(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2007-94922
(P2007-94922A)
(43)公開日　平成19年4月12日(2007.4.12)

(51)Int.Cl.		F I			テーマコード（参考）
G06F 9/44	(2006.01)	G06F	9/06	620B	5B076 5B176

審査請求 未請求　請求項の数 8　ＯＬ　（全 15 頁）

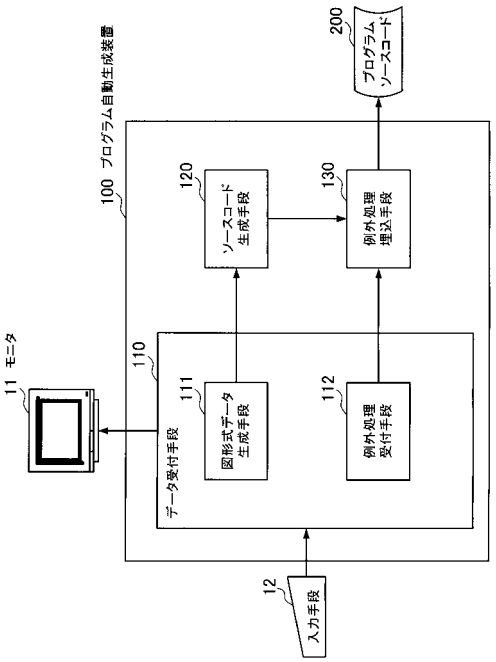
(21)出願番号　特願2005-285802 (P2005-285802)
(22)出願日　平成17年9月30日 (2005.9.30)

(71)出願人　000005234
富士電機ホールディングス株式会社
神奈川県川崎市川崎区田辺新田１番１号
(74)代理人　100092152
弁理士　服部　毅巖
(72)発明者　露木　正年
神奈川県横須賀市長坂二丁目２番１号　富士電機アドバンストテクノロジー株式会社内

Ｆターム(参考)　5B076 DD04　DE03
5B176 DD04　DE03

(54)【発明の名称】プログラム自動生成装置、方法、およびプログラム

(57)【要約】
【課題】プログラム自動生成装置がプログラムに自動で例外処理を追加できるようにする。
【解決手段】図形式データ生成手段１１１が、ユーザからの指示を受けて、図形式データを生成する。ソースコード生成手段１２０が、図形式データからコンパイル可能なプログラムソースコードを生成する。例外処理受付手段１１２が、図形式データで定義されたどの状態とも異なっている場合に行う例外処理のソースコードを受け付ける。例外処理埋込手段１３０が、プログラムソースコードに例外処理のソースコードを埋め込み、新たなプログラムソースコード２００として出力する。
【選択図】図１



【特許請求の範囲】
【請求項１】
プログラムの振る舞いを図形式で表現した図形式データからプログラムを生成するプログラム自動生成装置において、
ユーザからの指示を受けて、前記図形式データを生成する図形式データ生成手段と、
前記図形式データからコンパイル可能なプログラムソースコードを生成するソースコード生成手段と、
前記図形式データで定義されたどの状態とも異なっている場合に行う例外処理のソースコードを受け付ける例外処理受付手段と、
前記プログラムソースコードに前記例外処理のソースコードを埋め込み、新たなプログラムソースコードとして出力する例外処理埋込手段と、
を有することを特徴とするプログラム自動生成装置。
【請求項２】
前記例外処理受付手段は、前記例外処理のソースコードを前記図形式データの要素として取り扱うことを特徴とする請求項１記載のプログラム自動生成装置。
【請求項３】
前記例外処理受付手段は、前記例外処理のソースコードを前記図形式データの属性として取り扱うことを特徴とする請求項１記載のプログラム自動生成装置。
【請求項４】
前記例外処理受付手段は、ユーザから手入力で入力された前記例外処理のソースコードを受け付けることを特徴とする請求項１記載のプログラム自動生成装置。
【請求項５】
前記例外処理受付手段は、あらかじめ作成された前記例外処理のソースコードを受け付けることを特徴とする請求項１記載のプログラム自動生成装置。
【請求項６】
前記例外処理受付手段は、記録媒体に記録された前記例外処理のソースコードを受け付けることを特徴とする請求項１記載のプログラム自動生成装置。
【請求項７】
プログラムの振る舞いを図形式で表現した図形式データからプログラムを生成するプログラム自動生成方法において、
図形式データ生成手段が、ユーザからの指示を受けて、前記図形式データを生成するステップと、
ソースコード生成手段が、前記図形式データからコンパイル可能なプログラムソースコードを生成するステップと、
例外処理受付手段が、前記図形式データで定義されたどの状態とも異なっている場合に行う例外処理のソースコードを受け付けるステップと、
例外処理埋込手段が、前記プログラムソースコードに前記例外処理のソースコードを埋め込み、新たなプログラムソースコードとして出力するステップと、
を含むことを特徴とするプログラム自動生成方法。
【請求項８】
プログラムの振る舞いを図形式で表現した図形式データからプログラムを生成するプログラム自動生成プログラムにおいて、
コンピュータに、
ユーザからの指示を受けて、前記図形式データを生成する図形式データ生成手段と、
前記図形式データからコンパイル可能なプログラムソースコードを生成するソースコード生成手段、
前記図形式データで定義されたどの状態とも異なっている場合に行う例外処理のソースコードを受け付ける例外処理受付手段、
前記プログラムソースコードに前記例外処理のソースコードを埋め込み、新たなプログラムソースコードとして出力する例外処理埋込手段、

として機能させることを特徴とするプログラム自動生成プログラム。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、プログラム自動生成装置、方法、およびプログラムに関し、特にプログラムの振る舞いを図形式で表現した図形式データからプログラムを生成するプログラム自動生成装置、方法、およびプログラムに関する。

【背景技術】

【０００２】

従来より、プログラムの振る舞いが状態遷移図によって表されることはよく知られている。状態遷移図の標準的な表記方法は、ＵＭＬ（Unified Modeling Language）の状態マシン図としても定義されている。また、状態遷移図をベースに記述されたプログラムの仕様をもとに、プログラムのソースコードを自動生成する装置が提案されている（たとえば、特許文献１参照）。

【０００３】

図９は、従来のプログラム自動生成装置によるプログラムを自動生成する自動生成処理の手順を示すフローチャートである。

従来のプログラム自動生成装置は、キーボードやマウスなどの入力手段を介してユーザからの指示を受けて状態遷移図を生成する（ステップＳ３１）。そして、プログラム自動生成装置は、生成した状態遷移図に対応するソースコードを自動で生成する（ステップＳ３２）。

