変換ルールに分類される。換言すると、異なる複数の変換ルールは、ソースコードを特定のプログラミング言語に依存しない形式である中間形式へと変換する第１変換ルールと、前記中間形式に対して抽象化処理を行う第

２変換ルールと、前記中間形式から前記非機能検査コードに変換する第３変換ルールとに分類される。ソースコードから非機能検査コードへの変換は、ソースコードを、実装ー汎化変換ルールにより、汎化モデルへと変換するステップと、前記汎化モデルを、抽象化変換ルールにより抽象化するステップと、この抽象化された前記汎化モデルに非機能の制約を付加するステップと、この汎化モデルを、汎化ー検査変換ルールにより、非機能検査コードへと変換するステップと、の４つのステップを続けて行うことで実現される。

[0032] 換言すると、ソースコードから検査コードへの変換は、前記第１、第２、第３の変換ルールを各々１つ以上入力するステップと、前記第１変換ルールを用いて、ソフトウェアのソースコードを、前記中間形式へと変換するステップと、前記第２変換ルールを用いて、前記中間形式で表現されたソフトウェアを抽象化するステップと、非機能ルールを用いて前記中間形式に非機能の制約を付加するステップと、前記第３変換ルールを用いて前記中間形式を検証ツールの入力言語で記述された検証用コードに変換するステップの４つのステップを続けて行うことで実現される。

[0033] また、本発明において、検査対象のソースコードを、検査コードへと変換する一連の処理にて、内部的に保持される情報（モデル）は、その形式をメタモデルにより定義される。前記モデルは、検査対象のソースコードに対応する情報をもつ実装モデルと、前述の汎化モデルと、モデルチェッカの記述言語に対応する情報をもつ検査モデルと、に分類される。実装モデルは、そのメタモデルであるメタ・実装モデルにより定義され、汎化モデルは、そのメタモデルであるメタ・汎化モデルにより定義され、検査モデルは、そのメタモデルであるメタ・検査モデルにより定義される。前述のそれぞれのメタモデルは、データ構造の定義と、データに含まれる要素間の制約に関する情報とを保有する。

[0034] また、本発明の他の実施形態によれば、前記中間形式を、前記異なる複数の変換ルールにより、抽象化中間形式に変換し、さらに非機能検査コードに変換するステップと、前記抽象化中間形式に、観測ポイントを付加するステ

ップと、前記観測ポイントが付加された抽象化中間形式を、前記異なる複数の逆変換ルールにより、観測ポイントが付加された中間形式に変換するステップと、前記観測ポイントが付加された中間形式を、前記異なる複数の逆変換ルールにより、観測ポイントが付加されたソースコードに変換するステップを有する。

[0035] 以下、図面を参照して本発明の実施形態について詳しく説明する。

まず、図１、図２を参照しながら本発明の基本概念を説明する。本発明では、ソースコードを、複数の変換ルールの組み合わせた変換処理を行うことにより、既存モデル検査装置に適合した非機能検査コードへ変換する。すなわち、（ａ）“変換”を細粒度に分割し、複数の「変換ルール」の組み合わせとしてパッケージ化することで、柔軟な変換を実現する。（ｂ）利用者（検査者）は、検査対象のソースコード０００１を入力し、対象のソースコードと検査水準に応じた「変換ルール」０００２を選択し、変換処理１０００により中間形式で表現されたソフトウェアを抽象化し、この抽象化されたソフトウェアの各部品（機能）に非機能ルール０００５に基づいた非機能の制約を付加することで、所望の非機能検査コード０００６を得る。非機能検査コード０００６に対しては、さらに、モデル検査ツール２０００においてシステムの制約条件を満たすかの判定がなされる。制約条件を満たさない場合は、ソースコードを修正して変換処理１０００を繰り返す。このようにしてシステムの制約条件を満たす非機能検査コード０００６が生成される。

[0036] 本発明における、「変換ルール」の例を挙げると次の通りである。

（ａ）単純な構文変換

「C言語の条件分岐 （If文・Switch文）」を、「検査コードの条件分岐（If文）」に変換

「C言語の繰り返し文（for文・while文・…）」を、「検査コードの繰り返し（Do文）」に変換

（ｂ）外部環境のモデル化

「データ読込み」を、「ランダム入力」へ置き換え

（ｃ）抽象化

「繰り返しの除去」

「条件の簡略化」

また、本発明における、「非機能」ルールには、時間的制約や、容量的な制約がある。すなわち、本発明における、非機能要件には、処理時間、メモリ使用量、又はその両方を含む。

[0037] 図２により、本発明のソースコード変換処理における変換ルールの入力インタフェースについて、説明を補足する。

[0038] 本発明によれば、細粒度に分割された複数の変換ルール０００２を入力するインタフェースを所有することによって、利用者による抽象化の水準の変更は、複数の異なる変換ルールを選択・入力する操作により容易に実現される。すなわち、利用者による抽象化の水準の変更は、ドメイン情報や、検査したい性質、検査水準の情報（抽象化による性質への影響）に応じて、利用者が複数の異なる変換ルールを選択して呼び出し、ソースコード変換処理装置１０００へ入力する操作により容易に実現される。

[0039] 本発明は、特に、システム（ソフトウェア）の機能要求と非機能（処理性能）要求の検証に有効な技術である。すなわち、非機能ルール０００５を入力するインタフェースを所有することによって、利用者による抽象化されたソフトウェアの各部品（機能）に対する非機能の制約を付加した所望の非機能検査コード０００６の生成が可能になり、非機能要件の検証が容易となる。

[0040] また、非機能検査コード０００６は、細粒度に分割された状態の中間形式に「変換ルール」や「非機能」ルールが適用されて、生成される。この非機能検査コード０００６が、システム全体としての制約条件を満たすかを判定する必要があり、これをモデル検査ツール２０００に入力し、その結果を判定する機能（３０００）を有している。

[0041] 本発明によれば、ソースコードを利用者が選択した水準で検査コードに変換した後、非機能要件を追加する手段を提供することで、利用者の指定した

水準の非機能検査を可能にする。

[0042] また、本発明によれば、利用者が選択した抽象化の水準に合わせた実測データの観測が可能となり、理論値と実測値との乖離を観測でき、検査水準と状態数のトレードオフを容易に解消することができる。

実施例 1

[0043] 次に、本発明の第一の実施例になるソースコード変換装置および変換処理方法を、図３ないし図１１Ｂを参照しながら説明する。

[0044] 図３は、第一の実施例になるソースコード変換装置を含むソースコード変換システムの構成例を示す図である。本発明の実施形態に適用されるソースコード変換装置１０００は、検査対象のソースコードを非機能検査コード０００６に変換する装置であり、入力部１１００と、変換処理部１２００と、出力部１３００と、記憶部１４００と、制御部１５００、とを有する。２０００はモデル検査ツール、３０００は検査結果の判定機能を示す。記憶部１４００には、変換ルール０００２、メタモデル０００３、書出しルール０００４、及び、非機能ルール０００５が保持される。また、モデル検査ツー２０００に対してはシステム制約条件０００９が与えられる。

[0045] 図４に、ソースコード変換装置１０００の構成例を示す。入力部１１００は、ソースコード入力部１１０１と、変換ルール入力部１１０２と、非機能ルール入力部１１０３とを有する。変換処理部１２００は、モデル構築部１２０１、実装ー汎化モデル変換部１２０２、抽象化モデル変換部１２０３、非機能制約付加処理部１２０４、及び、汎化ー検査モデル変換部１２０９を備えている。出力部１３００は、非機能検査コード書出力部１３０１を有する。記憶部１４００は、変換ルールデータベース１４０１、メタモデルデータベース１４０２、および、書出しルールデータベース１４０３、及び、非機能ルールデータベース１４０４を有する。

[0046] ソースコード変換装置１０００は、例えば、１台のコンピュータ、あるいはネットワークを介して接続された複数のコンピュータ上で動作するプログラムとして実施される。ソースコード０００１と、変換ルール集合０００２

と、非機能ルール０００５は、例えば、コンピュータ上の記憶装置から読込む方法と、コンピュータに接続された入力デバイスにより直接入力される方法等の方法により入力される。また、非機能検査コード０００６は、例えば、コンピュータの記憶装置に書き出す方法と、コンピュータのディスプレイ装置に表示する方法等により出力される。

[0047] 入力部１１００は、ユーザによって入力されるデータを受付け、入力されたデータを変換処理部１２００へと供給する処理を行う。入力部１１００は、ソースコード０００１に関する情報を入力し、修正するソースコード入力・修正部１１０１と、細粒度に分割された複数の変換ルーすなわち「変換ルール集合」０００２に関する情報を入力する変換ルール入力部１１０２と、非機能ルール０００５を入力する非機能ルール入力部１１０３とを備え、受付けた各入力を変換処理部１２００へ供給する。ある実施形態においては、入力部１１００は、ユーザによる、変換処理部の駆動や制御、出力部の駆動や制御、に関する指示を受け付けてもよい。

[0048] 変換処理部１２００は、入力部より、ソースコード０００１の情報と、ソースコード０００１に適用すべき変換ルール集合０００２の情報と、非機能ルール０００５とを供給され、ソースコード０００１を、変換ルール集合０００２により変換し、非機能を付加する処理を行い、変換結果の情報を、出力部１３００に供給する。ある実施形態においては、入力部より供給される変換ルール集合０００２に関する情報は、記憶部に格納された変換ルールを指し示す識別情報のみが含まれていて、変換ルール集合０００２の実体を、前記識別情報を用いて記憶部１４００より取り出し、変換に用いてもよい。

[0049] 変換処理部１２００は、モデル構築部１２０１と、実装－汎化モデル変換部１２０２と、抽象化モデル変換部１２０３と、非機能制約付加処理部１２０４と、汎化－検査モデル変換部１２０９とを有する。本実施形態において、変換処理部１２００は、図６に示したように、メタモデルと、変換ルールを用いたモデル変換により、ソースコード情報１００１を、検査モデル１００８へと変換する。メタ・実装モデル１００２と、メタ・汎化モデル１００

３と、メタ・検査モデル１００４と、は、例えば、非特許文献２に記載のMOFにて記述する。また、例えば、非特許文献３に記載されているQVTにて、実装ー汎化変換ルール１００５と、抽象化変換ルール１００６と、汎化ー検査変換ルール１００７とを記述し、モデルの変換を行う。前記変換は、既に例示した方法の他のモデル変換方法でも良く、また、複数の方法を混在させてもよい。

[0050] また、ある実施例においては、実装ー汎化モデル変換部１２０２と、抽象化モデル変換部１２０３と、非機能制約付加処理部１２０４と、汎化ー検査モデル変換部１２０９とをそれぞれ独立させず、同一の部位により行われても良く、さらに、実装モデル１２０５と、汎化モデル１２０６と、検査モデル１００８と、のそれぞれのメタモデルとして、メタ・実装モデル１００２と、メタ・汎化モデル１００３と、メタ・検査モデル１００４とを個別に用意せず、単一のメタモデルにより、実装モデル１２０５と、汎化モデル１２０６と、検査モデル１００８とを定義してもよい。また、ある実施例においては、メタ・実装モデル１００２と、メタ・汎化モデル１００３と、メタ・検査モデル１００４と、は、複数の方式により、それぞれ実装モデル１２０５、汎化モデル１２０６、検査モデル１００８、の形式や制約を定義してもよい。例えば、それぞれのメタモデルは、前記MOFによる定義のほかに、非特許文献４に記載されているOCLにて記載された制約条件を含み、モデルの変換を行った際に、制約条件を満たしているかどうかの検証を行う方法があり得る。

[0051] モデル構築部１２０１は、ソースコード入力部１１０１からソースコード情報１００１を受け取り、実装モデル１２０５へと変換する。実装モデル１２０５の形式は、そのメタモデルであるメタ・実装モデル１００２により、定義される。実装モデル１２０５は、本発明の効果を最大限に得るためには、ソースコード情報１００１と相互に変換するのに必要十分な情報を持つことが望ましいが、ある実施形態によっては、検査コードを出力するという目的を逸しない程度にて、情報の省略や、追加があってもかまわない。

[0052] ある実施形態においては、モデル構築部１２０１は、ソースコード入力部１１０１と不可分な様態で実装され、ソースコード情報１００１が生じない形態で処理が進んでもよい。

[0053] 実装－汎化モデル変換部１２０２は、実装－汎化変換ルール１００５と、メタ・実装モデル１００２と、メタ・汎化モデル１００３とを用いて、実装モデル１２０５を、汎化モデル１２０６へと、変換する。汎化モデルは、複数のプログラミング言語における構造や処理を表現可能なデータ構造をもつ。例えば、汎化モデル中では、ソースコード０００１におけるIf文とSwitch文とを区別せず、選択的実行文として表現する。ある実施形態においては、実装モデル１２０５を、汎化モデル１２０６へと変換する際、実装－汎化変換ルール１００５のみを用いることもあり得る。また、ある実施形態において、実装－汎化変換ルール１００５が、複数の変換ルールを含む場合には、複数の変換ルールを統合して１つの変換ルールとし、実装モデル１２０５から汎化モデル１２０６への変換に利用する方法があり得る。また、ある実施形態において、実装－汎化変換ルール１００５が、複数の変換ルールを含む場合には、変換処理を複数回繰り返すことで、実装モデル１２０５から汎化モデル１２０６への変換を行う手順があり得る。

[0054] 抽象化モデル変換部１２０３は、抽象化変換ルール１００６と、メタ・汎化モデル１００３とを用いて、汎化モデル１２０６の抽象化を行う。抽象化の例としては、選択文における条件式を、恒真、もしくは恒偽、もしくは非決定的な選択で置き換える方法があり得る。ある実施形態においては、汎化モデル１２０６を抽象化する際、抽象化変換ルール１００６のみを用いることもあり得る。また、ある実施形態において、抽象化変換ルール１００６が、複数の変換ルールを含む場合には、複数の変換ルールを統合して１つの変換ルールとし、汎化モデル１２０６の抽象化に用いる方法があり得る。また、ある実施形態において、抽象化変換ルール１００６が、複数の変換ルールを含む場合には、変換処理を複数回繰り返すことで、汎化モデル１２０６の変換を行う手順があり得る。

[0055] 非機能制約付加処理部１２０４は、抽象化された汎化モデル１２０６の各部品（機能）に非機能ルール０００５に基づいた非機能の制約を付加し、汎化モデル１２０７を生成する。

[0056] 汎化－検査モデル変換部１２０９は、汎化－検査変換ルール１００７と、メタ・汎化モデル１００３と、メタ・検査モデル１００４とを用いて、非機能の制約が付加された汎化モデル１２０７を、検査モデル１００８へと、変換する。例えば、検査コードがPromela形式である場合、汎化モデルにおいて選択的実行文として表現された要素は、検査モデルにおいてはIf文として表現される。ある実施形態においては、汎化モデル１２０６を、検査モデル１００８へと変換する際、汎化－検査変換ルール１００７のみを用いることもあり得る。また、ある実施形態において、汎化－検査変換ルール１００７が、複数の変換ルールを含む場合には、複数の変換ルールを統合して１つの変換ルールとし、汎化モデル１２０６から検査モデル１００８への変換に利用する方法があり得る。また、ある実施形態において、汎化－検査変換ルール１００７が、複数の変換ルールを含む場合には、変換処理を複数回繰り返すことで、汎化モデル１２０６から検査モデル１００８への変換を行う手順があり得る。

[0057] 出力部１３００は、変換処理部１２００より供給された、変換結果の情報を用いて、非機能検査コード０００６を出力する。ある実施形態においては、検査コードの出力に際して、例えばモデルチェッカの記述言語の文法情報などの情報を記憶部から供給されてもよい。

