(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第3670278号
(P3670278)
(45)発行日　平成17年7月13日(2005.7.13)　(24)登録日　平成17年4月22日(2005.4.22)

(51)Int.Cl.[7]	F I		
G06F 9/06	G06F	9/06	410H
G06F 9/44	G06F	9/44	530P
G06F 9/445	G06F	12/00	520P
G06F 12/00	G06F	9/06	420A

請求項の数 20　(全 16 頁)

(21)出願番号	特願平6-522026
(86)(22)出願日	平成6年1月3日(1994.1.3)
(65)公表番号	特表平8-508594
(43)公表日	平成8年9月10日(1996.9.10)
(86)国際出願番号	PCT/US1994/000053
(87)国際公開番号	WO1994/023360
(87)国際公開日	平成6年10月13日(1994.10.13)
審査請求日	平成11年2月26日(1999.2.26)
審判番号	不服2002-9532(P2002-9532/J1)
審判請求日	平成14年5月27日(2002.5.27)
(31)優先権主張番号	08/042,959
(32)優先日	平成5年4月5日(1993.4.5)
(33)優先権主張国	米国(US)

(73)特許権者　500269956
オブジェクト　テクノロジー　ライセンシング　コーポレイション
アメリカ合衆国　９５０１４－２０８３
カリフォルニア州　クパチーノ　メールステーション　３－オーティーエル　インフィニト　ループ　１
(74)復代理人　100115635
弁理士　窪田　郁大
(74)代理人　100077481
弁理士　谷　義一
(74)代理人　100088915
弁理士　阿部　和夫

最終頁に続く

(54)【発明の名称】順序付けられた探索オブジェクトを使用する共有ライブラリ探索方法およびシステム

(57)【特許請求の範囲】

【請求項１】

幾つかのプログラム間で共有され、名前により参照され、コンピュータの記憶装置内で探索され、および開始のアプリケーションプログラムがロードされた後の要求でロードされるプログラムライブラリを探索するための方法において、

（ａ）ロードされたアプリケーションプログラムを調べて共有ライブラリの名前を決定するステップであって、そこでは、それらの名前が、アプリケーションプログラムおよび共有ライブラリに従属するさらなる共有ライブラリの名前により参照されるステップと、

（ｂ）前記記憶装置内で少なくとも１つの探索オブジェクトを構築するステップと、

（ｃ）当該構築された少なくとも１つの探索オブジェクトを使用して、ステップ（ａ）で決定された名前に対応する共有ライブラリのコピーを探索するステップと、

（ｄ）ステップ（ｃ）で単一の共有ライブラリのコピーが探索されたときには、前記探索オブジェクトを使用して前記単一の共有ライブラリのコピーを返し、

（ｅ）ステップ（ｃ）で複数の共有ライブラリのコピーが探索されたときには、前記構築された少なくとも１つの探索オブジェクトを使用して、当該探索された複数の共有ライブラリのコピーの互換性をチェックし、互換性の従属条件を満足する複数の共有ライブラリのコピーの互換性番号を決定し、および複数の共有ライブラリのコピーの中の最高の互換性番号を有するコピーを返すステップと、

（ｆ）返された共有ライブラリのコピーに対して名前をマッピングするステップと

を備えたことを特徴とする方法。

【請求項２】
請求項１に記載の方法において、ステップ（ｂ）では前記記憶装置の中に、前記少なくとも１つの探索オブジェクトとして、タイプが異なる複数の探索オブジェクトが構築されることを特徴とする方法。
【請求項３】
請求項２に記載の方法において、ステップ（ｂ）ではリスト上に複数の探索オブジェクトが置かれることを特徴とする方法。
【請求項４】
請求項３に記載の方法において、前記探索オブジェクトはラストイン・ファーストアウト（ＬＩＦＯ）の順で前記リスト上に置かれることを特徴とする方法。
【請求項５】
請求項４に記載の方法において、ステップ（ｃ）では、前記探索オブジェクトをＬＩＦＯの順で使用してステップ（ａ）で決定された名前に対応する共有ライブラリのコピーを探索することを特徴とする方法。
【請求項６】
請求項５に記載の方法において、ステップ（ｃ）はさらに、
（ｃ１）　前記リスト上の、最後の探索オブジェクトから始まる１つの探索オブジェクトを使用して、共有ライブラリのコピーを探索するステップと、
（ｃ２）　探索オブジェクトが共有ライブラリのいずれのコピーをも探索しない場合は、ステップ（ｃ１）で使用される探索オブジェクトより前の探索オブジェクトを使用するステップと、
（ｃ３）　ステップ（ｃ１）およびステップ（ｃ２）を、名前に対する共有ライブラリのコピーが探索されるまで、繰り返すステップと
を有することを特徴とする方法。
【請求項７】
請求項６に記載の方法は、さらに、名前に対してマッピングされた共有ライブラリのコピーを順次にロードするステップを有することを特徴とする方法。
【請求項８】
請求項３に記載の方法において、重複するような探索オブジェクトが前記リストにすでに存在する場合には、その探索オブジェクトを前記リストには置かないことを特徴とする方法。
【請求項９】
請求項８に記載の方法において、前記重複するような探索オブジェクトは前記リストのエンドに移動することを特徴とする方法。
【請求項１０】
請求項１に記載の方法において、ステップ（ｅ）は、複数の共有ライブラリのコピーが探索され、および各共有ライブラリのコピーの互換性番号が決定されないときは、複数の共有ライブラリのコピーの中から１つを、そのコピーが現在の探索場所とは別の場所にロードされているか否かの判定に基づいて、選択するステップを有することを特徴とする方法。
【請求項１１】
幾つかのプログラム間で共有され、名前により参照され、コンピュータの記憶装置内に探索され、および開始のアプリケーションプログラムがロードされた後の要求でロードされるプログラムライブラリを探索するための装置において、
ロードされたアプリケーションプログラムを調べて共有ライブラリの名前を決定するローダ手段であって、そこでは、それらの名前が、アプリケーションプログラムおよび共有ライブラリに従属するさらなる共有ライブラリの名前により参照されるローダ手段と、
前記記憶装置内で少なくとも１つの探索オブジェクトを構築する手段と、
前記ローダ手段内にあり、探索オブジェクトを使用して、前記ローダにより決定された名前に対応する共有ライブラリのコピーを前記記憶装置内で探索する探索手段と、