【０００４】

図１０は、従来のプログラム自動生成装置で生成されたプログラムの振る舞いを示すフローチャートである。

生成されたプログラムは、イベントの生起を待つ処理が行われ（ステップＳ４１）、イベントが発生すると、現在の状態を判定する処理が行われる（ステップＳ４２）。そして、判定された現在の状態別に生起イベントとガード条件によって遷移先の状態を決定する処理が行われ（ステップＳ４３）、決定された遷移先の状態に更新する処理が行われる（ステップＳ４４）。

【０００５】

図１１は、従来のプログラム自動生成装置が生成したプログラムソースコードの内容の概略を示す図である。

プログラムソースコード４００は、従来のプログラム自動生成装置が生成したプログラムソースコードの内容の概略を示している。プログラムソースコード４００は、Ｃ言語形式でプログラムソースコードが生成されたときのソースコードを示している。プログラムソースコード４００に記載されている内容は、図１０に示したフローチャートの各処理に対応している。

【０００６】

上から3行目の“ｉｆ”から始まる行は、図１０に示したフローチャートのステップＳ４１の処理に対応する。つまり、イベントの生起を待つ処理を行う。上から４行目の“ｓｗｉｔｃｈ”から始まる行は、図１０に示したフローチャートのステップＳ４２の処理に対応する。つまり、現在の状態を判定し、判定された現在の状態別に生起イベントとガード条件によって遷移先を決定する処理を行う。遷移先の処理は、この行の一番後ろの“｛”から始まるかっこで囲われている各処理である。

【０００７】

プログラムの状態が状態０から１、２、３、・・・、ｎ－１、ｎまであるとしたとき、上から5行目の“ｃａｓｅ　状態０；”と書いてある行を含めて“ｂｒｅａｋ；”と書いてある行までの3行が状態０と判定されたときに実行されるコードであり、図１０に示したフローチャートのステップＳ４３ａの処理に対応する。以下同様に、上から８行目の“ｃａｓｅ　状態１；”と書いてある行を含めて“ｂｒｅａｋ；”と書いてある行までの3

行が状態１と判定されたときに実行されるコードであり、図１０に示したフローチャートのステップＳ４３ｂの処理に対応し、上から１２行目の“ｃａｓｅ　状態ｎ；”と書いてある行を含めて“ｂｒｅａｋ；”と書いてある行までの３行が状態ｎと判定されたときに実行されるコードであり、図１０に示したフローチャートのステップＳ４３ｄの処理に対応する。また、下から３行目の“現在の”から始まる行は、図１０に示したフローチャートのステップＳ４４の処理に対応する。

【特許文献１】特開平７－７８０７６号公報

【発明の開示】

【発明が解決しようとする課題】

【０００８】

しかし、従来のプログラム自動生成装置で生成されるソースコードは、現在の状態を保持するメモリ領域が、このソースコードが示す処理以外の場所で何らかの原因により書き換えられ、現在の状態が想定したすべての状態に当てはまらなかった場合、つまり、上記例においては、状態０から１、２、３、・・・、ｎ－１、ｎまで定義されたいずれの状態にも当てはまらなかった場合の例外処理がない。現在の状態がすべての状態に当てはまらないということは、明らかにプログラムが異常な動作をしている状態であり、対象システムによっては異常状態を放置しておくことが致命的な障害につながる場合もあるため、何らかの警告を発したり、場合によってはシステムの機能を停止させるなどの対策が必要となる。つまり、実際のシステムの一部としてこのプログラムを実行させることを考えた場合、このソースコードでは信頼性が低いということになる。そのため、従来は例外処理がなくても問題がないと判断できる部分にしか自動生成を適用することができないという問題があった。また、例外処理がないと問題があると判断された場合には、自動生成されたソースコードに例外処理を人手で追加しなければならないという問題もあった。つまり、プログラムを自動で生成するメリットが損なわれていた。

【０００９】

本発明はこのような点に鑑みてなされたものであり、プログラムに自動で例外処理を追加できるプログラム自動生成装置、方法、および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】

【００１０】

本発明では上記問題を解決するために、プログラムの振る舞いを図形式で表現した図形式データからプログラムを生成するプログラム自動生成装置において、ユーザからの指示を受けて、前記図形式データを生成する図形式データ生成手段と、前記図形式データからコンパイル可能なプログラムソースコードを生成するソースコード生成手段と、前記図形式データで定義されたどの状態とも異なっている場合に行う例外処理のソースコードを受け付ける例外処理受付手段と、前記プログラムソースコードに前記例外処理のソースコードを埋め込み、新たなプログラムソースコードとして出力する例外処理埋込手段とを有することを特徴とするプログラム自動生成装置が提供される。

【００１１】

このようなプログラム自動生成装置によれば、図形式データ生成手段が、ユーザからの指示を受けて、図形式データを生成する。ソースコード生成手段が、図形式データからコンパイル可能なプログラムソースコードを生成する。例外処理受付手段が、図形式データで定義されたどの状態とも異なっている場合に行う例外処理のソースコードを受け付ける。例外処理埋込手段が、プログラムソースコードに例外処理のソースコードを埋め込み、新たなプログラムソースコードとして出力する。

【００１２】

また、本発明では、プログラムの振る舞いを図形式で表現した図形式データからプログラムを生成するプログラム自動生成方法において、図形式データ生成手段が、ユーザからの指示を受けて、前記図形式データを生成するステップと、ソースコード生成手段が、前記図形式データからコンパイル可能なプログラムソースコードを生成するステップと、例

外処理受付手段が、前記図形式データで定義されたどの状態とも異なっている場合に行う例外処理のソースコードを受け付けるステップと、例外処理埋込手段が、前記プログラムソースコードに前記例外処理のソースコードを埋め込み、新たなプログラムソースコードとして出力するステップとを含むことを特徴とするプログラム自動生成方法が提供される。

【００１３】

このようなプログラム自動生成方法によれば、図形式データ生成手段が、ユーザからの指示を受けて、図形式データを生成する。ソースコード生成手段が、図形式データからコンパイル可能なプログラムソースコードを生成する。例外処理受付手段が、図形式データで定義されたどの状態とも異なっている場合に行う例外処理のソースコードを受け付ける。例外処理埋込手段が、プログラムソースコードに例外処理のソースコードを埋め込み、新たなプログラムソースコードとして出力する。