[0058] 検査コード書出し部１３０１は、メタ・検査モデル１００４と、記憶部１４００の書出しルールデータベース１４０３から取得した検査コード書出しルール１００９とを用いて、検査モデル１００８を非機能検査コード０００６へと変換する。たとえば、非特許文献５に記載される方法にて、検査コード書出しルール１００９を記述し、検査モデル１００８から、非機能検査コード０００６への変換を行う。ある実施例においては、これに限らず、検査モデル１００８を、検査に利用するモデルチェッカの記述形式へと変換する

任意の方法でもよい。非機能検査コード０００６は、例えば、モデルチェッカとしてＳＰＩＮを用いる場合には、ＳＰＩＮの入力言語である、Ｐｒｏｍｅｌａにより記述される。

[0059] 記憶部１４００において、変換ルールデータベース１４０１と、メタモデルデータベース１４０２と、書出しルールデータベース１４０３と、非機能ルールデータベース１４０４のそれぞれは、例えば、関係データベース、もしくは、階層型データベースなどの、コンピュータ上にて実現される任意のデータ格納方式で実現される。変換ルールデータベース１４０１と、メタモデルデータベース１４０２と、書出しルールデータベース１４０３と、非機能ルールデータベース１４０４とは互いに同一の方式で実現されている必要性は無く、互いに異なる方式で実現されていてもよい。また、検査ルールデータベース１４０１と、メタモデルデータベース１４０２と、書出しルールデータベース１４０３と、非機能ルールデータベース１４０４のそれぞれは、単一の方式で実現されている必要性は無く、例えば、保有すべき情報の一部を関係データベースに格納し、保有すべき情報の一部を、発明を実現するコンピュータプログラム中に組込むなど、それぞれ異なる方式の組み合わせで実現されていてもよい。

[0060] 記憶部１４００は、入力部１１００と、変換処理部１２００と、出力部１３００とが、それぞれの処理を行うのに必要な情報を供給する。例えば、記憶部１４００は、変換ルールに関する情報と、メタモデルに関する情報と、モデルチェッカの記述言語の文法に関する情報とを格納する。

[0061] 変換ルールデータベース１４０１は、既に述べたとおり、変換ルールを、メタデータとともに保有する。前記メタデータは、変換ルールを選択するためのものであり、実装ー汎化変換ルール１００５と、抽象化変換ルール１００６と、汎化ー検査変換ルール１００７の、それぞれ異なる情報を持つ方法があり得る。実装ー汎化変換ルール１００５のメタデータは、例えば、変換元ソースコードの記述言語の種類があり得る。抽象化変換ルール１００６のメタデータは、例えば、抽象化を分かりやすく端的に表現した名前、簡単な

説明、「データの抽象化」、「処理の抽象化」などの抽象化の種別、自然語もしくはアルファベットや数値で表現された抽象化による状態数削減効果、自然語もしくはアルファベットや数値で表現された抽象化による性質への影響、抽象化の適用できるソフトウェアのドメイン、があり得る。汎化ー検査変換ルール１００７のメタデータは、例えば、検査に使用するモデルチェッカを指し示す名前があり得る。

[0062] 以降、図５ないし図６を参照しながら入力部１１００と、変換処理部１２００と、出力部１３００と、記憶部１４００と、制御部１５００の詳細に関し、説明する。

[0063] まず、本実施例におけるソースコード変換手順の一例を、図５を参照しながら説明する。図５は、第一の実施例の処理フローの例を示す図である。本実施例におけるソースコード変換手順は、ソースコード入力ステップＳ１０１と、変換ルール入力ステップＳ１０２と、変換ルール適用ステップＳ１０３と、検査コード出力ステップＳ１０４と、システム制約条件の入力ステップＳ１０５と、判定のステップＳ１０６と、修正のステップＳ１０７とからなる。

[0064] まず、ソースコード入力ステップＳ１０１において、ソースコード入力部１１０１により、ソースコード０００１がソースコード変換装置１０００に読み込まれ、ソースコード情報１００１へと変換される。

[0065] ソースコード入力部１１０１は、利用者から入力された検査対象のソースコード０００１を受付け、ソースコード情報１００１へと変換する。ソースコード０００１は、例えば、JIS X3010-1993に公開されるプログラミング言語Ｃにより記述される。ソースコード情報１００１は、具体的には、例えば抽象構文木の形式で保持される。ソースコード情報１００１の形式は、抽象構文木に限らず、構造や論理などソースコード０００１の検査に要する情報を保持する、任意の形式でもよい。

[0066] ソースコード入力ステップＳ１０１に続き、変換ルール入力ステップＳ１０２において、変換ルール入力部１１０２により、細粒度に分割された複数

の変換ルールである変換ルール集合０００２がソースコード変換装置１０００に読み込まれる。この変換ルール入力ステップＳ１０２では、変換前のモデルの要素と、変換後のモデル要素との対応関係と、変換によって変換前のモデルの要素に加えられる操作との一方もしくは両方が定義される。変換ルール集合０００２をソースコード変換装置１０００に読み込む処理は、利用者による一度の操作で行われる必要性は無く、繰り返し操作により行われてもよい。また、ソースコード入力ステップＳ１０１と、変換ルール入力ステップＳ１０２は、必ずしもこの順番で完了する必要性は無く、ソースコード入力部１１０１によりソースコード情報１００１が生成される前にソースコード０００１が入力され、かつ、変換ルール入力部１１０２が変換ルール入力処理のためにソースコード情報１００１を要求するまえにソースコード０００１が入力されていれば良く、したがって、ソースコード入力ステップＳ１０１の処理と、変換ルール入力ステップＳ１０２の処理とが混在する順番で処理がすすんでもよい。例えば、ソースコード入力部１１０２がソースコード０００１を受付け、続いて、変換ルール入力部が、変換ルール集合０００２を受付け、続いて、ソースコード入力部１１０２が、ソースコード０００１をソースコード情報１００１へ変換する、手順があり得る。

[0067] 変換ルール入力部１１０２は、利用者から入力された変換ルール集合０００２を受け付ける。利用者から変換ルール集合０００２を受け付ける方法は、例えば、次に示す方法があり得る。

[0068] １つめの変換ルール入力方法の例として、変換ルール入力部１１０２は、変換ルール集合０００２の一部として、利用者が手作業で直接入力した変換ルールを受け取る。

[0069] ２つめの変換ルール入力方法の例として、変換ルール集合０００２の少なくとも一部は、利用者が記述した変換ルール（記述）により入力しても良い。あるいはまた、入力部１１００が、記憶部１４００から変換ルールの一覧を取得し、それを利用者に一覧などの形式で提示し、前記一覧から利用者が選択すること、変換ルール集合０００２の入力を受け付けてもよい。すなわ

ち、利用者が、変換ルールの入力に前置して、変換ルールの検索条件を入力部１１００の変換ルール入力部１１０２に入力（記述）し、続いて、変換ルール入力部１１０２が、前記検索条件に合致する変換ルールを、記憶部１４００が有する変換ルールデータベース１４０１から取得し変換ルール一覧として前記利用者に提示する。続いて、前記利用者が、提示された前記変換ルール一覧に含まれる１つ以上の変換ルールを選択する。この利用者によって選択された１つ以上の変換ルールを、変換ルール入力部１１０２が、変換ルール集合０００２の一部として受け付ける。

[0070] ３つめの変換ルール入力方法の例として、まず、利用者が、変換ルールの入力に前置して、変換ルールの検索条件を変換ルール入力部１１０２に入力し、続いて、変換ルール入力部１１０２が、前記検索条件に合致する変換ルールを、変換ルールデータベース１４０１から取得して、変換ルール集合の一部として受け付けても良い。

[0071] ４つめの変換ルール入力方法の例として、入力されたソースコード０００１から、変換ルール入力部１１０２が、変換ルールの検索条件を抽出、生成し、さらに、前記検索条件に合致する変換ルールを、変換ルールデータベース１４０１から取得し、変換ルール集合の一部として受け付ける。

[0072] また、５つめの変換ルール入力方法の例として、変換ルール入力部１１０２は、何からの方法で入力された変換ルールを加工することにより、変換ルールを受け付ける。前記加工方法の例としては、変換ルールデータベース１４０１には、ソースコード０００１中の変数名などをパラメータ化した状態で変換ルールを保持しておき、例えば利用者の明示的な入力による方法などにより、ソースコード０００１の情報にてパラメータを埋めたものを、変換ルール集合に含める方法があり得る。

[0073] 変換ルール入力ステップＳ１０２に続き、変換ルール適用ステップＳ１０３において、モデル構築部１２０１がソースコード情報１００１を実装モデル１２０５へと変換し、続いて、実装ー汎化モデル変換部１２０２が実装モデル１２０５を汎化モデル１２０６へと変換し（Ｓ１０３１）、続いて、抽

象化モデル変換部１２０３が汎化モデル１２０６を抽象化する（Ｓ１０３２）。抽象化された汎化モデル１２０６の各部品（機能）に非機能ルール０００５に基づいた非機能の制約を付加し、汎化モデル１２０７を生成し（Ｓ１０３３）、続いて、汎化ー検査モデル変換部１２０９が汎化モデル１２０７を検査モデル１００８へと変換する（Ｓ１０３４）。変換ルール入力ステップＳ１０２と、変換ルール適用ステップＳ１０３とは、必ずしもこの順番で処理が完了する必要性は無く、実装ー汎化モデル変換部１２０２の処理までに実装ー汎化変換ルール１００５が入力され、かつ、抽象化モデル変換部１２０３の処理までに非機能ルール０００５及び抽象化変換ルール１００６が入力され、かつ、汎化ー検査モデル変換部の処理までに汎化ー検査変換ルール１００７が入力されていればよい。

[0074] 変換ルール適用ステップＳ１０３に続き、検査コード出力ステップＳ１０４において、検査コード書出し部１３０１により、検査モデル１００８が、非機能検査コード０００６として書き出される。非機能検査コード０００６の書出し先の指定は、必ずしも変換ルール適用ステップＳ１０３の後である必要性は無く、非機能検査コード０００６の書出しよりも先であればよい。例えば非機能検査コード０００６の書出し先の指定がソースコード入力ステップＳ１０１と平行して行われる手順があり得る。

[0075] なお、検査コード出力ステップＳ１０４に続き、変換ルール入力ステップＳ１０２へと進むことで、既に入力されたソースコード０００１を、繰り返し、異なる変換ルール集合０００２を用いて、変換する手順をとってもよい。また、ある実施例においては、検査コード出力ステップＳ１０４に続き、変換ルール入力ステップＳ１０２へと進み、既に入力された変換ルール集合０００２の全てまたは一部と、新たに変換ルール入力ステップＳ１０２で入力された変換ルール集合０００２をあわせて、変換ルール集合０００２として用いてもよい。

[0076] 非機能検査コード０００６は、モデル検査ツー２０００に入力される。このモデル検査ツー２０００には、システム制約条件の入力ステップＳ１０５

でシステム制約条件０００９が与えられる（ステップＳ１０５の入力のタイミングはこれ以前でも良い）。そして、ステップＳ１０６で、非機能検査コード０００６がシステム制約条件０００９を満たしているかの判定がなされる。非機能検査コード０００６がシステム制約条件０００９を満たしていない場合、ステップＳ１０７で、ソースコードに対してオペレータによる修正が加えられ、以下、システム制約条件０００９を満たすまで、同様処理が繰り返される。

[0077] 次に、図６、図７を用いて、変換処理部４１１１を詳細に説明する。図６は、第一の実施例のソースコード変換装置の動作を説明する図である。図７は、第一の実施例のソースコード変換の手順をより詳細に説明する図である。

[0078] 本実施形態において、変換処理部４１１１は、メタモデルと、変換ルールを用いたモデル変換により、ソースコード情報１１０１を、非機能検査モデル１００８へと変換する。まず、モデル構築部１２０１は、ソースコード入力部１００１からソースコード情報１００１を受け取り、実装モデル１２０５へと変換する。実装ー汎化モデル変換部１２０２は、変換ルールデータベースから実装ー汎化変換ルール１００５を、メタモデルデータベース１からメタ・実装モデル１００２とメタ・汎化モデル１００３を取り出し、実装モデル１２０５を、汎化モデル１２０６へと、変換する。抽象化モデル変換部１２０３は、変換ルールデータベースから抽象化変換ルール１００６を、メタモデルデータベースからメタ・汎化モデル１００３を取り出し、汎化モデル１２０６の抽象化を行う。非機能要件付加処理部１２０４は、非機能ルールデータベースから非機能要件１６００を取り出し、汎化モデル１２０６の対応するモデル要素に非機能要件の付加を行い非機能要件付き汎化モデル１２０７の生成を行う。例えば、非機能要件はモデル要素と要件項目のテーブル形式として表現される。

[0079] 汎化ー検査モデル変換部１２０９は、変換ルールデータベースから汎化ー検査変換ルール１００７を、メタモデルデータベースからメタ・汎化モデル

１００３とメタ・検査モデル１００４を取り出し、非機能要件付き汎化モデル１２０７を、検査モデル１００８へと、変換する。例えば、非機能要件が処理で、モデル検査ツールがUPPAALの場合では、検査コードを時間オートマトンの形式に変換する。検査コード出力部１３０１は、メタ・検査モデル１００４と、書出しルールデータベースから取得した検査コード書出しルールとを用いて、検査モデル１００８を非機能検査コード０００６へと変換する。例えば、モデル検査ツールとしてUPPAAL用いる場合には、モデルを直接入力することが可能であるため、変換処理することなく書き出せばよい。

[0080] このように、本発明では、モデル変換技術を利用し、段階的に変換するために、次のような処理を行う。

（１）ソースコード０００１をこれと（ほぼ）等価な「実装モデル」１２０５へと変換

（２）「実装モデル」を特定のプログラミング言語に依存しない形式にて構造や論理などのプログラム情報を表現する「汎化モデル」へと変換

すなわち、異なる複数の第１変換ルール１００５－１～１００５－ｎの少なくとも１つを用いて、「実装モデル」１２０５を特定のプログラミング言語に依存しない形式である中間形式の「汎化モデル」１２０６へと変換する。図７の例では、第１変換ルールとして、「“ｉｆ文”→条件分岐」、「“ｓｗｉｔｃｈ文”→条件分岐」、「“ｗｈｉｌｅ文”→“繰り返し”」、「“ｆｏｒ文”→“繰り返し”」、「“ｉｎｔ文”→“整理型”」、「“ｆｌｏｗ文”→“フロー型”」、「“ｄｏｕｂｌｅ文”→“フロー型”」の少なくとも７つの異なる変換ルールが選択されている。

[0081] （３）「汎化モデル」１２０６に非機能制約を付加する処理を実施

すなわち、非機能ルールを用いて、中間形式の「汎化モデル」１２０６を非機能制約付の汎化モデル１２０７へと変換する。図７の例では、時間の制約（単位：秒）として、「 “サブルーチン” →ｔ＞６０」、「“繰り返し”→ｔ＞５０」、「“条件分岐”→ｔ＞３０」、「“四則演算”→ｔ＞１０」の異なる非機能ルールが選択されている。また、容量の制約として、「“整

