(19)日本国特許庁(JP)

(12)特許公報(B2)

(11)特許番号
特許第7513204号
(P7513204)

(45)発行日　令和6年7月9日(2024.7.9)

(24)登録日　令和6年7月1日(2024.7.1)

(51)国際特許分類

Ｇ０６Ｆ	*8/65*	*(2018.01)*	Ｇ０６Ｆ	8/65
Ｇ０６Ｆ	*11/36*	*(2006.01)*	Ｇ０６Ｆ	11/36　１０４

ＦＩ

請求項の数　6　（全15頁）

(21)出願番号　特願2023-518603(P2023-518603)
(86)(22)出願日　令和3年5月7日(2021.5.7)
(86)国際出願番号　PCT/JP2021/017569
(87)国際公開番号　WO2022/234675
(87)国際公開日　令和4年11月10日(2022.11.10)
審査請求日　令和5年8月21日(2023.8.21)

(73)特許権者　000004226
日本電信電話株式会社
東京都千代田区大手町一丁目５番１号
(74)代理人　100107766
弁理士　伊東 忠重
(74)代理人　100070150
弁理士　伊東 忠彦
(74)代理人　100124844
弁理士　石原 隆治
(72)発明者　山口 大輔
東京都千代田区大手町一丁目５番１号
日本電信電話株式会社内
審査官　渡辺 一帆

最終頁に続く

(54)【発明の名称】　プログラム修正装置、プログラム修正方法及びプログラム

(57)【特許請求の範囲】

【請求項１】

マイグレーションが必要なＡＰＩを含むソースコードを構文解析して、前記ＡＰＩを呼び出した後の計算を構成する第１の要素について型及び式を抽出する解析部と、

前記第１の要素のうち、前記ＡＰＩのマイグレーション事例におけるマイグレーション前のプログラムとマイグレーション後のプログラムとのソースコードの差分を構成するいずれかの第２の要素と型が共通する前記第１の要素に対応する箇所を、前記ソースコードにおいて書き換えが必要な箇所として特定する特定部と、

を有すること特徴とするプログラム修正装置。

【請求項２】

前記書き換えが必要な箇所の式を、前記第２の要素の式で書き換える書換部、

を有することを特徴とする請求項１記載のプログラム修正装置。

【請求項３】

前記第２の要素において宣言されている変数を、前記ソースコード内で前言されている変数に置き換える翻訳部、

を有することを特徴とする請求項１又は２記載のプログラム修正装置。

【請求項４】

前記特定部は、前記差分に係る抽象構文木の根要素から下降する順序で当該抽象構文木の要素を選択し、選択した要素と型が共通する前記第１の要素が探索されると、探索された前記第１の要素の中で、前記ＡＰＩの呼び出し箇所に最も近い要素を前記書き換えが必

要な箇所として特定する、
ことを特徴とする請求項１乃至３いずれか一項記載のプログラム修正装置。
【請求項５】
マイグレーションが必要なＡＰＩを含むソースコードを構文解析して、前記ＡＰＩを呼び出した後の計算を構成する第１の要素について型及び式を抽出する解析手順と、
前記第１の要素のうち、前記ＡＰＩのマイグレーション事例におけるマイグレーション前のプログラムとマイグレーション後のプログラムとのソースコードの差分を構成するいずれかの第２の要素と型が共通する前記第１の要素に対応する箇所を、前記ソースコードにおいて書き換えが必要な箇所として特定する特定手順と、
をコンピュータが実行すること特徴とするプログラム修正方法。
【請求項６】
請求項１乃至４いずれか一項記載のプログラム修正装置としてコンピュータを機能させることを特徴とするプログラム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、プログラム修正装置、プログラム修正方法及びプログラムに関する。
【背景技術】
【０００２】
ライブラリの新バージョンリリースに際してＡＰＩの非推奨化や削除があると、当該ＡＰＩを利用するプログラム（クライアントコード）について、ＡＰＩのマイグレーション（呼び出すＡＰＩの変更）が必要になる。ライブラリの利用者は、マイグレーションガイドなどに示されたマイグレーション事例、又は事例の収集を可能にする技術によって、ＡＰＩをマイグレーションするために必要な具体的なコードの編集操作を知ることができる。但し、その編集操作をクライアントコードにそのまま適用することでＡＰＩのマイグレーションができるわけではない。
【０００３】
例えば、図１に示すようなマイグレーション事例を考える。以下のような新旧コードの対応が得られたとする。図１において、（１）は、マイグレーション前のプログラムを示し、（２）は、マイグレーション後のプログラムを示す。
【０００４】
図１の例は、型Ｒｅｃｔａｎｇｌｅの値を返すａｐｉ（）が、非推奨化又は削除されることによるマイグレーション事例を示す。マイグレーション先として型Ｓｈａｐｅ（ＳｈａｐｅはＲｅｃｔａｎｇｌｅの上位型である）の値を返すａｐｉ'（）があるため、コードの編集操作としては、ａｐｉ（）の呼び出し部分をａｐｉ'（）に置き換えをすれば良いことを示している。
【０００５】
但し、ａｎｏｔｈｅｒＣａｌｃｕｌａｔｉｏｎ（ａｐｉ（））；というプログラムに適用した場合にマイグレーションが成功するか否かは、ａｐｉ（）を呼び出した後のａｎｏｔｈｅｒＣａｌｃｕｌａｔｉｏｎの計算に依存する。ａｎｏｔｈｅｒＣａｌｃｕｌａｔｉｏｎが、ｓｏｍｅＣａｌｃｕｌａｔｉｏｎと同様に、必ずしもＲｅｃｔａｎｇｌｅ型の値は必要とせずＳｈａｐｅ型の値で計算が可能な場合、マイグレーションは成功する。しかし、ａｎｏｔｈｅｒＣａｌｃｕｌａｔｉｏｎがＲｅｃｔａｎｇｌｅ型の値を要求する場合、マイグレーションは失敗する。このようなマイグレーションの失敗を回避するために、クライアントコードのａｐｉ（）を呼び出した後の計算に即したコード編集操作を適用する必要がある。
【０００６】
従来、編集操作を自動的にクライアントコードに適用する技術として、Update example analysisが提案されている（非特許文献１）。Update example analysisの目的は、どのマイグレーション事例でも共通して行われているクライアントコードの編集操作を発