【００１４】

また、本発明では、プログラムの振る舞いを図形式で表現した図形式データからプログラムを生成するプログラム自動生成プログラムにおいて、コンピュータに、ユーザからの指示を受けて、前記図形式データを生成する図形式データ生成手段と、前記図形式データからコンパイル可能なプログラムソースコードを生成するソースコード生成手段、前記図形式データで定義されたどの状態とも異なっている場合に行う例外処理のソースコードを受け付ける例外処理受付手段、前記プログラムソースコードに前記例外処理のソースコードを埋め込み、新たなプログラムソースコードとして出力する例外処理埋込手段として機能させることを特徴とするプログラム自動生成プログラムが提供される。

【００１５】

このようなプログラム自動生成プログラムによれば、図形式データ生成手段が、ユーザからの指示を受けて、図形式データを生成する。ソースコード生成手段が、図形式データからコンパイル可能なプログラムソースコードを生成する。例外処理受付手段が、図形式データで定義されたどの状態とも異なっている場合に行う例外処理のソースコードを受け付ける。例外処理埋込手段が、プログラムソースコードに例外処理のソースコードを埋め込み、新たなプログラムソースコードとして出力する。

【発明の効果】

【００１６】

本発明のプログラム自動生成装置、方法、およびプログラムによれば、例外処理をプログラムに自動で追加できるので、例外処理がないと問題がある場合であるか否かに関係なくプログラムを自動で生成することができる。また、例外処理を人手で追加しなくてもよくなる。したがって、プログラムを自動で生成するメリットが向上する。

【発明を実施するための最良の形態】

【００１７】

以下、本発明の実施の形態を図面を参照して詳細に説明する。

図１は、本実施の形態に適用される発明の概念図である。図１に示すように、プログラム自動生成装置１００は、データ受付手段１１０、ソースコード生成手段１２０、および例外処理埋込手段１３０を備えている。また、データ受付手段１１０は、図形式データ生成手段１１１と例外処理受付手段１１２から構成されている。また、データ受付手段１１０は、入力手段１２を介してユーザの指示を受け、図形式データの生成や例外処理の受け付けを行う。生成した図形式データや受け付けた例外処理は逐次モニタ１１に表示する。図形式データ生成手段１１１は、入力手段１２を介してユーザからの指示を受けて、ユーザが生成したいプログラムを表す図形式データを生成する。例外処理受付手段１１２は、入力手段１２を介してユーザから例外処理を行う例外処理ソースコードを受け付ける。

【００１８】

ソースコード生成手段１２０は、図形式データ生成手段１１１から図形式データを受け取ると、図形式データとして表現されているプログラムを実現するためのソースコードを図形式データに基づいて生成する。例外処理埋込手段１３０は、例外処理受付手段１１２

が受け付けた例外処理ソースコードを受け取る。また、ソースコード生成手段１２０が生成したソースコード（以下、生成ソースコードという。）を受け取る。そして、生成ソースコードの所定の位置に例外処理ソースコードを埋め込む。そして、例外処理を行うことができる新たなプログラムソースコード２００として出力する。

【００１９】

このようにプログラム自動生成装置１００は、例外処理埋込手段１３０によって例外処理ソースコードを生成ソースコードに自動で埋め込み、新たに例外処理をすることができるプログラムソースコード２００として出力することができるので、例外処理が必要か不必要かに関係なくプログラム自動生成装置１００を用いてプログラムの自動生成を行わせることができる。また、例外処理を自動で必要な場所に埋め込むことができるので、容易に例外処理機能が盛り込まれたプログラムを自動で生成することができる。

【００２０】

図２は、本実施の形態に用いるプログラム自動生成装置のハードウェア構成例を示す図である。プログラム自動生成装置１００は、ＣＰＵ（Central Processing Unit）１０１によって装置全体が制御されている。ＣＰＵ１０１には、バス１０６を介してＲＡＭ（Random Access Memory）１０２、ハードディスクドライブ（ＨＤＤ:Hard Disk Drive）１０３、グラフィック処理装置１０４、および入力インタフェース１０５が接続されている。

【００２１】

ＲＡＭ１０２には、ＣＰＵ１０１に実行させるＯＳのプログラムやアプリケーションプログラムの少なくとも一部が一時的に格納される。また、ＲＡＭ１０２には、ＣＰＵ１０１による処理に必要な各種データが格納される。ＨＤＤ１０３には、ＯＳ（Operating System）やアプリケーションプログラムが格納される。

【００２２】

グラフィック処理装置１０４には、モニタ１１が接続されている。グラフィック処理装置１０４は、ＣＰＵ１０１からの命令に従って、画像をモニタ１１の画面に表示させる。入力インタフェース１０５には、キーボード１２ａとマウス１２ｂとが接続されている。入力インタフェース１０５は、キーボード１２ａやマウス１２ｂから送られてくる信号を、バス１０６を介してＣＰＵ１０１に送信する。以上のようなハードウェア構成によって、本実施の形態の処理機能を実現することができる。

【００２３】

図３は、本実施の形態のプログラム自動生成装置によるプログラムを自動生成する自動生成処理の手順を示すフローチャートである。以下、図３に示す処理をステップ番号に沿って説明する。

【００２４】

〔ステップＳ１１〕図形式データ生成手段１１１は、入力手段１２を介して入力されるユーザからの指示を受け付ける。この指示に基づいて、図形式データであり、プログラムの振る舞いを表現する状態遷移図を生成する。

【００２５】

〔ステップＳ１２〕例外処理受付手段１１２は、入力手段１２を介して入力される例外処理ソースコードを受け付ける。

〔ステップＳ１３〕ソースコード生成手段１２０は、図形式データ生成手段１１１から状態遷移図を受け取る。そして、受け取った状態遷移図に基づいて状態遷移図が表す処理を行うプログラムのソースコードを自動生成する。

【００２６】

〔ステップＳ１４〕例外処理埋込手段１３０は、ソースコード生成手段１２０から生成ソースコードを受け取る。また、例外処理受付手段１１２から例外処理ソースコードを受け取る。そして、生成ソースコードの所定の位置に例外処理ソースコードを埋め込み、新たなプログラムソースコード２００として出力する。