理型”→４バイト」、「“フロー型” →８バイト」の異なる非機能ルールが選択されている。

[0082] （４）非機能制約付の汎化モデル１２０７に対して、抽象化のための変換を実施

すなわち、異なる複数の第２変換ルール１００６－１～１００６－ｎの少なくとも１つを用いて、前記中間形式の汎化モデルに対して抽象化処理を行う。

[0083] 図７の例では、第２変換ルールとして、「データ読み込み→ランダム入力」、「データの抽象化」の少なくとも２つの異なる変換ルールが選択されている。

[0084] （５）汎化モデルを、「非機能検査モデル」に変換し、コード生成（出力）

すなわち、異なる複数の第３変換ルール１００７－１～１００７－ｎの少なくとも１つを用いて、前記中間形式の汎化モデル１２０７から非機能検査コードの生成に要する情報を有する非機能検査モデル１００８に変換する。図７の例では、第３変換ルールとして、第１変換ルールに対応した「“条件分岐”→“ｉｆ”文」、「“繰り返し”→“ｄｏ”文」の少なくとも２つの異なる変換ルールが選択されている。

[0085] また、実装モデル、汎化モデル、検査モデルは、それぞれ構文を定義する「メタモデル」によりデータ構造と意味論が定義される。

[0086] このように、実装モデルから汎化モデルへの変換に際しては、例えば、変換対象ソースコードの記述言語の文法が、繰り返し処理の記法として“ｆｏｒ文”や“ｗｈｉｌｅ文”を含むとき、使用者が“ｆｏｒ文”を「繰り返し」へ変換するルールと、“ｗｈｉｌｅ文”を「繰り返し」へ変換するルールとを、既に述べた変換ルール入力方法を用いて、第１変換ルールとして選択する。汎化モデルの抽象化の変換に際しては、使用者が検査水準（抽象化の度合い）を決定し、決定した検査水準を達成する変換ルールとして、例えば、外部データ読込みに関する命令および一連の処理をランダムな入力へと変

換するルールと、特定のデータ型をより抽象度の高い型へと変換するルールとを、既に述べた変換ルール入力方法を用いて、第２変換ルールとして選択する。さらに、汎化モデルから検査モデルへの変換に当たっては、例えば、モデルチェッカの入力言語の文法が、繰り返し処理の記法として“ｄｏ文”をもつとき、使用者が「繰り返し」を“ｄｏ文”へ変換するルールを、既に述べた変換ルール入力方法を用いて、第３変換ルールとして選択する。変換ルールは、複数のソフトウェアにまたがって適用可能な汎用的なルールなど、繰り返し利用可能なものがデータベース化される。データベースに格納された変換ルールは、使用者による検索やルール選択の判断材料として用いられるメタ情報として、ドメイン情報や検査水準（抽象化による検査への影響）の情報を有する。

[0087] また、変換ルールの選択方法としては、次のようなものがある。

（１）汎用的なルール： 常に選択

（２）特定のライブラリに依存したルール： 使用ライブラリや、検査対象のドメイン（カテゴリ）を入力することで、まとめて選択

（３）抽象化に対応したルール：（検査したい性質・検査水準を入力して得た）変換ルール一覧から、利用者が選択、もしくは利用者自身が記述して入力、もしくは検査したい性質などから、自動生成。

[0088] 本発明では、ソースコードから変換ルールに従って検査モデルを生成する際に、抽象化されたソフトウェアの各部品（機能）に、各々実行環境に対応した処理時間を記述することで、オートマトンに時間の概念を与えることができる。例えば、図８Ａのオートマトンに示したように、抽象化されたソフトウェアの各部品（機能）、すなわち各状態（Ａ）～（Ｅ）に対して、各々実行環境に対応した制約として処理時間（ＰＣ）を記述することができる。これにより、利用者が、対象ソースコードと、検査水準（抽象化の度合い）に応じて、変換ルールや非機能ルールを選択することで、検査可能な非機能検査コードに変換することが可能となり、利用者の選択した抽象化の水準での非機能検証の課題が解決される。

[0089] 本発明によれば、図８Ｂに示したように、抽象度に対応した機能要求や非機能要求の検証が可能であり、また、変換ルールの変更のみで種々の実行環境に対応可能となる。

[0090] すなわち、モデルの抽象化により、状態数を削減することができる。しかし、抽象化によりモデルの性質に影響を与えることがある。たとえば、検出された欠陥（反例）が、もとのシステムに存在しない、もとのシステムに存在する欠陥を発見できない、等である。一方で、性質に影響を与えない健全な抽象化は、状態数削減効果が小さい傾向がある。

[0091] 本発明によれば、細粒度に分割された複数の変換ルールを入力するインタフェースを所有することによって、利用者による抽象化の水準の変更や、非機能制約の付加は、変換ルールを入力する操作により容易に実現される。すなわち、複数の細粒度の変換ルールや非機能制約の付加を利用者が入力インタフェースにより選択できる。

[0092] ソースコード変換法は、複数の変換ルールを用いて検査対象のソースコードをモデルチェッカの入力言語で記述された検査コードへと変換する手順を有し、前記変換ルールは、実装－汎化変換ルールと、抽象化変換ルールと、汎化－検査変換ルールと、に分類され、変換が段階的に行われる。これにより、検査対象のソースコードの設計変更に追従する際には、複数の変換ルールの中の変更に関連する変換ルールや非機能制約の付加のみを変更すればよく、変更が最小限にとどめられる。加えて、実装モデルと、汎化モデルと、検査モデルとをそれぞれメタモデルで定義し、制約を加えることにより、変換ルールによる変換結果が不正でないことを検証可能となる。これにより、検査対象のソースコードを検査コードへと抽象化しながら変換する一連の処理を、細粒度の変換ルール及び非機能ルールと組み合わせることで実現することによって生じる、変換ルール及び非機能ルールの検証コストの増大を防ぐことが出来る。

[0093] また、細粒度に分割された複数の変換ルールを入力するインタフェースを所有することによって、利用者による抽象化の水準の変更は、検査したい性

質、検査水準に応じて、利用者が変換ルール及び非機能ルールを選択・入力する操作で容易に実現される。これにより、抽象化の水準の変更が困難で、あるという課題が解決され、モデル検査ツールの入力言語で検査コードを記述するコストを低減させることが可能となる。例えば、繰り返し実行時のみに発生する特定の不具合がある場合、その繰返しを除去することで、特定の不具合の検出はできなくなるものの、その繰返しを発生原因に含まない不具合の検出は可能であるままに、大幅に状態数を削減できる。

[0094] さらに、モデルの変換ルール及び非機能ルールをデータベースに蓄積・再利用することで、検査対象ソースコードの設計変更や、別ソフトウェアへの応用に、低コストで対応可能になる。

[0095] 本発明によるシステム（ソフトウェア）の非機能検証の例として、システムの時間制約に関する検証の例を挙げると、次のようなものがある。

（１）DTVのチャンネル切り替えが、１秒以内に切り替わること確認する。

（２）超解像アルゴリズムが、１画面あたり１／１２０秒以内で終了することを確認する。

（３）装置の異常をセンサにて検出した場合、５秒以内でユーザに通知することを確認する。

実施例 2

[0096] 次に、図９乃至図１４を使って、本発明の第２の実施例になるソースコード変換装置および変換処理方法を説明する。本実施例では、抽象化変換後の中間形式に対して抽象化フラグを付加するステップを有することを特徴とする。すなわち、ソースコード変換装置によるソースコード変換方法であって、ソースコード変換装置に、ソフトウェアのソースコードと、異なる複数の変換ルールと、異なる複数の逆変換ルールとを入力し、ソースコードを、異なる複数の変換ルールにより、中間形式に変換し、さらに、中間形式を、異なる複数の変換ルールにより、抽象化中間形式に変換する。そして、抽象化中間形式に、観測ポイントを付加する。さらに、観測ポイントが付加された抽象化中間形式を、異なる複数の逆変換ルールにより、観測ポイントが付加

された中間形式に変換し、この観測ポイントが付加された中間形式を、異なる複数の逆変換ルールにより、観測ポイントが付加されたソースコードに変換する。

[0097] 図９は、第２の実施例になるソースコード変換装置を含むソースコード変換システムの構成例を示す図である。本発明の実施形態に適用されるソースコード変換装置１０００は、検査対象のソースコード０００１を非機能検査コード０００７に変換し、計測付きソースコード０００８を出力する装置であり、入力部１１１０と、変換処理部１２００と、出力部１３００と、記憶部１４００と、制御部１５００とを有する。２０００はモデル検査ツール、４０００は実測装置を示す。

[0098] 図１０は、ソースコード変換装置１０００の構成例を示す図である。入力部１１００は、ソースコード入力・修正部１１０１と、変換ルール入力部１１０２と、非機能ルール入力部１１０３とを有する。変換処理部１２００は、モデル構築部１２０１、実装ー汎化モデル変換部１２０２、抽象化モデル変換部１２０３、非機能要件付加処理部１２０４、汎化ー検査モデル変換部１２０９、具体化モデル変換部１２１０、汎化ー実装モデル変換部１２１１、及び、コード生成部１２１２を備えている。出力部１３００は、非機能検査コード出力部１３０１と計測付きソースコード出力部１３０２とを有する。記憶部１４００は、変換ルールデータベース１４０１、メタモデルデータベース１４０２、書出しルールデータベース１４０３、非機能ルールデータベース１４０４、および、逆変換ルールデータベース１４０５を有する。

[0099] ソースコード変換装置１０００は、例えば、１台のコンピュータ、あるいはネットワークを介して接続された複数のコンピュータ上で動作するプログラムとして実施される。ソースコード０００１と変換ルール集合０００２とは、例えば、コンピュータ上の記憶装置から読込む方法と、コンピュータに接続された入力デバイスにより直接入力される方法等の方法により入力される。また、非機能検査コード０００７と計測付きソースコード０００８は、例えば、コンピュータ上の記憶装置に書出す方法と、コンピュータのディス

プレイ装置に表示する方法により出力される。入力部１１００は、ユーザによって入力されるデータであるソースコード０００１に関する情報と、細粒度に分割された複数の変換ルール、すなわち「変換ルール集合」０００２に関する情報とを受付け、変換処理部１２０２へ供給する。ある実施形態においては、入力部１１００は、ユーザによる、変換処理部の駆動や制御、出力部の駆動や制御、に関する指示を受け付けてもよい。

[0100] 図１１は、本発明の第二の実施例になるソースコード変換装置の処理フローの例を示す図である。変換処理部１２００は、入力部１１００からソースコード０００１の情報と、ソースコード０００１に適用すべき変換ルール集合０００２の情報とを供給され（図１１のＳ１０１、Ｓ１０２）、ソースコード０００１を、変換ルール集合０００２により変換する処理（Ｓ１０３１～１０３４、Ｓ１０４～Ｓ１０７）と逆変換処理（Ｓ１０３５～Ｓ１０３８）を行い、変換結果の情報と、ユーザの抽象化水準に会った計測処理が追加された計測付きソースコード０００８を出力部１３００に供給する（Ｓ１０８）。出力部１３０００は、変換処理部１２００より供給された、変換結果の情報を用いて、非機能検査コード０００７を出力する。

[0101] 以降、図１２～図１４を参照しながら変換処理部１２００の構成、機能を詳細に説明する。まず、図１２は、第二の実施例のソースコード変換装置の動作を説明する図である。図１３は、ソースコードー実装モデル変換対応表（Ｔａｂｌｅ－Ａ～Ｃ）の構成例及び機能について説明する図である。

[0102] 本実施形態において、変換処理部１２００は、メタモデルと、変換ルールを用いたモデル変換により、ソースコード情報０００１を、非機能検査モデル１２０７へと変換し、計測付きソースコード０００８を出力する。モデル構築部１２０１は、ソースコード入力・修正部１２０１からソースコード情報を受け取り、実装モデル１２０５へと変換し、その変換過程をソースコードー実装モデル変換対応表（Ｔａｂｌｅ－Ａ）として逆変換データベース１４０５に登録する。ソースコードー実装モデル変換対応表（Ｔａｂｌｅ－Ａ）には、例えば、“ｉｆ（）”→“ｉｆ文”、“ｓｗｉｔｃｈ”→“ｓｗｉ

ｔｃｈ文”、“ｗｈｉｌｅ”→“ｗｈｉｌｅ文”のように記録される。

[0103] 実装－汎化モデル変換部１２０２は、変換ルールデータベース１４０１から実装－汎化変換ルールを、メタモデルデータベース１４０２からメタ・実装モデルとメタ・汎化モデルとを取り出し、実装モデルを汎化モデルへと変換し、その変換過程を実装－汎化モデル変換対応表（Ｔａｂｌｅ－Ｂ）として逆変換データベース１４０５に登録する。ソースコード－実装モデル変換対応表（Ｔａｂｌｅ－Ｂ）には、例えば、位置１０、“繰り返し” →“ｗｈｉｌｅ文“Ｓｔｅｐ１０、 位置２０、“繰り返し” →“ｗｈｉｌｅ文“Ｓｔｅｐ１５、位置３０、“繰り返し” →“ｆｏｒ文“Ｓｔｅｐ３０のように記録される。

[0104] 抽象化モデル変換部１２０３は、変換ルールデータベース１４０１から抽象化変換ルールを、メタモデルデータベース１４０２からメタ・汎化モデルを取り出し、汎化モデルの抽象化を行い、その変換過程を抽象化変換対応表（Ｔａｂｌｅ－Ｃ）として逆変換データベース１４０５に登録する。ソースコード－実装モデル変換対応表（Ｔａｂｌｅ－Ｃ）には、例えば、位置２０、“ランダム入力” →位置５０、“データ読み込み”のように記録される。

[0105] 非機能要件付加処理部１２０４は、非機能ルールデータベース１４０４から非機能要件を取り出し、汎化モデルの対応するモデル要素に非機能要件の付加を行い、汎化－検査モデル変換部１２０９と具体化モデル変換部１２１０に出力する。汎化－検査モデル変換部１２０９は、変換ルールデータベースから汎化－検査変換ルールを、メタモデルデータベースからメタ・汎化モデルとメタ・検査モデルを取り出し、汎化モデルを、検査モデルへと変換する。具体化モデル変換部１２１０は、逆変換ルールデータベース１４０５から抽象化変換対応表（Ｔａｂｌｅ－Ｃ）を取得し、対応表に従って抽象化した汎化モデルの具体化を行う。汎化－実装モデル変換部１２１１は、逆変換ルールデータベース１４０５から実装－汎化モデル変換対応表（Ｔａｂｌｅ－Ｂ）を取得し、対応表に従って汎化モデルを実装モデルへと逆変換する。コード生成部１２１２は、逆変換ルールデータベース１４０５からソースコ

ードー実装モデル変換対応表（Ｔａｂｌｅ－Ａ）を取得し、対応表に従って実装モデルを計測付きソースコード０００８へと変換し、ソースコード出力部１３０２へ出力する。検査コード出力部は、メタ・検査モデルと、記憶部１４００の書出しルールデータベース１４０３から取得した検査コード書出しルールとを用いて、検査モデルを非機能検査コード０００７へと変換する。例えば、モデル検査ツールとしてUPPAAL用いる場合には、モデルを直接入力することが可能であるため、変換処理することなく書き出せばよい。ソースコード出力部では、入力された測定付きソースコードをシステム外部に出力する。例えば、外部記憶媒体へ保存したり、ディスプレイに表示したりする。

[0106] 次に、図１４は、ソースコード変換の手順を説明する図である。本発明による、非機能ルール、例えば、時間変換ルールを用いた、自動生成技術によれば、（１）検証したい抽象度まで抽象プログラムモデルを変換し、（２）検査対象の抽象プログラムモデルに計測用のチェックを追加し、（３）逆変換ルールに従ってソースコードに変換、計測チェックされた箇所に計測用のコードを埋め込み、計測コード付きソースコードとする。さらに、（４）実環境を使って実行時間を計測し、時間変換ルールを生成する。（５）上記生成された時間変換ルールを用いて非機能検査用コード０００７を生成する。なお、図１２、図１３で説明したように、時間変換ルールを付与してから計測コード付きソースコードを生成するようにしても良い。

