(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2013-164670
(P2013-164670A)
(43)公開日　平成25年8月22日(2013.8.22)

(51)Int.Cl.		F I			テーマコード（参考）
G06F 9/44	(2006.01)	G06F	9/06	620A	5B376

審査請求 未請求　請求項の数 8　ＯＬ　（全 21 頁）

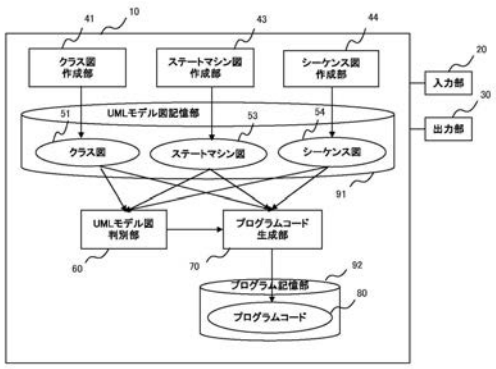
(21)出願番号　特願2012-26338(P2012-26338)
(22)出願日　平成24年2月9日(2012.2.9)

(71)出願人　000005234
富士電機株式会社
神奈川県川崎市川崎区田辺新田１番１号
(74)代理人　100150441
弁理士　松本　洋一
(72)発明者　宮崎　剛
神奈川県川崎市川崎区田辺新田１番１号
富士電機株式会社内
(72)発明者　杉野　一彦
東京都品川区大崎一丁目１１番２号　ゲートシティ大崎イーストタワー　富士電機リテイルシステムズ株式会社内

最終頁に続く

(54)【発明の名称】オブジェクト指向プログラム生成装置、その方法、プログラム

(57)【要約】
【課題】設計ドキュメントからオブジェクト指向のプログラムコードを自動で生成するオブジェクト指向プログラム生成装置、その方法、プログラムを提供する。
【解決手段】開発対象システムのUMLモデル図を作成する各UMLモデル図作成部と、各UMLモデル図の情報を記憶するUMLモデル図記憶部９１と、前記UMLモデル図記憶部９１から前記UMLモデル図の情報を読み出し、UMLモデル図間の整合性をチェックするUMLモデル図判別部６０と、整合性に問題が無い場合、オブジェクト指向プログラムを自動で生成するプログラムコード生成部７０とを備え、マルチタスク環境下で動作するプログラムを生成する。
【選択図】図１



【特許請求の範囲】
【請求項１】
オブジェクト指向プログラムを生成するオブジェクト指向プログラム生成装置であって、
開発対象システムの複数種類の設計ドキュメントを作成する設計ドキュメント作成部と、
前記設計ドキュメント作成部により作成された設計ドキュメントの情報を記憶する設計ドキュメント記憶部と、
前記設計ドキュメント記憶部から複数種類の前記設計ドキュメントの情報を各々読み出し、これら設計ドキュメント間の整合性をチェックする設計ドキュメント判別部と、
前記設計ドキュメント判別部で整合性に問題が無いと判別された場合、オブジェクト指向プログラムを生成するプログラムコード生成部と
を備えることを特徴とするオブジェクト指向プログラム生成装置。
【請求項２】
請求項１記載のオブジェクト指向プログラム生成装置において、
前記設計ドキュメントは、少なくともステートマシン図またはシーケンス図を含むUMLモデル図であることを特徴とするオブジェクト指向プログラム生成装置。
【請求項３】
請求項２記載のオブジェクト指向プログラム生成装置において、
前記UMLモデル図は、クラス図を含み、
前記設計ドキュメント判別部は、
ステートマシン図に記載されている全てのイベントがクラス図に記載されているクラスのイベントであるか否かと、ステートマシン図に記載されている全ての処理関数がクラス図に記載されているクラスの処理関数であるか否かの整合性、
シーケンス図に記載されている全てのイベントがクラス図に記載されているクラスのイベントであるか否かと、シーケンス図に記載されている全ての処理関数がクラス図に記載されているクラスの処理関数であるか否かの整合性、
または、シーケンス図に記載されている全ての状態がステートマシン図に記載されている状態であるか否かの整合性の少なくともいずれかの整合性のチェックをすることを特徴とするオブジェクト指向プログラム生成装置。
【請求項４】
請求項２または３記載のオブジェクト指向プログラム生成装置において、
前記設計ドキュメント作成部でクラス図を作成する際、クラス単位の機能に対応させてタスク名を記載し、プログラムコード生成部でプログラムコードを自動生成する際、マルチタスク（マルチスレッド）環境下で動作するプログラムコードを生成することを特徴とするオブジェクト指向プログラム生成装置。
【請求項５】
オブジェクト指向プログラムを生成するオブジェクト指向プログラム生成方法であって、
開発対象システムの複数種類の設計ドキュメントを作成する過程と、
作成された設計ドキュメントの情報を記憶する設計ドキュメント記憶部と、
前記設計ドキュメント記憶部から複数種類の前記設計ドキュメントの情報を各々読み出し、これら設計ドキュメント間の整合性をチェックする過程と、
前記整合性をチェックする過程で整合性に問題が無いと判別された場合、オブジェクト指向プログラムを生成する過程と
を具備することを特徴とするオブジェクト指向プログラム生成方法。
【請求項６】
請求項５記載のオブジェクト指向プログラム生成方法において、
前記UMLモデル図は、クラス図を含み、
前記設計ドキュメント間の整合性をチェックする過程は、

ステートマシン図に記載されている全てのイベントがクラス図に記載されているクラスのイベントであるか否かと、ステートマシン図に記載されている全ての処理関数がクラス図に記載されているクラスの処理関数であるか否かの整合性、
シーケンス図に記載されている全てのイベントがクラス図に記載されているクラスのイベントであるか否かと、シーケンス図に記載されている全ての処理関数がクラス図に記載されているクラスの処理関数であるか否かの整合性、
または、シーケンス図に記載されている全ての状態がステートマシン図に記載されている状態であるか否かの整合性の少なくともいずれかの整合性のチェックをすることを特徴とするオブジェクト指向プログラム生成方法。
【請求項７】
請求項５または６記載のオブジェクト指向プログラム生成方法において、
前記設計ドキュメント作成部でクラス図を作成する際、クラス単位の機能に対応させてタスク名を記載し、プログラムコード生成部でプログラムコードを自動生成する際、マルチタスク（マルチスレッド）環境下で動作するプログラムコードを生成することを特徴とするオブジェクト指向プログラム生成方法。
【請求項８】
コンピュータを、
請求項１ないし４の何れか１項に記載のオブジェクト指向プログラム生成装置が有する各手段（各部）として機能させるためのオブジェクト指向プログラム生成プログラム。