【００２７】

図４は、本実施の形態のプログラム自動生成装置が生成したプログラムの振る舞いを示

すフローチャートである。以下、図４に示す処理をステップ番号に沿って説明する。
〔ステップＳ２１〕イベントの生起を判断する。イベントが発生したときは処理をステップＳ２２へ進める。
【００２８】
〔ステップＳ２２〕現在の状態を判断する。状態０と判断した場合には、処理をステップＳ２３ａへ進め、状態１と判断した場合には、処理をステップＳ２３ｂへ進め、状態ｎ－１と判断した場合には、処理をステップＳ２３ｃへ進め、状態ｎと判断した場合には、処理をステップＳ２３ｄへ進め、状態０からｎまでに該当しないと判断した場合には、処理をステップＳ２５へ進める。
【００２９】
〔ステップＳ２３〕現在の状態別に生起イベントとガード条件によって遷移先の状態を決定する。
〔ステップＳ２４〕決定された遷移先の状態に更新し、処理をステップＳ２１へ進める。
【００３０】
〔ステップＳ２５〕例外処理ソースコードで表される例外処理を行い、処理をステップＳ２１へ進める。
図５は、本実施の形態のプログラム自動生成装置が生成したプログラムソースコードの概略を示す図である。
【００３１】
プログラムソースコード２０１は、プログラム自動生成装置１００が生成したプログラムソースコード２００の概略を示している。プログラムソースコード２０１は、Ｃ言語形式でプログラムソースコード２００が生成されたときのソースコードを示している。プログラムソースコード２０１に記載されている内容は、図４に示したフローチャートの各処理に対応している。
【００３２】
上から3行目の“ｉｆ”から始まる行は、図４に示したフローチャートのステップＳ２１の処理に対応する。つまり、イベントの生起を待つ処理を行う。上から４行目の“ｓｗｉｔｃｈ”から始まる行は、図４に示したフローチャートのステップＳ２２の処理に対応する。つまり、現在の状態を判定する。また、判定された現在の状態別に生起イベントとガード条件によって遷移先を決定する処理は、この行の一番後ろの“｛”から始まるかっこで囲われている各処理である。
【００３３】
プログラムの状態が状態０から１、２、３、・・・、ｎ－１、ｎまであるとしたとき、上から５行目の“ｃａｓｅ　状態０；”と書いてある行を含めて“ｂｒｅａｋ；”と書いてある行までの３行が状態０と判定されたときに実行されるコードであり、図４に示したフローチャートのステップＳ２３ａの処理に対応する。以下同様に、上から８行目の“ｃａｓｅ　状態１；”と書いてある行を含めて“ｂｒｅａｋ；”と書いてある行までの３行が状態１と判定されたときに実行されるコードであり、図４に示したフローチャートのステップＳ２３ｂの処理に対応し、上から１２行目の“ｃａｓｅ　状態ｎ；”と書いてある行を含めて“ｂｒｅａｋ；”と書いてある行までの３行が状態ｎと判定されたときに実行されるコードであり、図４に示したフローチャートのステップＳ２３ｄの処理に対応する。
【００３４】
下から7行目の“ｄｅｆａｕｌｔ；”と書いてある行を含めて“ｂｒｅａｋ；”と書いてある行までの３行が上記状態０からｎ以外の状態になったときに行われる処理である。つまり、図５に示しているプログラムソースコード２０１中において点線で囲われた部分のコードが例外処理ソースコード２１０であり、図４に示したフローチャートのステップＳ２５の処理に対応する。また、下から３行目の“現在の”から始まる行は、図４に示したフローチャートのステップＳ２４の処理に対応する。つまり、決定された遷移先の状

態に更新する。

【００３５】

図６は、データ受付手段が受け付けたデータの表示の一例を示す図である。

表示例３００は、入力手段１２を介して入力されたデータをデータ受付手段１１０が受け付け、モニタ１１に出力したときの表示例である。表示例３００には、状態遷移図および例外アクティビティが示されており、状態３１１、３１２、３１３、遷移３２１、３２２、３２３、３２４、３２５、ラベル３３１、３３２、３３３、３３４、３３５、および例外アクティビティ３７０によってプログラムの振る舞いを表している。また、ラベル３３１は、イベント３４１、ガード条件３５１、およびアクション式３６１で構成されている。同様に、ラベル３３２は、イベント３４２、ガード条件３５２、およびアクション式３６２で構成されており、ラベル３３３は、イベント３４３、ガード条件３５３、およびアクション式３６３で構成されており、ラベル３３４は、イベント３４４、ガード条件３５４、およびアクション式３６４で構成されており、ラベル３３５は、イベント３４５、ガード条件３５５、およびアクション式３６５で構成されている。

【００３６】

遷移３２１は、状態３１１から状態３１２に進むことを示しており、ラベル３３１が関連づけられている。つまり、状態３１１のときに、イベント３４１に示されている“ｅｖｅｎｔ１”が発生し、ガード条件３５１に示されている“ｘ＝＝１”の条件を満たすと、アクション式３６１に示されている“ａ＝ｘ＋１”が実行され、状態３１２に状態が遷移する。

【００３７】

同様に、遷移３２２は、状態３１２から状態３１１に進むことを示しており、ラベル３３２が関連づけられている。つまり、状態３１２のときに、イベント３４２に示されている“ｅｖｅｎｔ２＿１”が発生し、ガード条件３５２に示されている“ｘ＝＝２”の条件を満たすと、アクション式３６２に示されている“ａ＝ｘ＋２”が実行され、状態３１１に状態が遷移し、遷移３２３は、状態３１２から状態３１３に進むことを示しており、ラベル３３３が関連づけられている。つまり、状態３１２のときに、イベント３４３に示されている“ｅｖｅｎｔ２＿２”が発生し、ガード条件３５３に示されている“ｘ＝＝３”の条件を満たすと、アクション式３６３に示されている“ａ＝ｘ＋３”が実行され、状態３１３に状態が遷移する。

【００３８】

また、遷移３２４は、状態３１３から状態３１２に進むことを示しており、ラベル３３４が関連づけられている。つまり、状態３１３のときに、イベント３４４に示されている“ｅｖｅｎｔ３＿１”が発生し、ガード条件３５４に示されている“ｘ＝＝４”の条件を満たすと、アクション式３６４に示されている“ａ＝ｘ－３”が実行され、状態３１２に状態が遷移し、遷移３２５は、状態３１３から状態３１１に進むことを示しており、ラベル３３５が関連づけられている。つまり、状態３１３のときに、イベント３４５に示されている“ｅｖｅｎｔ３＿２”が発生し、ガード条件３５５に示されている“ｘ＝＝５”の条件を満たすと、アクション式３６５に示されている“ａ＝０”が実行され、状態３１１に状態が遷移する。

【００３９】

また、例外アクティビティ３７０には、状態遷移図には定義されていない状態になったときに実行される処理である例外処理のコードが属性として記述されている。この例外アクティビティ３７０は、入力手段１２を介して例外処理ソースコードが入力できるようになっている。