[0107] これにより、利用者による抽象化の水準の変更に応じた観測用プローブを実際のソフトウェアのソースコードに自動的に挿入が可能となり、利用者が選択した抽象化水準での非機能、例えば実行処理時間の観測が困難であるという課題が解決される。

[0108] すなわち、本実施例によれば、利用者による抽象化の水準の変更は、検査対象のソースコードに対応した、異なる複数の変換ルールを選択して入力する操作により、自動的に実現される。そして、利用者が選択した抽象化の水準に合わせて、ソフトウェアの理論上の位置と実際の位置を１：１に対応付

けた実測データの観測が可能となる。

符号の説明

[0109] ０００１　ソースコード

０００２　変換ルール

０００３　メタモデル

０００４　書出しルール

０００５　非機能ルール

０００６　非機能検査コード

０００７　非機能検査コード

０００８　計測付きソースコード

１０００　ソースコード検査装置

１００１　ソースコード情報

１００２　メタ・実装モデル

１００３　メタ・汎化モデル

１００４　メタ・検査モデル

１００５　実装－汎化変換ルール

１００６　抽象化変換ルール

１００７　汎化－検査変換ルール

１００８　検査モデル

１００９　検査コード書出ルール

１１００　入力部

１１０１　ソースコード入力部

１１０２　変換ルール入力部

１２００　変換処理部

１２０１　モデル構築部

１２０２　実装－汎化モデル変換部

１２０３　抽象化モデル変換部

１２０４　非機能制約付加処理部

１２０９　汎化－検査モデル変換部

１２０５　実装モデル

１２０６　汎化モデル

１３００　出力部

１３０１　検査コード書出し部

１４００　記憶部

１４０１　変換ルールデータベース

１４０２　メタモデルデータベース

１４０３　書出しルールデータベース

１４０４　非機能ルールデータベース

１５００　制御部

２０００　モデル検査ツール

４０００　実測装置

Ｓ１０１　ソースコード入力　ステップ

Ｓ１０２　変換ルール入力　ステップ

Ｓ１０３　変換ルール適用　ステップ

Ｓ１０４　検査コード出力　ステップ。

請求の範囲

［請求項1］ ソースコード変換装置によるソースコード変換方法であって、

ソフトウェアのソースコードを入力するステップと、

異なる複数の変換ルールを入力するステップと、

処理性能に関係する制約である非機能ルールを入力するステップと、

前記ソースコードを、前記異なる複数の変換ルール及び前記非機能ルールにより、検証ツールの入力言語で記述された非機能検査コードに変換するステップとを有する

ことを特徴とするソースコード変換方法。

［請求項2］ 請求項１において、

前記異なる複数の変換ルールは、検査対象の前記ソースコードを前記非機能検査コードへと変換し抽象化する一連の処理を、細粒度に分割したものであり、

前記非機能ルールは、時間的制約もしくは容量的な制約の少なくとも一方を含む

ことを特徴とするソースコード変換方法。

［請求項3］ 請求項１において、

前記変換ルールは、

ソースコードを特定のプログラミング言語に依存しない形式である中間形式へと変換する第１変換ルールと、

前記中間形式に対して抽象化処理を行う第２変換ルールと、

前記中間形式から前記検査コードに変換する第３変換ルールとを含み、

ソフトウェアのソースコードを入力するステップと、

少なくとも１つの前記第１変換ルールを入力するステップと、

少なくとも１つの前記第２変換ルールを入力するステップと、

少なくとも１つの前記第３変換ルールを入力するステップと、

前記第１変換ルールを用いて、前記ソフトウェアのソースコードを、前記中間形式へと変換するステップと、

前記第２変換ルールを用いて、前記中間形式で表現された前記ソフトウェアを抽象化するステップと、

前記中間形式に、前記非機能ルールにより、前記抽象化された中間形式に非機能制約を付加するステップと、

前記第３変換ルールを用いて、前記非機能制約の付加された前記中間形式を検証ツールの入力言語で記述された非機能検査コードに変換するステップとを有する

ことを特徴とする、ソースコード変換方法。

[請求項4] 請求項３において、

前記非機能ルールは、前記中間形式の条件分岐に与える時間的制約を含む

ことを特徴とする、ソースコード変換方法。

[請求項5] 請求項３において、

前記非機能ルールは、前記中間形式の繰り返しに与える時間的制約を含む

ことを特徴とする、ソースコード変換方法。

[請求項6] 請求項３において、

前記非機能ルールは、前記中間形式のサブルーチンに与える時間的制約を含む

ことを特徴とする、ソースコード変換方法。

[請求項7] 請求項３において、

前記非機能ルールは、前記中間形式の四則演算に与える時間的制約を含む

ことを特徴とする、ソースコード変換方法。

[請求項8] 請求項３において、

前記非機能ルールは、前記中間形式の整理型に与える容量的な制約

を含む

ことを特徴とする、ソースコード変換方法。

[請求項9] 請求項３において、

前記非機能ルールは、前記中間形式のフロー型に与える容量的な制約を含む

ことを特徴とする、ソースコード変換方法。

[請求項10] 請求項３において、

非機能検査コードがシステムの制約条件を満たすかの判定を行うステップと、

非機能検査コードが前記システムの制約条件を満たさない場合に、前記ソースコードを修正して再入力するステップとを含む、

ことを特徴とする、ソースコード変換方法。

[請求項11] ソースコード変換装置によるソースコード変換方法であって、

ソフトウェアのソースコードを入力するステップと、

異なる複数の変換ルールを入力するステップと、

異なる複数の逆変換ルールを入力するステップと、

前記ソースコードを、前記異なる複数の変換ルールにより、中間形式に変換するステップと、

前記中間形式を、前記異なる複数の変換ルールにより、抽象化中間形式に変換するステップと、

前記抽象化中間形式に、観測ポイントを付加するステップと、

前記観測ポイントが付加された抽象化中間形式を、前記異なる複数の逆変換ルールにより、観測ポイントが付加された中間形式に変換するステップと、

前記観測ポイントが付加された中間形式を、前記異なる複数の逆変換ルールにより、観測ポイントが付加されたソースコードに変換するステップを有する

ことを特徴とするソースコード変換方法。

[請求項12] 請求項１１において、

前記中間形式に対して検証要件を追加するステップを有する

ことを特徴とする、ソースコード変換方法。

[請求項13] 請求項１１において、

処理性能に関係する制約である非機能ルールを入力するステップと、

前記ソースコードを、前記異なる複数の変換ルール及び前記非機能ルールにより、検証ツールの入力言語で記述された非機能検査コードに変換するステップとを有する

ことを特徴とする、ソースコード変換方法。

[請求項14] 請求項１１において、

前記異なる複数の変換ルールは、前記ソースコードを、該ソースコードの記述言語に依存しない汎化されたプログラム情報をもつ汎化モデルへ変換する実装－汎化変換ルールと、前記汎化モデルを抽象化する抽象化変換ルールと、前記汎化モデルを検証ツールの記述言語へと変換する汎化－検査変換ルールとを含む

ことを特徴とするソースコード変換方法。

[請求項15] 請求項１４において、

前記ソースコード情報を受け取り、実装モデルへと変換し、その変換過程をソースコード－実装モデル変換対応表として逆変換データベースに登録するステップと、

前記実装モデルを前記汎化モデルへと変換し、その変換過程を実装－汎化モデル変換対応表として前記逆変換データベースに登録するステップと、

前記汎化モデルの抽象化を行い、その変換過程を抽象化変換対応表として前記逆変換データベースに登録するステップと、

前記汎化モデルの対応するモデル要素に非機能要件の付加を行い、出力するステップと、

前記汎化モデルを、非機能検査モデルへと変換するステップとを含む

ことを特徴とするソースコード変換方法。

[請求項16] 請求項１５において、

前記抽象化変換対応表に従って、抽象化した前記汎化モデルの具体化を行うステップと、

前記実装ー汎化モデル変換対応表に従って前記汎化モデルを前記実装モデルへと逆変換するステップと、

前記ソースコードー実装モデル変換対応表に従って前記実装モデルを計測付きソースコードへと変換し、出力するステップとを含む

ことを特徴とするソースコード変換方法。

[請求項17] 少なくとも１台のコンピュータ上で動作し、ソースコード変換装置を構成するプログラムであって、

前記コンピュータを、

ソフトウェアのソースコードを入力する手段、

異なる複数の変換ルールを入力する手段、

処理性能に関係する制約である非機能ルールを入力する手段、及び

前記ソースコードを、前記異なる複数の変換ルール及び前記非機能ルールにより、検証ツールの入力言語で記述された非機能検査コードに変換する手段

として機能させるためのソースコード変換プログラム。

[請求項18] 請求項１７において、

前記変換ルールは、

前記ソースコードを特定のプログラミング言語に依存しない形式である中間形式へと変換する第１変換ルールと、

前記中間形式に対して抽象化処理を行う第２変換ルールと、

前記中間形式から前記検査コードに変換する第３変換ルールとを含み、

前記コンピュータを、

前記ソースコードを入力する手段と、

少なくとも１つの前記第１変換ルールを入力する手段、

少なくとも１つの前記第２変換ルールを入力する手段、

少なくとも１つの前記第３変換ルールを入力する手段、

前記第１変換ルールを用いて、前記ソフトウェアのソースコードを、前記中間形式へと変換する手段、

前記第２変換ルールを用いて、前記中間形式で表現された前記ソフトウェアを抽象化する手段、

前記中間形式に、前記非機能ルールにより、前記抽象化された中間形式に非機能制約を付加する手段、及び

前記第３変換ルールを用いて、前記非機能制約の付加された前記中間形式を前記検証ツールの入力言語で記述された非機能検査検証用コードに変換する手段

として機能させるためのソースコード変換プログラム。

[請求項19] 請求項１８において、

前記コンピュータを、

前記抽象化中間形式に、観測ポイントを付加する手段、

前記観測ポイントが付加された抽象化中間形式を、前記異なる複数の逆変換ルールにより、観測ポイントが付加された前記中間形式に変換する手段、

前記観測ポイントが付加された中間形式を、前記異なる複数の逆変換ルールにより、観測ポイントが付加された前記ソースコードに変換する手段

として機能させるためのソースコード変換プログラム。

[請求項20] 請求項１８において、

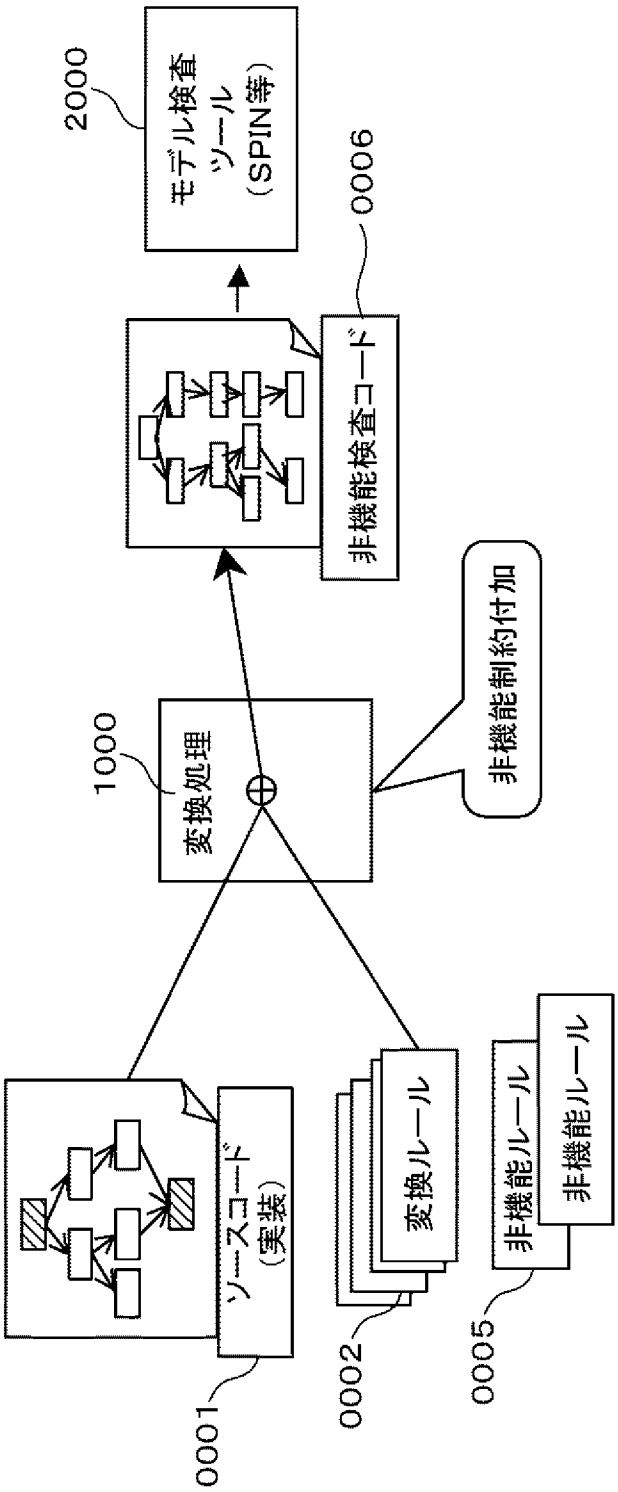
前記コンピュータを、

前記中間形式に対して検証要件を追加する手段

として機能させるためのソースコード変換プログラム。

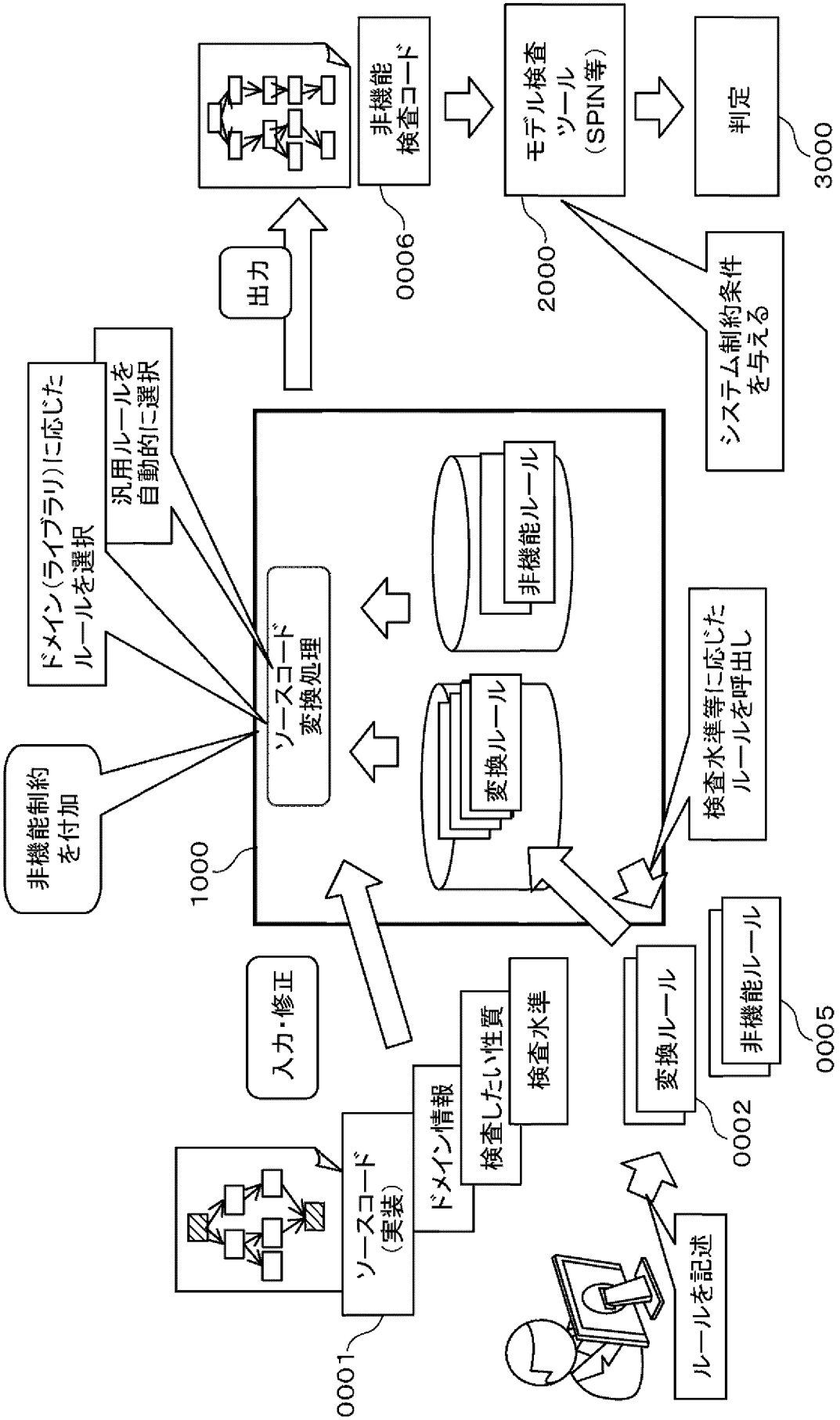
[図1]

