(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4938991号
(P4938991)
(45)発行日　平成24年5月23日(2012.5.23)　(24)登録日　平成24年3月2日(2012.3.2)

(51)Int.Cl.　　F I
G06F 9/44 (2006.01)　　G06F 9/06 620A
G06F 9/45 (2006.01)　　G06F 9/44 322F

請求項の数 9　(全 16 頁)

(21)出願番号　特願2005-103868(P2005-103868)
(22)出願日　平成17年3月31日(2005.3.31)
(65)公開番号　特開2006-285582(P2006-285582A)
(43)公開日　平成18年10月19日(2006.10.19)
審査請求日　平成20年3月28日(2008.3.28)

(73)特許権者　000001007
キヤノン株式会社
東京都大田区下丸子３丁目３０番２号
(74)代理人　100090273
弁理士　國分　孝悦
(72)発明者　野間口　徹
東京都大田区下丸子３丁目３０番２号　キヤノン株式会社内

審査官　新井　寛

最終頁に続く

(54)【発明の名称】プログラムコード生成装置、プログラムコード生成方法、及びコンピュータプログラム

(57)【特許請求の範囲】
【請求項１】
プラットフォームに非依存なオブジェクト指向で記述されている設計図面を用いてプログラムコードを生成するプログラムコード生成装置であって、
前記オブジェクト指向で記述されている設計図面における図面要素間の依存関係を解析し、前記設計図面において利用されていない機能を抽出する抽出手段と、
生成するプログラムコードの実行対象となるプラットフォームの情報を実行環境制約情報として取得する取得手段と、
前記取得手段により取得された実行環境制約情報に基づいて実行対象となるプラットフォームに応じたプログラムコードを生成する際に、前記抽出手段の抽出した前記設計図面において利用されていない機能に関するコードが除かれているプログラムコードを生成するプログラムコード生成手段とを有することを特徴とするプログラムコード生成装置。
【請求項２】
前記プログラムコード生成手段は、前記生成するプログラムコードに関するライブラリについて、前記抽出手段の抽出した前記設計図面において利用されていない機能に関するコードを除いたライブラリに最適化することを特徴とする請求項１に記載のプログラムコード生成装置。
【請求項３】
前記設計図面は、ドメイン図、クラス図、状態遷移図、及びアクション言語で記述されたクラスの挙動を規定する図面の少なくとも１つを含むことを特徴とする請求項１又は２

に記載のプログラムコード生成装置。
【請求項４】
前記抽出手段は、前記設計図面から、利用されていないクラス、メソッド、イベント、書き込みアクセスインタンス、クラス、及び属性値の少なくとも１つを前記利用されていない機能として抽出することを特徴とする請求項１乃至３のいずれか１項に記載のプログラムコード生成装置。
【請求項５】
前記利用されていない機能に関するイベントにより遷移する遷移線と、前記利用されていない機能に関して遷移されることのなくなった状態とを前記設計図面の含む状態遷移図から除去し、前記設計図面を最適化する最適化手段をさらに有することを特徴とする請求項１乃至４のいずれか１項に記載のプログラムコード生成装置。
【請求項６】
前記実行環境制約情報は、使用メモリ量の制限を示す情報を含むことを示す請求項１乃至５のいずれか１項に記載のプログラムコード生成装置。
【請求項７】
前記実行環境制約情報は、３２bit演算の禁止を示す情報を含むことを示す請求項１乃至６のいずれか１項に記載のプログラムコード生成装置。
【請求項８】
プラットフォームに非依存なオブジェクト指向で記述されている設計図面を用いてプログラムコードを生成することをコンピュータで実現される手段により実行するプログラムコード生成方法であって、
前記オブジェクト指向で記述されている設計図面における図面要素間の依存関係を解析し、前記設計図面において利用されていない機能を抽出手段により抽出する抽出ステップと、
生成するプログラムコードの実行対象となるプラットフォームの情報を取得手段により実行環境制約情報として取得する取得ステップと、
前記取得ステップにおいて取得された実行環境制約情報に基づいて実行対象となるプラットフォームに応じたプログラムコードを生成する際に、前記抽出ステップで抽出した前記設計図面において利用されていない機能に関するコードが除かれているプログラムコードをプログラムコード生成手段により生成するプログラムコード生成ステップとを有することを特徴とするプログラムコード生成方法。
【請求項９】
プラットフォームに非依存なオブジェクト指向で記述されている設計図面を用いてプログラムコードを生成する処理をコンピュータに実行させるためのコンピュータプログラムであって、
前記オブジェクト指向で記述されている設計図面における図面要素間の依存関係を解析し、前記設計図面において利用されていない機能を抽出する抽出ステップと、
生成するプログラムコードの実行対象となるプラットフォームの情報を実行環境制約情報として取得する取得ステップと、
前記取得ステップにおいて取得された実行環境制約情報に基づいて実行対象となるプラットフォームに応じたプログラムコードを生成する際に、前記抽出ステップで抽出した前記設計図面において利用されていない機能に関するコードが除かれているプログラムコードを生成するプログラムコード生成ステップとをコンピュータに実行させることを特徴とするコンピュータプログラム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、プログラムコード生成装置、プログラムコード生成方法、及びコンピュータプログラムに関し、特に、プログラムコードを生成するために用いて好適なものである。
【背景技術】

