(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第4146983号
(P4146983)

(45)発行日　平成20年9月10日(2008.9.10)　(24)登録日　平成20年6月27日(2008.6.27)

(51)Int.Cl.		F I		
G06F 9/46	(2006.01)	G06F	9/46	360F
G06F 15/16	(2006.01)	G06F	15/16	620T
G06F 9/44	(2006.01)	G06F	9/44	530M
G06F 13/00	(2006.01)	G06F	13/00	351B

請求項の数 11　(全 25 頁)

(21)出願番号　特願2000-45445(P2000-45445)
(22)出願日　平成12年2月23日(2000.2.23)
(65)公開番号　特開2000-250768(P2000-250768A)
(43)公開日　平成12年9月14日(2000.9.14)
審査請求日　平成12年2月23日(2000.2.23)
審判番号　不服2005-13206(P2005-13206/J1)
審判請求日　平成17年7月11日(2005.7.11)
(31)優先権主張番号　09/259141
(32)優先日　平成11年2月26日(1999.2.26)
(33)優先権主張国　米国(US)

(73)特許権者　390009531
インターナショナル・ビジネス・マシーンズ・コーポレーション
INTERNATIONAL BUSINESS MASCHINES CORPORATION
アメリカ合衆国10504　ニューヨーク州　アーモンク　ニュー　オーチャード　ロード
(74)代理人　100086243
弁理士　坂口　博
(74)代理人　100091568
弁理士　市位　嘉宏

最終頁に続く

(54)【発明の名称】サーバ・オブジェクトのメソッドを呼び出すプロセス方法及びデータ処理システム

(57)【特許請求の範囲】

【請求項１】

分散データ処理システム内の分散アプリケーションにおいて、サーバ・オブジェクトのメソッドを呼び出すデータ処理システムであって、

第１のプログラミング環境を実装するクライアントにおいて、前記第１のプログラミング環境のために記述されたクライアント・オブジェクトを実行する実行手段と、

前記第１のプログラミング環境と異なる第２のプログラミング環境を実装するサーバにおいて、前記第１のプログラミング環境のために記述されたサーバ・オブジェクトを実行する実行手段と、

前記サーバ・オブジェクト内のメソッドを呼び出しするための試みを開始する前記クライアント・オブジェクトを実行する実行手段と、

前記サーバ・オブジェクト内のメソッドを呼び出しするための試みを開始する前記クライアント・オブジェクトに応答して、前記サーバ・オブジェクトのためのオブジェクト参照を前記クライアント上で獲得する獲得手段と、

前記獲得したオブジェクト参照を前記クライアントのアダプタ内でラップするラッピング手段であって、前記アダプタは前記オブジェクト参照をクライアント・オブジェクトから分離し、かつ、前記オブジェクト参照と前記クライアント・オブジェクトとの間の異なるタイプのデータのためのデータ変換を実行する、前記ラッピング手段と、

前記サーバ・オブジェクト内の前記メソッドを前記クライアント・オブジェクトによって呼び出す呼び出し手段であって、前記呼び出しは前記アダプタのメソッドを呼び出し、

前記アダプタ内で呼び出された前記アダプタの前記メソッドは、前記サーバ・オブジェクト内の前記メソッドに対応する、前記呼び出し手段と、
前記オブジェクト参照内のメソッドを、前記アダプタ内で呼び出された前記メソッドによって呼び出す呼び出し手段であって、前記オブジェクト参照内の前記メソッドは前記アダプタ内で呼び出された前記メソッドに対応する、前記呼び出し手段と、
前記サーバ・オブジェクト内の前記メソッドを、前記オブジェクト参照によって呼び出す呼び出し手段であって、前記アダプタは、前記クライアント・オブジェクトが前記オブジェクト参照の操作に気づかないように、前記オブジェクト参照を用いて前記クライアント・オブジェクトに透過的に前記サーバ・オブジェクト内の前記メソッドを呼び出す、前記呼び出し手段と、
を含み、前記アダプタが前記第２のプログラミング環境から前記クライアント・オブジェクトを分離し、且つ前記第１のプログラミング環境から前記オブジェクト参照を分離し、
前記オブジェクト参照と前記クライアント・オブジェクトとの間の前記異なるタイプのデータのためのデータ変換を実行するために、更に
前記クライアント・オブジェクトによって呼び出されたメソッドを受信することに応答して、引き数について前記呼び出されたメソッドを解析する手段と、
必要に応じて前記引き数に対してデータ変換を実行する手段と、
前記引き数がアダプタによってラップされているかどうかを決定する手段と、
前記引き数がアダプタによってラップされている場合、前記引き数から前記アダプタをラップ解除して、前記引き数のオブジェクト参照を獲得する手段であって、前記引き数の前記オブジェクト参照が、前記引き数を表す前記第２のプログラミング環境におけるサーバ・オブジェクトへの参照を提供する手段と、
前記クライアント・オブジェクトによって呼び出された前記メソッドを、前記引き数の前記オブジェクト参照を引き数として前記サーバ・オブジェクトのための前記オブジェクト参照に委託する手段と、
戻り値を検出することに応答して、前記戻り値が前記第２のプログラミング環境のためのオブジェクト参照であるかどうかを決定する手段と、
前記戻り値が前記第２のプログラミング環境のためのオブジェクト参照である場合、前記オブジェクト参照を前記オブジェクト参照のタイプに基づき適切なアダプタでラップする手段と、
前記ラップされたオブジェクト参照を前記クライアント・オブジェクトに返す手段と
を含む、データ処理システム。
【請求項２】
前記アダプタが前記オブジェクト参照を使用して、前記サーバ上のスケルトンのメソッドを呼び出す、請求項１記載のデータ処理システム。
【請求項３】
前記スケルトンが前記サーバ・オブジェクトのメソッドを呼び出す、請求項２記載のデータ処理システム。
【請求項４】
分散データ処理システムにおいて分散アプリケーションを実装するデータ処理システムであって、
第１のプログラミング環境を実装するクライアントにおいて、前記第１のプログラミング環境のために記述されたクライアント・オブジェクトを実行する実行手段と、
前記第１のプログラミング環境と異なる第２のプログラミング環境を実装するサーバにおいて、前記第１のプログラミング環境のために記述されたサーバ・オブジェクトを実行する実行手段と、
前記サーバ・オブジェクト内のメソッドを呼び出しするための試みを開始する前記クライアント・オブジェクトを実行する実行手段と、
前記サーバ・オブジェクト内のメソッドを呼び出しするための試みを開始する前記クラ