図１


2000
モデル検査ツール（SPIN等）
0006
非機能検査コード
1000
変換処理
非機能制約付加
ソースコード（実装）
0001
変換ルール
0002
非機能ルール
非機能ルール
0005

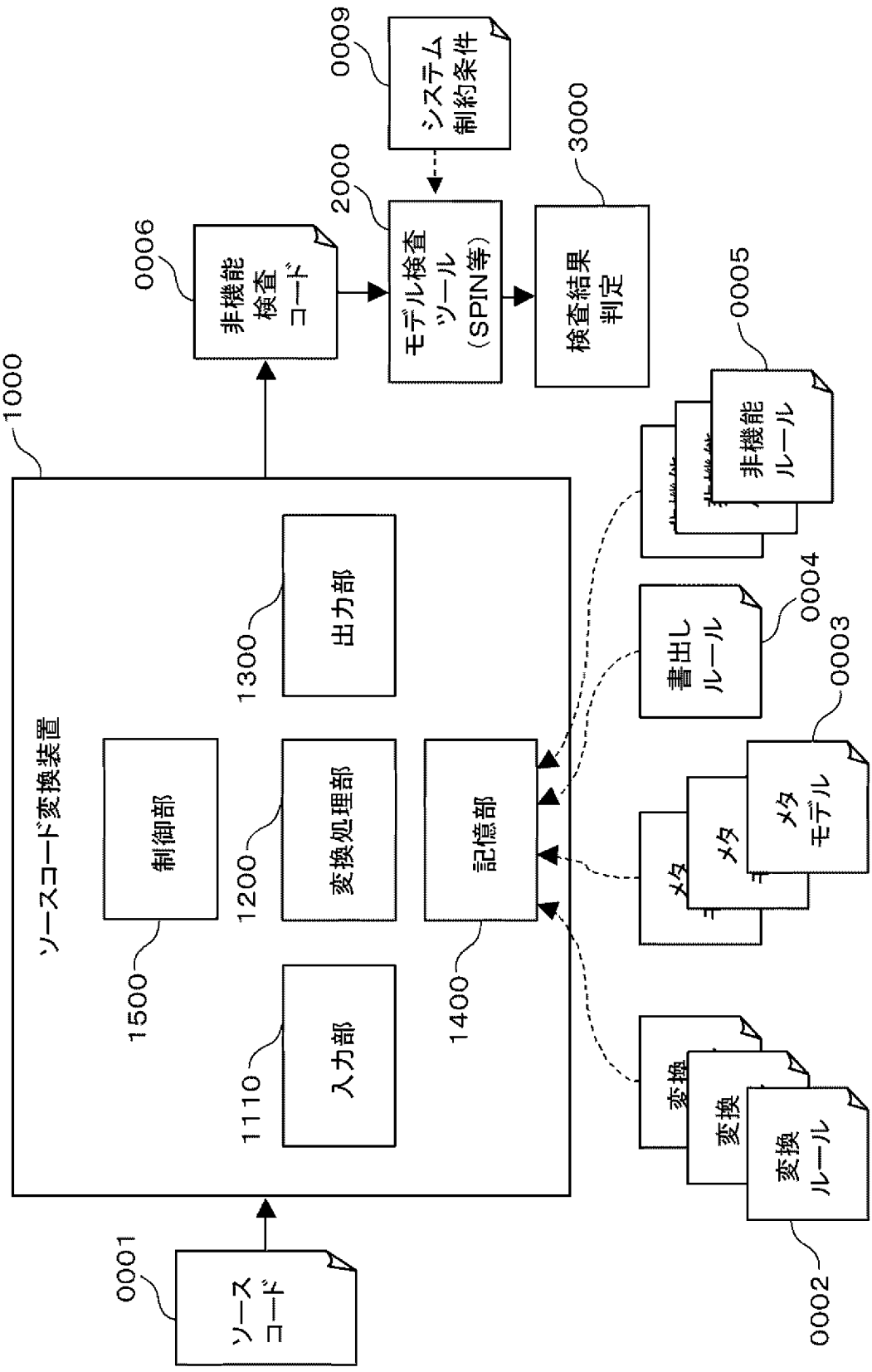
[図2]

図２


ソースコード（実装）
ドメイン情報
検査したい性質
検査水準
0001
入力・修正
ルールを記述
変換ルール
非機能ルール
0002
0005
検査水準等に応じたルールを呼出し
1000
非機能制約を付加
ドメイン（ライブラリ）に応じたルールを選択
汎用ルールを自動的に選択
ソースコード変換処理
変換ルール
非機能ルール
出力
0006
非機能検査コード
2000
モデル検査ツール（SPIN等）
システム制約条件を与える
判定
3000

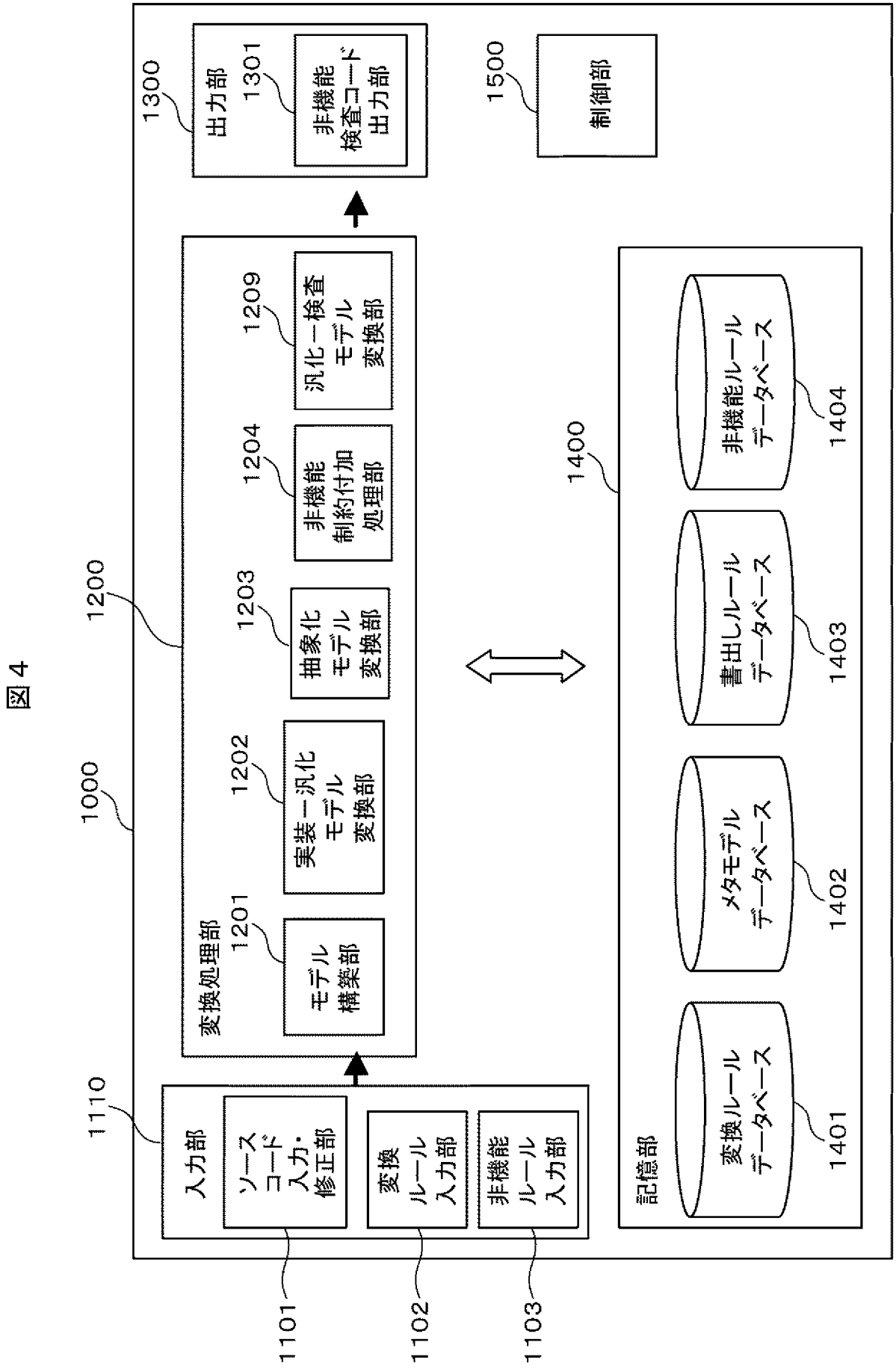
[図3]

図３


ソースコード
0001
ソースコード変換装置
1000
入力部
1110
制御部
1500
変換処理部
1200
出力部
1300
記憶部
1400
非機能検査コード
0006
モデル検査ツール（SPIN等）
2000
システム制約条件
0009
検査結果判定
3000
非機能ルール
0005
書出しルール
0004
メタモデル
0003
メタ
メタ
メタ
変換ルール
変換
変換
0002

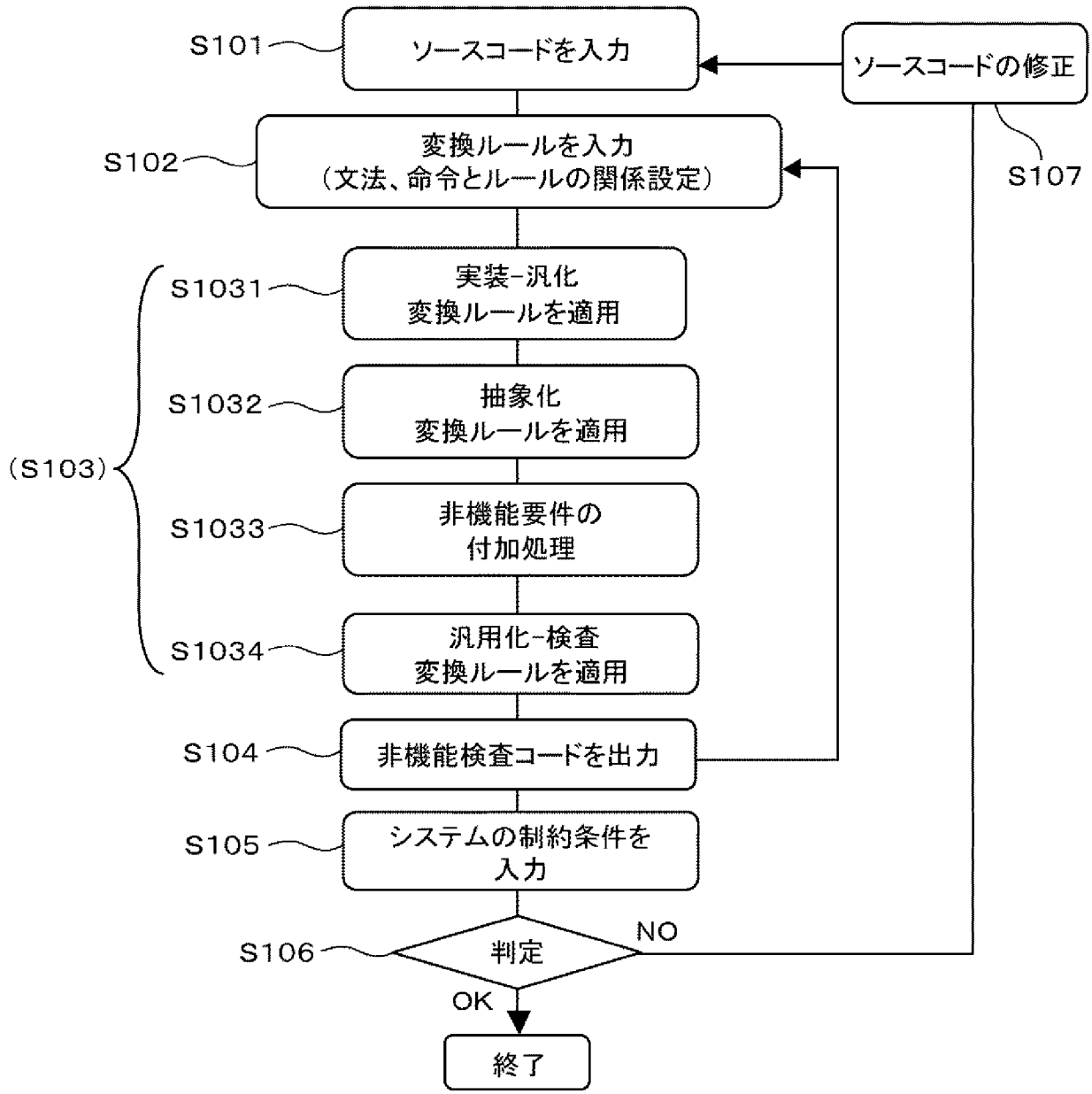
[図4]

図４


1000
1110
入力部
ソースコード入力・修正部
変換ルール入力部
非機能ルール入力部
1101
1102
1103
1200
変換処理部
1201
モデル構築部
1202
実装一汎化モデル変換部
1203
抽象化モデル変換部
1204
非機能制約付加処理部
1209
汎化一検査モデル変換部
1300
出力部
1301
非機能検査コード出力部
1400
記憶部
1401
変換ルールデータベース
1402
メタモデルデータベース
1403
書出しルールデータベース
1404
非機能ルールデータベース
1500
制御部

[図5]

図５


S101
ソースコードを入力
ソースコードの修正
S107
S102
変換ルールを入力
（文法、命令とルールの関係設定）
(S103)
S1031
実装-汎化
変換ルールを適用
S1032
抽象化
変換ルールを適用
S1033
非機能要件の
付加処理
S1034
汎用化-検査
変換ルールを適用
S104
非機能検査コードを出力
S105
システムの制約条件を
入力
S106
判定
NO
OK
終了

[図6]