見し、当該編集操作をクライアントコードに適用することである。したがって、大量のマイグレーション事例が収集され、それらの編集操作について共通部分が抽出される。

【先行技術文献】

【非特許文献】

【０００７】

【文献】Mattia Fazzini, Qi Xin, and Alessandro Orso、"Automated API-usage update for Android apps"、In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis(ISSTA 2019)、Association for Computing Machinery, New York, NY, USA, 204-215、［online］、インターネット＜ＵＲＬ：https://doi.org/10.1145/3293882.3330571＞

【発明の概要】

【発明が解決しようとする課題】

【０００８】

非特許文献１の方法は、どのマイグレーション事例でも共通して行われているクライアントコードの編集操作のみを使うため、クライアントコードに対し必要以上の編集を加えないが、ＡＰＩを呼び出した後の計算に対する編集が必要であったとしても、複数のマイグレーション事例の編集操作の共通部分が抽出されてしまうことでクライアントコードに適用される編集操作にならない（マイグレーションが失敗してしまう）。

【０００９】

例えば、マイグレーション事例の一つに、ａｎｏｔｈｅｒＣａｌｃｕｌａｔｉｏｎ（ａｐｉ（））；を、ａｎｏｔｈｅｒＣａｌｃｕｌａｔｉｏｎ（ａｐｉ'（）．ｇｅｔＲｅｃｔａｎｇｌｅ（））；に修正しているものがあったとしても、図１に示した例のように、ｇｅｔＲｅｃｔａｎｇｌｅ（）を呼ばず、ａｐｉ'（）を呼び出すだけマイグレーションが完了している事例が混在していると、それらの共通部分はａｐｉ（）をａｐｉ'（）に書き換える部分だけになってしまう。その結果、前述のマイグレーション失敗を回避することができない。

【００１０】

本発明は、上記の点に鑑みてなされたものであって、ＡＰＩのマイグレーションの成功率を高めることを目的とする。

【課題を解決するための手段】

【００１１】

そこで上記課題を解決するため、プログラム修正装置は、マイグレーションが必要なＡＰＩを含むソースコードを構文解析して、前記ＡＰＩを呼び出した後の計算を構成する第１の要素について型及び式を抽出する解析部と、前記第１の要素のうち、前記ＡＰＩのマイグレーション事例におけるマイグレーション前のプログラムとマイグレーション後のプログラムとのソースコードの差分を構成するいずれかの第２の要素と型が共通する前記第１の要素に対応する箇所を、前記ソースコードにおいて書き換えが必要な箇所として特定する特定部と、を有する。

【発明の効果】

【００１２】

ＡＰＩのマイグレーションの成功率を高めることができる。

【図面の簡単な説明】

【００１３】

【図１】マイグレーション事例を示す図である。

【図２】本発明の実施の形態におけるプログラム修正装置１０のハードウェア構成例を示す図である。

【図３】本発明の実施の形態におけるプログラム修正装置１０の機能構成例を示す図である。

【図４】プログラム修正装置１０が実行する処理手順の一例を説明するためのフローチャートである。

【図５】事後計算解析の処理手順の一例を説明するためのフローチャートである。
【図６】クライアントコードの一例を示す図である。
【図７】抽象構文木の一例を示す図である。
【図８】事後計算比較の処理手順の一例を説明するためのフローチャートである。
【図９】期待される書き換えの一例を説明するための図である。
【発明を実施するための形態】
【００１４】
本実施の形態では、収集されたマイグレーション事例を用いて、修正対象のクライアントコードを自動的に修正する技術が開示される。修正に際し、ＡＰＩを呼び出した後の計算で用いられる型のプログラムがマイグレーション事例から再構成される。すなわち、マイグレーション後のＡＰＩの戻り値の型がマイグレーション前と変わる場合、ＡＰＩを呼び出した後の計算で用いられるプログラムの型に対し、ＡＰＩ呼び出しによって得られる値の型の間で型の不整合が発生する。この不整合を解消ためには、ＡＰＩ呼び出しによって得られる値の型からＡＰＩを呼び出した後の計算で用いられるプログラムの型への変換を実現するプログラムが必要となる。「再構成」とは、斯かる変換を意味する。本実施の形態では、マイグレーション事例でのコード編集の一部を切り出してクライアントコードに反映させることで再構成が実現される。また、本実施の形態では、収集されたマイグレーション事例について共通部分を求める操作は行わないため、収集された事例の数はマイグレーションの成否に影響を与えない。
【００１５】
なお、クライアントコードとは、ＡＰＩを利用するプログラムのソースコードのことをいい、ＡＰＩを提供するライブラリのソースコードと区別するための用語である。また、ＡＰＩのマイグレーションとは、クライアントコードが呼び出すＡＰＩを変更することをいう。また、クライアントコードの一部、又はマイグレーション事例に関して使用する「プログラム」という用語は、プログラムの構文定義を意味する。プログラムの構文定義とは、文や式、及びそれらからなる文又は式の列のことをいい、文及び式はそれぞれ再帰的な構造を有する。
【００１６】
以下、図面に基づいて本発明の実施の形態を説明する。図２は、本発明の実施の形態におけるプログラム修正装置１０のハードウェア構成例を示す図である。図２のプログラム修正装置１０は、それぞれバスＢで相互に接続されているドライブ装置１００、補助記憶装置１０２、メモリ装置１０３、ＣＰＵ１０４、インタフェース装置１０５、表示装置１０６、及び入力装置１０７等を有する。
【００１７】
プログラム修正装置１０での処理を実現するプログラムは、ＣＤ－ＲＯＭ等の記録媒体１０１によって提供される。プログラムを記憶した記録媒体１０１がドライブ装置１００にセットされると、プログラムが記録媒体１０１からドライブ装置１００を介して補助記憶装置１０２にインストールされる。但し、プログラムのインストールは必ずしも記録媒体１０１より行う必要はなく、ネットワークを介して他のコンピュータよりダウンロードするようにしてもよい。補助記憶装置１０２は、インストールされたプログラムを格納すると共に、必要なファイルやデータ等を格納する。
【００１８】
メモリ装置１０３は、プログラムの起動指示があった場合に、補助記憶装置１０２からプログラムを読み出して格納する。ＣＰＵ１０４は、メモリ装置１０３に格納されたプログラムに従ってプログラム修正装置１０に係る機能を実現する。インタフェース装置１０５は、ネットワークに接続するためのインタフェースとして用いられる。表示装置１０６はプログラムによるＧＵＩ（Graphical User Interface）等を表示する。入力装置１０７はキーボード及びマウス等で構成され、様々な操作指示を入力させるために用いられる。
【００１９】
図３は、本発明の実施の形態におけるプログラム修正装置１０の機能構成例を示す図で