【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、設計ドキュメントからオブジェクト指向のプログラムコードを自動で生成するオブジェクト指向プログラム生成装置、その方法、プログラムに関する。
【背景技術】
【０００２】
オブジェクト指向をベースとするシステム開発において、分析・設計段階ではシステムの構造をソースコードよりも抽象化した形でモデリングする必要がある。モデリングとは、対象をある目的または観点から眺め、整理し表現することであり、このモデリングした結果、作成されたものをモデルと呼ぶ。
【０００３】
モデリングの表記方法を統一化することにより、分析の明確化や適切な設計、情報の共有化・再利用を行うことがきる。
統一モデリング言語としてUML(Unified Modeling Language)が知られている。UMLは、業務及びソフトウェアを記述するための図式記法として、オブジェクト指向の標準化団体である米国のOMG（オブジェクト・マネジメント・グループ）で制定された規格である。既に、デファクトスタンダードとして世界で受け入れられており、国際規格（ISO/IEC 19501）としても制定されている。
【０００４】
また、UMLはグラフィカルな記述で抽象化したシステムのモデルを生成する汎用モデリング言語であり、その実施において、複数種類の図が用いられている。UMLで用いられる図(以降、UMLモデル図と表記する)は、開発対象システムの分析や設計を行う際に作成される。また、UMLモデル図は、システムの静的な構造をモデルで表現する構造図とシステムの振る舞いをモデルで表現する振る舞い図に大別される。
【０００５】
構造図としては、クラス図、オブジェクト図などがある。振る舞い図としては、ステートマシン図(ステートチャート図や状態遷移図とも表現される)、シーケンス図、などがあ

る。
【０００６】
ところで、従来のオブジェクト指向プログラム開発においては、市販のUMLモデリングツールを用いて、UMLモデル図を作成し、作成したUMLモデル図からプログラムコードを自動で生成することで、ソフトウェアの生産性向上が図られてきた。
【０００７】
しかしながら、市販のUMLモデリングツールを用いて、自動生成できるUMLモデル図は構造図のみであり、振る舞い図からプログラムコードを自動生成することまではできていない。そのため、プログラム開発者は市販のUMLモデリングツールを用いて作成した振る舞い図を元に、手入力で振る舞い図に対応したプログラムコードを生成する必要があった。
【０００８】
また、特許文献１では、構造図と振る舞い図からプログラムコードを自動で生成するオブジェクト指向プログラムの自動生成装置を提供している。
【先行技術文献】
【特許文献】
【０００９】
【特許文献１】特開２００８－２９３１８６号公報
【発明の概要】
【発明が解決しようとする課題】
【００１０】
しかしながら、特許文献１のオブジェクト指向プログラムの自動生成装置では、構造図と振る舞い図からプログラムコードを自動生成するものの、UMLモデル図間の整合性チェックの方法については何ら具体的内容を開示していない。また、イベント毎に動作する機能をタスクとしてプログラムコードを生成することが記載されているが、イベント毎に動作する機能をタスクとして割り当てる方法について何ら具体的内容を開示していない。さらに、イベント毎に動作する機能をタスクとして割り当てた場合、プログラムとしての単位が細かすぎるため、複数の異なるタスク間での通信が頻発し処理におけるオーバーヘッドや、プログラム管理が複雑化する恐れがある。
【００１１】
上記の課題を解決するために、本発明では、UMLモデル図(クラス図、ステートマシン図、シーケンス図)間の整合性をチェックし、整合性に問題が無い場合にのみプログラムコードを自動で生成する装置、方法、プログラムを提供することを目的とする。
【００１２】
さらに、機能のタスク（スレッド）割り当てをクラス図で表現し、マルチタスク（マルチスレッド）環境下で動作するプログラムコードを自動で生成する装置、方法、プログラムを提供することを目的とする。
【課題を解決するための手段】
【００１３】
上記課題を解決するために、本発明に係るオブジェクト指向プログラム生成装置は、開発対象システムの複数種類の設計ドキュメントを作成する設計ドキュメント作成部と、前記設計ドキュメント作成部により作成された設計ドキュメントの情報を記憶する設計ドキュメント記憶部と、前記設計ドキュメント記憶部から複数種類の前記設計ドキュメントの情報を各々読み出し、これら設計ドキュメント間の整合性をチェックする設計ドキュメント判別部と、前記設計ドキュメント判別部で整合性に問題が無いと判別された場合、オブジェクト指向プログラムを生成するプログラムコード生成部とを備えることを特徴とする。
【００１４】
また、本発明に係るオブジェクト指向プログラム生成装置は、前記設計ドキュメントが、少なくともステートマシン図またはシーケンス図を含むUMLモデル図であることを特徴とする。

【００１５】
また、本発明に係るオブジェクト指向プログラム生成装置は、前記UMLモデル図は、クラス図を含み、前記設計ドキュメント判別部が、ステートマシン図に記載されている全てのイベントがクラス図に記載されているクラスのイベントであるか否かと、ステートマシン図に記載されている全ての処理関数がクラス図に記載されているクラスの処理関数であるか否かの整合性、シーケンス図に記載されている全てのイベントがクラス図に記載されているクラスのイベントであるか否かと、シーケンス図に記載されている全ての処理関数がクラス図に記載されているクラスの処理関数であるか否かの整合性、または、シーケンス図に記載されている全ての状態がステートマシン図に記載されている状態であるか否かの整合性の少なくともいずれかの整合性のチェックをすることを特徴とする。
【００１６】
尚、ここでは、クラス図作成部、ステートマシン図作成部、及び、シーケンス図作成部は別々の手段（部）としているが、クラス図、ステートマシン図、及び、シーケンス図の全てを同一の環境で作成できる手段であってもよい。
【００１７】
また、ここでは、装置として表現されているが、方法、プログラムにより実現されるとしてもよい。
【発明の効果】
【００１８】
本発明によれば、オブジェクト指向プログラム開発において、開発者は設計ドキュメントであるUMLモデル図からプログラムが自動で生成されるので、プログラム作成作業が大幅に軽減され、また、本発明により、プログラムコードを自動で生成する前に、UMLモデル図間の整合性をチェックすることにより、ソフトウェアテスト時の不具合を未然に防ぐことができ、結果としてプログラムの作成作業が大幅に軽減され、オブジェクト指向プログラム開発のトータル作業工数を大幅に削減することができる。
【００１９】
また、本発明によれば、プログラムコードを自動で生成する際に、クラス図を用いて機能のタスク割り当てを表現することにより、マルチタスク環境下で動作するプログラムコードも自動で生成されるため、プログラム作成作業が大幅に軽減され、オブジェクト指向プログラム開発のトータル作業工数を大幅に削減することができる。
【図面の簡単な説明】
【００２０】
【図１】本発明におけるオブジェクト指向プログラム生成装置の一例を示す図である。
【図２】クラス図（Task1）の一例を示す図である。
【図３】クラス図（Task2）の一例を示す図である。
【図４】ステートマシン図(クラス名が”ClassA”であるクラス)の一例を示す図である。
【図５】ステートマシン図(クラス名が”ClassB”であるクラス)の一例を示す図である。
【図６】ステートマシン図(クラス名が”ClassC”であるクラス)の一例を示す図である。
【図７】シーケンス図の一例を示す図である。
【図８】UMLモデル図判別部にて、ＵＭＬモデル図の整合性に問題がないかどうかを判別する際の処理を示すフローチャート図である。
【図９】異常なステートマシン図(クラス名が”ClassA”であるクラス）の一例（異常パターン１）を示す図である。
【図１０】異常なステートマシン図(クラス名が”ClassA”であるクラス)の一例（異常パターン２）を示す図である。
【図１１】異常なシーケンス図の一例（異常パターン１）を示す図である。
【図１２】異常なシーケンス図の一例（異常パターン２）を示す図である。
【図１３】異常なシーケンス図の一例（異常パターン３）を示す図である。
【図１４】異常なシーケンス図の一例（異常パターン４）を示す図である。
【図１５】異常なシーケンス図の一例（異常パターン５）を示す図である。