【００４０】

図７は、本実施の形態のプログラム自動生成装置が生成したプログラムソースコードの内容の一例を示す図である。

プログラムソースコード２０２は、図６に示した表示例３００の内容から自動生成されたプログラムソースコードの内容を示している。上から３行目の“ｉｆ”から始まる行は

、イベントの生起を待つ処理であり、図４に示したフローチャートのステップＳ２１の処理に対応する。図６に示したとおり、表示例３００に示した状態遷移図では、“ｅｖｅｎｔ１”、“ｅｖｅｎｔ２＿１”、“ｅｖｅｎｔ２＿２”、“ｅｖｅｎｔ３＿１”、および“ｅｖｅｎｔ３＿２”がイベントとして定義されており、プログラムソースコード例２０２においても上記各イベントが発生したときに遷移する旨が、上から４行目の“ｎｅｘｔ＿ｓｔａｔｅ”から始まる行に記載されている。

【００４１】

上から５行目の“ｓｗｉｔｃｈ”から始まる行は、図４に示したフローチャートのステップＳ２２の処理に対応する。つまり、現在の状態を判定する。また、判定された現在の状態別に生起イベントとガード条件によって遷移先を決定する処理は、この行の一番後ろの“｛”から始まるかっこで囲われている各処理である。本実施の形態のプログラム自動生成装置１００においては、図６の状態３１１、３１２、３１３にあるように、プログラムの状態がそれぞれ“ＳＴＡＴＥ１”、“ＳＴＡＴＥ２”、および“ＳＴＡＴＥ３”と定義されている。

【００４２】

このとき上から６行目の“ｃａｓｅ　ＳＴＡＴＥ１；”と書いてある行を含めて“ｂｒｅａｋ；”と書いてある行までの６行が、状態が“ＳＴＡＴＥ１”と判定されたときに実行されるコードであり、たとえば、図４に示したフローチャートのステップＳ２３ａの処理に対応する。以下同様に、上から１２行目の“ｃａｓｅ　ＳＴＡＴＥ２；”と書いてある行を含めて“ｂｒｅａｋ；”と書いてある行までの１０行が、状態が“ＳＴＡＴＥ２”と判定されたときに実行されるコードであり、たとえば、図４に示したフローチャートのステップＳ２３ｂの処理に対応し、上から２２行目の“ｃａｓｅ　ＳＴＡＴＥ３；”と書いてある行を含めて“ｂｒｅａｋ；”と書いてある行までの１０行が、状態が“ＳＴＡＴＥ３”と判定されたときに実行されるコードであり、図４に示したフローチャートのステップＳ２３ｄの処理に対応する。

【００４３】

また、下から８行目の“ｄｅｆａｕｌｔ；”と書いてある行を含めて“ｂｒｅａｋ；”と書いてある行までの４行が上記状態“ＳＴＡＴＥ１”、“ＳＴＡＴＥ２”、および“ＳＴＡＴＥ３”以外の状態になったときに行われる処理である。つまり、この４行のコードが例外処理ソースコード２１０であり、図４に示したフローチャートのステップＳ２５の処理に対応する。また、下から３行目の“ｎｏｗ＿ｓｔａｔｅ”から始まる行は、図４に示したフローチャートのステップＳ２４の処理に対応する。以下に、ソースコード生成手段１２０が表示例３００に示されている状態遷移図を使ってどのようにしてソースコードを生成するかを説明する。

【００４４】

まずは状態を“ＳＴＡＴＥ１”と判定したときに実行するコードについて説明する。ソースコード生成手段１２０は、図６に示した状態遷移図の状態３１１から伸びている矢印がどの状態に対して伸びているかを検索する。表示例３００においては、状態３１１から状態３１２に対してのみ遷移３２１が表示されている。つまり、状態３１１からは状態３１２にしか遷移しないということになる。また、ソースコード生成手段１２０は、遷移３２１に関連づけられているラベル３３１を読み取る。そして、プログラムソースコード２０２の上から７行目にあるように“ｉｆ”から始まる行にイベント３４１とガード条件３５１を所定の位置に挿入する。また、上から８行目にアクション式３６１を挿入する。また、上から９行目の“ｎｅｘｔ＿ｓｔａｔｅ＝”の後ろに遷移３２１の遷移先である状態３１２に対応する“ＳＴＡＴＥ２”を挿入する。

【００４５】

また、状態を“ＳＴＡＴＥ２”と判定したときに実行するコードについて説明する。ソースコード生成手段１２０は、状態３１２から伸びている矢印がどの状態に対して伸びているかを検索する。表示例３００においては、状態３１２から状態３１１と状態３１３に対して遷移３２２、３２３が表示されている。つまり、状態３１２からは状態３１１と状

態３１３に対して遷移するということになる。また、ソースコード生成手段１２０は、遷移３２２に対応づけられているラベル３３２を読み取る。そして、プログラムソースコード２０２の上から１３行目にあるように“ｉｆ”から始まる行にイベント３４２とガード条件３５２を所定の位置に挿入する。また、上から１４行目にアクション式３６２を挿入する。また、上から１５行目の“ｎｅｘｔ＿ｓｔａｔｅ＝”の後ろに遷移３２２の遷移先である状態３１１に対応する“ＳＴＡＴＥ１”を挿入する。また、ソースコード生成手段１２０は、遷移３２３に対応づけられているラベル３３３を読み取る。そして、上から１７行目にあるように“ｅｌｓｅ　ｉｆ”から始まる行にイベント３４３とガード条件３５３を所定の位置に挿入する。また、上から１８行目にアクション式３６３を挿入する。また、上から１９行目の“ｎｅｘｔ＿ｓｔａｔｅ＝”の後ろに遷移３２３の遷移先である状態３１３に対応する“ＳＴＡＴＥ３”を挿入する。

【００４６】

また、状態を“ＳＴＡＴＥ３”と判定したときに実行するコードについて説明する。ソースコード生成手段１２０は、状態３１３から伸びている矢印がどの状態に対して伸びているかを検索する。表示例３００においては、状態３１３から状態３１１と状態３１２に対して遷移３２４、３２５が表示されている。つまり、状態３１３からは状態３１１と状態３１２に対して遷移するということになる。また、ソースコード生成手段１２０は、遷移３２４に対応づけられているラベル３３４を読み取る。そして、プログラムソースコード２０２の上から２３行目にあるように“ｉｆ”から始まる行にイベント３４４とガード条件３５４を所定の位置に挿入する。また、上から２４行目にアクション式３６４を挿入する。また、上から２５行目の“ｎｅｘｔ＿ｓｔａｔｅ＝”の後ろに遷移３２４の遷移先である状態３１２に対応する“ＳＴＡＴＥ２”を挿入する。また、ソースコード生成手段１２０は、遷移３２５に対応づけられているラベル３３５を読み取る。そして、上から２７行目にあるように“ｅｌｓｅ　ｉｆ”から始まる行にイベント３４５とガード条件３５５を所定の位置に挿入する。また、上から２８行目にアクション式３６５を挿入する。また、上から２９行目の“ｎｅｘｔ＿ｓｔａｔｅ＝”の後ろに遷移３２５の遷移先である状態３１１に対応する“ＳＴＡＴＥ１”を挿入する。