【０００２】
プログラムコード生成装置は、オブジェクト指向で記述された設計図面からプログラムコードを自動生成するものであるが、既存のプログラムコード生成装置から生成されるプログラムコードを保持するためには、多くのメモリ量を必要としていた。また、コードを生成する実行速度が遅いといった問題があった。
この問題を解決するものとして、特開平１１－２３７９８０号公報には、生成されるプログラムコードのサイズを削減する方式が記述されている。この特開平１１－２３７９８０号公報に記述されている技術は、オブジェクト指向で記述された設計図面からプログラムコードを自動生成するプログラムコード自動生成装置に関するもので、オブジェクト指向機能を除去する特徴を持つ。
【０００３】
市販ツールのＳＤＬ(Specification & Description Language)には、ＳＤＬを用いて設計図面を記述した際に利用可能となる演算子の定義のうち、未使用のものについて、コンパイルマクロで取り外すといった機能がある。しかしながら、ＳＤＬで除去する対象としているのが演算子のみであるため、設計図面内で利用されないモデル内の使われない要素などから生成されるコードの大半を除去することができないという問題があった。
【０００４】
モデルからプログラムコードを自動生成する技術に類似した技術として、コードレベルのコンパイラ・リンカが挙げられる。コンパイラは、ファイルスコープ内で最適化をしてアセンブラで記述されたオブジェクトファイルを生成する。リンカは、オブジェクトファイルを繋ぎ合わせる。しかしながら、最適化はファイルスコープ内で行なわれるため局所的な最適化となる。したがって、プログラムの全体を最適化することが出来ないという問題があった。
【０００５】
【特許文献１】特開平１１－２３７９８０号公報
【発明の開示】
【発明が解決しようとする課題】
【０００６】
以上のように従来の最適化は、オブジェクト指向での記述により、付随する不要なコードや不要な演算子を除去するのみにとどまるものであった。そして、最適化による消費メモリ量の削減や実行速度の高速化を実行することは行われていなかった。
また、プログラムコード・コンパイラでは、ファイルスコープ内での閉じた最適化のみしか出来ないため、プログラム全体の最適化が困難であるという問題があった。
【０００７】
本発明は、前述の問題点に鑑みてなされたものであり、実行速度とメモリ使用量とが適切なプログラムコードを生成することができるようにすることを第１の目的とする。
また、プログラム全体の最適化を行なえるようにすることを第２の目的とする。
【課題を解決するための手段】
【０００８】
本発明のプログラムコード生成装置は、プラットフォームに非依存なオブジェクト指向で記述されている設計図面を用いてプログラムコードを生成するプログラムコード生成装置であって、前記オブジェクト指向で記述されている設計図面における図面要素間の依存関係を解析し、前記設計図面において利用されていない機能を抽出する抽出手段と、生成するプログラムコードの実行対象となるプラットフォームの情報を実行環境制約情報として取得する取得手段と、前記取得手段により取得された実行環境制約情報に基づいて実行対象となるプラットフォームに応じたプログラムコードを生成する際に、前記抽出手段の抽出した前記設計図面において利用されていない機能に関するコードが除かれているプログラムコードを生成するプログラムコード生成手段とを有することを特徴とする。
【０００９】
本発明のプログラムコード生成方法は、プラットフォームに非依存なオブジェクト指向

で記述されている設計図面を用いてプログラムコードを生成することをコンピュータで実現される手段により実行するプログラムコード生成方法であって、前記オブジェクト指向で記述されている設計図面における図面要素間の依存関係を解析し、前記設計図面において利用されていない機能を抽出手段により抽出する抽出ステップと、生成するプログラムコードの実行対象となるプラットフォームの情報を取得手段により実行環境制約情報として取得する取得ステップと、前記取得ステップにおいて取得された実行環境制約情報に基づいて実行対象となるプラットフォームに応じたプログラムコードを生成する際に、前記抽出ステップで抽出した前記設計図面において利用されていない機能に関するコードが除かれているプログラムコードをプログラムコード生成手段により生成するプログラムコード生成ステップとを有することを特徴とする。

【００１０】

本発明のコンピュータプログラムは、プラットフォームに非依存なオブジェクト指向で記述されている設計図面を用いてプログラムコードを生成する処理をコンピュータに実行させるためのコンピュータプログラムであって、前記オブジェクト指向で記述されている設計図面における図面要素間の依存関係を解析し、前記設計図面において利用されていない機能を抽出する抽出ステップと、生成するプログラムコードの実行対象となるプラットフォームの情報を実行環境制約情報として取得する取得ステップと、前記取得ステップにおいて取得された実行環境制約情報に基づいて実行対象となるプラットフォームに応じたプログラムコードを生成する際に、前記抽出ステップで抽出した前記設計図面において利用されていない機能に関するコードが除かれているプログラムコードを生成するプログラムコード生成ステップとをコンピュータに実行させることを特徴とする。

【発明の効果】

【００１１】

本発明によれば、包括的に最適化を行なうことが可能になり、プログラムコードを実行する際のメモリ使用量を可及的に削減することができると共に、実行速度を可及的に高速化することができる。

【発明を実施するための最良の形態】

【００１２】

（第１の実施形態）

次に、図面を参照しながら、本発明の一実施形態について説明する。

図１は、本実施形態のプログラムコード生成装置の構成の一例を示すブロック図である。

図１において、１はＣＰＵである。ＣＰＵ１は、プログラムコード生成装置を統括制御し、例えば、バス２に接続された装置３、４、５、７、９にアクセスして制御を行なう。

３は、バス２を介してＣＰＵ１からアクセス可能な読み出し専用メモリ（ＲＯＭ）であり、本実施形態では、処理プログラム３ａ及び処理プログラムにより使用されるパラメータ３ｂがＲＯＭ３に格納されている。

４は読み書き可能なメモリ（ＲＡＭ）である。ＲＡＭ４には、設計図面情報、プラットフォーム情報、最適化可能箇所情報、実行環境制約情報、最適化予定情報、プログラムコード、プロジェクトファイル、ライブラリ、及び実行可能形式をそれぞれ格納するための領域４ａ～４ｉが確保されている。これらデータ格納領域４ａ～４ｉに格納される情報は、それぞれ処理プログラム３ａにより作成及び変更される。