ある。図３に示されるように、プログラム修正装置１０は、事後計算解析部１１、事後計算比較部１２、変数翻訳部１３及びコード書換部１４を有する。これら各部は、プログラム修正装置１０にインストールされた１以上のプログラムが、ＣＰＵ１０４に実行させる処理により実現される。

【００２０】

事後計算解析部１１は、修正対象のクライアントコードにおける、ＡＰＩの呼び出し後の計算（以下、「事後計算」という。）を解析して後述のモデル（以下、「事後計算モデル」という。）を構築し、クライアントコードの事後計算で使用される型を、クライアントコードから抽出する。なお、計算とは、クライアントコード上の定義をいう。

【００２１】

事後計算比較部１２は、収集された複数のマイグレーション事例のそれぞれの事後計算とクライアントコードの事後計算とを比較し、クライアントコードで書き換えが必要な箇所を特定すると共に、マイグレーション事例におけるマイグレーション前後の差分を構成する計算要素（式及び型）のうち、クライアントコードの事後計算の要素と型が一致する要素（以下、「共通要素」という。）を特定する。

【００２２】

変数翻訳部１３は、マイグレーション事例におけるマイグレーション後のプログラムについて、コード内の変数とクライアントコードの変数の対応を取る。

【００２３】

コード書換部１４は、事後計算比較部１２が求めた共通要素を利用して、クライアントコードの書き換えを行う。

【００２４】

なお、図３における入力値及び出力値のそれぞれのデータ構造と、事後計算モデルとをＢＮＦ（Backus-Naur form）記法に基づく形式によって記すと以下の通りである。

【００２５】

［入力値］

＜マイグレーション事例＞：＝＜マイグレーション前のプログラム＞　＜マイグレーション後のプログラム＞

＜マイグレーション前のプログラム＞：＝＜Ｓｔｒｉｎｇ＞

＜マイグレーション後のプログラム＞：＝＜Ｓｔｒｉｎｇ＞

＜クライアントコード＞：＝＜Ｓｔｒｉｎｇ＞

［出力値］

＜修正済みクライアントコード＞：＝＜Ｓｔｒｉｎｇ＞

［事後計算モデル］

＜事後計算＞：＝（＜式＞　＜型＞）＊

以下、図Ａ１のプログラム修正装置１０の処理手順について説明する。図４は、プログラム修正装置１０が実行する処理手順の一例を説明するためのフローチャートである。

【００２６】

ステップＳ１０１において、事後計算解析部１１は、クライアントコードを入力する。入力したクライアントコードを、以下「対象クライアントコード」という。

【００２７】

続いて、事後計算解析部１１は、対象クライアントコードについて、事後計算解析を実行することで、事後計算モデルを構築する（Ｓ１０２）。事後計算解析の結果として得られる（抽出される）事後計算モデルは、クライアントコードの事後計算で宣言されている式とその型のリストであり、事後計算においてどのような型が使用されているかを示す。

【００２８】

続いて、事後計算比較部１２は、入力された複数のマイグレーション事例のうち、未処理のマイグレーション事例の有無を判定する（Ｓ１０３）。

【００２９】

未処理のマイグレーション事例が有る場合（Ｓ１０３でＹｅｓ）、事後計算比較部１２

は、未処理のマイグレーション事例のうちの一つのマイグレーション事例を処理対象（以下、「対象事例」という。）として選択する（Ｓ１０４）。

【００３０】

続いて、事後計算比較部１２は、対象事例と、事後計算解析で得られたクライアントコードの事後計算モデルとを比較することで、対象クライアントコードにおいて書き換えが必要な箇所を特定すると共に、対象事例について、対象クライアントコードの事後計算との間の共通要素を特定する（Ｓ１０５）。該当する共通要素が無い場合（Ｓ１０６でＮｏ）、ステップＳ１０３へ戻る。したがって、この場合、対象事例を利用したコード書き換えは実行されない。

【００３１】

該当する共通要素が有る場合（Ｓ１０６でＹｅｓ）、プログラム修正装置１０は、ステップＳ１０７及びＳ１０８において、対象事例に従ったコードの書き換えをクライアントコードに反映する。

【００３２】

具体的には、ステップＳ１０７において、変数翻訳部１３は、対象事例のマイグレーション後のプログラムで宣言されている変数（より厳密には、共通要素における変数）を、対象クライアントコード内で宣言されている変数に置き換えることで、当該マイグレーション後のプログラムを変更する。斯かる変更を、以下「変数翻訳」という。

【００３３】

続いて、コード書換部１４は、変数翻訳されたマイグレーション事例に従って、対象クライアントコードの複製に対して書き換えを行うことで、修正済みクライアントコードを生成する（Ｓ１０８）。斯かる書き換えは、対象クラアイントコードについて、ステップＳ１０５において特定された箇所を、（変数翻訳された）共通要素で置き換えることで実現される。

【００３４】

続いて、コード書換部１４は、修正済みクライアントコードを出力する。すなわち、修正済みクライアントコードごとは、共通要素が有ると判定されたマイグレーション事例ごとに生成及び出力される。

【００３５】

続いて、事後計算解析（Ｓ１０２）、事後計算比較（Ｓ１０５）、変数翻訳（Ｓ１０７）の３つの処理について詳細を説明する。

【００３６】

［事後計算解析］

図５は、事後計算解析の処理手順の一例を説明するためのフローチャートである。事後計算解析は、対象クライアントコードを入力として、事後計算のモデルを出力する。

【００３７】

ステップＳ２０１において、事後計算解析部１１は、対象クライアントコードを構文解析して、抽象構文木ａｓｔ＿ｗｈｏｌｅを生成する。

【００３８】

続いて、事後計算解析部１１は、ａｓｔ＿ｗｈｏｌｅからマイグレーションが必要なＡＰＩの呼び出し箇所ｅｒｒｏｒ＿ａｓｔを取得する（Ｓ２０２）。なお、ｅｒｒｏｒ＿ａｓｔは、抽象構文木ａｓｔ＿ｗｈｏｌｅにおいて、当該箇所に対応するノード以下の部分木である。また、マイグレーションが必要なＡＰＩの呼び出し箇所は、クライアントコードをコンパイルし、コンパイル時の検査結果に基づいて特定されてもよいし、他の方法によって特定されてもよい。また、図５では、当該箇所が１箇所であるという想定に基づいて、処理手順を説明する。当該箇所が複数である場合には、複数の箇所についてステップＳ２０２以降が実行されればよい。