【００４７】

また、状態が“ＳＴＡＴＥ１”、“ＳＴＡＴＥ２”、および“ＳＴＡＴＥ３”以外の状態と判定したときに実行するコードについて説明する。例外処理埋込手段１３０は、例外処理受付手段１１２が受け付けた、例外アクティビティ３７０に記載された例外処理ソースコードを読み取る。そして、ソースコード生成手段１２０が生成した生成ソースコードにある、いわゆる“ｓｗｉｔｃｈ文”の最後に“ｄｅｆａｕｌｔ文”を挿入する。このようにすることによって、“ｃａｓｅ文”に定義されている“ＳＴＡＴＥ１”、“ＳＴＡＴＥ２”、および“ＳＴＡＴＥ３”以外の状態になったときには下から８行目からの“ｄｅｆａｕｌｔ文”が実行されることになる。つまり、定義されている状態以外の状態になったときに例外処理が行われることになる。

【００４８】

以上のようなプログラム自動生成装置１００を用いることにより、例外アクティビティ３７０に記載した例外処理ソースコードが、ソースコード生成手段１２０が生成した生成ソースコードに自動で挿入されるので、新たに生成されたプログラムソースコード２００には想定されている状態以外の状態になっても例外処理が行われることになる。つまり、例外処理がないと問題がある場合であるか否かに関係なくプログラムを自動で生成することができる。また、例外処理を人手で追加しなくてもよくなる。したがって、プログラムを自動で生成するメリットが向上する。

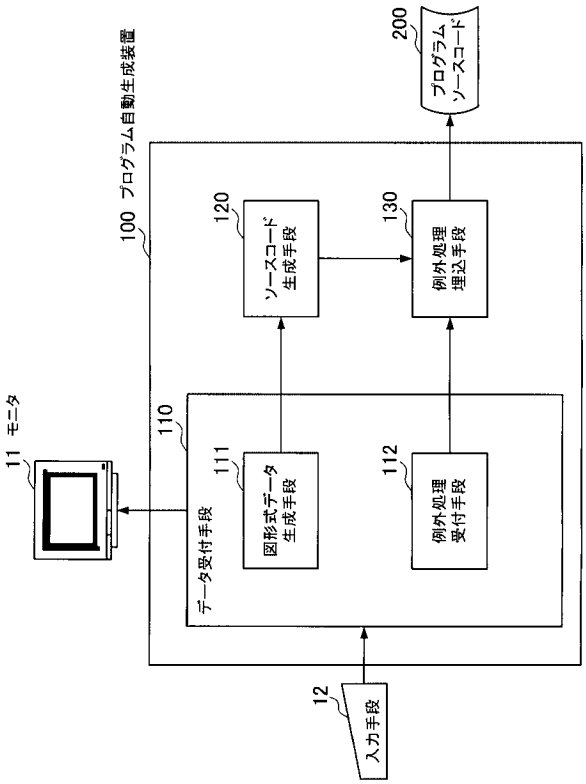
【００４９】

なお、図６において、入力手段１２を介して入力されたデータをデータ受付手段１１０が受け付け、モニタ１１に出力したときの表示例を表示例３００として示した。しかし、表示例３００のように例外アクティビティ３７０を必ずしも図形式データの一部として取り扱う必要はない。そこで、図形式データの属性として取り扱う場合の表示例を図８に示

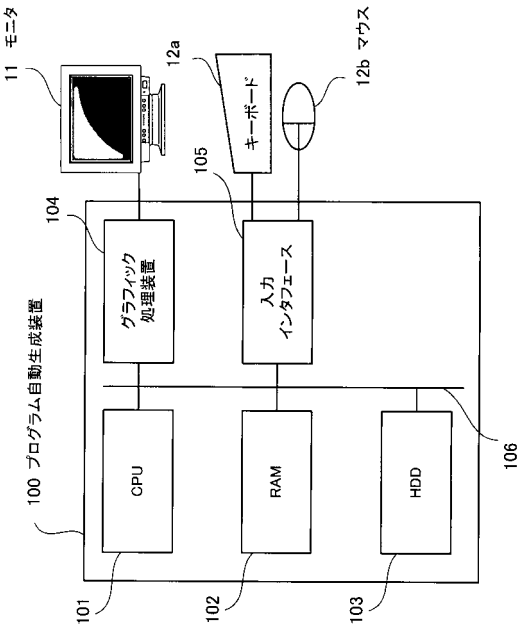
す。
【００５０】
図８は、データ受付手段が受け付けたデータの表示の別の例を示す図である。
表示例３０１には、図形データ表示部３０２と図面属性表示部３０３が表示されている。また、図面属性表示部３０３には、例外アクティビティ３７１が表示されている。図形データ表示部３０２は、図形データである状態遷移図を表示する。また、図面属性表示部３０３は、図形データ表示部３０２に表示されている状態遷移図の属性を表示する。その属性として例外アクティビティ３７１が表示されている。例外アクティビティ３７１に例外処理ソースコードを入力することによって、図形データ表示部３０２に表示されている状態遷移図の例外処理を行えるようにしてもよい。
【００５１】
このように、例外処理ソースコードを状態遷移図の属性データとして扱うことにより、例外アクティビティ３７１に入力した例外処理ソースコードを他の状態遷移図の例外処理のための例外処理ソースコードとして使うこともできる。したがって、状態遷移図を生成するたびに例外処理ソースコードを入力することなく例外処理を含んだプログラムソースコード２００を生成することができ、さらにプログラムを自動で生成するメリットが向上する。
【００５２】
また、例外アクティビティから入力手段を介して例外処理ソースコードを入力する旨を説明したが、例外処理ソースコードをプログラム自動生成装置とは別個に作成して、できあがった例外処理ソースコードを、たとえば、記録装置などに格納し、プログラム自動生成装置がその記録媒体を読み取ることによって例外処理ソースコードを受け取ってもよい。また、ネットワークを介して例外処理のソースコードを受け取ってもよい。その場合には、図２に示したハードウェアにネットワークインタフェースなどを設ける。
【図面の簡単な説明】
【００５３】
【図１】本実施の形態に適用される発明の概念図である。
【図２】本実施の形態に用いるプログラム自動生成装置のハードウェア構成例を示す図である。
【図３】本実施の形態のプログラム自動生成装置によるプログラムを自動生成する自動生成処理の手順を示すフローチャートである。
【図４】本実施の形態のプログラム自動生成装置が生成したプログラムの振る舞いを示すフローチャートである。
【図５】本実施の形態のプログラム自動生成装置が生成したプログラムソースコードの概略を示す図である。
【図６】データ受付手段が受け付けたデータの表示の一例を示す図である。
【図７】本実施の形態のプログラム自動生成装置が生成したプログラムソースコードの内容の一例を示す図である。
【図８】データ受付手段が受け付けたデータの表示の別の例を示す図である。
【図９】従来のプログラム自動生成装置によるプログラムを自動生成する自動生成処理の手順を示すフローチャートである。
【図１０】従来のプログラム自動生成装置で生成されたプログラムの振る舞いを示すフローチャートである。
【図１１】従来のプログラム自動生成装置が生成したプログラムソースコードの内容の概略を示す図である。
【符号の説明】
【００５４】
１１　モニタ
１２　入力手段
１００　プログラム自動生成装置