【図１６】本発明におけるオブジェクト指向プログラム生成装置によってクラス図とステートマシン図、及びシーケンス図から作成されるプログラムコードの一例を示す図である。
【図１７】本発明におけるオブジェクト指向プログラム生成装置によってクラス図から作成されるプログラムコードの一例を示す図である。
【発明を実施するための形態】
【００２１】
以下、本発明の実施の形態について、詳細に説明する。
＜オブジェクト指向プログラム生成装置の構成概要＞
図１は、本発明におけるオブジェクト指向プログラム生成装置の一例を示す図である。
【００２２】
図１に示すように、オブジェクト指向プログラム生成装置は、エディタなどの入力装置によって開発対象システムのクラス構造を表現するクラス図を作成するクラス図作成部４１、エディタなどの入力装置によって開発対象システムの各クラスにおける状態の変化を表現するステートマシン図を作成するステートマシン図作成部４３、エディタなどの入力装置によって開発対象システムのクラス間のメッセージの流れを表現するシーケンス図を作成するシーケンス図作成部４４、上記で作成したUMLモデル図の情報(クラス図５１、ステートマシン図５３、シーケンス図５４)を記憶するUMLモデル図記憶部９１、UMLモデル図記憶部９１から、UMLモデル図の情報を読み出しUMLモデル図間の整合性をチェックし問題がないかどうかを判別するUMLモデル図判別部６０、及び、UMLモデル図間の整合性に問題がない場合に上記により作成されたUMLモデル図からプログラムコードを自動で生成するプログラムコード生成部７０、生成されたプログラムコード８０を記憶するプログラムコード記憶部９２とで構成されている。
【００２３】
また、図示はしていないが、オブジェクト指向プログラム生成装置の機能を実現するためにオブジェクト指向プログラム生成装置本体１０は、一般的なコンピュータのハードウェア資源として、例えば、ＣＰＵ、記憶装置、入出力装置、各種インターフェースなど、を周知の構成として備えており、また当然ながら、上記のごとき機能を実現させるためのプログラムを上記記憶装置内に格納している。UMLモデル図記憶部９１、プログラム記憶部９２は上述の記憶装置により実現され、また、上述の各種作成部（４１～４４）、UMLモデル図判別部６０、プログラムコード生成部７０内の各構成は、上述のＣＰＵ、記憶装置、各種インターフェース等のハードウェア資源と後述する処理フローを実行する各種プログラムにより実現される。各種プログラムは、例えば汎用のパーソナルコンピュータ、サーバ等の記憶装置内にその実行プログラムをインストールすることにより、本発明における各部の処理等を実現することができる。
【００２４】
さらに、オブジェクト指向プログラム生成装置は、キーボード、マウス、タッチパネル等のインターフェースを用いて実現される入力部２０と、液晶または有機EL等からなる表示パネルや音声出力用のスピーカー等を有し、各種情報を出力する出力部３０を備える。
【００２５】
また、入力部２０及び出力部３０は、オブジェクト指向プログラム生成装置本体１０のクライアント端末（図示せず）にその機能があってもよい。このときは、ネットワークを介してオブジェクト指向プログラム生成装置本体１０とクライアント端末が通信可能に接続されている。なお、ネットワークは、有線、無線を問わず既存の公衆網、ＬＡＮ、ＷＡＮなどを用いることができる。
＜クラス図＞
図２、及び、図３は、本発明におけるクラス図の一例を示す図である。また、図２は、クラス図名が”Task1”であるクラス図であり、図３は、クラス図名が”Task2”であるクラス図である。クラス図は、クラス図作成部４１を用いて作成される。
【００２６】

図２、及び、図３に示すように、各クラスは、クラス名、属性、及び、操作の区画から構成される。また、操作は、ステレオタイプなどを使用し、処理関数とイベントに分類される。

【００２７】

図２に示す例では、クラス名が”ClassA”であるクラスは、属性として、int型である変数”a”、及び、char型である変数”b”を持つ。また、処理関数として、返り値の型が”void”である処理関数”Ma1”、及び、”Ma2”を持つ。また、イベントとして、返り値の型が”void”であるイベント”Ea1”、”Ea2”、及び、”Ea3”を持つ。

【００２８】

さらに、クラス名が”ClassB”であるクラスは、属性を持っていない。また、処理関数として、返り値の型が”void”である処理関数”Mb1”を持つ。また、イベントとして、返り値の型が”void”であるイベント”Eb1”、及び、”Eb2”を持つ。

【００２９】

図３に示す例では、クラス名が”ClassC”であるクラスは、属性を持っていない。また、処理関数として、返り値の型が”void”である処理関数”Mc1”、”Mc3”、及び、”Mc4”を持つ。また、イベントとして、返り値の型が”void”であるイベント”Ec1”、”Ec2”、”Ec3”、及び、”Ec4”を持つ。

【００３０】

また、クラス図名は、タスク名を表現している。図２より、タスク名が”Task1”であるタスクには、クラス名が”ClassA”であるクラス、及び、クラス名が”ClassB”であるクラスが割り当てられている。図３より、タスク名が”Task2”であるタスクには、クラス名が”ClassC”であるクラスが割り当てられている。

【００３１】

タスク名は、クラス図名として表現する代わりに、クラス図に付帯するノート欄などにタスク名を記載しておいても良い。

＜ステートマシン図＞

図４、図５、及び、図６は、本発明におけるステートマシン図の一例を示す図である。また、図４はクラス名が”ClassA”であるクラスのステートマシン図であり、図５はクラス名が”ClassB”であるクラスのステートマシン図であり、図６はクラス名が”ClassC”であるクラスのステートマシン図である。ステートマシン図は、ステートマシン図作成部４３を用いて作成される。

【００３２】

図４、図５、及び、図６に示すように、各ステートマシン図は、開始(図中に●で表示)、状態、及び、遷移を持つ。

図４に示す例では、状態として、”StateA1”、”StateA2”、及び、”StateA3”を持ち、初期状態(開始からの遷移で表現される)は”StateA1”である。また、遷移として、3つの遷移を持つ。

【００３３】

1つ目の遷移では、”StateA1”の状態でイベント”Ea1”を受け取ると、処理関数”Ma1”を実行し、”StateA2”へ状態を遷移する。

２つ目の遷移では、”StateA2”の状態でイベント”Ea2”を受け取ると、処理関数”Ma2”を実行し、”StateA3”へ状態を遷移する。

【００３４】

３つ目の遷移では、”StateA3”の状態でイベント”Ea3”を受け取ると、何も処理を実行することなく、”StateA1”へ状態を遷移する。

図５に示す例では、状態として、”StateB1”、及び、”StateB2”を持ち、初期状態は”StateB1”である。また、遷移として、2つの遷移を持つ。

【００３５】

1つ目の遷移では、”StateB1”の状態でイベント”Eb1”を受け取ると、処理関数”Mb1”を実行し、”StateB2”へ状態を遷移する。

２つ目の遷移では、”StateB2”の状態でイベント”Eb2”を受け取ると、処理関数”Mb1”を実行し、”StateB1”へ状態を遷移する。

【００３６】

図６に示す例では、状態として、”StateC1”、”StateC2”、”StateC3”、及び、”StateC4”を持ち、初期状態は”StateC1”である。また、遷移として、6つの遷移を持つ。