【００１３】

５は入力インターフェイスであり、キーボード、ボタン、マウス、ダイアル、タブレット等の入力装置６を介してなされる入力を受け取る。７は出力インターフェイスであり、例えばＣＲＴ（又はＬＣＤ）８ａや、プリンタ（又はプロッタ）８ｂ等を備える出力装置８に対し、データの表示指示や出力指示を行なう。９は外部記憶装置インターフェイスであり、ＨＤ、ＦＤ、ＣＤ－ＲＯＭ、ＭＤなどの外部記憶装置１０に対するデータの入出力を行なうものである。

【００１４】

本実施形態では、処理プログラム３ａやパラメータ３ｂがＲＯＭ３にあるものとして、また、処理対象となる各データの格納領域４ａ～４ｉがＲＡＭ４にあるものとして説明を行なうが、これら全て又は一部を、外部記憶装置１０に配置することも可能であり、更に、必要に応じて、外部記憶装置１０からＲＡＭ４にロードして使用することもできる。また、ＣＰＵ１のキャッシュメモリに配置することも可能である。

【００１５】

図２は、図１に示した処理プログラム３ａの構成要件と、その構成要件と図１に示したデータ格納領域４ａ～４ｉに格納されるデータとの関係と、を示す図である。

【００１６】

図２において、２００は入力インターフェイス５を介して入力されるデータを扱うＧＵＩ入力処理部である。オペレータは、このＧＵＩ入力処理部２００によりデータの入力や編集ができるようになっている。

【００１７】

２０１は、実行可能な設計図面を外部記憶装置１０内から読み出し、構文解析を行なう実行可能設計図面解析部である。処理プログラム３ａが解釈することが可能な設計図面情報２０３を実行可能設計図面解析部２０１が生成することで、処理プログラム３ａが実行可能設計図面を解釈することを可能にする。なお、設計図面は、プラットフォームに依存せず、オブジェクト指向であるのが好ましい。プラットフォーム非依存の設計図面に対して最適化を行なえば、プラットフォームに依存部分の最適化を行なうことが容易になるからである。

【００１８】

２０２は、外部記憶装置１０内から実行ファイルの動作するＯＳや、プログラミング言語や、利用フレームワークといったプラットフォームデータを読み込み、構文解析を行なうプラットフォーム依存解析部である。処理プログラム３ａが解釈することが可能なプラットフォーム情報２０４をプラットフォーム依存解析部２０２が生成することで、処理プログラム３ａがプラットフォーム情報を解釈することを可能にする。

【００１９】

２０５は、設計図面情報２０３を解析することで、利用されていないクラス、メソッド、及びイベントや、書き込みアクセスインスタンスや、クラスの有無といった最適化が可能な変更箇所を特定する最適化可能箇所抽出部である。最適化可能な変更箇所を特定するための情報である最適化可能箇所２０７を最適化可能箇所抽出部２０５が生成することで、実行可能な設計図面内における最適化が可能な変更箇所を絞り込むことを可能にする。

【００２０】

２０６はプラットフォーム情報２０４から、３２ｂｉｔ演算の禁止や、使用メモリ量の制限といった、最適化の際に行なうべき制約を解析する実行環境制約解析部である。実行環境における制約条件の情報である実行環境制約２０８を実行環境制約解析部２０６が抽出することで、速度やメモリ量といったトレードオフ選択を可能にする。

【００２１】

２０９は、最適化可能箇所２０７と実行環境制約２０８とを用いて、プログラムコードやプロジェクトファイルを自動生成する際にどのような最適化が効果的且つ適用可能かを示す情報（最適化予定リスト２１０）を抽出する最適化予定リスト抽出部である。

２１１は、設計図面情報２０３と、プラットフォーム依存情報２０４と、最適化予定リスト２１０とを用いて、プログラムコードを自動生成するプログラムコード生成部である。このプログラムコード生成部２１１によって、設計図面の依存関係や実行対象となるプラットフォームに応じて最適化されたプログラムコード２１３の自動生成が可能である。

２１２は、設計図面情報２０３と、プラットフォーム依存情報２０４と、最適化予定リスト２１０とを用いて、プログラムコードやライブラリを管理するプロジェクトファイル２１４を自動生成するプロジェクトファイル生成部である。なお、プロジェクトファイル２１４は、そのライブラリ内の不要なコードブロックを除去するコンパイルスイッチなども管理するのが好ましい。

【００２２】
２１５は、プログラムコードから参照されるライブラリである。２１６は、プロジェクトファイル２１４が管理するプログラムコード２１３と、ライブラリ２１５とをコンパイル・リンクして、実行可能形式２１７を作成するコンパイラ・リンカである。
【００２３】
次に、図３の処理プログラム３ａに基づいて実行するＣＰＵ１の動作処理フローチャートを参照しながら、本実施形態のプログラムコード生成装置の処理手順の概要の一例を説明する。
まず、ステップＳ３０１において、処理プログラム３ａに基づくプログラムコード生成処理を起動する。
次に、ステップＳ３０２において、実行可能な設計図面を読み込む。
次に、ステップＳ３０３において、プログラムコード生成処理によって生成されるアプリケーションソフトを動作させる際の基盤となるＯＳの種類や環境、設定情報を含むプラットフォーム情報２０４の有無を確認する。この確認の結果、プラットフォーム情報２０４がある場合には、ステップＳ３０４に進み、プラットフォーム情報２０４を読み込む。
次に、ステップＳ３０５において、実行環境制約解析部２０６は、プラットフォーム情報２０４からプログラムの実行環境制約情報の解析をすることで実行環境制約２０８を取得する。
【００２４】
次に、ステップＳ３０６において、最適化可能箇所抽出部２０５は、設計図面情報２０３から図面の依存関係の解析をすることで最適化可能箇所２０７を作成する。最適化可能箇所の抽出に関しては後述する。
前記ステップＳ３０３において、プラットフォーム情報２０４がない場合には、以上のステップＳ３０４～Ｓ３０５を省略してステップＳ３０６に進む。
そして、ステップＳ３０７において、最適化予定リスト作成部２０９を起動して、図面情報の依存関係を保持する最適化可能箇所２０７と実行環境制約２０８とを用いて、適用可能な最適化予定リスト２１０を作成する。
【００２５】
次に、ステップＳ３０８において、プロジェクト管理ファイル作成部２１２を起動して、設計図面情報２０３と、プラットフォーム情報２０４と、最適化予定リスト２１０とを用いて、プロジェクト管理ファイル２１４を作成する。
次に、ステップＳ３０９において、プログラムコード生成部２１１を起動して、設計図面情報２０３と、プラットフォーム情報２０４と最適化予定リスト２１０とを用いて、プログラムコード２１３を生成する。
最後に、ステップＳ３１０において、コード・コンパイラ２１６は、プロジェクトファイル２１４で設定されたプログラムコード２１３とライブラリ２１５とに対してコンパイルとリンクを実施し、実行可能形式２１７を生成する。
【００２６】
図４は、実行可能な設計図面の全体構成の一例を示す図である。なお、このような図は、一般にドメイン図と呼ばれている。
図４において、４０１は実行可能な設計図面を構成するドメイン（Domain）と呼ばれる集合である。４０２は、ドメイン同士の結びつきを表現したものであり、ここでは関連と呼ぶ。
【００２７】
図５は、設計図面を構成する要素の静的な構造の一例を示す図である。なお、このような図は、一般にクラス図と呼ばれている。
図５において、四角形で記述されている図形はクラスと呼ばれるモデル構成要素である。本実施形態では、クラスは、クラス名５０１、属性名５０２、及びメソッド名５０３から構成される。また、クラス間の関連５０４はクラス同士を線で結ぶことで表現することができる。