【００３９】

続いて、事後計算解析部１１は、リスト形式の変数であるモデルを空の状態に初期化する（Ｓ２０３）。

【００４０】
続いて、事後計算解析部１１は、ｅｒｒｏｒ＿ａｓｔを利用するプログラムｕｓａｇｅを、対象クライアントコード内においてｅｒｒｏｒ＿ａｓｔを含むコードブロックから取得する（Ｓ２０４）。コードブロックとは、括弧で囲まれた範囲をいう。「ｅｒｒｏｒ＿ａｓｔを利用するプログラム」とは、「マイグレーションが必要なＡＰＩ」を呼び出した結果を利用しているプログラムのことをいう。事後計算解析部１１は、ｅｒｒｏｒ＿ａｓｔの親ノードが以下のいずれかにマッチした場合に、ｅｒｒｏｒ＿ａｓｔを利用するプログラムとして当該親ノードを取得する。
・ｒｅｔｕｒｎ文
・メソッド呼び出し
・フィールドアクセス
・コンストラクタ呼び出し
・代入文
なお、メソッド呼び出しとは、ＡＰＩの呼び出しではなく、任意のメソッド呼び出しである。また、フィールドアクセスとは、或るデータ型の値が持つフィールドへの参照のことをいう。例えば、Ｃ言語であれば構造体のメンバへのポインタがフィールドアクセスに相当する。
【００４１】
続いて、事後計算解析部１１は、ｕｓａｇｅの型と式のペアを１つの要素としてｍｏｄｅｌへ追加する（Ｓ２０５）。続いて、事後計算解析部１１は、ｕｓａｇｅを新たなｅｒｒｏｒ＿ａｓｔとする（Ｓ２０６）。すなわち、ｅｒｒｏｒ＿ａｓｔが指すノードが、ａｓｔ＿ｗｈｏｌｅにおいて根ノード側に移動する。但し、ｕｓａｇｅが代入文である場合には、事後計算解析部１１は、例外的に、当該代入文において代入先に対応するノードを新たなｅｒｒｏｒ＿ａｓｔとする。
【００４２】
続いて、事後計算解析部１１は、ａｓｔ＿ｗｈｏｌｅにおけるｕｓａｇｅの親ノード（部分木ｕｓａｇｅの根ノードの親ノード）が、コードブロックであるか否かを判定する（Ｓ２０７）。例えば、｛ｒｅｔｕｒｎ　ｇ（ｆ（ｘ））；｝というプログラムについて、ｕｓａｇｅがｒｅｔｕｒｎ　ｇ（ｆ（ｘ））；（但し、実際には抽象構文木に変換されている。）であるならば、ｕｓａｇｅの親ノードは｛ｒｅｔｕｒｎ　ｇ（ｆ（ｘ））；｝になるため、この場合、ｕｓａｇｅの親ノードがコードブロックに相当する。
【００４３】
当該親ノードがコードブロックでない場合（Ｓ２０７でＮｏ）、事後計算解析部１１は、ステップＳ２０４以降を繰り返す。当該親ノードがコードブロックである場合（Ｓ２０７でＹｅｓ）、事後計算解析部１１は、ｍｏｄｅｌを事後計算のモデルとして出力する（Ｓ２０８）。最終的に、ｍｏｄｅｌにはｅｒｒｏｒ＿ａｓｔを再帰的に利用するプログラムのリストが得られる。
【００４４】
例えば、対象クライアントコードの一部分が図６の（１）の通りであるとすると、ステップＳ２０１では、図７に示すような抽象構文木が生成される。但し、クライアントコードにおいて、ｖａｒ．ｎｕｍｂｅｒＯｆＥｄｇｅｓのスコープ内では、図６の（２）のクラス定義がなされているとする。
【００４５】
ここで、ａｐｉ（）がマイグレーションの必要なＡＰＩであるとすると、ステップＳ２０２では、図７のノードｎ１がｅｒｒｏｒ＿ａｓｔとして取得される。
【００４６】
ステップＳ２０４では、図７のノードｎ２がｕｓａｇｅとして取得される。ノードｎ２は、代入文に相当するからである。したがって、ステップＳ２０５では、ノードｎ２の式及び型である（Ｒｅｃｔａｎｇｌｅ，ｖａｒ）がｍｏｄｅｌに追加される。
【００４７】

ステップＳ２０６では、ｕｓａｇｅが代入文であるため、図７において、ノードｎ２の代入先のノードであるノードｎ３が新たなｅｒｒｏｒ＿ａｓｔとされる。

【００４８】

新たなｅｒｒｏｒ＿ａｓｔに関して実行される２巡目のステップＳ２０４では、図７のノードｎ４がｕｓａｇｅとして取得される。ノードｎ４は、フィールドアクセスに相当するからである。したがって、２巡目のステップＳ２０５では、ノードｎ４の式及び型である（ｉｎｔ，ｖａｒ．ｎｕｍｂｅｒＯｆＥｄｇｅｓ）がｍｏｄｅｌに追加される。その結果、ｍｏｄｅｌは、［（Ｒｅｃｔａｎｇｌｅ，ｖａｒ），（ｉｎｔ，ｖａｒ．ｎｕｍｂｅｒＯｆＥｄｇｅｓ）］となる。

【００４９】

２巡目のステップＳ２０６では、ノードｎ４が新たなｅｒｒｏｒ＿ａｓｔとされる。

【００５０】

３巡目のステップＳ２０４では、図７のノードｎ５がｕｓａｇｅとして取得される。ノードｎ５は、ｒｅｔｕｒｎ文に相当するからである。したがって、３巡目のステップＳ２０５では、ノードｎ５の式及び型である（ｉｎｔ，ｖａｒ．ｎｕｍｂｅｒＯｆＥｄｇｅｓ）がｍｏｄｅｌに追加されるが、同じ要素が既にｍｏｄｅｌ内にあるため、単に、上書きされる。従って、ｍｏｄｅｌの内容は変化しない。