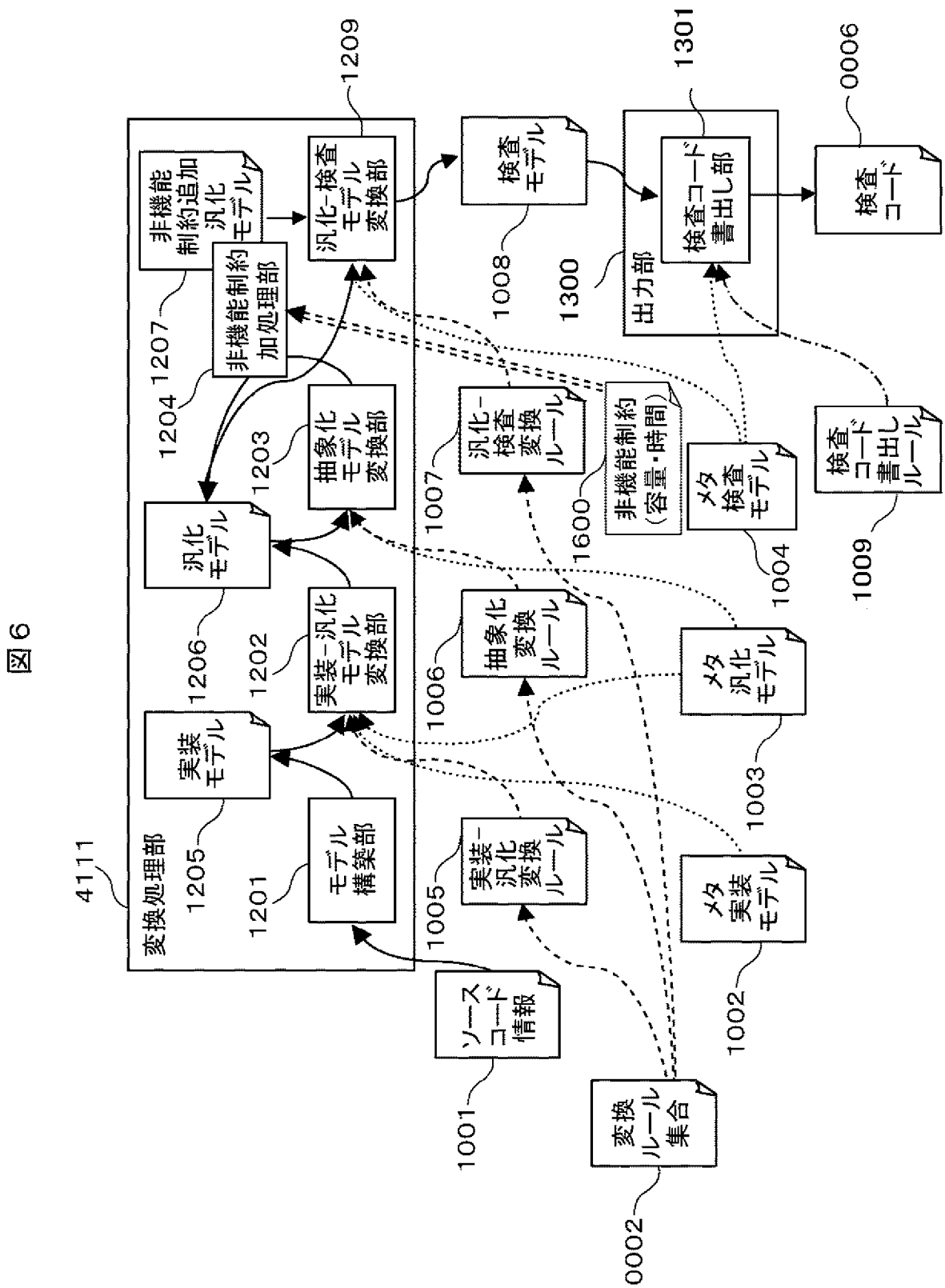
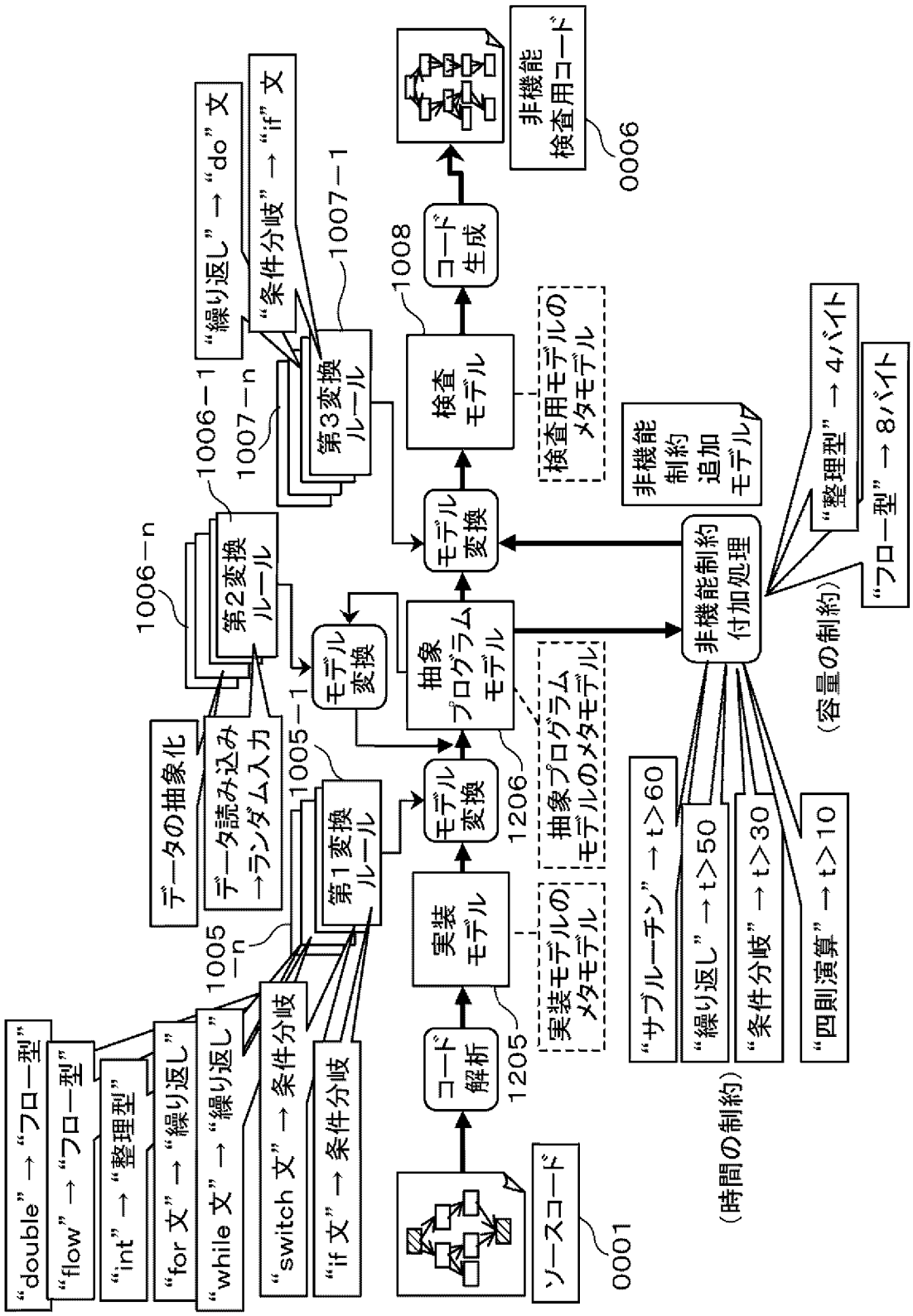

変換処理部
4111
モデル構築部
1201
実装モデル
1205
実装-汎化モデル変換部
1202
汎化モデル
1206
抽象化モデル変換部
1203
非機能制約加処理部
1204
非機能制約追加汎化モデル
1207
汎化-検査モデル変換部
1209
ソースコード情報
1001
実装-汎化変換ルール
1005
抽象化変換ルール
1006
汎化-検査変換ルール
1007
非機能制約（容量・時間）
1600
変換ルール集合
0002
メタ実装モデル
1002
メタ汎化モデル
1003
メタ検査モデル
1004
検査モデル
1008
出力部
1300
検査コード書出し部
1301
検査コード
0006
検査コード書出しルール
1009

図6

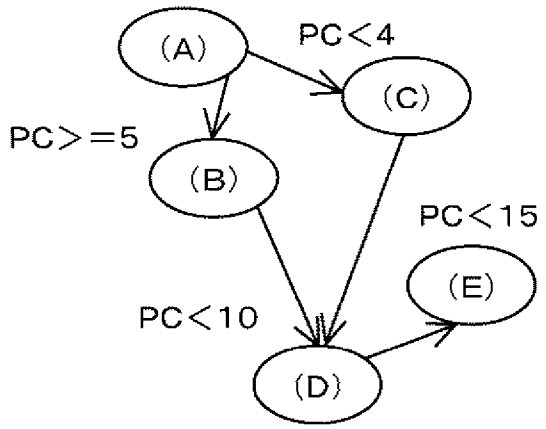
[図7]

図 7


“double” → “フロー型”
“flow” → “フロー型”
“int” → “整理型”
“for 文” → “繰り返し”
“while 文” → “繰り返し”
“switch 文” → 条件分岐
“if 文” → 条件分岐
1005−n
1005−1
データの抽象化
データ読み込み→ランダム入力
第1変換ルール
第2変換ルール
1006−n
1006−1
“繰り返し” → “do” 文
“条件分岐” → “if” 文
1007−n
1007−1
第3変換ルール
モデル変換
モデル変換
モデル変換
ソースコード
0001
コード解析
1205
実装モデル
実装モデルのメタモデル
抽象プログラムモデル
1206
抽象プログラムモデルのメタモデル
検査モデル
1008
コード生成
非機能検査用コード
0006
検査用モデルのメタモデル
非機能制約追加モデル
非機能制約付加処理
“サブルーチン” → t>60
“繰り返し” → t>50
“条件分岐” → t>30
“四則演算” → t>10
（時間の制約）
“整理型” → 4バイト
“フロー型” → 8バイト
（容量の制約）

[図8A]

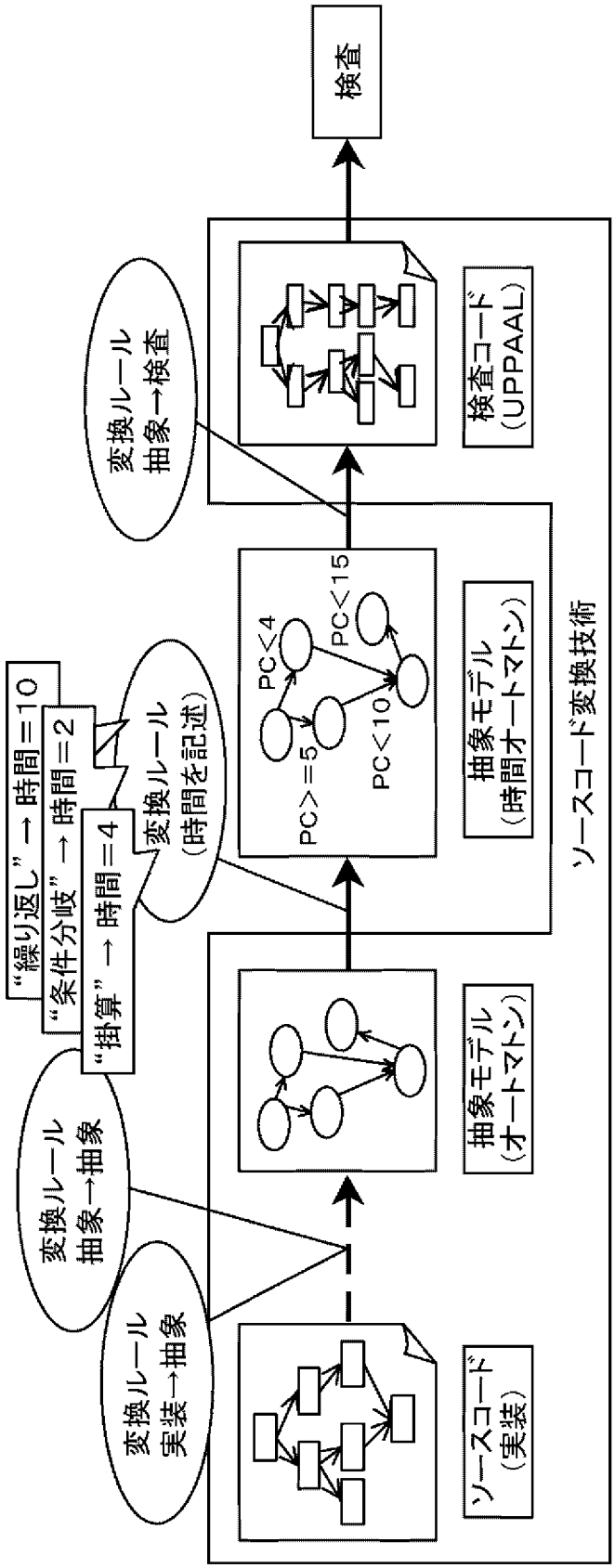
図８Ａ



非機能制約＝時間変換ルール(PC)

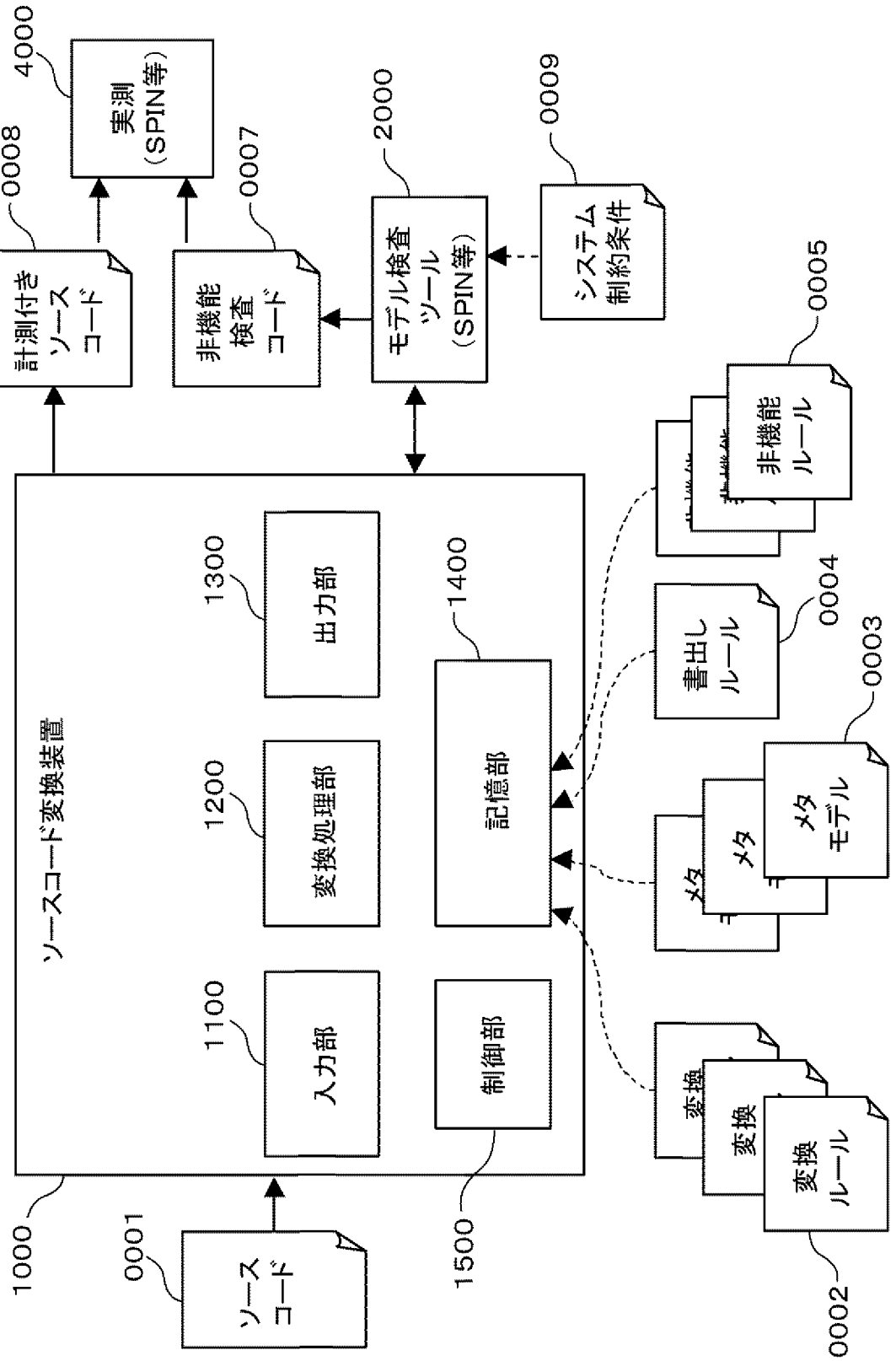
[図8B]