イアント・オブジェクトに応答して、前記サーバ・オブジェクトのプロキシを前記クライアント上で獲得する獲得手段と、
前記プロキシを前記クライアントのアダプタ内でラップするラッピング手段であって、前記アダプタは、前記プロキシをクライアント・オブジェクトから分離し、かつ、前記プロキシと前記クライアント・オブジェクトとの間の異なるタイプのデータのためのデータ変換を実行する、前記ラッピング手段と、
前記アダプタのメソッドを呼び出す呼び出し手段であって、前記アダプタは前記プロキシ内のメソッドに対応するメソッドを呼び出す、前記呼び出し手段と
を含み、
前記プロキシと前記クライアント・オブジェクトとの間の前記異なるタイプのデータのためのデータ変換を実行するために、更に
前記クライアント・オブジェクトによって呼び出されたメソッドを受信することに応答して、引き数について前記呼び出されたメソッドを解析する手段と、
必要に応じて前記引き数に対してデータ変換を実行する手段と、
前記引き数がアダプタによってラップされているかどうかを決定する手段と、
前記引き数がアダプタによってラップされている場合、前記引き数から前記アダプタをラップ解除して、前記引き数のプロキシを獲得する手段であって、前記引き数の前記プロキシが、前記引き数を表す前記第２のプログラミング環境におけるサーバ・オブジェクトへの参照を提供する手段と、
前記クライアント・オブジェクトによって呼び出された前記メソッドを、前記引き数の前記プロキシを引き数として前記サーバ・オブジェクトの前記プロキシに委託する手段と、
戻り値を検出することに応答して、前記戻り値が前記第２のプログラミング環境のためのプロキシであるかどうかを決定する手段と、
前記戻り値が前記第２のプログラミング環境のためのプロキシである場合、前記プロキシを前記プロキシのタイプに基づき適切なアダプタでラップする手段と、
前記ラップされたプロキシを前記クライアント・オブジェクトに返す手段と
を含む、データ処理システム。
【請求項５】
前記アダプタが、前記サーバ・オブジェクトによりサポートされるインタフェースを実装するＪａｖａクラスである、請求項４記載のデータ処理システム。
【請求項６】
前記サーバ・オブジェクトがEnterprise Java Beanである、請求項４記載のデータ処理システム。
【請求項７】
前記オブジェクト参照が命名サービスから獲得される、請求項４記載のデータ処理システム。
【請求項８】
前記プロキシがＣＯＲＢＡプロキシである、請求項４記載のデータ処理システム。
【請求項９】
前記アダプタが前記ＣＯＲＢＡプロキシのメソッドを呼び出す、請求項８記載のデータ処理システム。
【請求項１０】
前記ＣＯＲＢＡプロキシがクライアント・コンピュータ上に存在するＪａｖａクラスである、請求項８記載のデータ処理システム。
【請求項１１】
前記ＣＯＲＢＡプロキシがメソッド要求をオブジェクト・リクエスト・ブローカに受け渡す、請求項８記載のデータ処理システム。
【発明の詳細な説明】
【０００１】

【発明の属する技術分野】
本発明は一般に、改善された分散データ処理システムに関し、特に、分散データ処理システム内のクライアント及びサーバ上にオブジェクトを含む、分散アプリケーションのための方法及び装置に関する。
【０００２】
【関連技術】
本発明は、本願と同時に出願された米国特許出願"Method and System for Persisting Beans as Container-Managed Fields"（出願人整理番号：ＡＴ９９８９０７）に関連する。
【０００３】
【従来の技術】
ソフトウェア開発者は、企業全体に及ぶアプリケーションの作成が困難であり、分散アプリケーションの作成が更に困難である根本的な問題に直面している。更に、企業は１つのプラットフォームに閉ざされること無く、できる限り速くアプリケーションを作成することを願望する。理想的には、企業開発者は一旦アプリケーションを作成したら、それを彼らの全てのプラットフォーム上で実行することを希望する。Enterprise JavaBeans（商標）技術は、この能力を提供しようとする。
【０００４】
Enterprise JavaBeans（ＥＪＢ）コンポーネント・アーキテクチャは、企業がスケーラブルで安全なマルチプラットフォーム業務用アプリケーションを、再利用可能なサーバ側コンポーネントとして作成することを可能にするように設計される。その目的は、企業開発者が業務論理（またはビジネス論理）の作成だけに注力できるようにすることにより企業問題を解決することである。
【０００５】
サーバ側環境、及びそれをサービスするために必要とされるツールは、ＥＪＢ技術のための設計目標に多大に影響する。１つの主要な設計目的は、分散アプリケーションを作成するプロセスをできる限り低減することである。この目的は、Enterprise JavaBeansの単純な宣言属性に、通常、手作業でコーディングされる必要のあるフィーチャを転換することにより達成された。これらの宣言属性は、開発効率の多大な向上をもたらす。なぜなら、セキュリティ及びトランザクションなどの特定の振舞いがコード内にセットされるのではなく、ビーン自身上の"フラグ"であるからである。
【０００６】
ＥＪＢ仕様は、トランザクション、セキュリティ、スレッディング、名前付け、オブジェクト・ライフサイクル、資源プーリング、リモート・アクセス、及び持続性などのシステム・レベルのプログラミングを世話するインフラストラクチャを作成する。ＥＪＢ仕様はまた、既存のアプリケーションへのアクセスを単純化し、ツール作成使用のために均等なアプリケーション開発モデルを提供する。
【０００７】
Ｊａｖａは、Ｊａｖａクライアントがリモート・メソッド呼び出し（ＲＭＩ）と呼ばれる方法により、別のプロセスで動作しているＪａｖａサーバ上のメソッドを呼び出す機構を提供する。しかしながら、例えば共通オブジェクト・リクエスト・ブローカ・アーキテクチャ（ＣＯＲＢＡ：Common Object Request Broker Architecture）準拠のサーバなど、サーバがＪａｖａ環境で動作していない場合、ＪａｖａクライアントはＣＯＲＢＡサーバ上のメソッドに対してメソッド呼び出しを発行できない。なぜなら、ＪａｖａはＣＯＲＢＡオブジェクトと通信するための固有の機構を提供しないからである。
【０００８】
要するにＣＯＲＢＡは、アプリケーションがどこに配置されようと、或いは誰がそれらを設計したかに関わらず、アプリケーションが互いに通信することを可能にするオブジェクト・リクエスト・ブローカ（ＯＲＢ）である。オブジェクト・リクエスト・ブローカは、オブジェクト間のクライアント－サーバ関係を確立するミドルウェアである。企業は、様々なソフトウェア・アプリケーション間の相互運用性を提供するための解決策としてＣＯ