【００５１】

３巡目のステップＳ２０６では、ノードｎ５が新たなｅｒｒｏｒ＿ａｓｔとされる。

【００５２】

３巡目のステップＳ２０７では、ｕｓａｇｅ（ノードｎ５）の親ノードがコードブロックであるため、ステップＳ２０８において、この時点のｍｏｄｅｌの内容である［（Ｒｅｃｔａｎｇｌｅ，ｖａｒ），（ｉｎｔ，ｖａｒ．ｎｕｍｂｅｒＯｆＥｄｇｅｓ）］が出力される。

【００５３】

［事後計算比較］

図８は、事後計算比較の処理手順の一例を説明するためのフローチャートである。事後計算比較は、対象クライアントコードの事後計算のモデルと対象事例を入力として受け取り、クライアントコードで書き換えが必要な箇所を特定するとともに、対象事例についての対象クライアントコードとの共通要素を特定する。

【００５４】

ステップＳ３０１において、事後計算比較部１２は、対象クライアントコードの事後計算のモデルｍｏｄｅｌを取得する。続いて、事後計算比較部１２は、対象事例からマイグレーション前のプログラムｂｅｆｏｒｅを取得する（Ｓ３０２）。続いて、事後計算比較部１２は、対象事例からマイグレーション後のプログラムａｆｔｅｒを取得する（Ｓ３０３）。続いて、事後計算比較部１２は、ｂｅｆｏｒｅとａｆｔｅｒとのコード差分（ｂｅｆｏｒｅに対するａｆｔｅｒの差分（相違部分））ｄｉｆｆを抽象構文木形式で取得する（Ｓ３０４）。

【００５５】

抽象構文木形式のコード差分は非特許文献１のUpdate example analysisを利用して取得することができる。例えば、マイグレーション事例が、ｃａｌｃ（ａｐｉ（））をｃａｌｃ（ａｐｉ'（）．ｇｅｔＲｅｃｔａｎｇｌｅ（））に書き換えている場合の抽象構文木形式のコード差分は以下のようになる。

```
[（T，calc（api'（）．getRectangle（））），
    [（Rectangle，api'（）．getRectangle（）），
        [（Shape，api'（））]
]
]
```

続いて、事後計算比較部１２は、ｄｉｆｆのノード（T_d，e_d）ごとにループ処理Ｌ１を実行する。以下、ループ処理Ｌ１において処理対象とされているノードを「対象ノー

ド」という。対象ノードは、ｄｉｆｆの根要素から下降する順序で選択される。なお、Ｔｄは、対象ノードの型を示し、ｅｄは、対象ノードの式を示す。

【００５６】

続いて、事後計算比較部１２は、ｍｏｄｅｌのうち、対象ノードの型Ｔｄと同じ型を含む要素をリストｓａｍｅに代入する（Ｓ３０５）。

【００５７】

続いて、事後計算比較部１２は、ｓａｍｅが空であるか否かを判定する（Ｓ３０６）。ｓａｍｅが空である場合（Ｓ３０６でＮｏ）、対象ノードについてステップＳ３０７以降が実行されない。

【００５８】

ｓａｍｅが空でない場合（対象ノードの型Ｔｄと同じ型を含む要素が探索されると）（Ｓ３０６でＹｅｓ）、ｓａｍｅの先頭要素を図４のステップＳ１０８においてクライアントコードで書き換えが必要な箇所として特定する（Ｓ３０７）。すなわち、ステップＳ１０８では、当該先頭要素に対応する箇所が書き換えられる。

【００５９】

なお、先頭の要素のみを書き換えが必要な箇所とするのは、ｓａｍｅの要素の順序とｍｏｄｅｌの要素の順序が共通であることに基づく。事後計算解析によって生成されたｍｏｄｅｌは、その要素の順序が「マイグレーションが必要なＡＰＩの呼び出し箇所」から抽象構文木の深さ方向の順になっている。つまり、ｍｏｄｅｌの中に隣接する２要素ａ，ｂがあるとき、ａとｂの抽象構文木上での位置関係は、ｂはａの子ノードとなっている。この順序がｓａｍｅでも共通してあることと、なるべく書き換える範囲は小さいことが望ましいことを踏まえると、ｓａｍｅの要素が複数であったとしても先頭要素以外は無視し、先頭の要素だけを「クライアントコードで書き換えが必要な箇所」とするのが合理的であるといえるからである。

【００６０】

続いて、事後計算比較部１２は、（Ｔｄ，ｅｄ）を対象事例について、対象クライアントコードとの共通要素として出力して（Ｓ３０８）、図８の処理を終了する。ループの途中で抜ける可能性が有るのは、一つでも共通要素が見つかればよいからである。当該（Ｔｄ，ｅｄ）は、ｄｉｆｆの深さ方向の位置関係において、「マイグレーションが必要なＡＰＩの呼び出し箇所」最も近い要素に相当する。そうすることで、最も近い要素において、型の不整合の解消が行われる。

【００６１】

例えば、クライアントコードの事後計算のモデルが、［（Ｒｅｃｔａｎｇｌｅ，ｖａｒ），（ｉｎｔ，ｖａｒ．ｎｕｍｂｅｒＯｆＥｄｇｅｓ）］、抽象構文木形式のコード差分が前述の例の通りであるならば、１回目のループではｓａｍｅは空となる。２回目のループでは（Ｒｅｃｔａｎｇｌｅ，ｖａｒ）∈ｍｏｄｅｌは、抽象構文木形式のコード差分の（Ｒｅｃｔａｎｇｌｅ，ａｐｉ'（）．ｇｅｔＲｅｃｔａｎｇｌｅ（））の型Ｒｅｃｔａｎｇｌｅに一致する。したがって、ｓａｍｅは［（Ｒｅｃｔａｎｇｌｅ，ｖａｒ）］となる。ｓａｍｅは空でないため、クライアントコードで書き換えが必要な箇所を（Ｒｅｃｔａｎｇｌｅ，ｖａｒ）とし、ループ処理から抜け（Ｒｅｃｔａｎｇｌｅ，ａｐｉ'（）．ｇｅｔＲｅｃｔａｎｇｌｅ（））をマイグレーション事例から構成可能な型とクライアントコードの事後計算で用いられる型の共通要素として出力する。

【００６２】

なお、上記の仕組みにより、マイグレーション事例では、

（１）ａｐｉ'（）という新しいメソッドの呼び出しに置き換え

（２）さらに、その呼び出し結果を使ってｇｅｔＲｅｃｔａｎｇｌｅ（）というメソッド呼び出しを追加する

という二つの編集がなされているが、このうの（１）及び（２）の両方を含めるべきか、又は（１）だけを含めるべきかという判断をクライアントコードの事後の計算（今回の例ではＲｅｃｔａｎｇｌｅの値が必要）にあわせて行うことができる。