【００２８】
図６は、クラスの挙動の一例を示す図である。なお、このような図は、一般に状態遷移図と呼ばれている。
図６において、６０１は初期状態、６０２ａ～ｃは状態、６０３は遷移線、６０４は終了状態である。遷移線６０３ａ～ｇにそれぞれ記述されているイベント名は、その遷移が発生するきっかけとなるイベントを表す。
【００２９】
図７は、クラスの挙動を詳細に規定する言語の一例を示す図である。この言語７０１は、状態６０２内やメソッド５０３内に記述される。
【００３０】
図４～図６は、オブジェクト指向において、一般的に利用されている図面である。また、図７に示したのは、アクション言語と呼ばれる言語である。設計図面からコードを自動生成する技術分野における標準技術であるモデル・ドリブン・アーキテクチャでは、図４～図７に示したようにして構成される設計図面からコードを自動生成することが一般的である。
【００３１】
ここで、本実施の形態における情報処理装置によって、生成コードが参照するライブラリ内の利用されていないコードブロックを除去する処理の方法の一例について説明する。
【００３２】
ＣＰＵ１は、プログラム３ａに従って、設計図面内で利用されていない機能を探索する。利用されていない機能があれば、図３のステップＳ３０７において、以下のような最適化可能箇所（コードブロックを除去可能な変更可能箇所）２０７のリストを作成する。なお、ここで例に挙げたのは、未使用のイベントや遅延イベントやイベントキャンセルといった機能の探索である。
【００３３】
```
Function,Event,NotUsed          // イベント未使用
Function,DelayEvent,NotUsed    // 遅延イベント未使用
Function,CancelEvent,NotUsed  // イベントキャンセル未使用
```
【００３４】
イベントや遅延イベントやイベントキャンセルといった機能の利用の有無の探索は、これらの機能を使用する際に必ず記述される構文が設計図面の全て箇所に存在するか否かを判断することによって行われる。例えば、設計図面の全ての箇所に遅延イベント機能を示す構文が存在しなければ、遅延イベント機能が使われていないと判断することで行われる。以下に、前述した機能を利用する際に必ず記述される構文の例を示す。ただし、構文はＥＢＮＦ（拡張バッカス・ナウア記法）で表記した。
【００３５】
・通常イベントを利用する場合に必ず記述される構文の例
```
<event call> ::= "GENERATE " <class name> ":" <event name>
"(" <argument expression> ");"
<arguments expression> ::= <insntace expression> [ "," <insntace expression> ]
例) GENERATE Light:evOn();
```
【００３６】
・遅延イベントを利用する場合に必ず記述される構文の例
```
<delay event call> ::= "GENERATE " <class name> ":" <event name>
             "(" <argument expression> ")"  "AFTER (" <number expression> ");"
例) GENERATE Light:evOn() AFTER(10);
```
【００３７】
・イベントキャンセルを利用する場合に必ず記述される構文の例
```
<cancel event> ::= "CANCEL " <class name> ":" <event name> ";"
例) CANCEL Light:evOn;
```

【００３８】
そして、前述した構文による探索により得られた機能の使用の有無の確認結果から、これらの機能が使用されていないことを表すコンパイルマクロのリストを自動生成する。例えば、遅延イベントが使われていない場合にコンパイルマクロ「NO__DELAY__EVENT」が、設定されるようにすることで、ライブラリ内から遅延イベントに関する全てのコードを除去することが可能になる。
ライブラリ内の、クラス、属性、メソッド、及びメソッド内のコードブロックなど、あらゆるコードブロックが最適化の対象となる。以下に、属性、メソッド、及びメソッド内のコードブロックを除去する例を示す。
【００３９】
・属性を除去する例：

```
class Task [
public:
#ifndef NO_DELAY_EVENT
        Queue* delayQueue;
#endif /* NO_DELAY_EVENT */
        Queue* queue;
        …
];
```

【００４０】
・メソッドの定義を除去する例：

```
#ifndef NO_DELAY_EVENT
void Task::udateDelayQueue()
[
        …
]
#endif /* NO_DELAY_EVENT */
```

【００４１】
・メソッド内のコードブロックを除去する例：

```
void Task::main_loop()[
      Incident* incidnet = NULL;
      while(state != EndOfTask)[
              incident = dispatch();
              if(incident)[
              ] else [
#ifndef NO_DELAY_EVENT
                    updateDelayQueue();
#endif /* NO_DELAY_EVENT */
              ]
        ]
]
```