1つ目の遷移では、”StateC1”の状態でイベント”Ec1”を受け取ると、処理関数”Mc1”を実行し、”StateC2”へ状態を遷移する。

【００３７】

２つ目の遷移では、”StateC1”の状態でイベント”Ec3”を受け取ると、処理関数”Mc3”を実行し、”StateC4”へ状態を遷移する。

３つ目の遷移では、”StateC2”の状態でイベント”Ec2”を受け取ると、何も処理を実行することなく、”StateC3”へ状態を遷移する。

【００３８】

４つ目の遷移では、”StateC3”の状態でイベント”Ec3”を受け取ると、何も処理を実行することなく、”StateC4”へ状態を遷移する。

５つ目の遷移では、”StateC4”の状態でイベント”Ec1”を受け取ると、処理関数”Mc1”を実行し、”StateC2”へ状態を遷移する。

【００３９】

６つ目の遷移では、”StateC4”の状態でイベント”Ec4”を受け取ると、処理関数”Mc4”を実行し、状態は遷移しない。

＜シーケンス図＞

図７は、本発明におけるシーケンス図の一例を示す図である。シーケンス図は、シーケンス図作成部４４を用いて作成される。

【００４０】

図７に示すように、シーケンス図は、メッセージ終了点(図中に●で表示)、ライフライン、及び、メッセージを持つ。また、各シーケンスは、メッセージ終了点からあるライフライン(以降、ライフライン1と表記する)に対してメッセージを送信し、そのメッセージを送信されたライフライン1からあるライフライン(以降、ライフライン2と表記する)に対してメッセージを送信するように表記される。

【００４１】

図７に示す例では、１つ目のシーケンスでは、クラス名が”ClassA”であるクラスが”StateA1”の状態でイベント”Ea1”を受け取ると、クラス名が”ClassB”であるクラスに対してイベント”Eb1”を送信し、クラス名が”ClassC”であるクラスに対してイベント”Ec1”を送信する。

【００４２】

２つ目のシーケンスでは、クラス名が”ClassA”であるクラスが”StateA2”の状態でイベント”Ea2”を受け取ると、クラス名が”ClassB”であるクラスに対してイベント”Eb2”を送信する。

【００４３】

３つ目のシーケンスでは、クラス名が”ClassB”であるクラスが”StateB1”の状態でイベント”Eb2”を受け取ると、クラス名が”ClassA”であるクラスに対してイベント”Ea3”を送信する。

【００４４】

４つ目のシーケンスでは、クラス名が”ClassC”であるクラスが処理関数”Mc1”を実行すると、クラス名が”ClassA”であるクラスに対してイベント”Ea2”を送信し、クラス名が”ClassB”であるクラスに対してイベント”Eb1”を送信する。

＜UMLモデル図判別＞

図８は、UMLモデル図判別部６０にて、UMLモデル図間の整合性に問題がないかどうかを判別する際の処理を示すフローチャート図である。

【００４５】

UMLモデル図判別処理では、ステートマシン図とクラス図の関係に問題がないかどうかをチェックする(S701)。ステートマシン図とクラス図の関係に問題がある場合、異常で終了する。S701では、ステートマシン図に記載されている全てのイベントがクラス図に記載されているクラスのイベントであるかどうかをチェックする。また、S701では、ステートマシン図に記載されている全ての処理関数がクラス図に記載されているクラスの処理関数であるかどうかをチェックする。

【００４６】

ここでS701の詳細について図２、図３、図４、図５、図６、図９、図１０を例に説明する。

例えば、図４、図５、及び、図６に記載されている全てのイベントは図２、及び、図３に記載されているクラスのイベントであり、図４、図５、及び、図６に記載されている全ての処理関数は図２、及び、図３に記載されているクラスの処理関数であるため、図４、図５、及び、図６に示すステートマシン図と図２、及び、図３に示すクラス図の関係には問題がないと判別し、処理を続行する。

【００４７】

また、図９に記載されているステートマシン図は、クラス名が”ClassA”であるクラスのステートマシン図であるが、図９において、状態”StateA1”から状態”StateA2”への遷移で使用されているイベント”Ex1”は図２に記載されているクラス名が”ClassA”であるクラスのイベントではないため、図９に示すステートマシン図と図２に示すクラス図の関係には問題があると判別し、異常で終了する。

【００４８】

また、図１０に記載されているステートマシン図は、クラス名が”ClassAであるクラスのステートマシン図であるが、図１０において、状態”StateA2”から状態”StateA3”への遷移で使用されている処理関数”Mx1”は図２に記載されているクラス名が”ClassA”であるクラスの処理関数ではないため、図１０に示すステートマシン図と図２に示すクラス図の関係には問題があると判別し、異常で終了する。

【００４９】

図８に戻って、続いて、シーケンス図とクラス図の関係に問題がないかどうかをチェックする(S702)。シーケンス図とクラス図の関係に問題がある場合、異常で終了する。S702では、シーケンス図に記載されている全てのクラスがクラス図に記載されているクラスであるかどうかをチェックする。また、S702では、シーケンス図に記載されている全てのイベントがクラス図に記載されているクラスのイベントであるかどうかをチェックする。また、S702では、シーケンス図に記載されている全ての処理関数がクラス図に記載されているクラスの処理関数であるかどうかをチェックする。

【００５０】

ここでS702の詳細について図２、図３、図７、図１１、図１２、図１３、図１４を例に説明する。

例えば、図７に記載されている全てのクラスは図２、及び、図３に記載されているクラスであり、図７に記載されている全てのイベントは図２、及び、図３に記載されているクラスのイベントであり、図７に記載されている全ての処理関数は図２、及び、図３に記載されているクラスの処理関数であるため、図７に示すシーケンス図と図２、及び、図３に示すクラス図の関係には問題がないと判別し、処理を続行する。

【００５１】

また、図１１に記載されているシーケンス図において、クラス名が”ClassX”であるクラスは図２、及び、図３に記載されていないため、図１１に示すシーケンス図と図２、及び、図３に示すクラス図の関係には問題があると判別し、異常で終了する。