ＲＢＡに注目した。
【０００９】
【発明が解決しようとする課題】
データ整備、すなわち、ＪａｖａクライアントとＣＯＲＢＡサーバ間での異なるタイプのデータのためのデータ変換を実行するための標準的な機構は存在しない。ＪａｖａクライアントがＣＯＲＢＡサーバなどの非Ｊａｖａ環境で実行されるＥＪＢなどの、別のＪａｖａアプリケーションと通信することを可能にする機構を有することが有利である。
【００１０】
【課題を解決するための手段】
本発明は、ＣＯＲＢＡサーバ上で実行されるＥＪＢのインタフェース上で内観（introspect）することにより生成されるアダプタを提供する。アダプタはＪａｖａクライアント側に存在し、ＥＪＢを実行するＣＯＲＢＡサーバのリモート・プロキシを含む。アダプタはＥＪＢのビジネス・メソッドを呼び出すために、ＥＪＢにより指定されるインタフェースを実装するＪａｖａクラスである。アダプタは、クライアントからの全てのビジネス・メソッド呼び出しをサーバ上のＣＯＲＢＡプロキシに委託し、ＪａｖａクライアントからＣＯＲＢＡプロキシへの、及びその逆のデータ整備を実行する。クライアントによりアダプタに発行されるビジネス・メソッド呼び出しは適切なデータ変換の後、アダプタによりＣＯＲＢＡプロキシに委託される。従って、アダプタはＪａｖａクライアントとＣＯＲＢＡサーバ上のＥＪＢとの間の透過的な接着剤として作用する。
【００１１】
【発明の実施の形態】
図１を参照すると、本発明が実装され得る分散データ処理システムの図を示す。分散データ処理システム１００は、本発明が実現され得るコンピュータのネットワークである。分散データ処理システム１００はネットワーク１０２を含み、これは分散データ処理システム１００内で一緒に接続される様々な装置及びコンピュータ間で、通信リンクを提供するために使用される媒体である。ネットワーク１０２は電線または光ファイバ・ケーブルなどの永久接続、または電話接続を通じて形成される一時接続を含み得る。
【００１２】
図示の例では、サーバ１０４が記憶ユニット１０６と共にネットワーク１０２に接続される。更に、クライアント１０８、１１０及び１１２がネットワーク１０２に接続される。これらのクライアント１０８、１１０及び１１２は、例えばパーソナル・コンピュータまたはネットワーク・コンピュータである。本願の目的上、ネットワーク・コンピュータはネットワークに接続される任意のコンピュータであり、ネットワークに接続される別のコンピュータからプログラムまたは他のアプリケーションを受信する。図示の例では、サーバ１０４はブート・ファイル、オペレーティング・システム・イメージ、及びアプリケーションなどのデータをクライアント１０８乃至１１２に提供する。クライアント１０８、１１０及び１１２は、サーバ１０４のクライアントである。分散データ処理システム１００は図示されない追加のサーバ、クライアント及び他の装置を含み得る。図示の例では、分散データ処理システムは１００はインターネットであり、ネットワーク１０２が、ＴＣＰ／ＩＰプロトコル一式を使用し互いに通信するネットワーク及びゲートウェイの世界的な集合を表す。主要ノードまたはホスト・コンピュータ間の高速データ通信回線の中枢がインターネットの中心部にあり、データ及びメッセージを経路指定する数千の商業用、政府用、教育用、及び他のコンピュータ・システムを含む。勿論、分散データ処理システム１００は、例えばイントラネット、ローカル・エリア・ネットワーク（ＬＡＮ）、または広域ネットワーク（ＷＡＮ）などの、多数の異なるタイプのネットワークとしても実現され得る。図１は例として示されただけであり、本発明のプロセスの体系的な制限を意図するものではない。
【００１３】
図２を参照すると、このブロック図は、図１のサーバ１０４など、本発明に従いサーバとして実現されるデータ処理システムを示す。データ処理システム（サーバ）２００は、シ

ステム・バス２０６に接続される複数のプロセッサ２０２及び２０４を含む対称マルチプロセッサ（ＳＭＰ）・システムであり得る。或いは、単一プロセッサ・システムが使用され得る。システム・バス２０６には更に、メモリ制御装置／キャッシュ２０８が接続され、これはローカル・メモリ２０９とのインタフェースを提供する。Ｉ／Ｏバス・ブリッジ２１０はシステム・バス２０６に接続され、Ｉ／Ｏバス２１２とのインタフェースを提供する。メモリ制御装置／キャッシュ２０８及びＩ／Ｏバス・ブリッジ２１０は、図示のように統合され得る。

【００１４】

Ｉ／Ｏバス２１２に接続される周辺コンポーネント相互接続（ＰＣＩ）バス・ブリッジ２１４は、ＰＣＩローカル・バス２１６とのインタフェースを提供する。多数のモデム２１８乃至２２０がＰＣＩバス２１６に接続される。一般的なＰＣＩバス・インプリメンテーションは、４つの拡張スロットまたは増設コネクタをサポートする。図１のネットワーク・コンピュータ１０８乃至１１２への通信リンクは、増設ボードを介してＰＣＩローカル・バス２１６に接続されるモデム２１８及びネットワーク・アダプタ２２０を介して提供され得る。

【００１５】

追加のＰＣＩバス・ブリッジ２２２及び２２４は、ＰＣＩバス２２６及び２２８のためのインタフェースを提供し、これらのバスから追加のモデムまたはネットワーク・アダプタがサポートされ得る。このように、サーバ２００は複数のネットワーク・コンピュータへの接続を可能にする。また図示のように、メモリマップド・グラフィックス・アダプタ２３０及びハード・ディスク２３２も、直接的にまたは間接的にＩ／Ｏバス２１２に接続され得る。

【００１６】

当業者には明らかなように、図２に示されるハードウェアは変化し得る。例えば、光ディスク・ドライブなどの他の周辺装置も、追加で、または図示のハードウェアの代わりに使用され得る。図示の例は、本発明に関する体系的な制限を意味するものではない。

【００１７】

図２に示されるデータ処理システムは、例えばＩＢＭの製品であるIBM RISC/System6000システムであり、ＡＩＸ（Advanced Interactive Executive）オペレーティング・システムを実行する。

【００１８】

次に図３を参照すると、本発明が実現され得るデータ処理システムのブロック図が示される。データ処理システム３００は、クライアント・コンピュータの例である。データ処理システム３００は、周辺コンポーネント相互接続（ＰＣＩ）ローカル・バス・アーキテクチャを使用する。図示の例はＰＣＩバスを使用するが、マイクロチャネル及びＩＳＡなどの他のバス・アーキテクチャも使用され得る。プロセッサ３０２及び主メモリ３０４がＰＣＩブリッジ３０８を介してＰＣＩローカル・バス３０６に接続される。ＰＣＩブリッジ３０８は、プロセッサ３０２のための統合型メモリ制御装置及びキャッシュ・メモリを含み得る。ＰＣＩローカル・バス３０６への追加の接続は、直接コンポーネント相互接続を介してまたは増設ボードを介して形成され得る。図示の例では、ローカル・エリア・ネットワーク（ＬＡＮ）・アダプタ３１０、ＳＣＳＩホスト・バス・アダプタ３１２、及び拡張バス・インタフェース３１４が直接コンポーネント接続によりＰＣＩローカル・バス３０６に接続される。それに対して、音声アダプタ３１６、グラフィックス・アダプタ３１８、及びスマートカード・アダプタ３１９は、拡張スロットに挿入される増設ボードにより、ＰＣＩローカル・バス３０６に接続される。拡張バス・インタフェース３１４は、キーボード及びマウス・アダプタ３２０、モデム３２２、及び追加のメモリ３２４のための接続を提供する。ＳＣＳＩホスト・バス・アダプタ３１２は、ハード・ディスク・ドライブ３２６、テープ・ドライブ３２８、及びＣＤ－ＲＯＭドライブ３３０のための接続を提供する。一般的なＰＣＩローカル・バス・インプリメンテーションは、３つまたは４つのＰＣＩ拡張スロットまたは増設コネクタをサポートする。