【００４２】
なお、コンパイルマクロのリストの生成は、例えば、図３のステップＳ３０７とステップＳ３０８との間で行なうようにする。
このように、生成コードが参照するライブラリ内の利用されていないコードブロックを除去するようにすれば、生成コードが参照するライブラリを生成コードに応じて最適化することが可能となる。
【００４３】
（最適化可能箇所抽出部の処理例）
次に、最適化可能箇所抽出部２０５の処理例を示す。

設計図面内における、ドメイン（Domain）、クラス（Class）、状態（State）、イベント（Event）、メソッド（Operation）、属性（Attribute）、遷移線（Transition）の利用状況を探索して、使われていないものがあれば以下のような最適化可能箇所のリストを作成する。

【００４４】

・Domain,<domain name>,NotUsed
・Class,<domain name>.<class name>,NotUsed
・State,<domain name>.<class name>.<state name>,NotUsed
・Event,<domain name>.<class name>.<event name>,NotUsed
・Method,<domain name>.<class name>.<method name>,NotUsed
・Attribute,<domain name>.<class name>.<attribute name>,NotUsed
・Transition,<domain name>.<class name>.<state name>.<state name>,NotUsed
・Insntace,<domain name>.<instance name>,NotUsed
・Insntace,<domain name>.<class name>.<method name>.<instance name>,NotUsed

【００４５】

ここで、以下のＥＢＮＦ（拡張バッカス・ナウア記法）でメソッドやイベントや属性などの呼び出しが記述される設計図面を例にして、先のリストを作成するアルゴリズムの一例を挙げる。

【００４６】

<method call> ::= <instance name> "." <method name> "("
[<arguments expression>] ");"
<event call> ::= "GENERATE " <class name> ":" <event name>
"(" [<arguments expression>] ") TO " <instance name> ";"
<attribute call> ::= <instance name> "." <attribute name>
<arguments expression> ::= <insntace expression> ["," <insntace expression>]
<instance expression> ::= <insntace name> | <method call>

【００４７】

そして、以下の（１）～（３）の処理を実行する。

（１）　設計図面全体から、利用されていないメソッド、イベント、及び属性値を探索する。具体的には、以下の（ａ）～（ｄ）の処理を行なう。

（ａ）　設計図面内の最適化により無効になっていない部分に対して以下の箇所を探索する。

（ｂ）　特定のメソッドの所属するドメイン名<domain name>、所属するクラス<class name>、及び名前<method name>を入力とし、<method call>構文と一致する箇所を探索する。

【００４８】

（ｃ）　特定のイベントの所属するドメイン名<domain name>、所属するクラス名<class name>、及びイベント名<event name>を入力とし、<event call>構文と一致する箇所を探索する。

（ｄ）　特定の属性の所属するドメイン名<domain name>、所属するクラス名、<class name>、及び属性名<attribute name>を入力とし、最適化により無効になっていない部分で<attribute call>構文と一致する箇所を探索する。

【００４９】

ただし、<insntace name>の所属するクラスが<class name>に一致する制約条件と、<insntace name>の所属するドメインが<domain name>に一致という制約条件との元で探索を行なう。

前述の探索を、全てのメソッド、イベント、属性について行い、指定された構文に一致する箇所が無いメソッド、イベント、属性を、最適化可能箇所２０７のリストに登録し、その設計図面情報を無効にする。

【００５０】

（２）　利用されていないイベントから利用されていない遷移線や状態を探索する。
　利用されていないイベントにより遷移する遷移線、他の状態から遷移されることのなくなった状態、及び参照されなくなった属性を最適化可能箇所２０７として登録し、無効にする。
（３）　最適化可能箇所が新たに見つからなくなるまで前記（１）～（２）を繰り返す。
【００５１】
　図６に示すように、Ｌｉｇｈｔクラス１０００には、次の２つの特徴があったとする。すなわち、メソッドgetName()１００３と、イベントevCleanup()１００２は利用されていない。
　name属性１００１は、メソッドgetName()１００３のみで利用される。
　前述した（１）の処理により、メソッドgetName()１００３とイベントevCleanup()１００２が除去される。
　そして、前述した（２）の処理により、name属性１００１、Cleaning状態６０２ｃ、OnからCleaningへの遷移線６０３ａ、OffからCleaningへの遷移線６０３ｄが除去される。その結果、図６に示したＬｉｇｈｔクラスは、図８に示すように変換され、図６に示した状態遷移図は図９に示すように変換される。
【００５２】
　本例の最適化により設計図面の構成要素の数が、遷移線が８から５に、イベントが３から２に、メソッド１から０に、状態が３から２に、属性が１から０に激減する。このように、本例では、設計図面全体から、利用されていないメソッド、イベント、及び属性値を探索し、利用されていないイベントから利用されていない遷移線や状態を探索して、設計図面の構成要素の数を削減するようにしたので、使用不使用といったオブジェクト指向以外の観点からも最適化が可能になる。これにより、メモリ容量の大幅な削減が実現可能になるとともに、実行速度の高速化が可能になる。
【００５３】
　（実行環境制約解析部の処理例）
　次に、実行環境制約解析部２０６の処理例を示す。
　実行環境制約解析部２０６の入力であるプラットフォーム情報２０４の例としては、ソフトウェア構成（使用ＯＳ、ライブラリ、フレームワーク）、ハードウェア構成（ＣＰＵ、メモリなど）、スレッドマッピング、及びプロセスマッピングといった情報が挙げられる。これらの情報は、以下のような形式で表現することが可能である。
【００５４】

```
Platform,CPU,SH-1 // CPUはSH-1
Platform,OS,mItron // OSはμITRON
Platform,RAM,512kByte // RAM使用可能量は512kByteまで
Platform,ROM,128kByte // ROM使用可能量は128kByteまで
Thread,<thread name>,threadid="<thread id>", priority="<piority>" // スレッド定義
Process,<process name>,processid="<process id>", priority="<priority>" // プロセス定義
Domain,<domain name>,ThreadId="<thread id>" // Domainにスレッドマッピング
Class,<domain name>.<class name>,ThreadId="<thread id>"
      // Classにスレッドマッピング
Instance,<domain name>.<instance name>,ThreadId="<thread id>"
      // Instanceにスレッドマッピング
Domain,<domain name>,ProcessId="<process id>"
      // Domainにプロセスマッピング
Class,<domain name>.<class name>, ProcessId="<process id>"
      // Classにプロセスマッピング
Instance,<domain name>.<instance name>, ProcessId="<process id>"
```