単一の共有ライブラリのコピーが探索されたときには、構築された前記少なくとも１つの探索オブジェクトを使用して前記単一の共有ライブラリのコピーを返す第１の手段と、
前記探索オブジェクト内にあり、複数の共有ライブラリのコピーが探索されたときには、構築された前記少なくとも１つの探索オブジェクトを使用して当該探索された複数の共有ライブラリのコピーの互換性をチェックし、互換性の従属条件を満足する複数の共有ライブラリのコピーの互換性番号を決定し、および複数の共有ライブラリのコピーの中の最高の互換性番号を有するコピーを返す第２の手段と、
返された共有ライブラリのコピーに対して名前をマッピングする手段と
を備えたことを特徴とする装置。

【請求項１２】
請求項１１に記載の装置において、前記記憶装置の中に複数の探索オブジェクトを構築する手段をさらに備えたことを特徴とする装置。

【請求項１３】
請求項１２に記載の装置において、前記探索手段は、リスト上に複数の探索オブジェクトを置く手段を有することを特徴とする装置。

【請求項１４】
請求項１３に記載の装置において、前記複数の探索オブジェクトを置く手段は、前記探索オブジェクトを、ラストイン・ファーストアウト（ＬＩＦＯ）の順で前記リスト上に置くことを特徴とする装置。

【請求項１５】
請求項１４に記載の装置において、前記探索手段は、前記探索オブジェクトをＬＩＦＯの順で使用して前記ローダ手段により決定された名前に対応する共有ライブラリのコピーを探索することを特徴とする装置。

【請求項１６】
請求項１５に記載の装置において、前記探索手段は、前記リスト上の、最後の探索オブジェクトから始まる１つの探索オブジェクトを使用して、共有ライブラリのコピーを順次に探索する手段を有し、探索オブジェクトが共有ライブラリのいずれのコピーをも探索しない場合は、第１の探索オブジェクトより前の第２の探索オブジェクトが使用されることを特徴とする装置。

【請求項１７】
請求項１６に記載の装置において、前記ローダ手段は、各共有ライブラリの名前が共有ライブラリのコピーの返されたロケーションに対してマッピングされた後、名前にマッピングされた共有ライブラリのコピーを順次にロードする手段を有することを特徴とする装置。

【請求項１８】
請求項１３に記載の装置において、前記リスト上に探索オブジェクトを置く手段は、重複するような探索オブジェクトが前記リストにすでに存在する場合には、その探索オブジェクトを前記リストには置かないことを特徴とする装置。

【請求項１９】
請求項１８に記載の装置において、前記リスト上に探索オブジェクトを置く手段は、前記重複するような探索オブジェクトは前記リストのエンドに移動することを特徴とする装置。

【請求項２０】
請求項１１に記載の装置において、前記少なくとも１つの探索オブジェクトは、複数の共有ライブラリのコピーが探索され、および各共有ライブラリのコピーの互換性番号が決定されないときは、複数の共有ライブラリのコピーの中から１つを、そのコピーが現在の探索場所とは別の場所にロードされているか否かの判定に基づいて、選択する手段をさらに有することを特徴とする装置。

【発明の詳細な説明】
関連特許出願の相互参照

本出願は、(i)米国特許出願（発明者Debra L.Orton、David B.Goldsmith、Christopher P.Moeller、およびAndrew G.Heninger、発明の名称「オブジェクト指向フレームワーク・システム」(Object Oriented Framework System)、1992年12月23日出願、被譲渡人Taligent社）および(ii)米国特許出願（発明者Robert Dickinson、発明の名称「オブジェクト指向オペレーティング・システム」(Object Oriented Operating System)、1993年１月22日出願、被譲渡人Taligent）の関連出願であり、これらの開示事項は引用により本明細書の一部を構成するものである。

発明の分野

本発明は、一般的には、共有ライブラリ(shared libraries)のローディングに関し、より具体的には、共有ライブラリ名を探索オブジェクト(search objedt)を使用して物理共用ライブラリのロケーションと対応づける（マッピング）ことに関する。

発明の背景

ライブラリとは、あるロケーションにストアされたデータとコードの集まりである。ライブラリには古いデータ項目（アイテム）が保存されていると共に、新しいデータ項目が受け入れられる。さらに、既存のデータ項目は動的であり、変更を受ける対象になっている。１つのライブラリは、複数のバージョンがコンピュータ・システムに存在することもできる。

ライブラリは共用にすることも可能である。つまり、ライブラリは、複数のアプリケーションやプロセスが同時並行に使用することができる。そのようなライブラリは「共用ライブラリ」(shared library)と呼ばれている。

あるアプリケーションまたはプログラムが開始されるとき、そのプログラムを実行するために必要なすべての共用ライブラリを「ロード」する必要がある。プログラムは、一般に、共用ライブラリの名前を参照する。そのために、プログラムの中で参照された共用ライブラリは、その共用ライブラリが従属する他のすべての共用ライブラリと共に、ローディングが行われる前に、探し出さなければならない。

現在のコンピュータ・システムは、必要な共用ライブラリのロケーションを、プログラムによって参照された名前から効率よく確かめることができない。このようなことが起こるのは、現在のコンピュータ・システムには、共用ライブラリのロケーションを探し出す複雑な探索（サーチ）を行う能力がないためである。さらに、現在のコンピュータ・システムは、共用ライブラリの複数のバージョンやコピーを区別することができない。さらに、現在のコンピュータ・システムは異なる共用ライブラリの互換性属性（compatibility attributes）を正しく埋め合わせることができない。以上のように、本件出願人が承知している従来技術によるシステムは、共用ライブラリをロードするときの複雑さと効率性が不十分である。

発明の概要

本発明は、物理共用ライブラリのロケーションを共用ライブラリ名から効率よく探索し、探し出す方法および装置を提供することによって、従来技術の上記欠点を解消している。

本発明によれば、操作は、開始されたプログラムにとって必要な共用ライブラリの名前を定めることから開始される。そのあと、探索オブジェクトが使用されて、定められた名前の各々に対応する物理ロケーションが探索されて、突き止められる。チーム内に複数の探索オブジェクトを編成することも可能である。ここでチーム(team)とは、アドレス空間(address space)およびそのアドレス空間内で実行される複数の実行スレッド(thread)のことである。このようにすると、探索オブジェクトの集合を、選択した順次順序で使用することにより、物理ロケーションを探し出すことができる。例えば、探索オブジェクトは、それがチームに追加されたときの順序で連続的に使用することができる。