【００６３】
その結果は、マイグレーション事例のコード編集からｇｅｔＲｅｃｔａｎｇｌｅ（）というメソッド呼び出しを追加する編集がクライアントコードに反映するべきコード編集に追加されたことに現れている。従来であれば、ａｐｉ（）がａｐｉ'（）に置き換えられ、Ｒｅｃｔａｎｇｌｅ型の値はａｐｉ'（）の戻り値のＳｈａｐｅ型の値に置き換えられてＲｅｃｔａｎｇｌｅ型の値は失われてしまうが、ｇｅｔＲｅｃｔａｎｇｌｅ（）というメソッド呼び出しを追加する編集が加わることで、Ｒｅｃｔａｎｇｌｅ型の値が再構成される。
【００６４】
［変数翻訳］
マイグレーション事例が使用している型を実際に構成するために変数へのアクセスが必要な場合はクライアントコードでも当該変数を宣言する必要がある。
【００６５】
但し、クライアントコードで既にその変数が宣言されているならばその限りではない。この時、マイグレーション事例でアクセスされている変数とクライアントコードで宣言されている変数の対応を取れば、すでに宣言されている変数を利用することができる。この対応は、非特許文献１のように変数と型のペアを格納した変数テーブルを利用した方法やＣｏｃｃｉｎｅｌｌｅ４Ｊ（「Kang, H. J., Thung, F., Lawall, J., Muller, G., Jiang, L., & Lo, D. (2019). Semantic patches for Java program transformation (experience report). In 33rd European Conference on Object-Oriented Programming (ECOOP 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik」）を利用することでとることができる。
【００６６】
すなわち、クライアントコードに反映させるべきマイグレーション事例で行われた編集の要素が定まった段階で、その編集はクライアントコードとは別のプログラムにおける編集である。そのため、別のプログラムのコード編集を反映させる際に、それぞれのプログラムでの変数間で片方には定義されていて片方では未定義といった不整合や同じ型の変数はどちらにも定義されているが変数名が異なっているということが発生する。事後計算比較で示した例のｇｅｔＲｅｃｔａｎｇｌｅが図形の色を引数にとることを仮定して説明する。マイグレーション事例が、ｃａｌｃ（ａｐｉ（））をｃａｌｃ（ａｐｉ'（）．ｇｅｔＲｅｃｔａｎｇｌｅ（ｃｏｌｏｒ））に修正することである場合は、図９に示すようなクライアントコードの書き換えが期待される。
【００６７】
ｃｏｌｏｒという変数がクライアントコードのどこかで定義されているとは限らないので、クライアントコード中にｃｏｌｏｒを新たに宣言する必要が生じることになる。したがって、このような場合に、変数翻訳部１３は、対象事例のマイグレーション後のプログラムに対して新たな宣言を追加する。
【００６８】
また、上記において「マイグレーション事例でアクセスされている変数とクライアントコードで宣言されている変数の対応を取る」とは、「同じ型の変数はどちらにも定義されているが変数名が異なっている」場合に行われる処理である。クライアントコードで定義された変数の集合と新たに定義が必要な変数（先の例ではｃｏｌｏｒ）とを比較し、或る基準で同じ変数と判断できるものをクライアントコードで定義された変数の集合から取り出す処理をいう。或る基準は、変数翻訳の方法によって異なるが、例えば、非特許文献１では、型が一致するものを同じ変数とみなす基準が採用されている。
【００６９】
上述したように、本実施の形態によれば、クライアントコードの事後計算との間で共通要素を有するマイグレーション事例に基づいて、クラアイントコードについてのマイグレーションが実行される。したがって、ＡＰＩのマイグレーションの成功率を高めることができる。
【００７０】

また、事後計算解析部１１が、クライアントコードの事後計算で用いられる型を特定し、事後計算比較部１２が、マイグレーション事例が使用している型からクライアントコードの事後計算で用いられる型の値を構成するため、ＡＰＩを呼び出した後の計算をするために必要なコードの編集をクライアントコードに反映することができる。

【００７１】

なお、本実施の形態において、事後計算解析部１１は、解析部の一例である。事後計算比較部１２は、特定部の一例である。変数翻訳部１３は、翻訳部の一例である。コード書換部１４は、書換部の一例である。

【００７２】

以上、本発明の実施の形態について詳述したが、本発明は斯かる特定の実施形態に限定されるものではなく、請求の範囲に記載された本発明の要旨の範囲内において、種々の変形・変更が可能である。

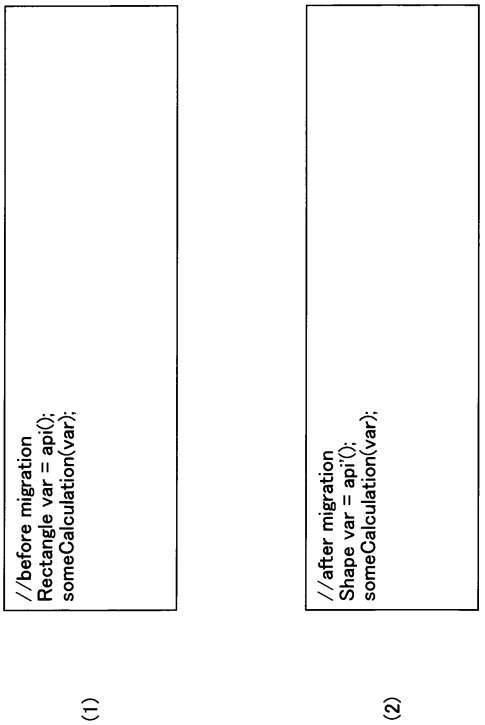
【符号の説明】

【００７３】

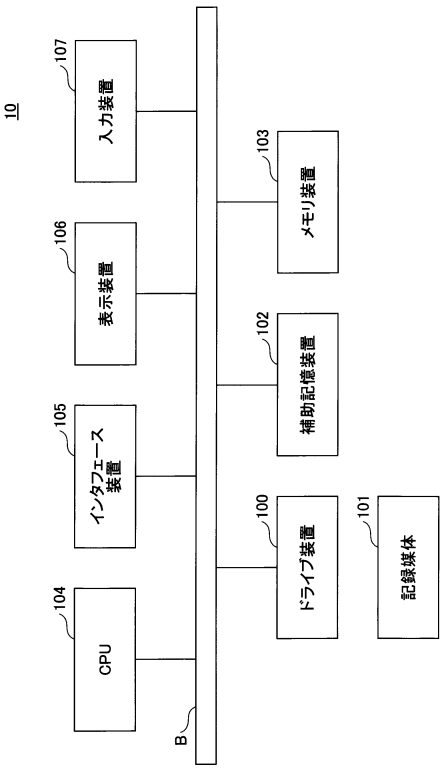
１０　プログラム修正装置
１１　事後計算解析部
１２　事後計算比較部
１３　変数翻訳部
１４　コード書換部
１００　ドライブ装置
１０１　記録媒体
１０２　補助記憶装置
１０３　メモリ装置
１０４　ＣＰＵ
１０５　インタフェース装置
Ｂ　バス