// Instanceにプロセスマッピング
【００５５】
実行環境制約解析部２０６で、ＣＰＵ名やＯＳ名から基本ビット長を取得したり、ＲＯＭの使用可能量の指定の有無からＲＯＭが使用可能か否かを判定したり、使用可能なＲＡＭ量やＲＯＭ量をモデル要素の数と比較することでメモリ量を優先して最適化すべきか否かを判定したり、複数プロセスや複数スレッドからアクセスされるか否かを判定したりしてプラットフォーム情報２０４を解析することで、実行環境制約２０８が得られる。このような解析結果は、以下のような形式で保存することができる。
【００５６】
System,OS,mItron // システムで利用されるOSはμITRON
System,BitLength,16bit // CPU名やOS名から基本ビット長を取得
System,ReadOnlyMemoryEnable
// ROM領域の有無からReadOnlyMemoryが利用可能かを判別
System,EnoughMemory
// 十分なメモリ量を確保することが可能
Domain,<domain name>,IsThreadModel
Class,<domain name>.<class name>,IsThreadModel
Instance,<domain name>.<instance name>,IsThreadModel
Instance,<domain name>.<class name>.<method name>.<instance name>,IsThreadModel
Domain,<domain name>,IsProcessModel
Class,<domain name>.<class name>,IsProcessModel
Instance,<domain name>.<instance name>,IsProcessModel
Instance,<domain name>.<class name>.<method name>.<instance name>,IsProcessModel
【００５７】
（最適化予定リスト作成部の処理例）
次に、最適化予定リスト作成部２０９の処理例を示す。
最適化予定リスト作成部２０９の入力である最適化可能箇所２０７と実行環境制約２０８として以下のものが渡されて最適化予定リスト２１０が生成される場合を例に挙げて説明する。
【００５８】
最適化可能箇所２０７としては、
「Instance,<domain.name>.<instance name>,ReadOnly
// リードオンリーの静的なインスタンス
Method,<domain name>.<class name>.<method name>,NotUsed
// 利用されていないメソッド」が渡される。
【００５９】
実行環境制約２０８としては、
「Option,OptimizationType,Speed
// 速度優先の最適化
System,ReadOnlyMemoryEnable
// ROMを使用可能」が渡される。
【００６０】
そうすると、最適化予定リスト２１０として、
「Option,OptimizationType,Speed
// 速度優先の最適化
Method,<domain name>.<class name>.<method name>,NotGenerate
// 指定のメソッドを生成しない」が生成される。
【００６１】
リードオンリーの静的なインスタンスはＲＯＭ３に配置して軽量化することが可能であるが、ＲＯＭ３に配置することで速度が遅くなるため、実行環境制約２０８で指定された

速度優先の最適化というオプションを考慮して、リードオンリーの静的なインスタンスの生成する旨が、最適化予定リスト２１０に登録される。

また、最適化可能箇所２１０に登録されている利用されていないメソッドを生成しないことが可能であり、利用されていないメソッドを生成しないことにより、ＲＯＭ３のサイズを削減することが可能になるため、前記のように、指定のメソッドを生成しない旨が最適化予定リスト２１０に登録される。

【００６２】

以上のように本実施形態では、設計図面内の構成要素である、ドメイン、クラス、状態、アクション間の全ての依存関係を解析して、最適化可能箇所２０７を作成し、最適化可能箇所２０７と実行環境制約２０８とを用いて、適用可能な最適化予定リスト２１０を作成し、最適化予定リスト２１０を用いて、プログラムコード２１３を作成することにより、コンパイラのような局所的な最適化ではなく、包括的に最適化をすることができる。これにより、メモリ使用量の削減や実行速度の高速化を実現することが可能になる。

また、生成コードが参照するライブラリ内の利用されていないコードブロックを除去するようにしたので、生成コードが参照するライブラリを生成コードに応じて最適化することが可能となる。

さらに、設計図面における利用されていないイベントに基づいて、利用されていない遷移線や状態を探索して、設計図面の構成要素の数を削減するようにしたので、使用不使用といったオブジェクト指向以外の観点からも最適化が可能になる。これにより、メモリ容量の大幅な削減が実現可能になるとともに、実行速度のより一層の高速化が可能になる。

【００６３】

（本発明の他の実施形態）

上述した実施形態の機能を実現するべく各種のデバイスを動作させるように、該各種デバイスと接続された装置あるいはシステム内のコンピュータに対し、前記実施形態の機能を実現するためのソフトウェアのプログラムコードを供給し、そのシステムあるいは装置のコンピュータ（ＣＰＵあるいはＭＰＵ）に格納されたプログラムに従って前記各種デバイスを動作させることによって実施したものも、本発明の範疇に含まれる。

【００６４】

また、この場合、前記ソフトウェアのプログラムコード自体が上述した実施形態の機能を実現することになり、そのプログラムコード自体、及びそのプログラムコードをコンピュータに供給するための手段、例えば、かかるプログラムコードを格納した記録媒体は本発明を構成する。かかるプログラムコードを記憶する記録媒体としては、例えばフレキシブルディスク、ハードディスク、光ディスク、光磁気ディスク、ＣＤ－ＲＯＭ、磁気テープ、不揮発性のメモリカード、ＲＯＭ等を用いることができる。