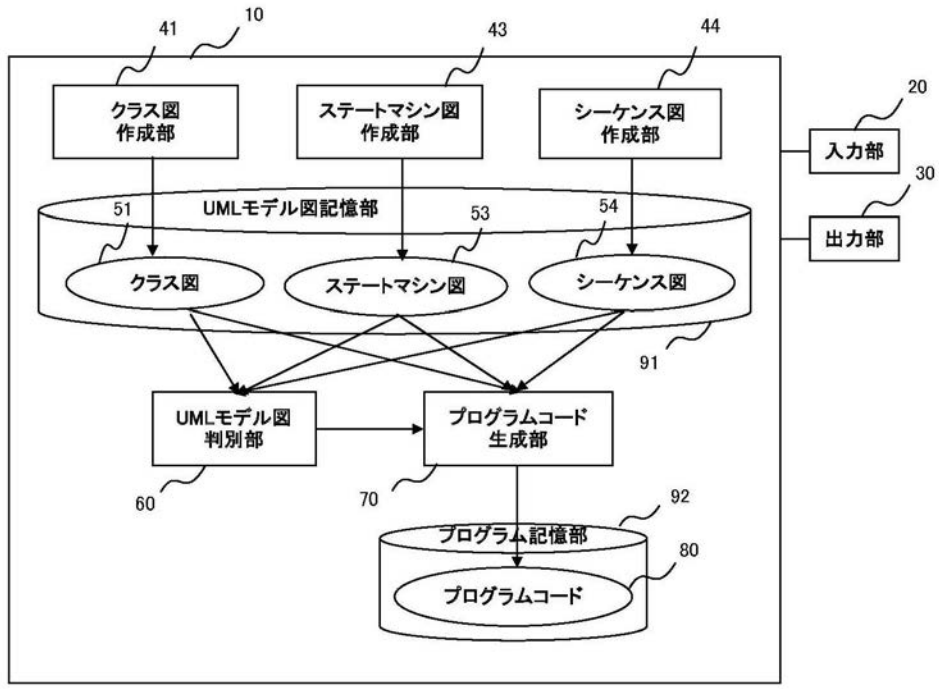
【００５２】

また、図１２に記載されているシーケンス図において、クラス名が”ClassA”であるクラスが”StateA1”の状態で受け取るイベント”Ex2”は図２に記載されているクラス名が”ClassA”であるクラスのイベントではないため、図１２に示すシーケンス図と図２に示

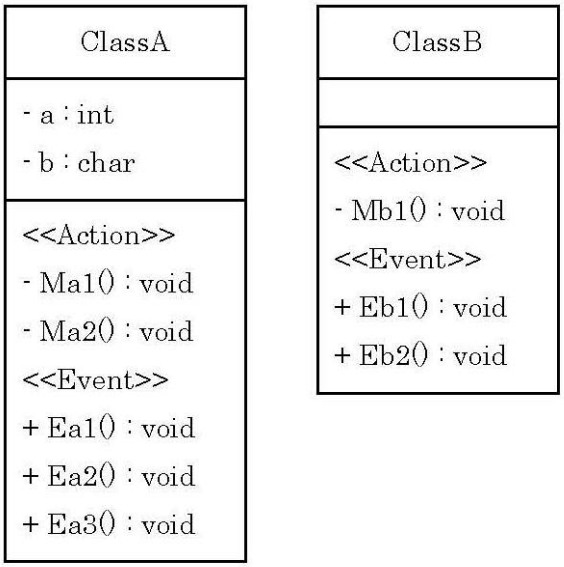
すクラス図の関係には問題があると判別し、異常で終了する。
【００５３】
また、図１３に記載されているシーケンス図において、クラス名が”ClassA”であるクラスがクラス名が”ClassB”であるクラスに対して送信するイベント”Ex3”は図２に記載されているクラス名が”ClassB”であるクラスのイベントではないため、図１３に示すシーケンス図と図２に示すクラス図の関係には問題があると判別し、異常で終了する。
【００５４】
また、図１４に記載されているシーケンス図において、クラス名が”ClassC”であるクラスが処理を実行する処理関数”Mx2”は図３に記載されているクラス名が”ClassC”であるクラスの処理関数ではないため、図１４に示すシーケンス図と図３に示すクラス図の関係には問題があると判別し、異常で終了する。
【００５５】
図８に戻って、続いて、シーケンス図とステートマシン図の関係に問題がないかどうかをチェックする(S703)。シーケンス図とステートマシン図の関係に問題がある場合、異常で終了する。S703では、シーケンス図に記載されている全ての状態がステートマシン図に記載されている状態であるかどうかをチェックする。
【００５６】
ここでS703の詳細について図４、図５、図６、図７、図１５を例に説明する。
例えば、図７に記載されている全ての状態は図４、図５、及び、図６に記載されている状態であるため、図７に示すシーケンス図と図４、図５、及び、図６に示すステートマシン図の関係には問題がないと判別し、正常で終了する。
【００５７】
また、図１５に記載されているシーケンス図において、クラス名が”ClassA”であるクラスがイベント”Ea1”を受け取る状態”StateX1”は図４に記載されているクラス名が”ClassA”であるクラスの状態ではないため、図１５に示すシーケンス図と図４に示すステートマシン図の関係には問題があると判別し、異常で終了する。
【００５８】
以上により、UMLモデル図間の関係性をチェックし問題がないかどうかを判別する。
なお、ここでは、UMLモデル図間の関係性のチェックの順番をステートマシン図とクラス図、シーケンス図とクラス図、シーケンス図とステートマシン図のようにしているが、順番は任意であってもよい。
＜プログラムコード生成＞。
【００５９】
図１６は、本発明におけるクラス図、ステートマシン図、及び、シーケンス図から作成されるプログラムコードの一実施例を示す図である。図１６に示すプログラムコードは、プログラムコード生成部７０を用いて自動で作成される。
【００６０】
図１６に示すプログラムコードは、クラス名が”ClassA”であるクラスのプログラムコードである。図１６において、２行目、８行目～１０行目、１６行目～１７行目、及び、２７行目～３２行目は図２に示すクラス図から作成され、４行目～６行目、１４行目、３７行目～４０行目、４７行目～５０行目、及び、５６行目は図４に示すステートマシン図から作成され、４１行目～４４行目、及び、５１行目～５３行目は図７に示すシーケンス図から作成される。
【００６１】
同様に、クラス名が”ClassB”であるクラス、及び、クラス名が”ClassC”であるクラスのプログラムコードもプログラムコード生成部７０を用いて自動で作成される。図１６の例では、クラス”ClassA”にタスク”Task1”が割り当てられている。
【００６２】
図１７は、本発明におけるクラス図から作成されるプログラムコードの一実施例を示す図である。プログラムコードは、プログラムコード生成部７０を用いて自動で作成される