【００１９】
オペレーティング・システムはプロセッサ３０２上で実行され、図３のデータ処理システム３００内の様々なコンポーネントの制御を調整及び提供するために使用される。オペレーティング・システムはＩＢＭから提供されるＯＳ／２などの、市販のオペレーティング・システムであってよい。"ＯＳ／２"はＩＢＭの商標である。Ｊａｖａなどのオブジェクト指向のプログラミング・システムが、オペレーティング・システムと共に実行され、データ処理システム３００上で実行されるＪａｖａプログラムまたはアプリケーションからオペレーティング・システムに呼び出しを提供する。"Ｊａｖａ"はサン・マイクロシステムズ社の商標である。オペレーティング・システムの命令、オブジェクト指向オペレーティング・システム、及びアプリケーションまたはプログラムは、ハード・ディスク・ドライブ３２６などの記憶装置上に配置され、主メモリ３０４にロードされ、プロセッサ３０２により実行される。
【００２０】
当業者には明らかなように、図３のハードウェアはインプリメンテーションに応じて変化し得る。フラッシュＲＯＭ（または等価な不揮発性メモリ）または光ディスク・ドライブなどの、他の内部ハードウェアまたは周辺装置が更に追加で、または図３に示されるハードウェアの代わりに使用され得る。また、本発明のプロセスはマルチプロセッサ・データ処理システムに適用され得る。
【００２１】
例えば、データ処理システム３００は、ネットワーク・コンピュータとして任意的に構成される場合、図３に点線３３２で示されるボックス内のＳＣＳＩホスト・バス・アダプタ３１２、ハード・ディスク・ドライブ３２６、テープ・ドライブ３２８、及びＣＤ－ＲＯＭ３３０を含まなくてよい。すなわち、点線ボックス内のコンポーネントは任意に含まれ得ることを示す。その場合、正確にはクライアント・コンピュータと呼ばれるコンピュータは、ＬＡＮアダプタ３１０やモデム３２２などの、特定タイプのネットワーク通信インタフェースを含まねばならない。別の例として、データ処理システム３００は自身が特定タイプのネットワーク通信インタフェースを含むか否かに関わらず、特定タイプのネットワーク通信インタフェースに頼ることなく、ブート可能に構成される独立型のシステムである。更に別の例として、データ処理システム３００はパーソナル・デジタル・アシスタント（ＰＤＡ）装置であり、これはオペレーティング・システム・ファイルまたはユーザ生成データを記憶する不揮発性メモリを提供するために、ＲＯＭまたはフラッシュＲＯＭにより構成される。
【００２２】
図３の図示の例、及び前述の例は、本発明に関する体系的な制限を意味するものではない。
【００２３】
本発明は、ＪａｖａクライアントがＣＯＲＢＡサーバ上のＥＪＢと通信可能な分散アプリケーションを提供する方法、データ処理システム、及び命令を提供する。分散アプリケーション内における本発明について述べる都合上、従来の総称的な分散アプリケーションについて詳述することにする。
【００２４】
本発明のプロセスは、オブジェクト指向プログラミング・システムであるＪａｖａプログラミング・システムを用いて実現され得る。オブジェクト指向プログラミング技術は、"オブジェクト"の定義、作成、使用及び命令を含み得る。これらのオブジェクトは、データ要素または属性、及びデータ要素を操作するメソッドを含むソフトウェア・エンティティである。オブジェクトはまた、オブジェクト内のメソッドをトリガまたは制御するオブジェクト外の事象に関連するデータを含み得る。
【００２５】
オブジェクトは、"クラス"を定義することにより定義される。クラスはオブジェクトではなくテンプレートであり、コンパイラに実際のオブジェクトの作成方法を指示する。例え

ば、クラスはデータ変数の数及びタイプや、データを操作する機能に関わるステップを指定し得る。オブジェクトは実際、コンストラクタと呼ばれる特殊機能によりプログラム内で作成される。コンストラクタは、対応するクラス定義、及びオブジェクト作成の間に提供される引き数などの追加の情報を使用し、オブジェクトを構成する。オブジェクトはデストラクタと呼ばれる特殊機能により破壊される。Ｊａｖａはまた、インタフェースとして知られる完全な抽象クラスの作成を許容し、これは他のクラスが如何にメソッドを処理しているかに関わり無く、複数のクラスと共用され得るメソッドの定義を可能にする。

【００２６】

図４を参照すると、従来の分散アプリケーションを示す図が示される。図示のように、クライアント・オブジェクト４００は、分散アプリケーションのクライアント側に存在し、一方、サーバ・オブジェクト４０２は分散アプリケーションのサーバ側を形成する。クライアント・オブジェクト４００は、例えば図１の分散データ処理システム１００内のクライアント１０８などの、クライアント・コンピュータ上に配置される。サーバ・オブジェクト４０２は、図１のサーバ１０４などのサーバ上に配置される。クライアント・オブジェクト４００は、サーバ・オブジェクト４０２内に実装される様々なビジネス規則またはビジネス論理にもとづき、データベース４０４をアクセスするために、サーバ・オブジェクト４０２への呼び出しを開始する。データベース４０４はサーバ内に配置されるか、リモート・データベースである。サーバ・オブジェクト４０２は現企業アプリケーション４１０へのアクセス、及びレガシ・アプリケーション４０８へのアクセスも提供し得る。分散アプリケーションを作成するユーザが、サーバ・オブジェクト４０２を実装する第２階級（tier）のコンピュータの、ホスト名またはインターネット・プロトコル（ＩＰ）・アドレスを構成することを可能にするためにカストマイザが提供され得る。この例では、Ｊａｖａリモート・メソッド呼び出し（ＲＭＩ）プロトコル４０６が、クライアント・オブジェクト４００及びサーバ・オブジェクト４０２などのオブジェクト間の分散通信のために使用される。

【００２７】

サーバ・オブジェクト４０２は、アプリケーション・プログラミング・インタフェース（ＡＰＩ）を用いて実装される実際のビジネス論理を含む。ＡＰＩはＪａｖａ定義Ｊａｖａデータベース接続性（ＪＤＢＣ）構造化照会言語（ＳＱＬ）データベース・アクセス・インタフェースを使用し、これは広範囲のリレーショナル・データベースへの均等なアクセスを提供する。図示の例では、これらのデータベースはデータベース４０４内で見いだされ得る。サーバ・オブジェクト４０２は、クライアント・オブジェクト４００から呼び出される必要な機能を提供するために使用されるメソッドを含む。前述のメソッドは、他の後置システム（すなわちＣＩＣＳ、ＩＭＳ、ＭＱ、ＳＡＰなど）をアクセスするために作成され、ＪＤＢＣまたはデータベース・アクセスだけに制限されるべきではない。

【００２８】

分散アプリケーション内におけるデータベース・アクセスの例について、引き続き述べると、データベース・アクセス機能は２つのオブジェクト、すなわちクライアント・オブジェクト及びサーバ・オブジェクトに分けられる。サーバ・オブジェクトは、クライアント・オブジェクトから呼び出され、ＪＤＢＣを使用するメソッドを含むインタフェースを実装する。