【００６５】

また、コンピュータが供給されたプログラムコードを実行することにより、上述の実施形態の機能が実現されるだけでなく、そのプログラムコードがコンピュータにおいて稼働しているＯＳ（オペレーティングシステム）あるいは他のアプリケーションソフト等と共同して上述の実施形態の機能が実現される場合にもかかるプログラムコードは本発明の実施形態に含まれることは言うまでもない。

【００６６】

さらに、供給されたプログラムコードがコンピュータの機能拡張ボードやコンピュータに接続された機能拡張ユニットに備わるメモリに格納された後、そのプログラムコードの指示に基づいてその機能拡張ボードや機能拡張ユニットに備わるＣＰＵ等が実際の処理の一部または全部を行い、その処理によって上述した実施形態の機能が実現される場合にも本発明に含まれることは言うまでもない。

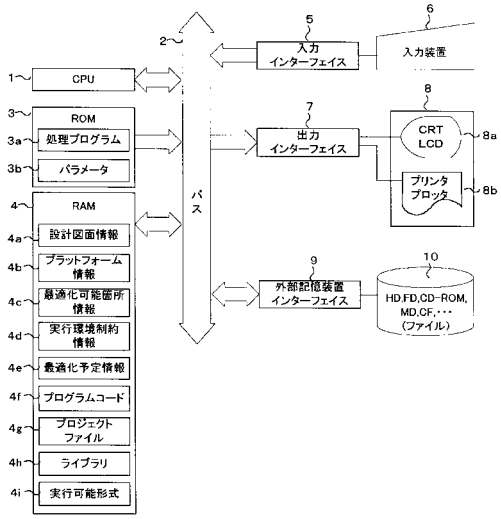
【図面の簡単な説明】

【００６７】

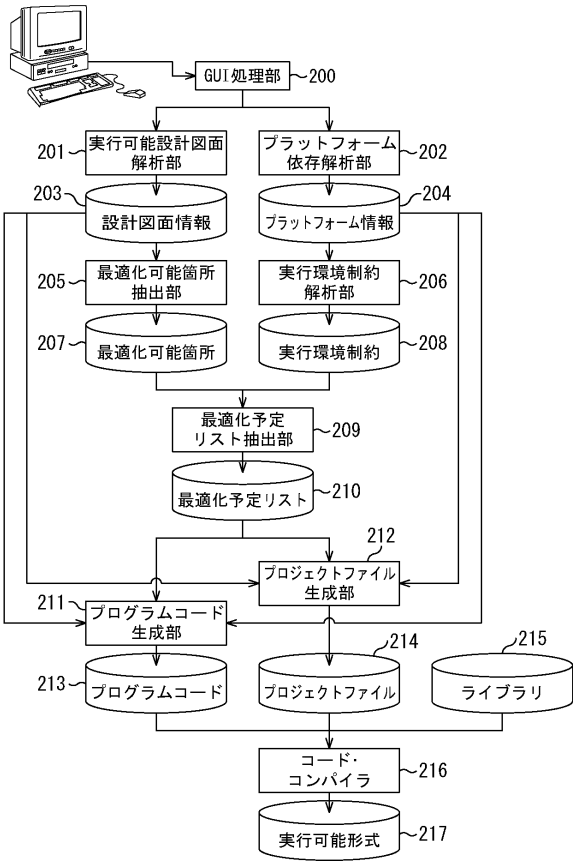
【図１】本発明の実施形態を示し、プログラムコード生成装置の構成の一例を示すブロック図である。

【図２】本発明の実施形態を示し、処理プログラムの構成要件と、その構成要件とデータ格納領域に格納されるデータとの関係との一例を示す図である。
【図３】本発明の実施形態を示し、プログラムコード生成装置の処理手順の概要の一例を説明するフローチャートである。
【図４】本発明の実施形態を示し、実行可能な設計図面の全体構成の一例を示す図である。
【図５】本発明の実施形態を示し、設計図面を構成する要素の静的な構造の一例を示す図である。
【図６】本発明の実施形態を示し、クラスの挙動の一例を示す図である。
【図７】本発明の実施形態を示し、クラスの挙動を詳細に規定する言語の一例を示す図である。
【図８】本発明の実施形態を示し、変換されたＬｉｇｈｔクラスの構造の一例を示す図である。
【図９】本発明の実施形態を示し、変換されたＬｉｇｈｔクラスの保持する状態遷移図である。
【符号の説明】
【００６８】
１　ＣＰＵ
３　ＲＯＭ
４　ＲＡＭ
２００　ＧＵＩ入力処理部
２０１　実行可能設計図面解析部
２０２　プラットフォーム依存解析部
２０３　設計図面情報
２０４　プラットフォーム情報
２０５　最適化可能箇所抽出部
２０６　実行環境制約解析部
２０７　最適化可能箇所
２０８　実行環境制約
２０９　最適化予定リスト作成部
２１０　最適化予定リスト
２１１　プログラムコード生成部
２１２　プロジェクトファイル生成部
２１３　プログラムコード
２１４　プロジェクトファイル
２１５　ライブラリ
２１６　コード・コンパイラ
２１７　実行可能形式