。
【００６３】
図１７に示すプログラムコードは、タスク、及び、オブジェクトの生成のプログラムコードである。図１７において、４行目、７行目～８行目、１３行目、１６行目～１７行目、２０行目、及び、２３行目～３０行目は図２に示すクラス図から作成され、５行目、９行目、１４行目、１８行目、２１行目、及び、３１行目～３４行目は図３に示すクラス図から作成される。
【００６４】
図１７の例では、メインプログラムにタスク”Task1”と”Task2”が定義され、タスクに起動がかけられると該当するオブジェクトが呼び出され処理を実行する。これにより、クラスの機能を最小の単位として、一つのタスクと対応させることで、他のタスクとの排他制御やレベル付け（優先付け）を容易にしている。
【００６５】
ここでは、”ClassA”と”ClassB”は、同じタスク”Task1”であるため、”ClassA”と”ClassB”のプログラムをマルチ処理（並行処理）することはできない。一方、”ClassA”と”ClassC”のプログラムはそれぞれ別なタスクに割り付けられているため、マルチ処理が可能となる。
【００６６】
なお、ここでは、プログラムコードとしてJava(登録商標)言語を用いて表記しているが、その他の言語(C言語、C++言語など)を用いて表記してもよい。
＜実施例の効果＞
以上のように、本発明を適用することにより、オブジェクト指向プログラム開発において、クラス図、ステートマシン図、及び、シーケンス図を作成し、オブジェクト指向プログラム生成装置を使用することで、プログラムコードが自動で生成されるため、プログラム作成作業が大幅に軽減され、トータル作業工数を大幅に削減することが可能となった。
【００６７】
また、本発明を適用することにより、UMLモデル図間の整合性をチェックすることで、ソフトウェアテスト時の不具合を未然に防ぐことができ、テスト作業を軽減することが可能となった。
【００６８】
さらに、本発明を適用することにより、クラス図を用いて機能のタスク割り当てを表現することで、マルチタスク環境下で動作するプログラムコードも自動で生成されるため、プログラム作成作業が大幅に軽減され、オブジェクト指向プログラム開発のトータル作業工数を大幅に削減することが可能となった。
【符号の説明】
【００６９】
１０　オブジェクト指向プログラム生成装置本体
２０　入力部
３０　出力部
４１　クラス図作成部
４３　ステートマシン図作成部
４４　シーケンス図作成部
５１　クラス図データ
５３　ステートマシン図データ
５４　シーケンス図データ
６０　UMLモデル図判別部
７０　プログラムコード生成部
８０　プログラムコード
９１　UMLモデル図記憶部
９２　プログラム記憶部

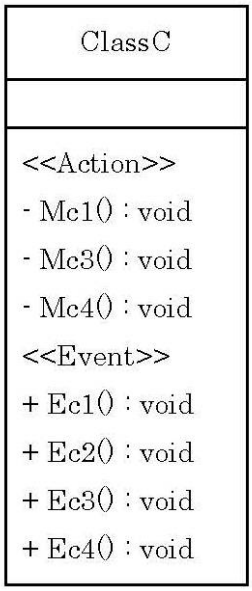
【図１】


10
41
43
44
クラス図
作成部
ステートマシン図
作成部
シーケンス図
作成部
20
入力部
30
出力部
51
UMLモデル図記憶部
53
54
クラス図
ステートマシン図
シーケンス図
91
UMLモデル図
判別部
プログラムコード
生成部
60
70
92
プログラム記憶部
80
プログラムコード

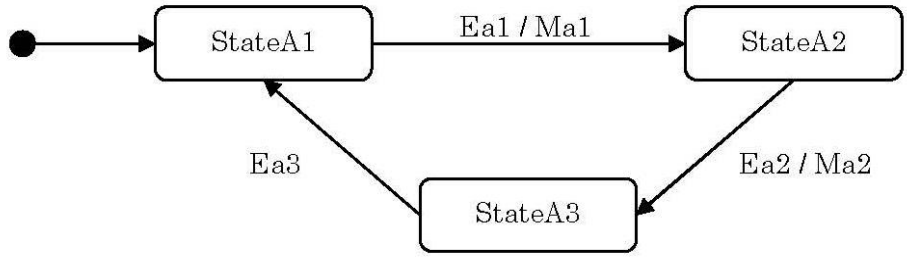
【図２】


ClassA
- a : int
- b : char
<<Action>>
- Ma1() : void
- Ma2() : void
<<Event>>
+ Ea1() : void
+ Ea2() : void
+ Ea3() : void
ClassB
<<Action>>
- Mb1() : void
<<Event>>
+ Eb1() : void
+ Eb2() : void

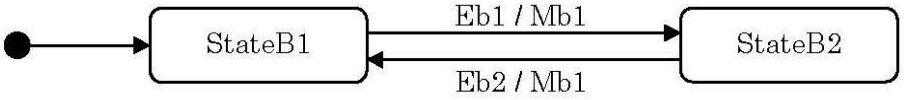
【図３】


ClassC
<<Action>>
- Mc1() : void
- Mc3() : void
- Mc4() : void
<<Event>>
+ Ec1() : void
+ Ec2() : void
+ Ec3() : void
+ Ec4() : void