探索オブジェクトが共用ライブラリ名の各々に対応する該当の物理ロケーションに到達すると、共用ライブラリ名の各々は、それぞれに対応する物理ロケーションにマッピングされる。そのあと、到達した物理共用ライブラリ・ロケーションの内容は、プログラムの目的のためにロードすることができる。

【図面の簡単な説明】

本発明の上記および他の目的と利点は、添付図面を参照して以下に詳述する本発明の好適実施例の説明から明らかにする。添付図面において、
第１図は、本発明によるコンピュータ・システムを示すブロック図である。
第２図は、本発明によるロジックの流れを示すフローチャートである。
第３図は、本発明による探索オブジェクトのクラス継承階層を示す図である。
第４図は、本発明に従って探索オブジェクトがたどっていく、名前付き共用ライブラリの探索を示すフローチャートである。
第５図は、本発明による例に従って、異なる互換性番号をもつ共用ライブラリの例を示す図である。

発明の好適実施例の詳細説明

コンピュータ・システム

第１図は代表的なハードウェア環境を示したもので、本発明によるワークステーション40の適切なハードウェア構成を示している。ワークステーション40は、従来のマイクロプロセッサのような中央演算処理ユニット10と、システム・バス12を介して相互接続された複数の他のユニットとを装備している。第１図に示す図示のワークステーション40は、ランダム・アクセス・メモリ(RAM)14と、リード・オンリ・メモリ(ROM)16と、ディスク・ユニットなどの周辺デバイスをバス12に接続するための入出力(I/O)アダプタ18と、キーボード24、マウス26、スピーカ28、マイクロホン32、および／またはタッチ・スクリーン・デバイス（図示せず）などの他のユーザ・インタフェース・デバイスをバス12に接続するためのユーザ・インタフェース・アダプタ22とを備えている。ワークステーション40は、ワークステーション40をデータ処理ネットワーク30に接続するための通信アダプタ34と、バス12を表示デバイス38に接続するための表示アダプタ36とを装備することも可能である。本発明は、好ましくは、「オブジェクト指向フレームワーク・システム」の名称で1992年12月23日に出願された米国特許出願および「オブジェクト指向オペレーティング・システム」の名称で1993年１月22日に出願された米国特許出願に記載されているオブジェクト指向オペレーティング・システムの下で実施される。両米国特許出願はTaligent社に譲渡されている。

本発明のロジック・オペレーション

本発明は、プログラムの実行時にコンピュータ・システムと併用されて、共用ライブラリをロードし、共用ライブラリを相互にリンクするためのフレームワークを備えている。かかるフレームワークは、プログラムによって参照された共用ライブラリ名が、そのプログラムの開始時に従属している共用ライブラリとリンクされるときの仕方を提供している。すべての共用ライブラリは指定された名前によって識別され、指定された名前空間内に置かれている。共用ライブラリは任意の数の他の共用ライブラリに従属し、必要ならば、それらの共用ライブラリとリンクされている。特定の共用ライブラリに対応する共用ライブラリ・ロード・モジュールは、(i)特定共用ライブラリの指定された名前と、(ii)その特定共用ライブラリが従属する共用ライブラリの指定された名前を格納する。

ある特定共用ライブラリがプログラムに続いてロードされるとき、その特定共用ライブラリが従属するすべての共用ライブラリもロードされなければならない。従って、プログラムに続いてロードされる共用ライブラリの名前を確かめるためには、共用ライブラリ・ロード・モジュールの助けが必要になる。共用ライブラリの名前空間はグローバルであるので、該当する共用ライブラリが一緒にリンクされるためには、各共用ライブラリに異なる名前をつけなければならない。本発明は、各共用ライブラリが一意的(unique)な名前をもっていることを前提としている。

プログラムが開始と同時に実行されるためには、共用ライブラリは他の共用ライブラリとリンクされていなければならない。このリンキング・プロセス(linking process)は、該当の共用ライブラリがロードされると行われる。

ローダと呼ばれるメカニズムは、ロードする必要のある共用ライブラリの名前を最初に決める。そのあと、定められた共用ライブラリ名に対応する物理共用ライブラリを、ローダに知らせなければならない。物理共用ライブラリのロケーションには、コンピュータ・デ

ィスク、RAM(「ランダム・アクセス・メモリ」)およびROM(「リードオンリ・メモリ」)が置かれているのが代表的である。しかし、ローダはライブラリがどこに置かれているかを定めない。その代わりに、ローダは探索オブジェクトを使用して共用ライブラリのロケーションを決定する。探索オブジェクトはライブラリ名を物理共用ライブラリ・ロケーションに対応づける（マッピングする）。結果として得たマッピング情報を使用して、ローダは、該当の共用ライブラリをロードすることが可能になる。

第２図は、ある共用ライブラリを別の共用ライブラリとリンクするステップを記載したフローチャートを示す図である。プログラムを開始するために、通常は、共用ライブラリをロードし、相互にリンクする必要がある。これは、ボックス202に示されている。そのあと、開始されたプログラムに必要な共用ライブラリの名前の決定が行われる。これは、ボックス204に示されている。プログラムに必要な共用ライブラリ名には、プログラムによって参照される名前の他に、開始されたプログラムまたは参照された共用ライブラリが従属する共用ライブラリの名前も含まれる。これらの名前は前述した共用ライブラリ・ロード・モジュールから確かめられる。

共用ライブラリの名前が確かめられると、次に、探索オブジェクトが使用されて、その名前に対応するそれぞれの物理共用ライブラリ・ロケーションが突き止められる。これは、ボックス206に示されている。クラスTLibrarySearcherは、以下で詳しく説明するように、使用された探索オブジェクトのインタフェースを定義している派生クラス(derived class)である。共用ライブラリの確かめられた名前が与えられているとき、探索オブジェクトは、その探索オブジェクトの探索命令(search instructions)をたどっていくことにより、物理共用ライブラリに到達する。

対応する物理共用ライブラリに到達すると、ボックス208に示すように、マッピングが行われる。マッピングは共用ライブラリ名を、到達した共用ライブラリ・ロケーションに対応づけることである。そのあと、到達した物理共用ライブラリはロードされ、相互にリンクされる。これは、それぞれ参照符号210と212で示されている。