【００２９】

クライアント・オブジェクトは、様々な事象の事象ソースまたは事象シンクとして作用する。生成時、クライアント・オブジェクトはクライアント側ビルダ環境で使用され、他のクライアント側ソフトウェア・コンポーネントに接続される。例えば、クライアント側生成時にアプリケーション・アセンブラがクライアント側ビルダ環境を使用し、ＧＵＩオブジェクトをクライアント・オブジェクトに接続する。従って、ボタンが押下されるとき、特定のデータを検索するための事象がオブジェクトに送信される。サーバ側生成時には、アプリケーション・アセンブラがビルダ環境を使用し、サーバ・オブジェクトを接続する。

【００３０】
実行時、ユーザはクライアント上のボタンを押下し、これは特定のデータが表示のために要求されることを示す。ＧＵＩオブジェクトは事象を生成し、これがクライアント・オブジェクトに送信される。クライアント・オブジェクトは、必要なデータを要求するサーバ・オブジェクト上のメソッドを呼び出す。クライアント・オブジェクトは、リモート・メソッド呼び出し（ＲＭＩ）またはＩＩＯＰ（Internet InterOrb Protocol）を使用するオブジェクト・リクエスト・ブローカ（ＯＲＢ）などのプロトコルを使用することにより、その対応するサーバ・オブジェクトと通信する。サーバ側オブジェクトはデータを検索し、データをクライアント側オブジェクトに返送する。データが次にクライアント側オブジェクトによりＧＵＩオブジェクトに返送され、エンドユーザに表示される。

【００３１】
図示の例では、クライアント・オブジェクト４００及びサーバ・オブジェクト４０２を実装するＪａｖａビーンズが使用され得る。ビーンを純粋なオブジェクトと区別するのは、それが属性インタフェースと呼ばれる外部インタフェースを有することであり、これはツールがコンポーネントが実行しようとしていることを読出し、それを他のビーンにフック留めし（hook up）、それを別の環境にプラグ・インすることを可能にする。２つの異なるタイプのビーン、すなわちJavaBeans及びEnterprise JavaBeans（ＥＪＢ）が使用され得る。JavaBeansは単一のプロセスに対して局所的であるようにもくろまれ、しばしば実行時に見ることができる。このビジュアル・コンポーネントは、ボタン、リスト・ボックス、グラフィックまたはチャートなどである。

【００３２】
ＥＪＢは、サーバ上で実行され、クライアントにより呼び出されるように設計される、見ることができないリモート・オブジェクトである。ＥＪＢは複数の見ることができないJavaBeansから構成される。ＥＪＢはあるマシン上に常駐し、別のマシンから遠隔的に呼び出されるようにもくろまれる。ＥＪＢは、ツールにより読出され得るビーンに関する記述としてもくろまれる展開記述子（deployment descriptor）を有する。ＥＪＢはまたプラットフォーム独立であり、Ｊａｖａをサポートする任意のプラットフォーム上で使用され得る。

【００３３】
サーバ・ビーンズすなわちＥＪＢは、サーバ上に展開される遠隔的に実行可能なコンポーネントまたはビジネス・オブジェクトである。ＥＪＢはそれらが遠隔的にアクセスされることを可能にするプロトコルを有し、このプロトコルは、ＥＪＢが特定のサーバ上に導入または展開されることを可能にする。ＥＪＢはサービスの主要な品質、セキュリティ、トランザクション振舞い、同時性（２つ以上のクライアントにより一度にアクセスされる能力）、及び持続性（どのようにそれらの状態が保管され得るか）をＥＪＢサーバ上においてそれらが配置されるコンテナに委託する一連の機構を有する。ＥＪＢはそれらの振舞いを異なるサービス品質を提供するコンテナ内に導入される。展開ツールの使用を通じて、プラットフォーム独立のJavaBeanがプラットフォーム特定のＥＪＢに取り入れられる。後者は、既存のビジネス・システム及びアプリケーションの特定の要求に合致するために使用可能な適正なサービス品質を有する。

【００３４】
クライアント・ビーン４００とサーバ・ビーン４０２間のこの分離により、サーバ・ビーン４０２内の様々なビジネス論理の変更がクライアント・ビーン４００の変更無しに実行され得る。これは単一のサーバをアクセスする数千のクライアントが存在し得ることを考慮すると望ましい。更にこれらのプロセスは、例えばＣＯＢＯＬなどの今日的でないプログラミング言語により作成されたプログラムにも適用され得る。こうしたプログラムの動的変更は、そのプログラムをＪａｖａなどのオブジェクト指向プログラミング・システムと互換にするインタフェースを作成することにより実行され得る。

【００３５】
分散アプリケーション内及び間の通信サービスは、例えばＯＭＧ（Object Management Gr

oup）コンソーシアムにより設計されたＣＯＲＢＡ規格など、Ｊａｖａリモート・メソッド呼び出し（ＲＭＩ）以外の他のタイプの分散プロトコルによっても同様に実現され得る。ＣＯＲＢＡは、今日使用可能な急増しつつあるハードウェア及びソフトウェア製品の間の相互運用性のニーズに対するＯＭＧの回答である。要するに、ＣＯＲＢＡはアプリケーションがどこに配置されようと、また誰がそれらを設計したかに関わらず、アプリケーションが互いに通信することを可能にするオブジェクト・リクエスト・ブローカ（ＯＲＢ）である。

【００３６】

オブジェクト・リクエスト・ブローカは、オブジェクト間のクライアント－サーバ関係を確立するミドルウェアである。オブジェクト・リクエスト・ブローカを使用することにより、クライアントは同一マシン上に存在する、またはネットワークを介して存在するサーバ・オブジェクト上のメソッドを透過的に呼び出すことができる。オブジェクト・リクエスト・ブローカは呼び出しを横取りし、要求を実現するオブジェクトを見いだす責任があり、それにパラメータを受け渡し、そのメソッドを呼び出し、結果を返却する。クライアントは、オブジェクトが配置されている場所、そのプログラミング言語、そのオペレーティング・システム、またはオブジェクトのインタフェースの一部でない他のシステム態様を知る必要はない。そうすることにより、オブジェクト・リクエスト・ブローカは、異種の分散環境内の異なるマシン上のアプリケーション間の相互運用性を提供し、複数のオブジェクト・システムを継ぎ目無く相互接続する。

【００３７】

典型的なクライアント／サーバ・アプリケーションを扱うために、開発者は彼ら自身の設計または認識された規格を使用し、装置間で使用されるプロトコルを定義する。プロトコル定義はインプリメンテーション言語、ネットワーク・トランスポート、及び幾つかの他のファクタに依存する。オブジェクト・リクエスト・ブローカはこのプロセスを単純化し、柔軟性を提供する。オブジェクト・リクエスト・ブローカは、構成中のシステムの各コンポーネントのために、プログラマが大抵の適切なオペレーティング・システム、実行環境、及びプログラミング言語さえも選択し、使用することを可能にする。更に重要な点は、オブジェクト・リクエスト・ブローカは既存のコンポーネントの統合を可能にする。ＯＲＢベースの解決策では、開発者は新たなオブジェクトを作成するために使用するのと同一のインタフェースを用いて、レガシ・コンポーネントをモデル化し、次に標準化バスとレガシ・インタフェース間を変換する"ラッパ（wrapper）"・コードを作成する。