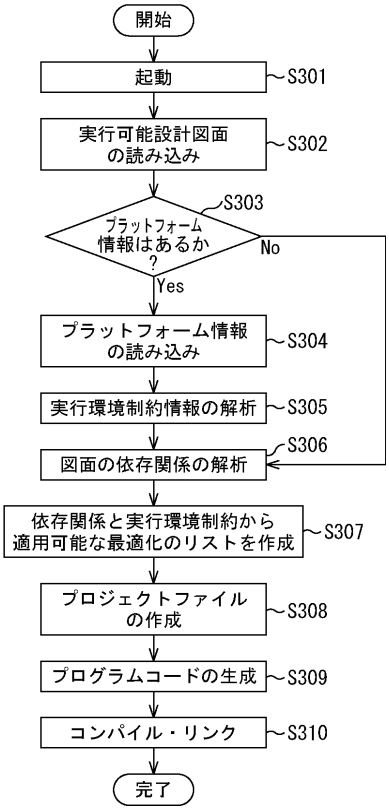
【図１】


CPU
ROM
処理プログラム
パラメータ
RAM
設計図面情報
プラットフォーム情報
最適化可能箇所情報
実行環境制約情報
最適化予定情報
プログラムコード
プロジェクトファイル
ライブラリ
実行可能形式
バス
入力インターフェイス
入力装置
出力インターフェイス
CRT LCD
プリンタプロッタ
外部記憶装置インターフェイス
HD,FD,CD-ROM,MD,CF,・・・(ファイル)

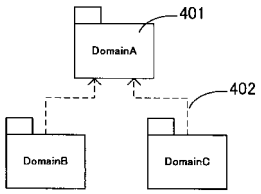
【図２】


GUI処理部
実行可能設計図面解析部
プラットフォーム依存解析部
設計図面情報
プラットフォーム情報
最適化可能箇所抽出部
実行環境制約解析部
最適化可能箇所
実行環境制約
最適化予定リスト抽出部
最適化予定リスト
プロジェクトファイル生成部
プログラムコード生成部
プログラムコード
プロジェクトファイル
ライブラリ
コード・コンパイラ
実行可能形式

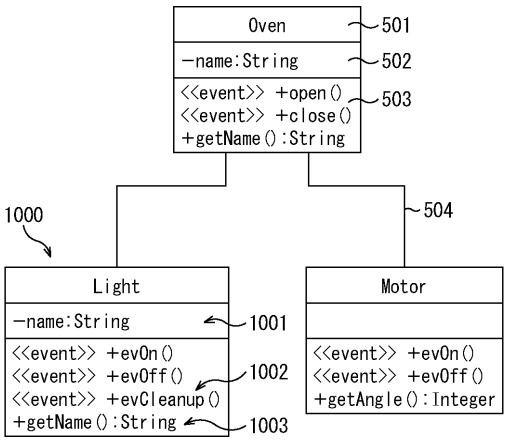
【図３】


開始
起動
実行可能設計図面の読み込み
プラットフォーム情報はあるか？
No
Yes
プラットフォーム情報の読み込み
実行環境制約情報の解析
図面の依存関係の解析
依存関係と実行環境制約から適用可能な最適化のリストを作成
プロジェクトファイルの作成
プログラムコードの生成
コンパイル・リンク
完了

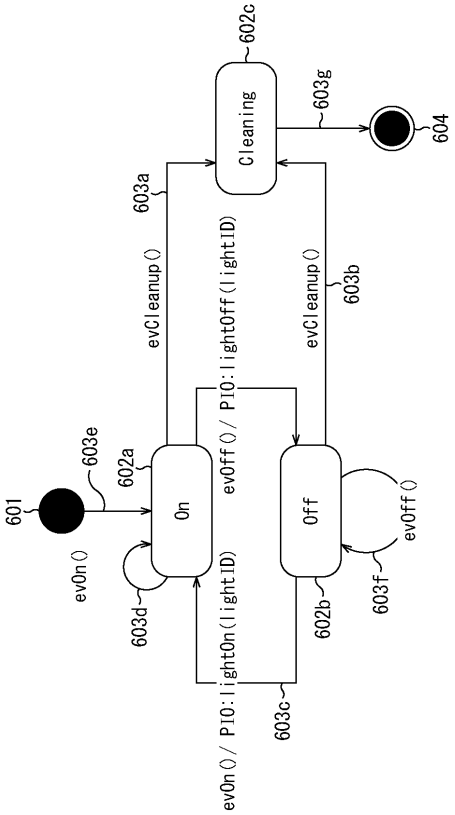
【図４】


DomainA
DomainB
DomainC

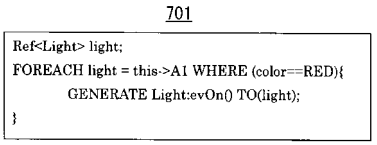
【図５】


Oven
-name:String
<<event>> +open()
<<event>> +close()
+getName():String
Light
-name:String
<<event>> +evOn()
<<event>> +evOff()
<<event>> +evCleanup()
+getName():String
Motor
<<event>> +evOn()
<<event>> +evOff()
+getAngle():Integer

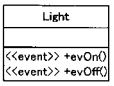
【図６】


602c
Cleaning
603g
604
603a
evCleanup()
evOff() / PIO:lightOff(lightID)
evCleanup()
603b
603e
602a
601
On
Off
evOff()
evOn()
603d
603f
602b
evOn() / PIO:lightOn(lightID)
603c

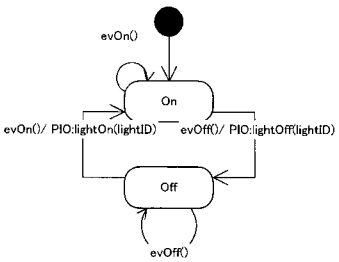
【図７】


701
Ref<Light> light;
FOREACH light = this->A1 WHERE (color==RED){
GENERATE Light:evOn() TO(light);
}

【図８】


Light
<<event>> +evOn()
<<event>> +evOff()

【図９】


evOn()
On
evOn()/ PIO:lightOn(lightID)
evOff()/ PIO:lightOff(lightID)
Off
evOff()

フロントページの続き

(56)参考文献 特開２００２－２０２８８６（ＪＰ，Ａ）
特開２００５－０１８４２５（ＪＰ，Ａ）
特開平１１－２３７９８０（ＪＰ，Ａ）
特開２００３－０７６５４３（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　　９／４４
Ｇ０６Ｆ　　９／４５