探索オブジェクト

オブジェクト指向(object-orientation)とは、複数のオブジェクトからなるシステム手法である。各オブジェクトは、通常、実世界のエンティティ（実体）に対応しているが、いくつかのオペレーションと、そのオペレーションのステート（状態）を覚えているステート（状態）を提供する。例えば、人物オブジェクト(person object)は年齢、アドレス、電話番号、職業などの、ある人物に対応する情報を所有している。オペレーションは、この情報をアクセスし、あるいは影響を与えるために作成される。さらに、オペレーションは、ウォーキング(walking)のような、どの内部情報にも影響を与えない作用(behavior)を実行するために作成することもできる。

クラスとは、新しいオブジェクトの作成を可能にする定義、テンプレートまたはモールド(mold)である。従って、クラスは、複数のオブジェクトの共通特性を記述したものである。あるクラスを含むオブジェクトはこのテンプレートを共通にもっている。クラスの特性は、サブクラス(subclass)と呼ばれる別のクラスに継承させることができる。従って、サブクラスはクラスの情報構造と作用を継承する。そのあと、別の情報や作用をサブクラスに追加することが可能である。

継承階層において、あるクラスの下位に置かれたクラスは、クラスの子孫と呼ばれる。上位に置かれたクラスは祖先と呼ばれる。子孫とサブクラスの用語は同じ意味で使用されている。

第３図は、探索オブジェクト・クラスの階層を示す図である。クラスMCollectible 300は、すべての探索オブジェクトの基底クラス(base class)を表している。つまり、クラスMCollectible 300は、他のすべてのクラスがそこから子孫として作成されるクラスである。クラスMCollectible 300の主要子孫は２つあり、それは、クラスTLibrarySearcher 302とクラスTTeamHandle 308である。クラスTLibrarySearcher 302は別の子孫を２つもっている。それは、それぞれ参照符号304と306で示すように、クラスTStandardLibrarySearcherとクラスTWorkspaceSearcherである。

クラスTStandardLibrarySearcher 304は、単一ディレクトリ内の物理共用ライブラリを探索するデフォルトの探索オブジェクトを定義している。探索する必要があるディレクトリは２つ以上であるのが普通であるので、クラスTStandardLibrarySearcher 304は、通常、システム内のすべての共用ライブラリを探し出すことができない。さらに、クラスTStandardLibrarySearcher 304のオブジェクトは、ライブラリの複数のコピーに遭遇したときの互換性を考慮しない。

従って、クラスTStandardLibrarySearcher 304に従って作成された探索オブジェクトに代わって、物理共用ライブラリを探すためのクラスTWorkspaceSearcher 306が用意されている。クラスTWorkspaceSearcher306により、ワークステーションから探索オブジェクトを作成することができる。クラスTWorkspaceSearcher 306によって定義された探索オブジェクトは、２つの以上のディレクトリ内を探索できると共に、必要時に互換性を考慮することができる。

クラスTLibrarySearcher 302から派生したオブジェクトは任意的に複雑な探索を実現することができる。例えば、この種のオブジェクトは、(i)ファイル属性を調べ、(ii)ライブラリのディレクトリまたはデータベースを維持し、(iii)ライブラリ導入プロシージャとやりとりすることができる。TLibrarySearcher302サブクラスである、クラスTStandardLibrarySearcher 304とクラスTWorkspaceSearcher 306は、共用ライブラリのあるバージョンがすでに使用中であるかどうかをシステムに確かめるために作成されることもある。

第４図は、共用ライブラリを探索するときのロジックの流れを示すフローチャートである。初めに、探索オブジェクトが共用ライブラリの名前と一緒に提示される。提示されたこの名前付き共用ライブラリから、共用ライブラリのロケーションの探索が行われる。これは、参照符号400で示されている。

名前付きライブラリが探索されると、(i)共用ライブラリに複数のコピーがあること、(ii)共用ライブラリ名に対応するロケーションがないこと、あるいは(iii)共用ライブラリに１つのコピーがあることが明らかになる。これらの結果は、それぞれ参照符号402,410、および420を付けたボックスに示されている。

共用ライブラリが見つからないと、ボックス412に示すように、探索オブジェクトから“NIL”が返される。共用ライブラリのコピーが１つだけ見つかったときは、ボックス422に示すように、その共用ライブラリがローダに返される。

しかし、共用ライブラリのコピーが複数見つかったときは、コピーが返される前に別の分析が必要になる。この別の分析は、まず、共用ライブラリ・コピーの互換性番号が確認できるかどうかを判断することから始まる。これはボックス404に示されている。共用ライブラリ・コピーの互換性番号が確認できれば、最高互換性番号をもつ共用ライブラリ・コピーが返される。これは、ボックス406に示されている。

共用ライブラリ・コピーの互換性番号が確認できなければ、別の選択方法が必要になる。この別の選択方法には、共用ライブラリのコピーで、現在別の場所に置かれているコピーがあるかどうかの判断を含めることができる。最終的には、この別の選択方法からは、ボックス416に示すように、共用ライブラリの選択されたコピーが返されるはずである。

ボックス406,416,422に示すように、共用ライブラリの返されたコピーは、共用ライブラリ名と対応づけられる（マッピング）。これは、ボックス430に示されている。そのあと、マップ情報がローダに知らされる。

互換性は、ファイルである共用ライブラリのファイル・システム属性である。互換性は互換性番号で示され、最高互換性番号をもつ共用ライブリーのコピーが他の従属共用ライブラリと互換性をもつ可能性がある。従って、探索オブジェクトからは、最高互換性番号をもつ共用ライブラリのコピーが返される。共用ライブラリがファイルであるとき、探索オブジェクトが常にそうすることができるのは、ファイルが常に互換性番号へのアクセスを許しているからである。しかるに、ファイルでない共用ライブラリは、コピーの互換性番号をアクセスすることを認められていない。そのような場合は、探索オブジェクトは最高度の互換性をもつコピーを確かめることができない。

共用ライブラリは、２つの条件が満足されると連続的にロードされる。第１は、最高互換

性番号をもつコピーが探索オブジェクトから返されたときである。第２は、探索オブジェクトが探索した個所のロケーションの間で、一貫性のある共用ライブラリの集合が存在し、それらがロードされるときである。