【００３８】

ＣＯＲＢＡは、オブジェクト指向標準化及び相互運用性に向けての重要な進歩を表すオブジェクト・リクエスト・ブローカである。ＣＯＲＢＡにより、ユーザは、情報がどのソフトウェアまたはハードウェア・プラットフォーム上に存在するか、或いは情報がネットワーク内のどこに配置されるかを知る必要無しに、情報へのアクセスを透過的に獲得できる。ＣＯＲＢＡオブジェクト開発の目的は、オブジェクト・サーバまたは単にサーバの生成及び登録である。サーバはプログラムであり、１つ以上のオブジェクト・タイプのインプリメンテーションを含み、オブジェクト・リクエスト・ブローカに登録される。

【００３９】

ＣＯＲＢＡは、オブジェクトが企業及び大陸さえも横断して通信することを可能にするオブジェクト・バスを詳述する。ＣＯＲＢＡは、高機能コンポーネントが互いに発見し合い、オブジェクト・バスを介して相互運用することを可能にするように設計された。しかしながら、ＣＯＲＢＡは単なる相互運用性を超越する。ＣＯＲＢＡはバス関連サービスの拡張セットを指定することにより、オブジェクトを作成及び消去し、それらを名前によりアクセスし、それらを永久記憶装置に記憶し、それらの状態を外部化し、それらの間の臨時の関係を定義する。

【００４０】

ＪＤＫバージョン１．１の発表により、Ｊａｖａはリモート・メソッド呼び出し（ＲＭＩ）と呼ばれる、それ自身の組み込み型の固有のオブジェクト・リクエスト・ブローカを有

する。リモート・メソッド呼び出しは、リモート・オブジェクト上のメソッド呼び出しを行うといった総称的な意味では、オブジェクト・リクエスト・ブローカであるが、それはＣＯＲＢＡ準拠のオブジェクト・リクエスト・ブローカではない。リモート・メソッド呼び出しはＪａｖａに固有である。リモート・メソッド呼び出しは、本来、中核を成すＪａｖａ言語の拡張である。リモート・メソッド呼び出し（ＲＭＩ）は、Ｊａｖａオブジェクト直列化、移植性のあるダウンロード可能なオブジェクト・インプリメンテーション、及びＪａｖａインタフェース定義などの、他の多くのフィーチャに依存する。他方、リモート・メソッド呼び出しは幾つかの制限を有し、その最も偉大な強み、すなわちＪａｖａとのその堅い統合の結果である原理上の制限が、他の言語により作成されたオブジェクトまたはアプリケーションとの使用を非現実的なものにする。

【００４１】

Ｊａｖａは、拡張によるリモート・メソッド呼び出しと共に具体的なプログラミング技術である。Ｊａｖａは基本的に、実行可能コードを作成及び編成する問題を解決するために設計された。Ｊａｖａはそれなりにプログラミング技術の間に、特定のポイントを築き上げる。Ｊａｖａと他のプログラミング言語との間に存在する隔たりは、時に行き交うことが困難である。例えば、ＪａｖａコードからＡｄａコードに呼び出しを行うために使用される技術は、ＪａｖａコードからＣ＋＋コードに呼び出しを行うために使用されるものと幾分異なる。このことは多言語環境において、システムの生成を複雑化させ、複雑化は使用される言語の数と共に著しく、時に非線形的に増加する。

【００４２】

ＪａｖａはＪａｖａネイティブ・インタフェース（ＪＮＩ：Java Native Interface）と呼ばれるＡＰＩを提供し、これはＪａｖａコードが他の言語によるルーチンを呼び出す、または呼び出されることを可能にする。ＪＮＩは主にＣ及びＣ＋＋言語との相互運用のために適合化され、習得するのにかなり難しいインタフェースである。リモート・メソッド呼び出しはＪａｖａ間の技術である。Ｊａｖａクライアントがリモート・メソッド呼び出しを使用し、別の言語のリモート・オブジェクトと通信したい場合、"異質の（foreign）"リモート・オブジェクトと一緒に配置されるＪａｖａ媒介を介する必要がある。この場合の根本的な問題は、Ｊａｖａが定義上、言語自身の境界内で作用するプログラミング技術であることである。

【００４３】

それに対してＣＯＲＢＡは統合技術であり、プログラミング技術ではない。ＣＯＲＢＡは特に、異種のプログラミング技術を一緒に結合する接着剤として設計される。ＣＯＲＢＡはプログラミング空間内のポイントとして存在するのではなく、個々の言語を表すポイント間の空間を占有する。例えば、ＪａｖａクライアントがＣＯＲＢＡ技術を使用し、Ｃ＋＋オブジェクトと通信するとき、Ｃ＋＋プログラマ及びＪａｖａプログラマの両者は、完全にそれぞれの言語環境内で作業する。ＣＯＲＢＡのオブジェクト・リクエスト・ブローカは、ＪａｖａクライアントにＪａｖａスタブ・インタフェースを提供し、Ｃ＋＋プログラマにＣ＋＋スケルトン・インタフェースを提供する。ＣＯＲＢＡは言語間問題を自動的に解決する。

【００４４】

ＣＯＲＢＡは統合指向の見地を提供し、そこでは設計努力がシステムの要素間の境界に絞られる。基礎となるインタフェース技術（例えばＩＩＯＰ）は、それらの境界をできる限り柔軟で、適応的で、プログラミング技術独立にするように設計される。ＣＯＲＢＡなどのインタフェース技術は、プログラミング技術よりも長い半減期を有するだけでなく、廃れたプログラミング言語への依存による、アプリケーションの追加及び死去に対する最善の防御である。

【００４５】

図５を参照すると、ＣＯＲＢＡ規格を使用する従来の分散アプリケーションが示される。クライアント・オブジェクト５００は、通信リンク５１０として示されるＩＩＯＰ規格を使用し、ＣＯＲＢＡサーバ５２０と通信する。ＣＯＲＢＡサーバ５２０は、異種のタイプ

のソフトウェア・オブジェクトのための統合及び相互運用性を提供する。JavaBean５２１、Ｃ＋＋オブジェクト５２２、Enterprise JavaBean（ＥＪＢ）５２３、及びＣＯＢＯＬオブジェクト５２４は、サーバ５２０により提供されるＣＯＲＢＡサービスを用いて通信し、サービス機能を登録する。クライアント・オブジェクト５００は、ＣＯＲＢＡサービスを通じて、オブジェクト５２１乃至５２４により提供される機能及びメソッドを呼び出す。