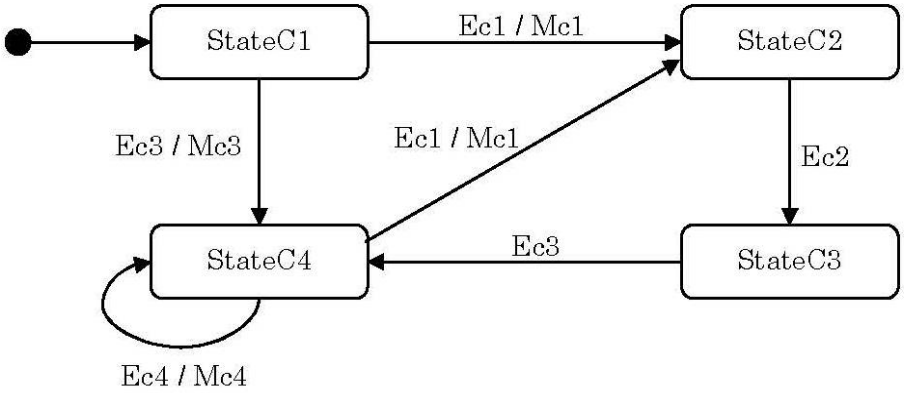
【図４】


StateA1
Ea1 / Ma1
StateA2
Ea3
Ea2 / Ma2
StateA3

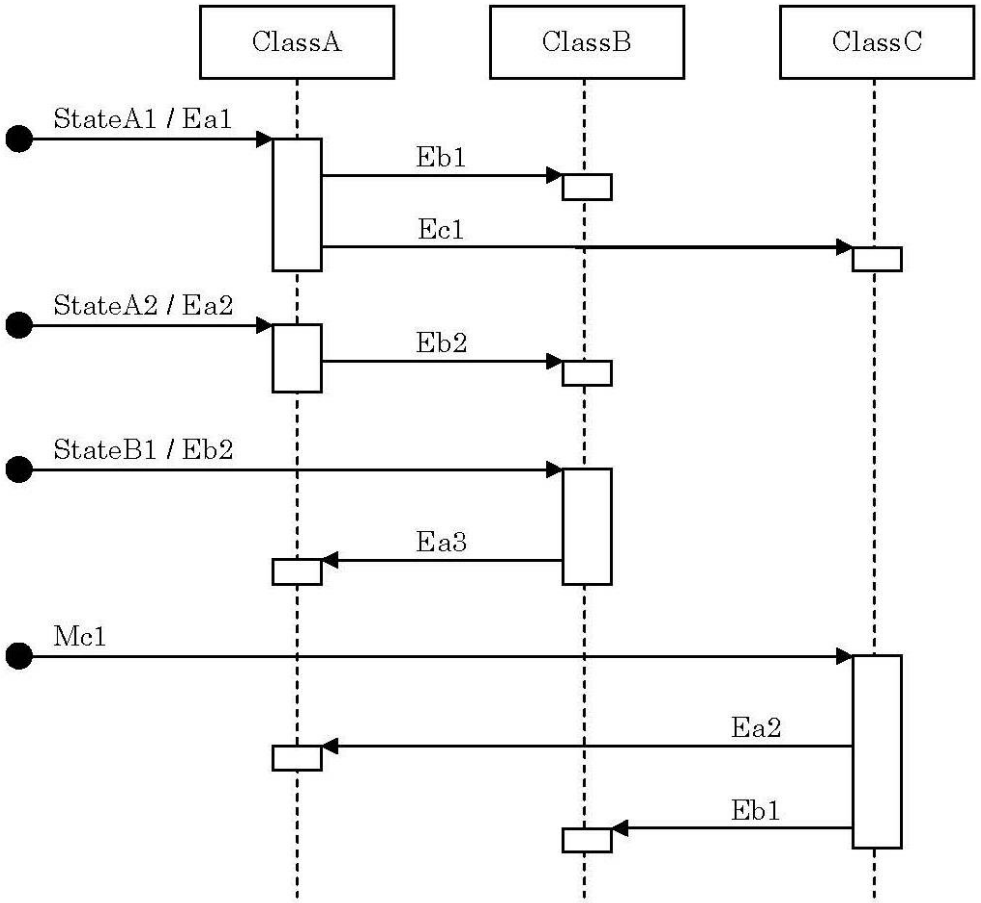
【図５】


StateB1
Eb1 / Mb1
Eb2 / Mb1
StateB2

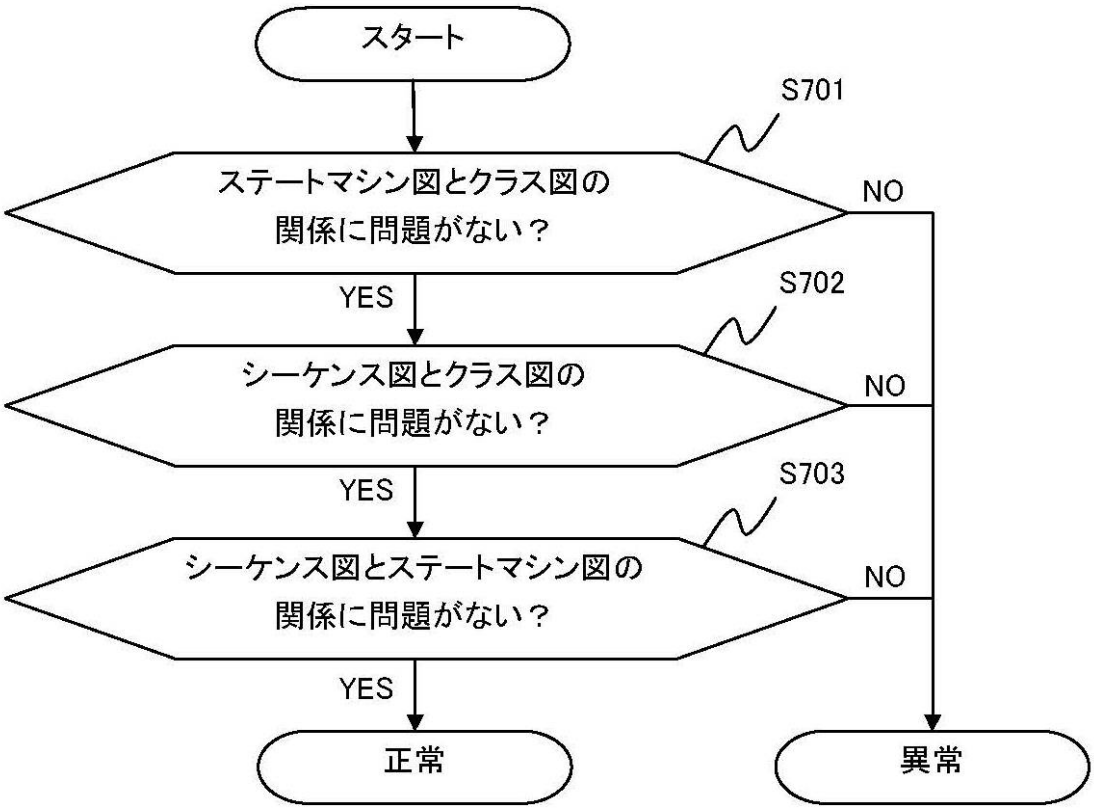
【図６】


StateC1
Ec1 / Mc1
StateC2
Ec3 / Mc3
Ec1 / Mc1
Ec2
StateC4
Ec3
StateC3
Ec4 / Mc4

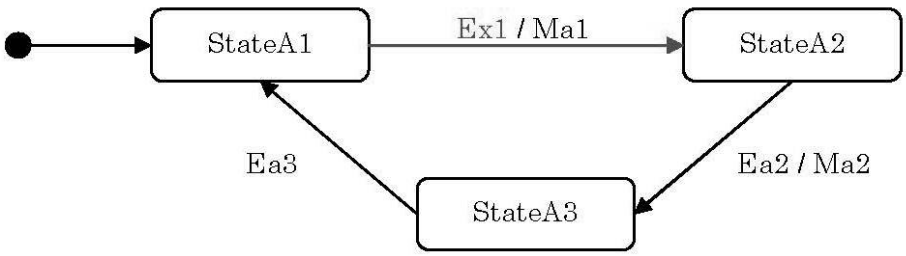
【図７】


ClassA
ClassB
ClassC
StateA1 / Ea1
Eb1
Ec1
StateA2 / Ea2
Eb2
StateB1 / Eb2
Ea3
Mc1
Ea2
Eb1

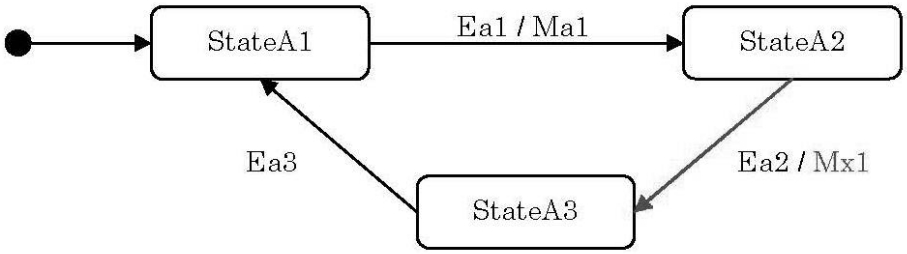
【図８】


スタート
S701
ステートマシン図とクラス図の
関係に問題がない？
NO
YES
S702
シーケンス図とクラス図の
関係に問題がない？
NO
YES
S703
シーケンス図とステートマシン図の
関係に問題がない？
NO
YES
正常
異常