１１０　データ受付手段
１１１　図形式データ生成手段
１１２　例外処理受付手段
１２０　ソースコード生成手段
１３０　例外処理埋込手段
２００　プログラムソースコード

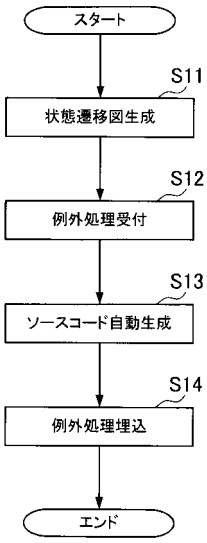
【図１】



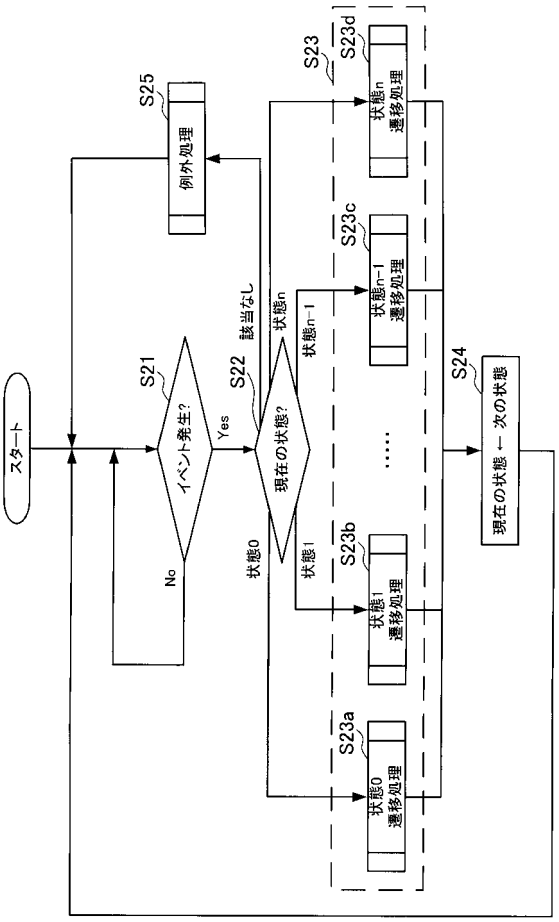
【図２】



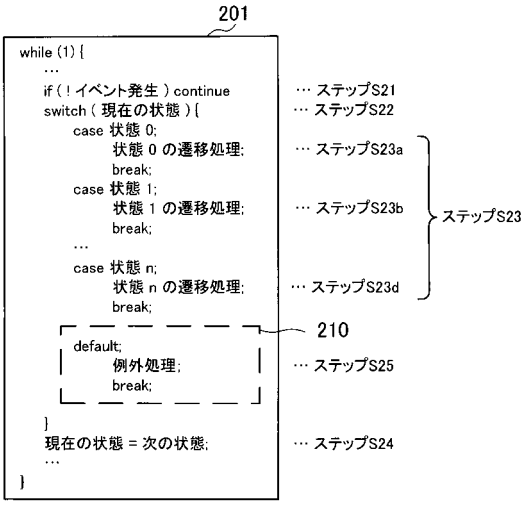
【図３】


スタート
S11
状態遷移図生成
S12
例外処理受付
S13
ソースコード自動生成
S14
例外処理埋込
エンド

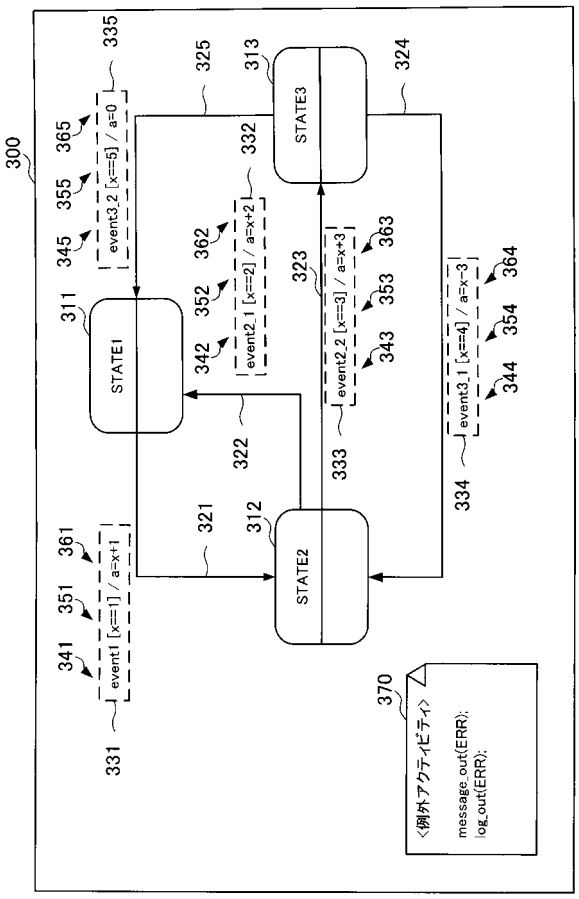
【図４】


スタート
S21
イベント発生?
No
Yes
S22
該当なし
現在の状態?
状態0
状態1
状態n-1
状態n
S23
S23a
状態0遷移処理
S23b
状態1遷移処理
S23c
状態n-1遷移処理
S23d
状態n遷移処理
S24
現在の状態 ← 次の状態
S25
例外処理