【図面】

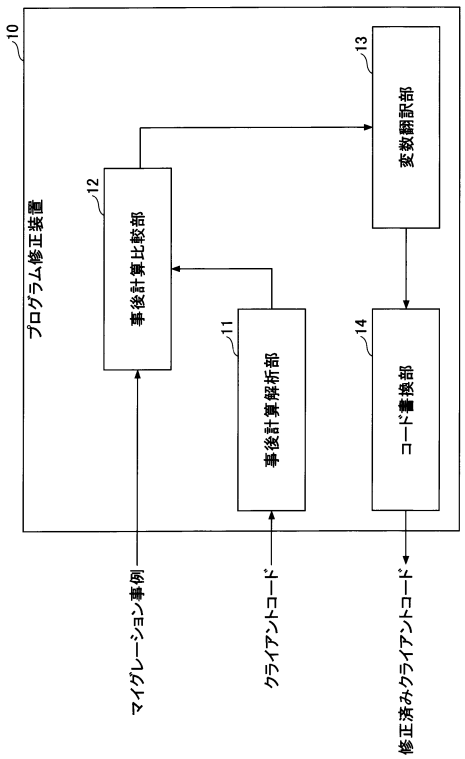
【図１】


//before migration
Rectangle var = api();
someCalculation(var);
(1)
//after migration
Shape var = api();
someCalculation(var);
(2)

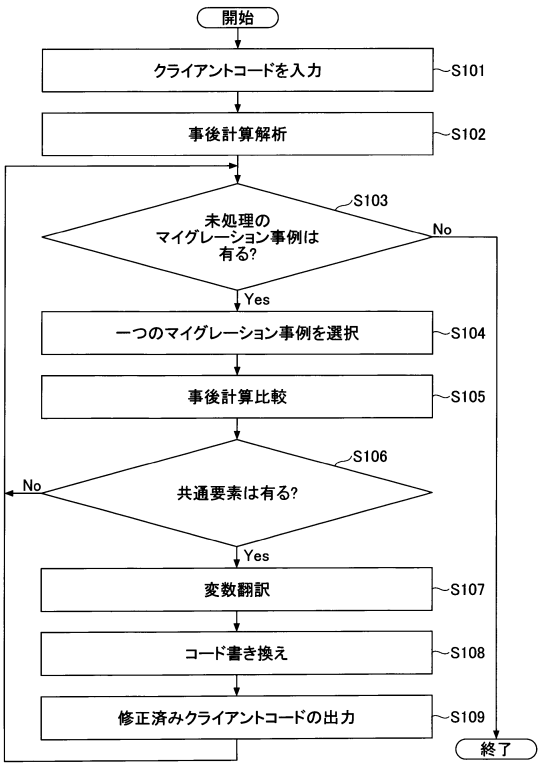
【図２】


10
107
入力装置
103
メモリ装置
106
表示装置
102
補助記憶装置
105
インタフェース装置
100
ドライブ装置
101
記録媒体
104
CPU
B

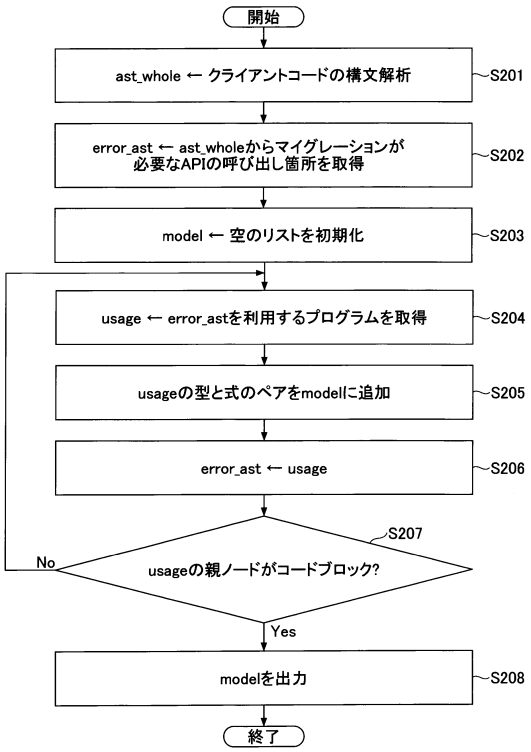
【図３】


10
プログラム修正装置
13
変数翻訳部
12
事後計算比較部
11
事後計算解析部
14
コード書換部
マイグレーション事例
クライアントコード
修正済みクライアントコード

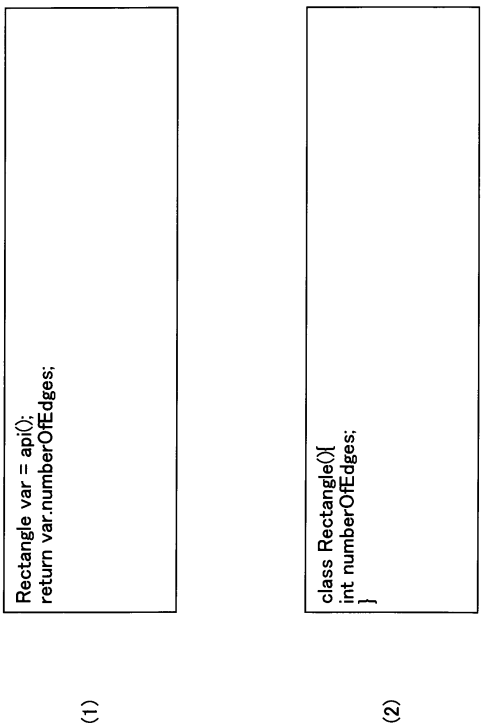
【図４】


開始
クライアントコードを入力
S101
事後計算解析
S102
未処理の
マイグレーション事例は
有る?
S103
No
Yes
一つのマイグレーション事例を選択
S104
事後計算比較
S105
共通要素は有る?
S106
No
Yes
変数翻訳
S107
コード書き換え
S108
修正済みクライアントコードの出力
S109
終了