第５図は、異なる互換性番号をもつ３つのライブラリの例を示す図である。Ａ，ＢおよびＣと名づけた３つのライブラリがロードされることを想定している。互換性番号１をもつライブラリＡ(参照符号502で示すように、"library(A,1)")は、互換性番号２をもつライブラリＢ（参照符号504で示すように、"library(B,2)"）と互換性番号２をもつライブラリＣ（参照符号506で示すように、"library(C,2)"）に従属している。互換性番号３をもつライブラリＢ（参照符号510で示すように、"library(B,3)"）は、互換性番号３をもつライブラリＣ（参照符号508で示すように、"library(C,3)"）に従属している。ライブラリ(B,2)504はライブラリ(C,2)506に従属している。しかし、ライブラリ(C,2)506とライブラリ(C,3)508は他のどのライブラリにも従属していない。

ローダがライブラリ(A,1)502のロードを試みるとき、ライブラリ(A,1)502が従属するライブラリのコピーもロードする必要がある。ロードされるコピーは従属共用ライブラリの互換性番号と等しいか、それより大の互換性番号をもっていなければならない。従って、ローダがロードできるのは、(i)ライブラリ(B,2)504またはライブラリ(B,3)510および(ii)ライブラリ(C,2)506とライブラリ(C,3)508である。しかし、ライブラリ(B,3)510はライブラリ(C,3)508と一緒にロードされなければならない。ライブラリ(B,3)510の従属を満たすには、ライブラリ(C,2)506では不十分である。

探索オブジェクトまたは複数の探索オブジェクトはライブラリ(B,2)504、ライブラリ(B,3)510、ライブラリ(C,2)506、およびライブラリ(C,3)508を探し出す。最大互換性番号をもつコピーが探索オブジェクトから返されると、ライブラリ(A,1)502、ライブラリ(B,3)510、およびライブラリ(C,3)508が返される。これらの３ライブラリはすべての従属条件が満たされているので、ロードされる。

前記の３ライブラリが探索オブジェクトから返されなかったときは、ローディングはできないであろう。例えば、探索オブジェクトからライブラリ(A,1)502、ライブラリ(B,3)510、およびライブラリ(C,2)506が返されることがある。そのような場合は、ライブラリ(B,3)510がライブラリ(C,3)508に従属することが満たされないので、これらのライブラリはロードされない。このようなことが起こるのを避けるには、探索オブジェクトは最高互換性番号をもつライブラリのコピーを返さなければならない。もちろん、これが行えるのは、探索オブジェクトが共用ライブラリの互換性番号をアクセスできる場合だけである。

探索オブジェクトのチーム属性

探索オブジェクトはチーム内に置かれる。各チームは１つまたは２つ以上の探索オブジェクトのリストをもち、このリストは、そのチームに関連する共用ライブラリをロードするときローダによって調べられる。チーム内に置いておくことができる探索オブジェクトの数には制限がない。新しいチームが作られるときは、クラスTTeamHandle内に作られているコンストラクタ・オブジェクト(constructor object)内に探索オブジェクトの集合を渡すことができる（第３図に参照符号308で示されている）。そのあと、探索オブジェクトの集合は新しいチームに加えられる。

クラスTLibrarySearcher 302から派生した探索オブジェクトは２種類があり、チーム内に置かれている。第１は、作成されるすべてのチームに追加される探索オブジェクトであり、これは「システム探索オブジェクト」と呼ばれる。第２は、特定のチームにだけ追加するために作成できる探索オブジェクトである。これらはチーム特定オブジェクト(team specfic object)と呼ばれる。あるチームがアクセスされるとき、チーム特定探索オブジェクトとシステム探索オブジェクトの両方がストリーム化されて、そのチームのアドレス空間に入れられる。従って、TLibrarySearcher 302のサブクラスTStandardLibrarySearcher 304とTWorkspaceSearcher 306はストリーミング演算子(streaming operators)をもっていないければならない。

ローダはチーム内の探索オブジェクトの集合を使用して、ライブラリ名をライブラリ・ロケーションまたはモジュールに対応づける（マッピング）。そのあと、共用ライブラリは

チームのアドレス空間内に置かれる。ローダは、好ましくは、探索オブジェクトを「LIFO順」（後入れ先出し）に使用する。つまり、探索オブジェクトは、チームに追加されたときの順序で使用される。従って、最も最近に追加された探索オブジェクトが最初に使用される。探索オブジェクトは共用ライブラリが見つかるまで、LIFO順に連続的に使用される。探索オブジェクトのどれも共用ライブラリを見つけることができないと、ローダは先に進むことができないので例外を引き起こすことになる。例外に応答して、チームにアクセスしてから、探索オブジェクトをチームに追加することができる。しかし、あとで追加された探索オブジェクトが見つけることができるのは、探索オブジェクトの追加のあとでチームのアドレス空間に動的にロードされる共用ライブラリだけである。

最近に追加された探索オブジェクトは既存の探索オブジェクトをオーバライド(override)するものと想定されているので、ローダは探索オブジェクトをLIFO順に使用することが好ましい。従って、チーム探索オブジェクトの集合は、常に、クラスTStandardLibrarySearcher 304によって定義されるようなデフォルトの探索オブジェクトの前に使用される。例えば、共用ライブラリＸは、２つのバージョンがシステムに存在することがある。デフォルト探索オブジェクトはライブラリＸをライブラリＸの第１バージョンにマップする。他方、クラスTWorkspaceSearcher 306によって定義された探索オブジェクトはライブラリＸの第２バージョンにマップする。ワークスペース定義探索オブジェクトがデフォルト探索オブジェクトの前に使用されると、ライブラリＸはライブラリＸの第２バージョンにマップされる。従って、ワークスペース定義探索オブジェクトはシステム探索オブジェクトの前に調べられるので、ライブラリＸの正しいバージョンがチームのアドレス空間にロードされることになる。

別の方法として、一時的チーム探索オブジェクトを指定しても同じ効果が得られる。一時的探索オブジェクトは、“TLibrary::LoadLibrary for LibraryＸ”へのコールの中で指定できる。従って、チームはコールが存続している間一時的探索オブジェクトを使用するので、ライブラリＸの正しいバージョンがロードされることになる。一時的探索オブジェクトは、チームが開始されたあと使用すると特に便利である。その理由は、一時的探索オブジェクトはチーム探索オブジェクトの集合に追加されないためである。これとは逆に、探索オブジェクトをチームに追加すると、他の共用ライブラリに対するチームの使用効率に好ましくない影響を与えることになる。そのために、一時的ライブラリ探索オブジェクトは、ライブラリを稼働中のチームに動的にロードするとき使用するのが望ましい。