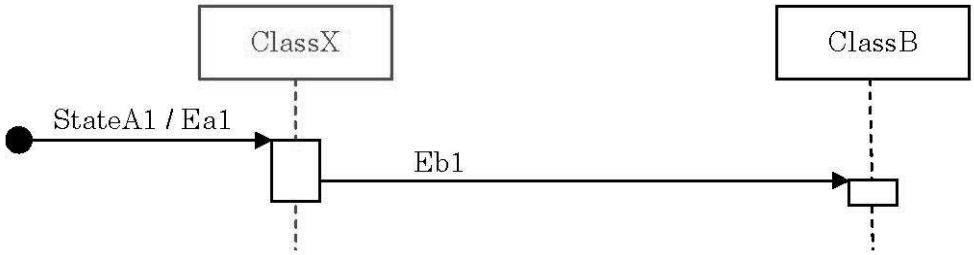
【図９】


StateA1
Ex1 / Ma1
StateA2
Ea3
Ea2 / Ma2
StateA3

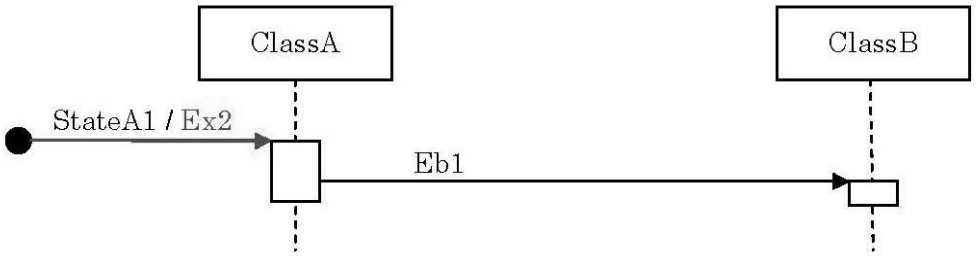
【図１０】


StateA1
Ea1 / Ma1
StateA2
Ea3
Ea2 / Mx1
StateA3

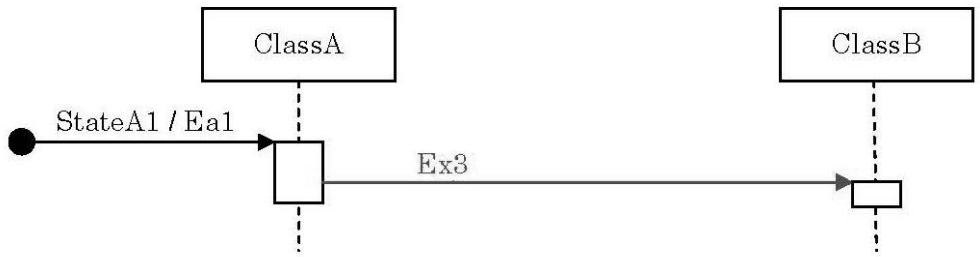
【図１１】


ClassX
ClassB
StateA1 / Ea1
Eb1

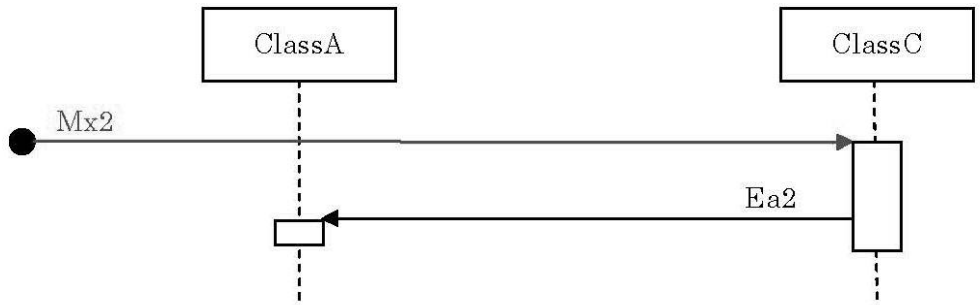
【図１２】


ClassA
ClassB
StateA1 / Ex2
Eb1

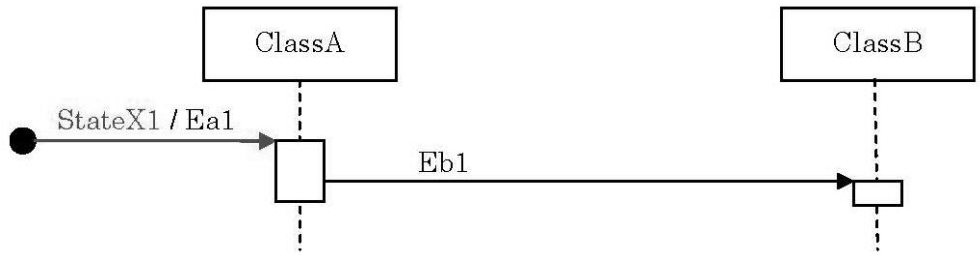
【図１３】


ClassA
ClassB
StateA1 / Ea1
Ex3

【図１４】


ClassA
ClassC
Mx2
Ea2

【図１５】


ClassA
ClassB
StateX1 / Ea1
Eb1

【図１６】

```
1| /** ClassA */
2| public class ClassA {
3|   /* 状態 */
4|   int StateA1 = 1;
5|   int StateA2 = 2;
6|   int StateA3 = 3;
7|   /* イベント */
8|   public static int Ea1 = 1;
9|   public static int Ea2 = 2;
10|  public static int Ea3 = 3;
11|  /* タスク */
12|  Task task;
13|  /* 状態変数 */
14|  int state = StateA1;
15|  /* 属性 */
16|  private int a;
17|  private char b;
18|  /* コンストラクタ */
19|  public ClassA(Task t) {
20|    task = t;
21|  }
22|  /* タスク取得 */
23|  public Task getTask() {
24|    return task ;
25|  }
26|  /* 処理関数 */
27|  private void Ma1() {
28|    // Ma1 の処理
29|  }
30|  private void Ma2() {
31|    // Ma2 の処理
32|  }
```

```
33|  /* 状態遷移処理 */
34|  public void stateTransition(int e) {
35|    switch(state) {
36|      case StateA1:
37|        if(e==Ea1) {
38|          Ma1();
39|          state = StateA2;
40|        }
41|        if(e==Ea1) {
42|          Main.getClassB().getTask().sendE
vent(ClassB.Eb1);
43|          Main.getClassC().getTask().sendE
vent(ClassC.Ec1);
44|        }
45|        break;
46|      case StateA2:
47|        if(e==Ea2) {
48|          Ma2();
49|          state = StateA3;
50|        }
51|        if(e==Ea2) {
52|          Main.getClassB().getTask().sendE
vent(ClassB.Eb2);
53|        }
54|        break;
55|      case StateA3:
56|        if(e==Ea3) state = StateA1;
57|        break;
58|    }
59|  }
60| }
```

【図１７】

```
/** Main */
public class Main {
  /* タスク(スレッド) */
  Task task1;
  Task task2;
  /* オブジェクト */
  static ClassA classA;
  static ClassB classB;
  static ClassC classC;
  /* メイン処理 */
  public static void main(String args[]) {
    /* タスク(スレッド)生成 */
    task1 = new Task();
    task2 = new Task();
    /* オブジェクト生成 */
    classA = new ClassA(task1);
    classB = new ClassB(task1);
    classC = new ClassC(task2);
    /* タスク(スレッド)スタート */
    task1.start();
    task2.start();
  }
  /* classA 取得 */
  public static ClassA getClassA() {
    return classA;
  }
  /* classB 取得 */
  public static ClassB getClassB() {
    return classB;
  }
  /* classC 取得 */
  public static ClassC getClassC() {
    return classC;
  }
}
```

フロントページの続き

(72)発明者　曽山　和弘
東京都品川区大崎一丁目１１番２号　ゲートシティ大崎イーストタワー　富士電機リテイルシステムズ株式会社内

Ｆターム(参考) 5B376 BB03　BB05　BB06　BB11　BB20　BC06　BC33　BC44　FA25