図８Ｂ


検査
検査コード
(UPPAAL)
変換ルール
抽象→検査
PC<4
PC>=5
PC<15
PC<10
抽象モデル
(時間オートマトン)
抽象モデル
(オートマトン)
ソースコード
(実装)
ソースコード変換技術
変換ルール
抽象→抽象
変換ルール
実装→抽象
“繰り返し”→ 時間＝10
“条件分岐”→ 時間＝2
“掛算”→ 時間＝4
変換ルール
(時間を記述)

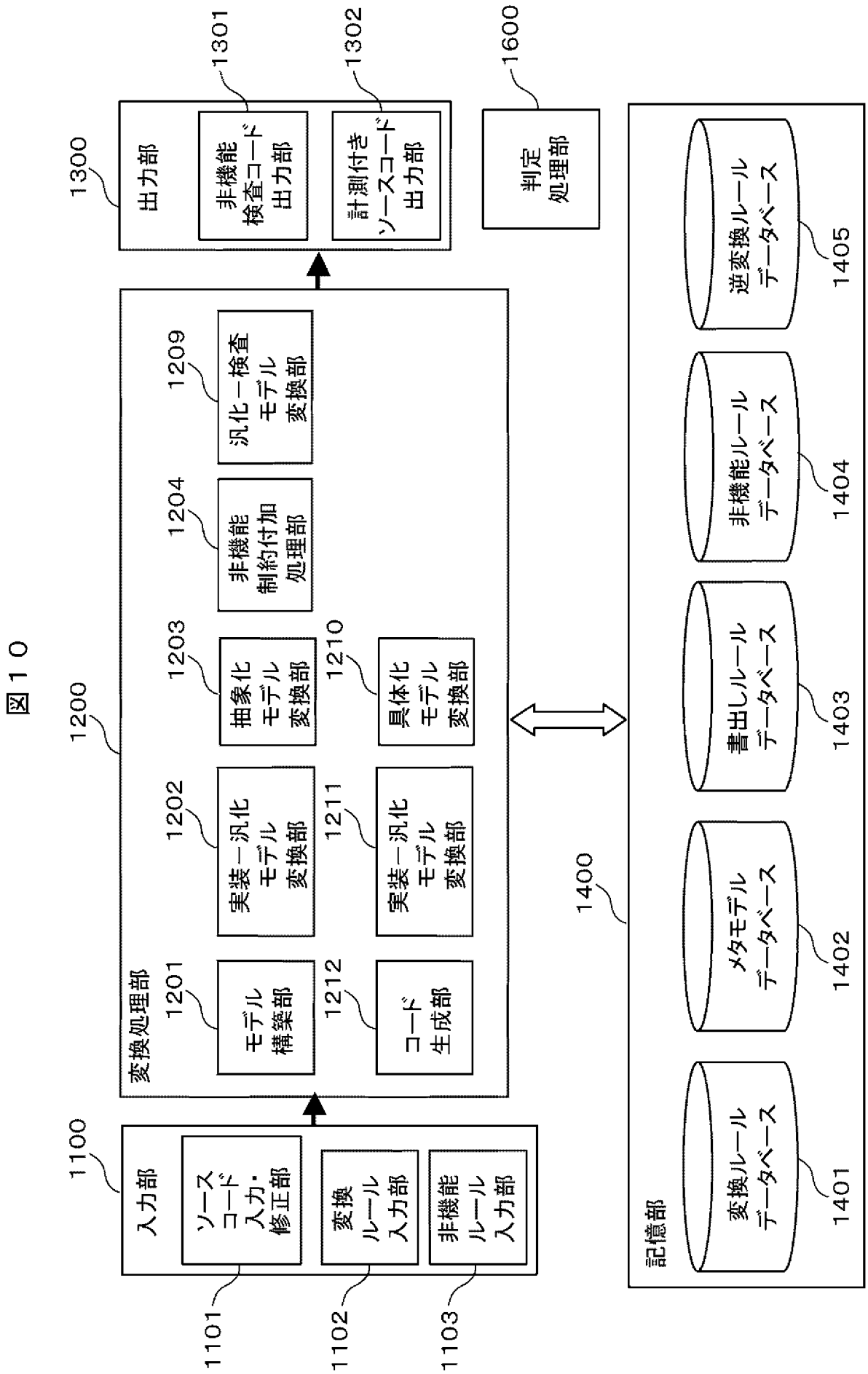
[図9]

図９


4000
実測
（SPIN等）
0008
計測付き
ソース
コード
0007
非機能
検査
コード
2000
モデル検査
ツール
（SPIN等）
0009
システム
制約条件
0005
非機能
ルール
0004
書出し
ルール
0003
メタ
モデル
メタ
メタ
変換
ルール
変換
変換
0002
1000
ソースコード変換装置
1300
出力部
1400
記憶部
1200
変換処理部
1100
入力部
制御部
1500
0001
ソース
コード

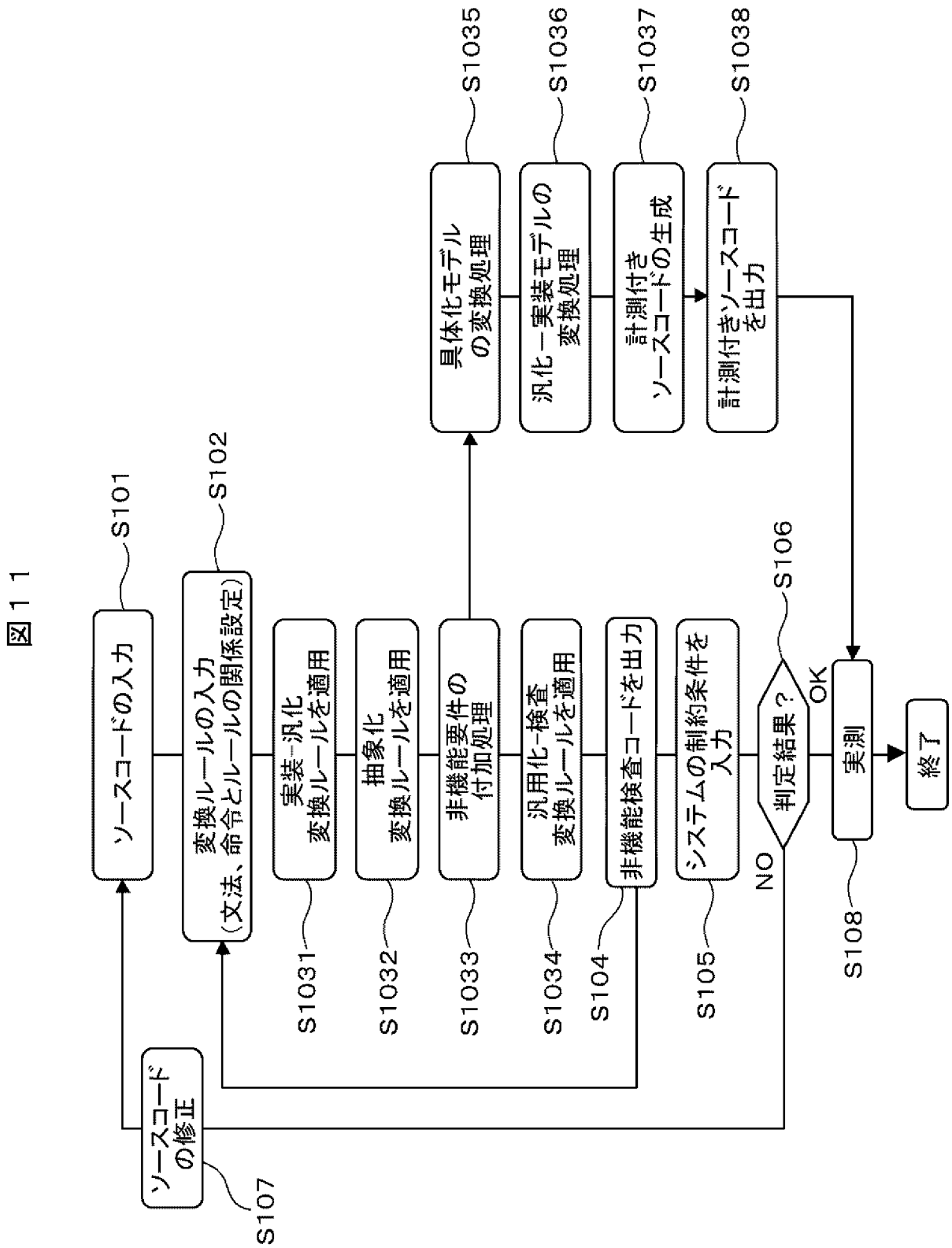
[図10]

図１０


1100
入力部
ソースコード入力・修正部
変換ルール入力部
非機能ルール入力部
1101
1102
1103
1200
変換処理部
1201
モデル構築部
1212
コード生成部
1202
実装一汎化モデル変換部
1211
実装一汎化モデル変換部
1203
抽象化モデル変換部
1210
具体化モデル変換部
1204
非機能制約付加処理部
1209
汎化一検査モデル変換部
1300
出力部
1301
非機能検査コード出力部
1302
計測付きソースコード出力部
1600
判定処理部
1400
記憶部
1401
変換ルールデータベース
1402
メタモデルデータベース
1403
書出しルールデータベース
1404
非機能ルールデータベース
1405
逆変換ルールデータベース

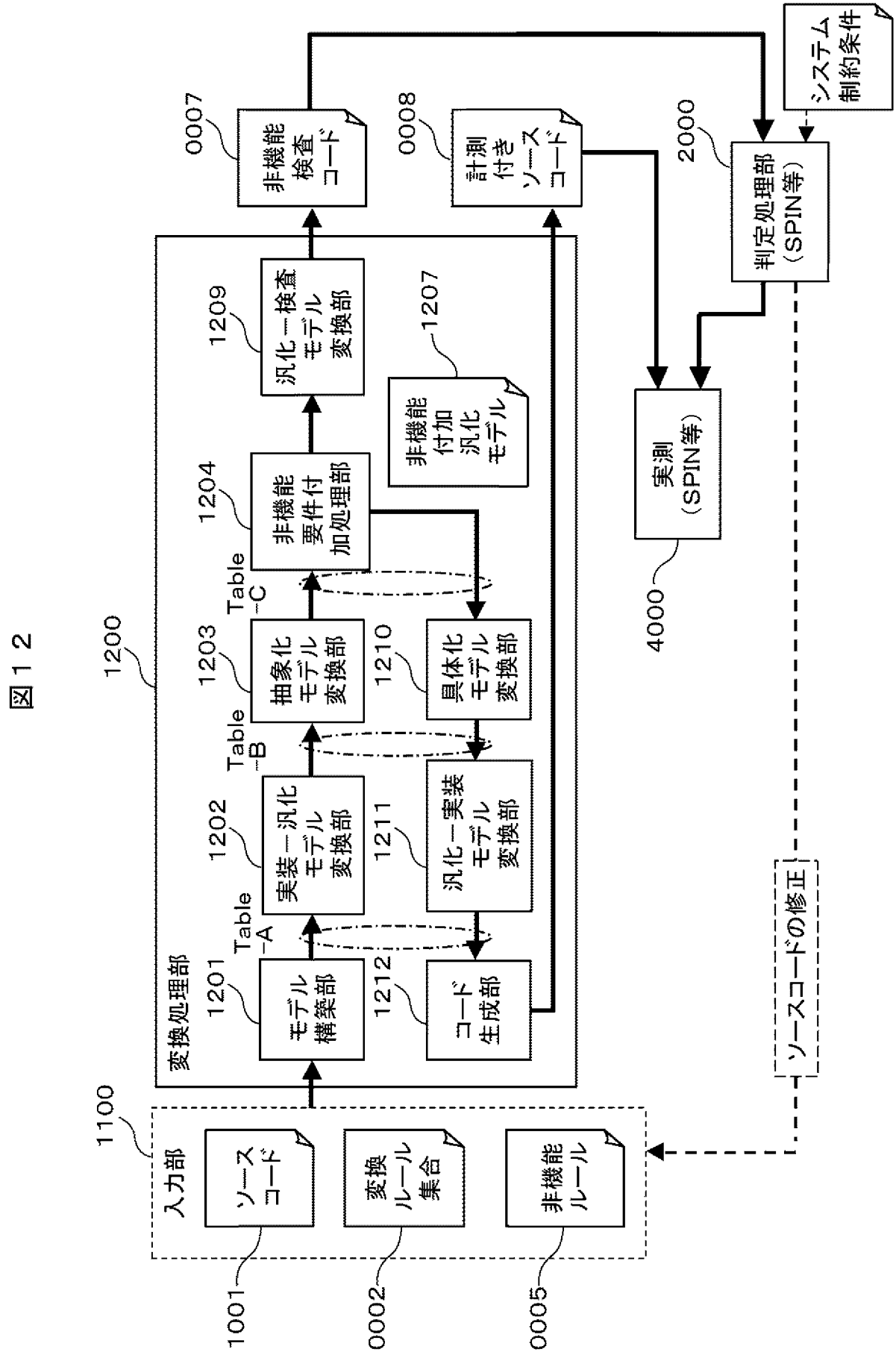
[図11]

図１１


ソースコードの入力
S101
変換ルールの入力
（文法、命令とルールの関係設定）
S102
実装−汎化
変換ルールを適用
S1031
抽象化
変換ルールを適用
S1032
非機能要件の
付加処理
S1033
汎用化−検査
変換ルールを適用
S1034
非機能検査コードを出力
S104
システムの制約条件を
入力
S105
判定結果？
S106
NO
OK
実測
S108
終了
ソースコード
の修正
S107
具体化モデル
の変換処理
S1035
汎化−実装モデルの
変換処理
S1036
計測付き
ソースコードの生成
S1037
計測付きソースコード
を出力
S1038