【００４６】

図６を参照すると、ＣＯＲＢＡを用いて相互運用性機能を提供する従来の分散アプリケーション内のコンポーネントが示される。図６は、クライアント６０２から、サーバ内のＣＯＲＢＡオブジェクト・インプリメンテーションに送信されるメソッド要求６１８を示す。クライアントは、ＣＯＲＢＡサーバ上のメソッドを呼び出す任意のコードであり、ことによるとそれ自身がＣＯＲＢＡオブジェクトである。サーバ・オブジェクト６２０は、オブジェクト・インプリメンテーションのインスタンス、すなわちＣＯＲＢＡオブジェクトを実現する実際のコード及びデータである。

【００４７】

ＣＯＲＢＡサーバ・オブジェクトのクライアント６０２は、サーバ・オブジェクトに対するオブジェクト参照６１６を有し、クライアントはこのオブジェクト参照を使用し、メソッド要求６１８を発行する。

【００４８】

オブジェクト参照は、オブジェクト・リクエスト・ブローカ内のオブジェクトを指定するために必要とされる情報である。クライアントは通常、幾つかの異なる方法によりオブジェクト参照を獲得する。第１に、クライアントはオブジェクトを作成するためにオブジェクト上の"作成"メソッドを呼び出す。作成メソッドは、新たなオブジェクトに対するオブジェクト参照をクライアントに返却する。第２に、クライアントは、命名サービスに要求を発行することにより、オブジェクト参照を獲得する。命名サービスはオブジェクト参照を名前によりデータ構造内に記憶し、クライアントは特定タイプのハードコード化ネットワーク・アドレスではなしに、オブジェクトの関連付けられる名前によりオブジェクト参照を探索または解析する。すなわち、オブジェクトを同一の物理マシン内で、またはネットワーク上のどこかで突き止める。最後に、クライアントは、オブジェクト参照をストリング化することにより特定的に作成されたストリングからオブジェクト参照を獲得する。

【００４９】

一旦オブジェクト参照が獲得されると、クライアントはＣＯＲＢＡオブジェクトを自身上に呼び出せるように、適切なタイプに制限する。

【００５０】

サーバ・オブジェクトが遠隔的である場合、オブジェクト参照はスタブ機能６０４を指し示し、これがオブジェクト・リクエスト・ブローカ・マシンを使用して、呼び出しをサーバ・オブジェクトに転送する。ＣＯＲＢＡクライアントは、その全てのデータ整備及びＩＯＰ作業を実行するために、ローカル・オブジェクト・リクエスト・ブローカ・オブジェクトを必要とする。スタブ・コードはオブジェクト・リクエスト・ブローカ６０６により、サーバ・オブジェクトを実行するマシンを識別し、そのマシンのオブジェクト・リクエスト・ブローカ６１０に、オブジェクトのサーバ６１４への接続を依頼する。スタブ・コードが接続を有するとき、それはオブジェクト参照及びパラメータを宛先オブジェクトのインプリメンテーションにリンクされるスケルトン・コード６１２に送信する。スケルトン・コードは呼び出し及びパラメータを要求されたインプリメンテーション特定の形式に変換し、オブジェクトを呼び出す。あらゆる結果または例外が、同一の経路に沿って返却される。

【００５１】

クライアントは、ＣＯＲＢＡオブジェクトの位置、インプリメンテーション詳細、及びオブジェクトをアクセスするために使用されるオブジェクト・リクエスト・ブローカを知ら

ない。異なるオブジェクト・リクエスト・ブローカが、ＩＩＯＰ６０８を介して通信し得る。

【００５２】

クライアントは、ＣＯＲＢＡオブジェクトのインタフェース内で指定されるメソッドだけを呼び出し得る。インタフェースはオブジェクト・タイプを定義し、命名メソッド及びパラメータのセットの他に、これらのメソッドが返却し得る例外タイプを指定する。サーバ・オブジェクトのクライアントは、サーバ・オブジェクトに対応するオブジェクト参照へのアクセスを有し、そのオブジェクトに対するオペレーションを呼び出す。クライアント・オブジェクトは、そのインタフェースに従い、サーバ・オブジェクトの論理構造だけを知り、メソッド呼び出しを通じてサーバ・オブジェクトの振舞いに遭遇する。重要な点は、クライアント－サーバ関係が２つの特定のオブジェクトに関連することである。すなわち、一方のサーバ・オブジェクトのインプリメンテーションは、他のサーバ・オブジェクトのクライアントで有り得る。

【００５３】

スタブ及びスケルトン・ファイルは、様々な方法により生成される。スタブ・ファイルは、クライアント・プログラミング言語により、クライアントにサーバ・メソッドへのアクセスを提供する。サーバ・スケルトン・ファイルは、オブジェクト・インプリメンテーションをオブジェクト・リクエスト・ブローカ（ＯＲＢ）実行時に結び付ける。オブジェクト・リクエスト・ブローカはスケルトンを使用し、メソッドをオブジェクト・インプリメンテーション・インスタンス（サーバント）にディスパッチする。

【００５４】

従来技術の説明から本発明の説明に目を向けると、図７及び図８は、Ｊａｖａ及びＣＯＲＢＡの利点を結合する本発明の詳細を示す。前述のように、リモート・メソッド呼び出しは、Ｊａｖａクライアントがリモート・メソッド呼び出しによりリモート・オブジェクトと通信することを強要する、Ｊａｖａ間の技術である。Ｊａｖａは定義上、言語自身の境界内で作用するプログラミング技術である。それに対して、ＣＯＲＢＡは統合技術であり、プログラミング技術ではない。本発明は、ＪａｖａクライアントがＣＯＲＢＡクライアント上に存在するＥＪＢのビジネス・メソッドを呼び出すことを可能にする。

【００５５】

図７を参照すると、このブロック図は、ＣＯＲＢＡサーバ内で実行されるEnterprise JavaBean（ＥＪＢ）のリモート・ビジネス・メソッドを呼び出す方法を実現するために使用されるコンポーネントを示す。図７のシステムは、図４及び図６に示されるシステムに類似する。図７は、クライアント７００及びサーバ７２６が、Ｊａｖａ仮想マシンＪＶＭ７０２及びＪＶＭ７２４を含むＪａｖａ実行時環境によりイネーブルされる以外は、図４に示される分散アプリケーションと類似する。図７は図６と類似する。なぜなら、クライアント６０２及びサーバ６１４と同様に、Ｊａｖａクライアント・オブジェクト７０４がリモートＥＪＢ７２８内のメソッドを呼び出そうとしているからである。

【００５６】

更に、図７の分散アプリケーションは、オブジェクト・リクエスト・ブローカＯＲＢ７１４及びＯＲＢ７１８を含み、オブジェクト要求及び応答をＩＩＯＰ７１６を介して伝達する。図６の総称的なアーキテクチャは、クライアント、サーバ、スタブ、及びスケルトンを含み、オブジェクト・リクエスト・ブローカが図７のソフトウェア・アーキテクチャ内に反映され、これは本発明の方法に従い、ＪａｖａオブジェクトがＣＯＲＢＡを用いて通信することを可能にする。