【図５】


201
while (1) {
…
if (! イベント発生) continue … ステップS21
switch (現在の状態) { … ステップS22
case 状態 0;
状態 0 の遷移処理; … ステップS23a
break;
case 状態 1;
状態 1 の遷移処理; … ステップS23b
break;
…
case 状態 n;
状態 n の遷移処理; … ステップS23d
break;
ステップS23
210
default;
例外処理; … ステップS25
break;
}
現在の状態 = 次の状態; … ステップS24
…
}

【図６】

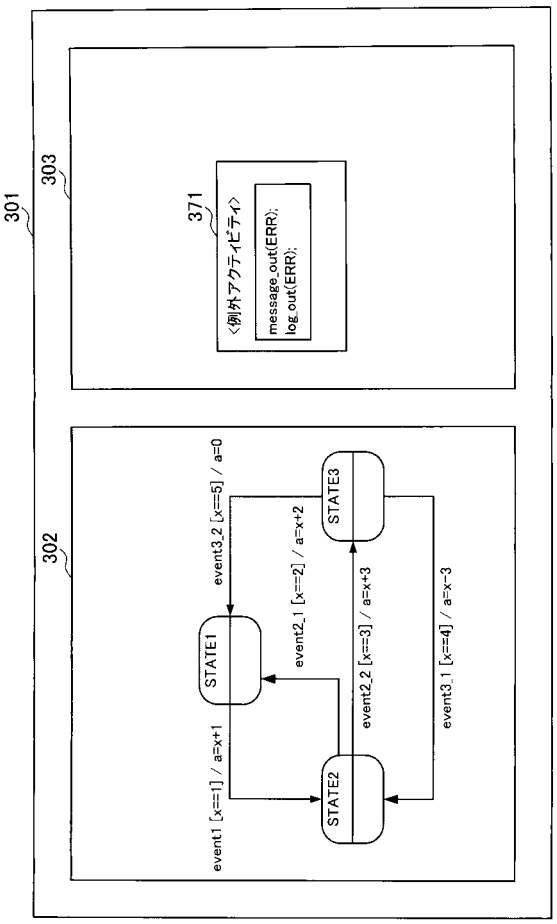

300
STATE1
STATE2
STATE3
311
312
313
321
322
323
324
325
331
332
333
334
335
341
342
343
344
345
351
352
353
354
355
361
362
363
364
365
370
event1 [x==1] / a=x+1
event2_1 [x==2] / a=x+2
event2_2 [x==3] / a=x+3
event3_1 [x==4] / a=x-3
event3_2 [x==5] / a=0
〈例外アクティビティ〉
message_out(ERR);
log_out(ERR);

【図７】

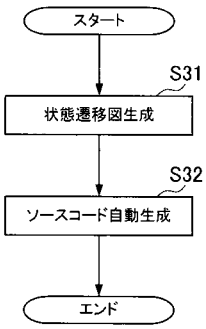
202

```
while (1) {
    …
    if ( ! ( event1 || event2_1 || event2_2 || event3_1 || event3_2 ) ) continue;
    next_state = now_state;
    switch ( now_state ) {
        case STATE1;
            if ( event1 && ( x==1 ) ) {
              a = x+1;
              next_state = STATE2;
            }
            break;
        case STATE2;
            if ( event2_1 && ( x==2 ) ) {
              a = x+2;
              next_state = STATE1;
            }
            else if ( event2_2 && ( x==3 ) ) {
              a = x+3;
              next_state = STATE3;
            }
            break;
        case STATE3;
            if ( event3_1 && ( x==4 ) ) {
              a = x-3;
              next_state = STATE2;
             }
            else if ( event3_2 && ( x==5 ) ) {
              a = 0;
              next_state = STATE1;
            }
            break;
        default;
            message_out ( ERR );
            log_out ( ERR );
            break;
    }
    now_state = next_state;
    …
}
```

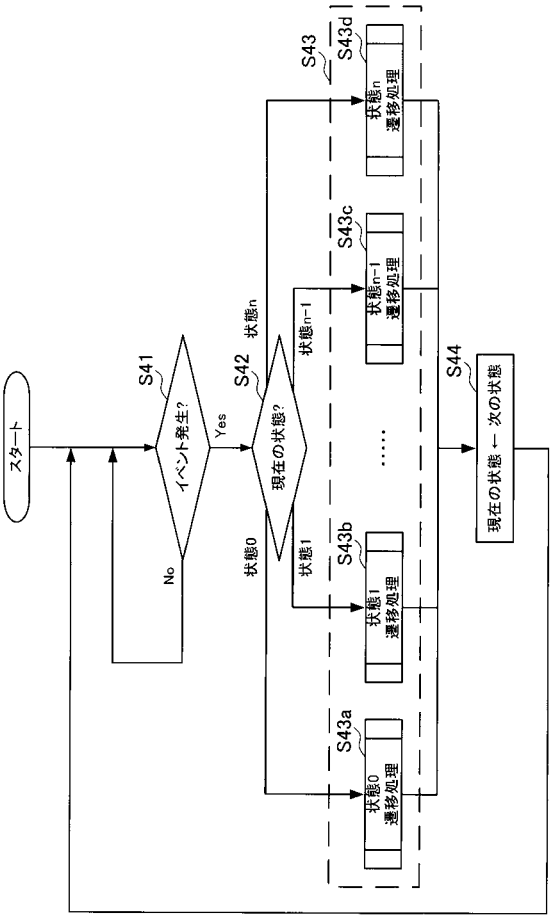
【図８】



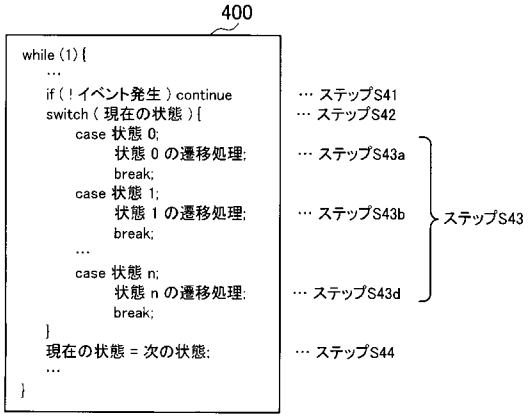
【図９】



【図１０】



【図１１】


400
while (1) {
…
if (! イベント発生) continue … ステップS41
switch (現在の状態) { … ステップS42
case 状態 0;
状態 0 の遷移処理; … ステップS43a
break;
case 状態 1;
状態 1 の遷移処理; … ステップS43b
break;
…
case 状態 n;
状態 n の遷移処理; … ステップS43d
break;
}
現在の状態 = 次の状態; … ステップS44
…
}
ステップS43