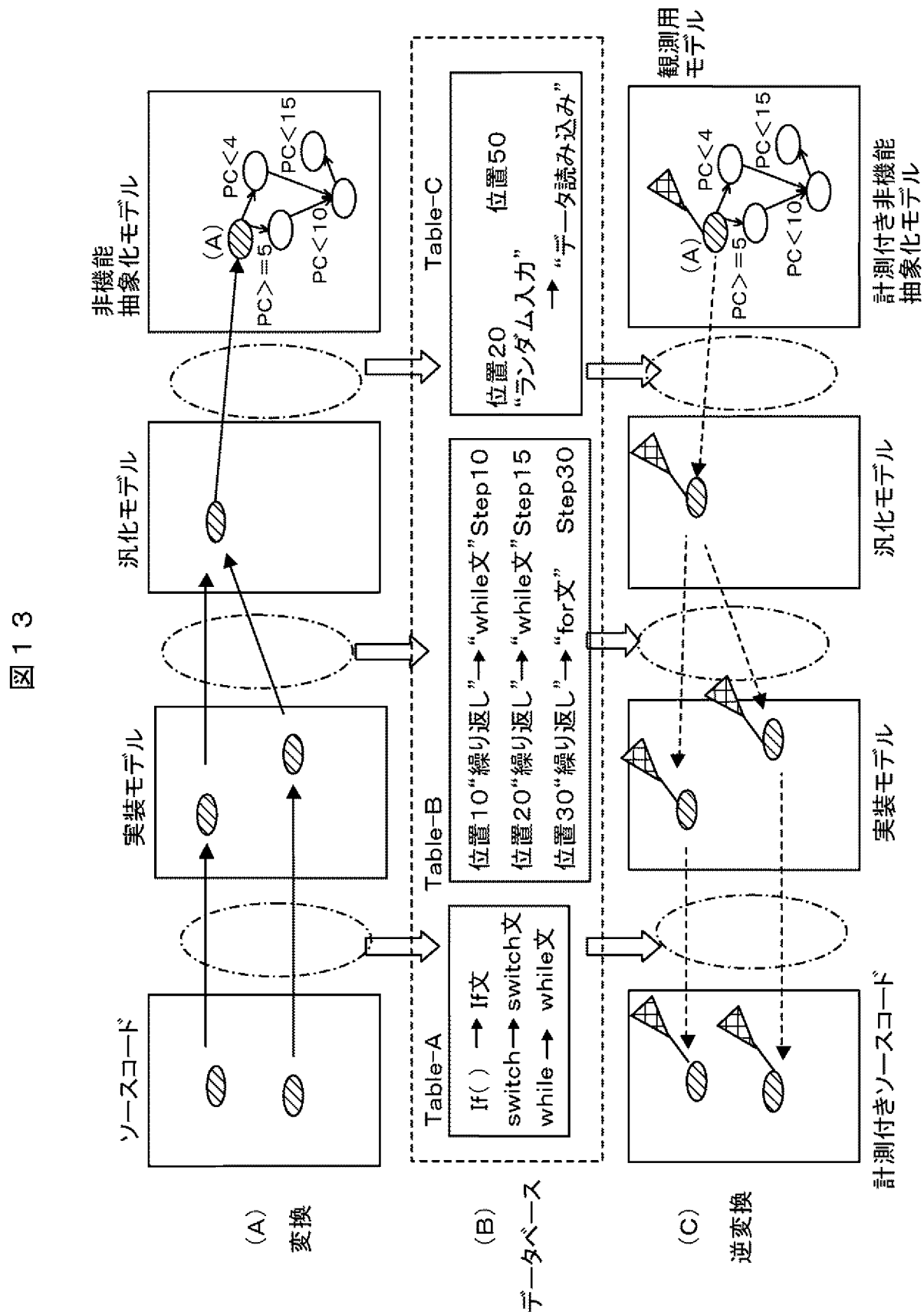
[図12]

図１２


1100
入力部
ソースコード
変換ルール集合
非機能ルール
1001
0002
0005
変換処理部
1200
モデル構築部
1201
コード生成部
1212
Table-A
実装－汎化モデル変換部
1202
汎化－実装モデル変換部
1211
Table-B
抽象化モデル変換部
1203
具体化モデル変換部
1210
Table-C
非機能要件付加処理部
1204
汎化－検査モデル変換部
1209
非機能付加汎化モデル
1207
非機能検査コード
0007
計測付きソースコード
0008
実測（SPIN等）
4000
判定処理部（SPIN等）
2000
システム制約条件
ソースコードの修正

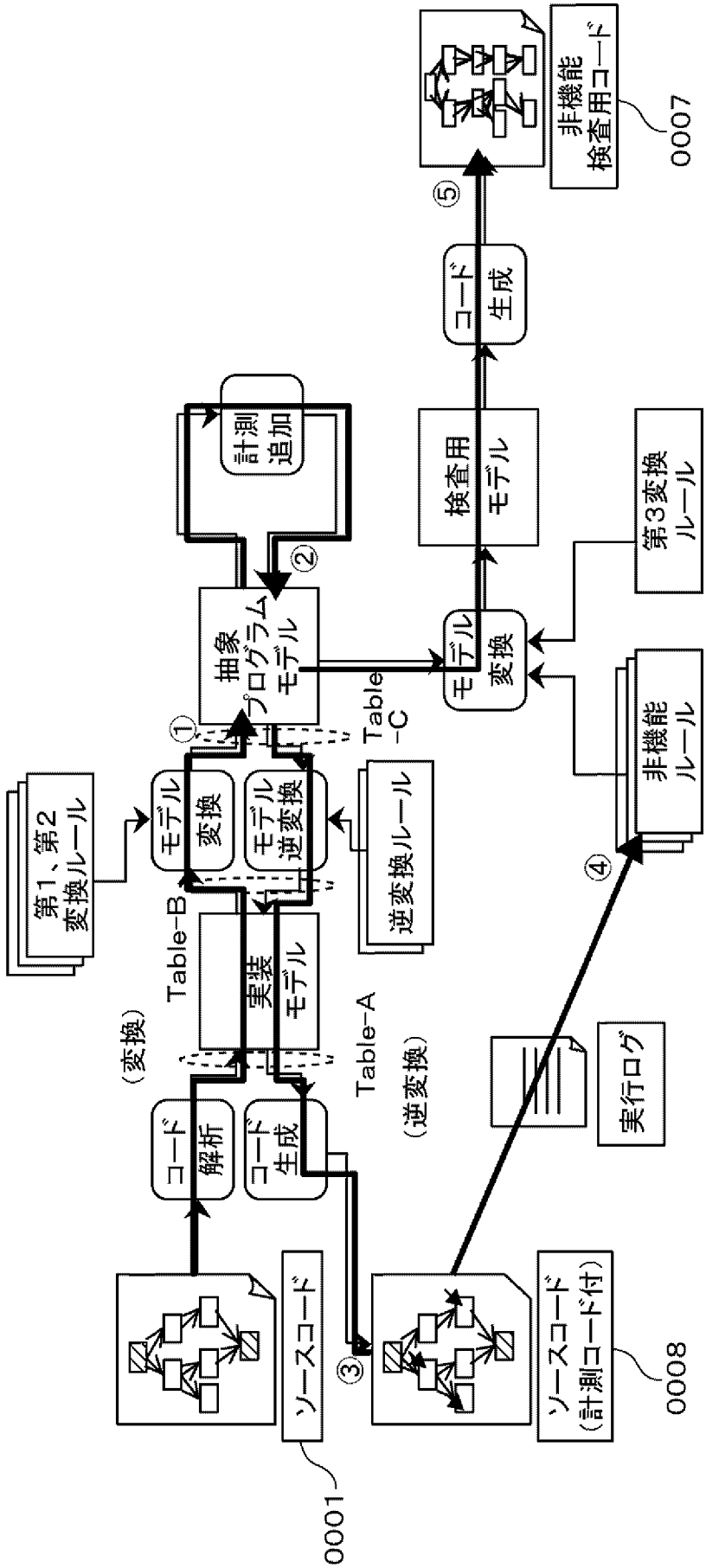
[図13]

図１３


ソースコード
実装モデル
汎化モデル
非機能
抽象化モデル
Table-A
If( ) → If文
switch→ switch文
while → while文
Table-B
位置10“繰り返し”→ “while文”Step10
位置20“繰り返し”→ “while文”Step15
位置30“繰り返し”→ “for文” Step30
Table-C
位置50
位置20
“ランダム入力”
→ “データ読み込み”
(A)
PC<4
PC>=5
PC<15
PC<10
観測用
モデル
計測付きソースコード
実装モデル
汎化モデル
計測付き非機能
抽象化モデル
(A)
変換
(B)
データベース
(C)
逆変換

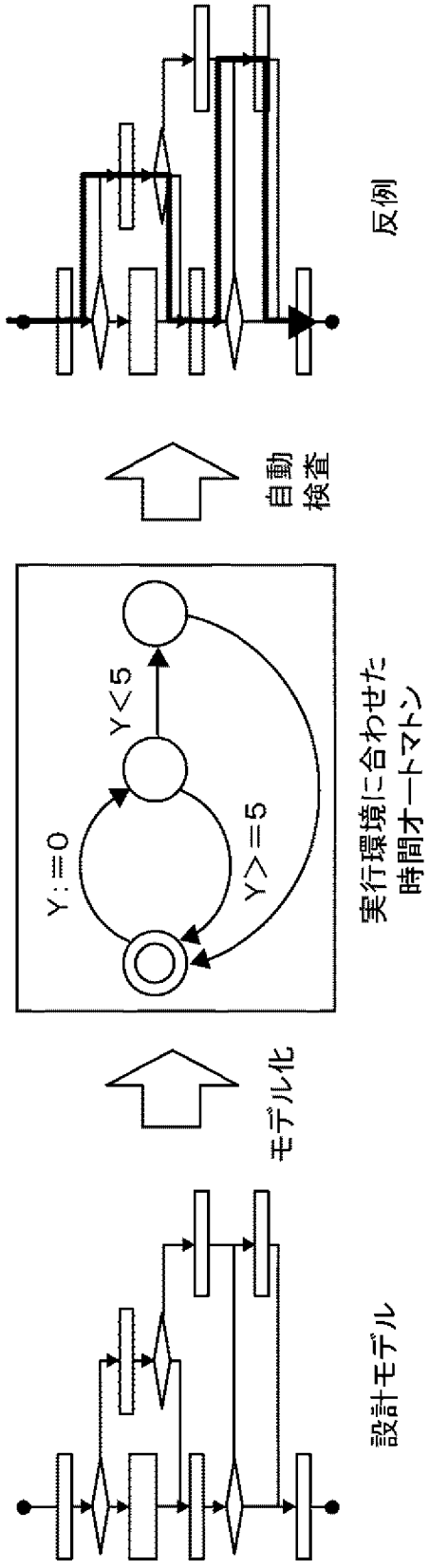
[図14]

図１４


ソースコード
0001
コード解析
コード生成
実装モデル
計測追加
抽象プログラムモデル
モデル変換
モデル逆変換
第1、第2変換ルール
逆変換ルール
Table-B
Table-A
Table-C
（変換）
（逆変換）
①
②
③
④
⑤
検査用モデル
コード生成
非機能検査用コード
0007
第3変換ルール
非機能ルール
実行ログ
ソースコード（計測コード付）
0008

[図15]

図15



INTERNATIONAL SEARCH REPORT

International application No.
PCT/JP2011/074595

A. CLASSIFICATION OF SUBJECT MATTER
G06F11/36(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F11/36

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

Jitsuyo Shinan Koho	1922-1996	Jitsuyo Shinan Toroku Koho	1996-2011
Kokai Jitsuyo Shinan Koho	1971-2011	Toroku Jitsuyo Shinan Koho	1994-2011

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

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y A	Yoshitaka AOKI et al., "Development and Practice of Program Defect Detecting Work Supporting Tool Based on Model Checking", Dai 72 Kai (Heisei 22 Nen) Zenkoku Taikai Koen Ronbunshu (1), Information Processing Society of Japan, 08 March 2010 (08.03.2010), 1-463 to 1-464	1,2,17 3-16,18-20
Y A	JP 2010-140408 A (Nomura Research Institute, Ltd.), 24 June 2010 (24.06.2010), paragraphs [0010] to [0028], [0045]; fig. 1 to 4 (Family: none)	1,2,17 3-16,18-20

☒ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
25 November, 2011 (25.11.11)	06 December, 2011 (06.12.11)
Name and mailing address of the ISA/ Japanese Patent Office Facsimile No.	Authorized officer Telephone No.

INTERNATIONAL SEARCH REPORT

International application No.
PCT/JP2011/074595

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y A	JP 2010-26704 A (Toshiba Corp.), 04 February 2010 (04.02.2010), paragraphs [0013] to [0017]; fig. 1 (Family: none)	1,2,17 3-16,18-20
A	JP 2010-504572 A (National ICT Australia Ltd.), 12 February 2010 (12.02.2010), entire text; all drawings & US 2009/0307664 A1 & EP 2074507 A & WO 2008/034170 A1 & KR 10-2009-0071596 A & AU 2007299571 A	1-20

国際調査報告

国際出願番号　ＰＣＴ／ＪＰ２０１１／０７４５９５

Ａ．　発明の属する分野の分類（国際特許分類（ＩＰＣ））

Int.Cl.　G06F11/36(2006.01)i

Ｂ．　調査を行った分野

調査を行った最小限資料（国際特許分類（ＩＰＣ））

Int.Cl.　G06F11/36

最小限資料以外の資料で調査を行った分野に含まれるもの

日本国実用新案公報	１９２２－１９９６年
日本国公開実用新案公報	１９７１－２０１１年
日本国実用新案登録公報	１９９６－２０１１年
日本国登録実用新案公報	１９９４－２０１１年

国際調査で使用した電子データベース（データベースの名称、調査に使用した用語）

Ｃ．　関連すると認められる文献

引用文献のカテゴリー＊	引用文献名　及び一部の箇所が関連するときは、その関連する箇所の表示	関連する請求項の番号
Y	青木善貴 他，モデル検査に基づくプログラム欠陥抽出作業支援ツールの開発と実践，第７２回（平成２２年）全国大会講演論文集（１），社団法人情報処理学会，2010.03.08，1-463～1-464	1,2,17
A		3-16,18-20
Y	JP 2010-140408 A（株式会社野村総合研究所）2010.06.24，段落【００１０】－【００２８】，【００４５】，図１－４	1,2,17
A	（ファミリーなし）	3-16,18-20

☑　Ｃ欄の続きにも文献が列挙されている。　　☐　パテントファミリーに関する別紙を参照。

＊　引用文献のカテゴリー
「Ａ」特に関連のある文献ではなく、一般的技術水準を示すもの
「Ｅ」国際出願日前の出願または特許であるが、国際出願日以後に公表されたもの
「Ｌ」優先権主張に疑義を提起する文献又は他の文献の発行日若しくは他の特別な理由を確立するために引用する文献（理由を付す）
「Ｏ」口頭による開示、使用、展示等に言及する文献
「Ｐ」国際出願日前で、かつ優先権の主張の基礎となる出願の日の後に公表された文献
「Ｔ」国際出願日又は優先日後に公表された文献であって出願と矛盾するものではなく、発明の原理又は理論の理解のために引用するもの
「Ｘ」特に関連のある文献であって、当該文献のみで発明の新規性又は進歩性がないと考えられるもの
「Ｙ」特に関連のある文献であって、当該文献と他の１以上の文献との、当業者にとって自明である組合せによって進歩性がないと考えられるもの
「＆」同一パテントファミリー文献

国際調査を完了した日	国際調査報告の発送日
２５．１１．２０１１	０６．１２．２０１１

国際調査機関の名称及びあて先
日本国特許庁（ＩＳＡ／ＪＰ）
郵便番号１００－８９１５
東京都千代田区霞が関三丁目４番３号

特許庁審査官（権限のある職員）　５Ｂ　３３５１
石川　亮
電話番号　０３－３５８１－１１０１　内線　３５４５

Ｃ（続き）．　関連すると認められる文献

引用文献のカテゴリー＊	引用文献名　及び一部の箇所が関連するときは、その関連する箇所の表示	関連する請求項の番号
Y	JP 2010-26704 A (株式会社東芝) 2010.02.04, 段落【００１３】－【００１７】, 図１	1,2,17
A	（ファミリーなし）	3-16,18-20
A	JP 2010-504572 A (ナショナル　アイシーティー　オーストラリア リミテッド) 2010.02.12, 全文, 全図 & US 2009/0307664 A1 & EP 2074507 A & WO 2008/034170 A1 & KR 10-2009-0071596 A & AU 2007299571 A	1-20