システム探索オブジェクトはすべてのチームと関係があるので、これがローダによって使用されるのは、すべてのチーム特定探索オブジェクトが使用されたあとだけである。従って、探索オブジェクトは次のような順序で使用される。すなわち、(i)チーム特定探索オブジェクト、(ii)システム探索オブジェクト、(iii)デフォルト探索オブジェクト。さらに、本発明ではシステム・オブジェクトはLIFO順に使用されるので、追加されたシステム探索オブジェクトは、それがシステムに追加されたあと作成された共用ライブラリのローディングだけに影響する。

探索オブジェクトは、システム上のすべてのチームに対して適用するためにも、特定のチームに対して適用するためにも追加することが可能である。すべてのチームに対して適用するためにシステム探索オブジェクトが追加されると(“TLibrarySearcher::AddSystemSearcher”を呼び出すことにより)、そのシステム・オブジェクトはそのあとで作成されたすべてのチームに適用される。これに対して、あるチームに対して適用するために探索オブジェクトが追加されると(“TTeamHandle::AddLibrarySearcher”を呼び出すことにより)、新しく作成された探索オブジェクトは、そのチームによるライブラリ探索だけに適用される。

探索オブジェクトは、あるチームにいったん追加されたら、除去できないようにすることが好ましい。これは追加された探索オブジェクトがチーム特定探索オブジェクトであるか、システム探索オブジェクトであるかに関係ない。このような制約を設けたのは、探索オブジェクトがチームのすべてのタスクによって共用されるグローバル資源であるためである。あるタスクに対してチーム探索オブジェクトの除去を認めると、その探索オブジェク

トに従属する別のタスクに好ましくない影響を与えることになる。例えば、同じチーム上の２つの従属タスク（「タスク１」と「タスク２」）が同じ追加探索オブジェクトをもっているとする。この場合、探索オブジェクトがタスク１から除去されたとしても、タスク２はその探索オブジェクトがまだ残っているものと思っている。そのために、両方のタスクは探索オブジェクトの同じリストを保持している必要がある。一時的探索オブジェクトは、探索オブジェクトがある特定タスクに対してだけ使用されるときに利用するのが好ましく、そうすれば、その探索オブジェクトをチームに追加しないで済むことになる。

また、本発明によれば、あるチームに追加しようとしている探索オブジェクトがすでにそのチームに存在するかどうかを確かめるための分析も行われる。探索オブジェクトと重複するものがすでに置かれているチームに探索オブジェクトが追加されると、探索コピーの二番目のコピーの追加は行われない。しかし、チーム内にすでに置かれている重複探索オブジェクトはチーム・ライブラリ探索オブジェクトのリストの最後に移される。このようにしたのは、チームの探索オブジェクト・リストの最後にある探索オブジェクトがチームによって最初に使用されるためである。

本発明によれば、重複探索オブジェクトを見つけるために探索オブジェクトのタイプが分析される。例えば、C++プログラミング言語では、"Is Equal"メソッドを実行することにより比較が行われる。従って、すべてのTLibrarySearcher 302サブクラスは、IsEqual"メソッドまたは同等のメソッドをオーバライド(override)しなければならない。

TLibrarySearcher 302のサブクラスは、共用ライブラリに入れておくことができる。しかし、ローダがある共用ライブラリを見つけるためには、その所定のライブラリには、それを見つけるために必要なTLibrarySearcher 302サブクラスが入っていてはならない。従って、TLibrarySearcher 302サブクラスは、TLibrarySearcher 302サブクラスに探索させる共用ライブラリの外部のロケーションに置いておかなければならない。言い換えれば、TLibrarySearcher 302サブクラスを収めている共用ライブラリは、それを見つけるためにライブラリに置かれているTLibrarySearcher 302サブクラスに頼らなくても、見つけることができるロケーションの場所に置いておかなければならない。

TLibrarySearcher 302サブクラスを収めている所定共用ライブラリを、ローダに確実に見つけさせる１つの方法は、デフォルトのシステム探索オブジェクトがそれを見つけることができる場所に、その所定の共用ライブラリを置いておくことである。クラスTStandardLibrarySearcher 304は、デフォルト探索オブジェクトを定義し、すべてのチームと関連づけられているので、所定の共用ライブラリを常に探し出すことができる。従って、新しく作成されたチームは、クラスTWorkspaceSearcher 306によって定義されたワークスペース探索オブジェクトがチームに追加されてから初めて、デフォルト・システム探索オブジェクトに頼ることができる。ある探索オブジェクトがあるチームに追加されたときは、そのチームはその探索オブジェクトを収めている共用ライブラリをロードしなければならない。これはチームにすでに追加されているシステム・ライブラリ探索オブジェクトと、もしあれば、チーム特定探索オブジェクトだけを使用して行われる。

好ましくは、共用ライブラリの１つのコピーだけがチームのアドレス空間にロードされる。従って、ローダは、単一チームを扱っているとき、ある共用ライブラリを２回以上見つけたときは、現在ロードされている共用ライブラリが正しいものと想定する。従って、ローダは重複するライブラリを探すために探索オブジェクトを使用することがない。チームの探索オブジェクトの集合が修正され、チームのアドレス空間にすでに置かれているライブラリをロードするために“LoadLibrary”が呼び出されると、修正されたチームはライブラリの異なるバージョンを返すことになる。従って、LoadLibrary を呼び出しても、ライブラリの旧バージョンはチームのアドレス空間に残っているので、所望とする効果は得られない。

同じワークステーションに置かれている異なるチームは、同じ共用ライブラリの異なるコピーを使用することができる。しかし、同じ共用ライブラリの異なるコピーを使用する複数のチームは、オブジェクト・クラスが共用ライブラリ内に定義されていても、共用メモリによって別のチームのオブジェクトを共用することはできない。むしろ、所定のチーム

が使用できるオブジェクトは、所定のチームに関連するものだけに制限されている。共用ライブラリの異なるバージョンを使用するチームはチーム同士でオブジェクトを共用できないが、そのクラスが共用ライブラリ内に定義されているオブジェクトをストリーム化して相互にやりすることは許される。

発明者が開発した本発明の実施例は、本明細書の付録に記載されている。これは、本発明の詳細ロジックを理解しやすくするために示されたものである。実施例の各部分の重要要素も、詳しく解説されている。