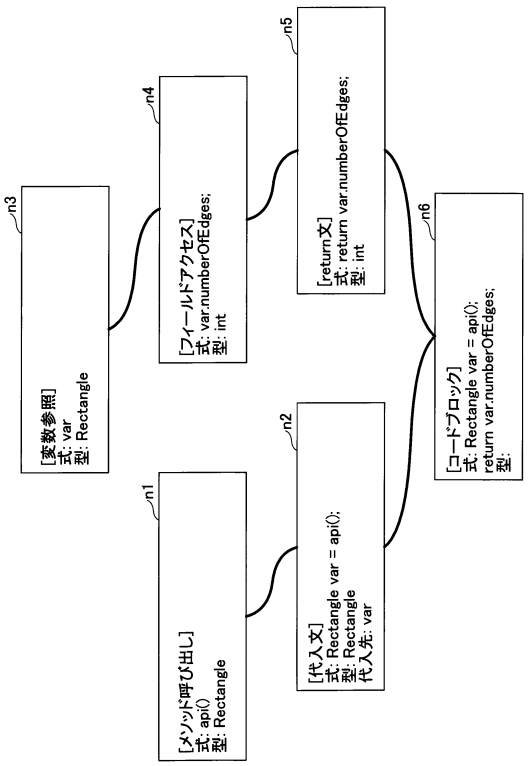
【図５】


開始
ast_whole ← クライアントコードの構文解析
S201
error_ast ← ast_wholeからマイグレーションが必要なAPIの呼び出し箇所を取得
S202
model ← 空のリストを初期化
S203
usage ← error_astを利用するプログラムを取得
S204
usageの型と式のペアをmodelに追加
S205
error_ast ← usage
S206
S207
usageの親ノードがコードブロック?
No
Yes
modelを出力
S208
終了

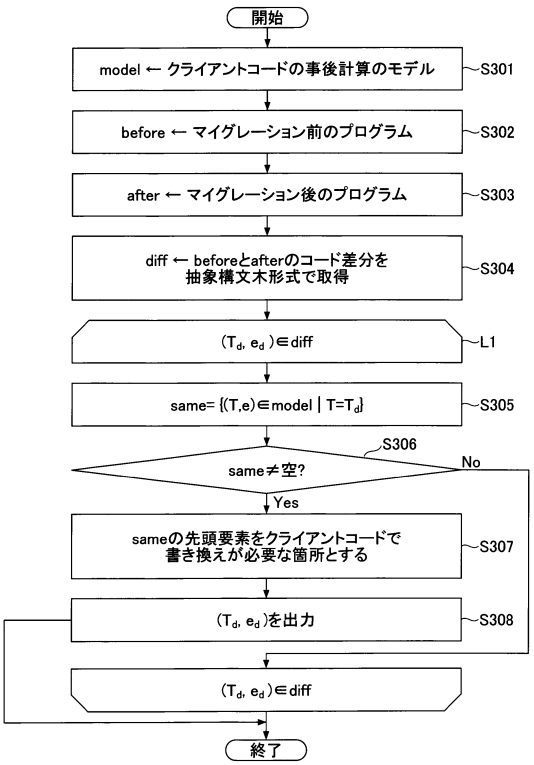
【図６】


Rectangle var = api();
return var.numberOfEdges;
(1)
class Rectangle(){
int numberOfEdges;
}
(2)

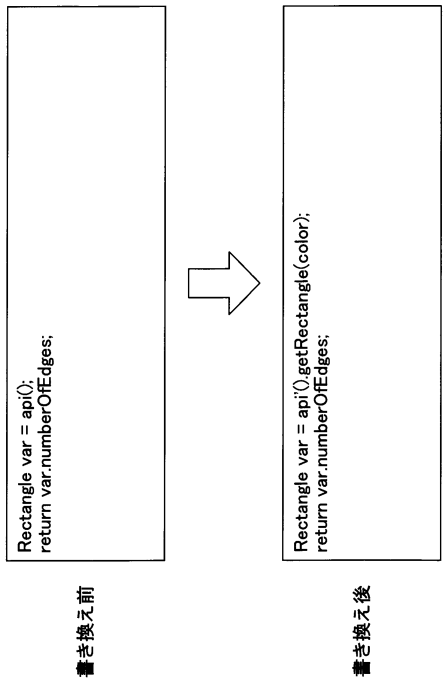
【図７】


n3
[変数参照]
式: var
型: Rectangle
n4
[フィールドアクセス]
式: var.numberOfEdges;
型: int
n5
[return文]
式: return var.numberOfEdges;
型: int
n6
[コードブロック]
式: Rectangle var = api();
return var.numberOfEdges;
型:
n1
[メソッド呼び出し]
式: api()
型: Rectangle
n2
[代入文]
式: Rectangle var = api();
型: Rectangle
代入先: var

【図８】


開始
model ← クライアントコードの事後計算のモデル
S301
before ← マイグレーション前のプログラム
S302
after ← マイグレーション後のプログラム
S303
diff ← beforeとafterのコード差分を抽象構文木形式で取得
S304
(Td, ed)∈diff
L1
same= {(T,e)∈model | T=Td}
S305
S306
same≠空?
No
Yes
sameの先頭要素をクライアントコードで書き換えが必要な箇所とする
S307
(Td, ed)を出力
S308
(Td, ed)∈diff
終了

【図９】


Rectangle var = api();
return var.numberOfEdges;
書き換え前
Rectangle var = api'().getRectangle(color);
return var.numberOfEdges;
書き換え後

フロントページの続き

(56)参考文献 特開平１０－１５４０７９（ＪＰ，Ａ）
米国特許出願公開第２０２０／０１９２６６２（ＵＳ，Ａ１）
FAZZINI, M et al.，"Automated API-usage update for Android apps"，Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis [online]，2019年，pp. 204-215，[retrieved on 2021.06.28], Retrieved from the Internet: 〈URL: https://dl.acm.org/doi/10.1145/3293882.3330571〉，〈DOI: 10.1145/3293882.3330571〉

(58)調査した分野 (Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　８／６５－８／６５８
Ｇ０６Ｆ　１１／３６