付録：本発明の好適実施例

ソース・コード：

```
//Class TLibrarySearcher derived from base class MCollectible
class TLibrarySearcher : public MCollectible
        {
        ....
 public                                  ~TLibrarySearcher();
        virtual TMappableDataContainer *CreateContainerAndFindLibrary(
                                                        constTText&LibraryName,
                                                        constCompatibilityNumber
                                                    desiredRelease)
                                                        = 0;
        virtual Boolean                 IsEqual(const MCollectible *) const = 0;
        //the following public methods are for use only by the workspace.


          static void                   AddSystemLibrarySearcher(TLibrarySearcher &);
          static TIterator   *CreateSystemLibrarySearcherIterator();

   protected:
                                        TLibrarySearcher();
                                        TLibrarySearcher(const TLibrary Searcher&);
          TLibrarySearcher&  operator= (const TLibrary Searcher &);
          virtual TStream&   operator>>=(TStream& toWhere) const;
          virtual TStream&    operator<<=(TStream& fromWhere);
          }
```

```
//Subclass library searcher used by the run time during system boot.
//Not intended for subclassing.
classTStandardLibrarySearcher : public TLibrarySearcher
        {
        ....
 public:
        virtual                                 ~TStandardLibrarySearcher();
                                                TStandardLibrarySearcher();
                                                TStandardLibrarySearcher(const
                                                TStandardLibrarySearcher &);
        virtual TMappableDataContainer *CreateContainerAndFindLibrary(const
                                                TText &libraryName, const
                                                CompatabilityNumber desiredRelease);

        virtual Boolean                         IsEqual(const TLibrary Searcher *) const;
        TStandardLibrarySearcher & operator=(const TStandardLibrarySearcher     &);
        virtual TStream &  operator>>=(TStream& toWhere) const;
        virtual TStream &  operator<<=(TStream& fromWhere);
        }


//Excerpt of TTeamHandle.  These are the methods of TTeamHandle
//that use TLibrary Searcher
class TTeamHandle : public MCollectible
        {
        ....
                                                TTeamHandle(const TTaskProgram
&whatToRun,
                                                 const TOrderedCollection & library
                                                Searchers);
        virtual void       AddLibrarySearcher(TLibrarySearcher &);
        virtual TIterator  *CreateLibrarySearcherIterator();
        ....
        }
```

ソース・コードの注釈：

//Class TLibrarySearcher derived from base class MCollectible (//基底クラスMCollectibleから派生されたクラスTLibrarySearcher)

～TLibrarySearcher():デストラクタ。

TMappableDataContainer*CreateContainerAndFindLibrary(constTText&libraryName,constCompatibilityNumberdesireRelease):特定の名前をもつライブラリを探索する。そのライブラリのTMappableDataContainerを返す。呼出し側はTMappableDataContainerのための記憶域を所有している。ライブラリ・サーチャは共用ライブラリの複数のコピーを見つけたときは、どのコピーを返すべきかを決定しなければならない。実行中のチームにより現在ロードされたのがどのライブラリであるのかを、その決定には考慮する必要がある。また、共用ライブラリの必要とする互換性番号も考慮する必要がある。サーチャが共用ライブラリの互換性番号をアクセスできる場合は（互換性番号はファイルである共用ライブラリの特性である）、返すことを考慮している共用ライブラリの互換性番号が、必要とする互換性番号より大か、あるいは等しいかをチェックする必要がある。ライブラリが見つからなければ、NILが返される。このメソッドは、すべてのTLibrarySearcherサブクラスによって無視しなければならない。

Boolean IsEqual(constMCollectible*):２つのライブラリ・サーチャが同等のものであれば、TRUEが返され、そうでなければ、FALSEが返される。

以下のメソッドは、ワークスペースだけがコールすることが可能である。

static AddSystemLibrarySearcher(const TLibrarySearcher &):その作成時にチームによって自動的に継承されるライブラリ・サーチャのグローバル・リストにライブラリ・サーチャを追加する。この関数は、コールされたあとに作成されたチームにだけ作用する。既存のチームには作用しない。
static TIterator*CreateSystemLibrarySearcherIterator():システム・ライブラリ・サーチャのためのイタレータ(iterator)を生成する。このイタレータは、TLibrarySearcherサブクラス・オブジェクトを指しているポインタを返す。呼出し側はイタレータのための記憶域を所有している。イタレータは返すTLibrarySearcherのための記憶域を所有している。イタレータはシステム・ライブラリ・サーチャを、適用される順に返す。現在のスレッドがシステム・ライブラリ・サーチャを繰り返している途中でシステム・ライブラリ・サーチャが別のスレッドによって追加されると、kIteratorOutOfSync例外が引き起こされる。
//Subclass library searcher used by the run time during system boot.
//Not intended for subclassing.
(//システム・ブート期間の実行時に使用されるサブクラス・ライブラリ・サーチャ。
//サブクラス化を目的としてない。)
TStandardLibrarySearcher():コンストラクタ。
～TStandardLibrarySearcher():デストラクタ。
TMappableDataContainer*CreateContainerAndFindLibrary(constTText&libraryName,const CompabilityNumberdesiredRelease):ディスク上の「共用ライブラリ」フォルダに入っている特定名をもつライブラリを探索し、そのライブラリのTMappableDataContainerを返す。ライブラリが見つからなければ、NILが返される。このライブラリ・サーチャはその決定のとき要求された互換性番号を考慮しない。呼出し側はTMappableDataContainerのための記憶域を所有している。
BooleanIsEqual(constMCollectible*):２つの標準ライブラリ・サーチャが同等のものであればTRUEを返し、そうでなければELSEを返す。
特定のシステム環境における好適実施例を参照して本発明を説明してきたが、本発明は、請求の範囲に明確化された本発明の精神と範囲内において、修正を加えることにより他の異なるハードウェアおよびソフトウェア環境において実施することが可能であることはもちろんである。

【図１】

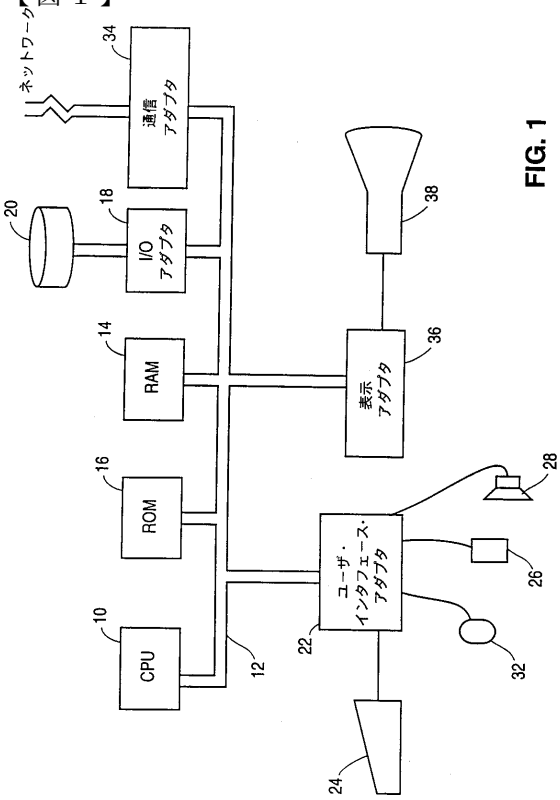

FIG. 1

【図２】

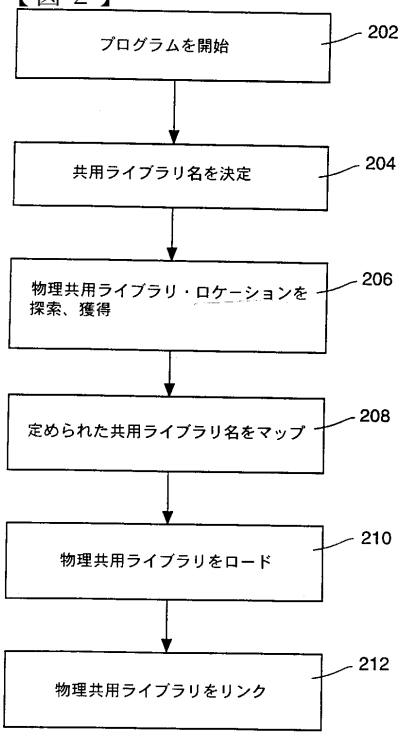

FIG. 2

【図３】

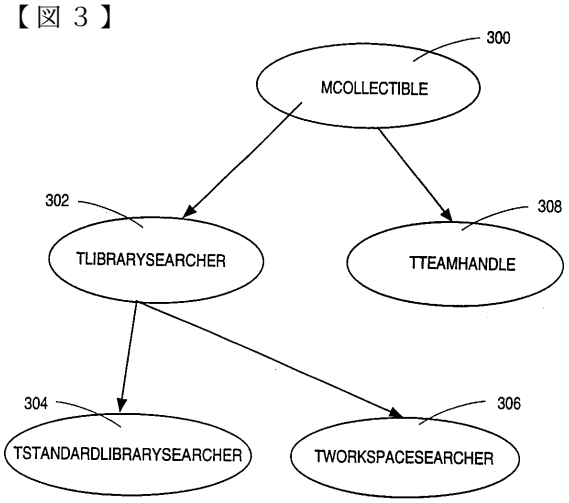

FIG. 3

【図４】

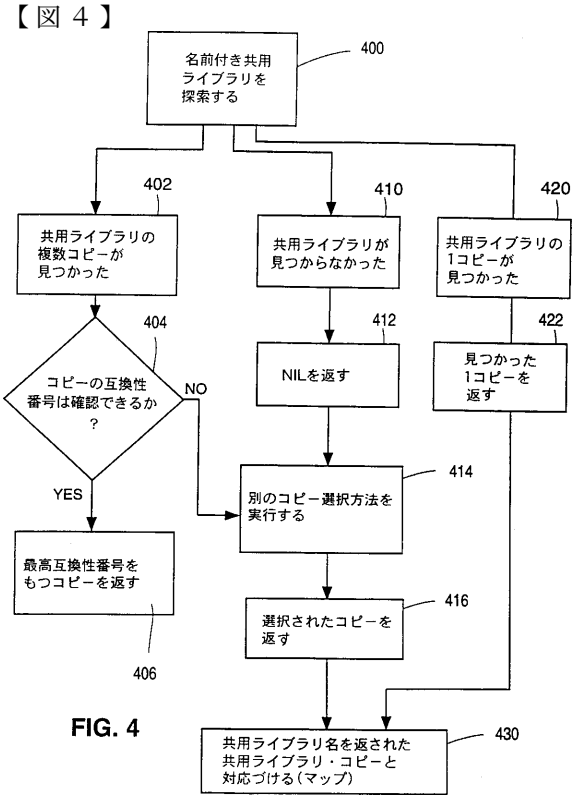

FIG. 4

【図５】

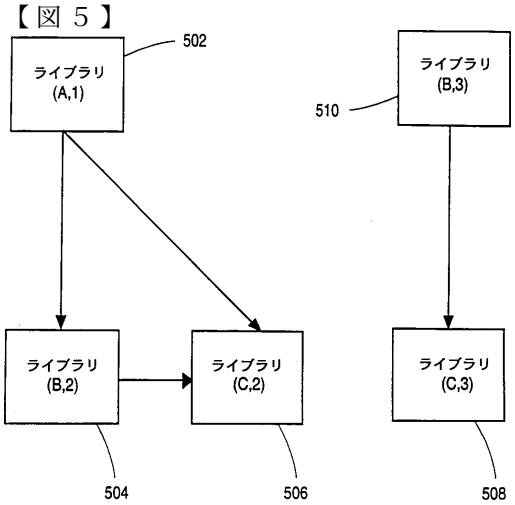
ライブラリ
(A,1)
502
ライブラリ
(B,3)
510
ライブラリ
(B,2)
ライブラリ
(C,2)
ライブラリ
(C,3)
504
506
508

FIG. 5

フロントページの続き

(72)発明者 ヘニンガー，アンドリュー，ジー.
アメリカ合衆国 ９４０２４ カリフォルニア州 ロス アルトス ボルテ ヴィア １６１１
(72)発明者 ナカノ，ラッセル，ティー.
アメリカ合衆国 ９４０８７ カリフォルニア州 サニーベイル アーリッジ ドライブ １３２６

合議体
審判長 吉岡 浩
審判官 松浦 功
審判官 山中 実

(58)調査した分野(Int.Cl.[7]，ＤＢ名)
G06F9/06-9/46,12/00