【００５７】

ＪＶＭ７０２は、ＥＪＢ７２８内に存在するビジネス・メソッドを呼び出そうとしているＪａｖａクライアント・オブジェクト７０４を含む。Ｊａｖａ仮想マシン（ＪＶＭ）は、メモリ内に存在する仮想コンピュータ・コンポーネントである。特定のケースでは、ＪＶＭはプロセッサ内で実現され得る。ＪＶＭはＪａｖａプログラムが、コードがコンパイルされたあるプラットフォームだけではなしに、異なるプラットフォーム上で実行されるこ

とを可能にする。ＪａｖａプログラムはＪＶＭのためにコンパイルされる。このように、Ｊａｖａは、様々な中央処理ユニット及びオペレーティング・システム・アーキテクチャを含み得る多くのタイプのデータ処理システムのためにアプリケーションをサポートできる。

【００５８】

Ｊａｖａアプリケーションが異なるタイプのデータ処理システム上で実行されるようにするために、コンパイラは一般に、アーキテクチャ中立ファイル形式を生成する。すなわち、Ｊａｖａ実行時システムが存在する場合、コンパイルされたコードが多くのプロセッサ上で実行可能である。Ｊａｖａコンパイラは、特定のコンピュータ・アーキテクチャには特定的でないバイトコード命令を生成する。バイトコードは、Ｊａｖａコンパイラにより生成されるマシン独立なコードであり、Ｊａｖａインタプリタにより実行される。Ｊａｖａインタプリタは、バイトコードを交互に解読及び実行するＪＶＭ内のモジュールである。これらのバイトコード命令は、任意のコンピュータ上で容易に解釈され、また固有マシン・コードに即座に容易に変換されるように設計される。

【００５９】

ＪＶＭ７０２は、ＥＪＢインタフェース７０８をクライアント・オブジェクト７０４に提供するアダプタ７０６を含み、それにより、クライアント・オブジェクト７０４はリモート・メソッド呼び出しの既知のメソッドを使用し、ＥＪＢインタフェース７０８内のメソッドを呼び出す。ＥＪＢインタフェース７０８はＣＯＲＢＡプロキシ７１０を呼び出し、サーバ７２６とのＣＯＲＢＡ通信を開始する。ＣＯＲＢＡプロキシ７１０は、オブジェクト要求をオブジェクト・リクエスト・ブローカＯＲＢ７１４に受け渡す。この例では、ＯＲＢ７１４はＪａｖａにより実現される。ＯＲＢがＣ＋＋により実現される場合、オブジェクト要求はＪａｖａネイティブ・インタフェース（ＪＮＩ）を通じて受け渡される。ＯＲＢ７１４及びＯＲＢ７１８はＩＩＯＰ７１６を介して通信し、オブジェクト要求がクライアント及びサーバ・オブジェクトにより、オブジェクト・リクエスト・ブローカを用いて透過的にサポートされることを保証する。一旦ＯＲＢ７１８がオブジェクト要求を受信すると、ＥＪＢスケルトン７２０内のコードが呼び出され、ＥＪＢ７２８から要求されたビジネス・メソッドの呼び出しを開始する。ＥＪＢスケルトン７２０はＪＮＩ７２２を用いてＥＪＢ７２８を呼び出し、適切な引き数をＪＶＭ７２４内に含まれるＥＪＢ７２８に受け渡す。この特定の例では、ＥＪＢスケルトン７２０はＣ＋＋により実現され得るが、ＪＮＩ７２２の使用を要求するＪａｖａ以外の別の言語によっても実現され得る。

【００６０】

図８を参照すると、このブロック図は、サーバ・オブジェクトのリモート・メソッドを呼び出す方法を実現するために使用されるコンポーネントを示す。図８のシステムは、クライアント７５０がある言語により実装されるクライアント・オブジェクト７５２を有し、サーバ７７２が異なる言語により実装されるサーバ・オブジェクト７７０を有するように、環境が一般化される以外は、図７に示されるシステムと類似する。

【００６１】

クライアント７５０はアダプタ７５４を含む。アダプタはクライアント・オブジェクト７５２のためにオブジェクト参照により、サーバ・オブジェクト７７０上のリモート・メソッド呼び出しをシミュレートする。クライアント・オブジェクト７５２はアダプタ７５４内のメソッドを呼び出し、これがオブジェクト参照７５６を呼び出し、サーバ７７２とのオブジェクト通信を開始する。オブジェクト参照７５６は、データ整備モジュール７５８を通じてオブジェクト要求をオブジェクト・ディスパッチャ７６０に受け渡す。オブジェクト・ディスパッチャ７６０及びオブジェクト・ディスパッチャ７６４は、オブジェクト通信リンク７６２を介して通信し、オブジェクト要求がクライアント及びサーバ・オブジェクトにより適切なオブジェクト要求プロトコルを用いて、透過的にサポートされることを保証する。オブジェクト・ディスパッチャ７６４はオブジェクト要求をデータ整備モジュール７６６を介して、リモート呼び出しスケルトン７６８に受け渡す。データ整備モジュール７５８及びデータ整備モジュール７６６は、異なる言語により実装されるオブジェ

クトを含む相互運用環境のために、適切なデータ変換及び呼び出し変換機構を提供する。リモート呼び出しスケルトン７６８は、サーバ・オブジェクト７７０内のメソッドを呼び出し、適切な引き数をサーバ・オブジェクト７７０に受け渡す。

【００６２】

図９を参照すると、ＣＯＲＢＡサーバ内で実行されるEnterprise JavaBean（ＥＪＢ）上のリモート・ビジネス・メソッドを呼び出す方法を示すフロー図が示される。図７に関連して前述したように、Ｊａｖａクライアント・オブジェクトからＪａｖａアダプタ、ＯＲＢ及びＥＪＢスケルトンを介して所望のＥＪＢに至るプロセス・フローは、図９に示されるプロセス・フローに類似する。図９は、本発明のＪａｖａ及びＣＯＲＢＡ環境内の実際の呼び出し及びクラス操作を示す。

【００６３】

プロセスは、クライアント内のコードがリモート・オブジェクトの名前を探索するとき開始する（ステップ８０２）。クライアントは、ＥＪＢ内のビジネス・メソッドなど、リモート・サーバ内のオブジェクトのメソッドの実行を要求する。ルックアップ・メソッド内のコードが所望のメソッドを含むリモート・オブジェクトに対応するＣＯＲＢＡプロキシの名前を見いだす（ステップ８０４）。適切なＣＯＲＢＡプロキシが見いだされた後、その対応するＣＯＲＢＡプロキシのための適切なアダプタ・クラスが突き止められる（ステップ８０６）。ＣＯＲＢＡプロキシは適切なアダプタ・クラスによりラップ（wrap）され（ステップ８０８）、アダプタ・クラスが呼び出し側クライアント・コードに総称オブジェクトとして返却される（ステップ８１０）。クライアント・コードは、返却された総称オブジェクトを必要なクラス・タイプに固定する（ステップ８１２）。クライアント・コードは次に、新たに獲得されたオブジェクトから所望のメソッドを呼び出し（ステップ８１４）、新たに獲得されたアダプタ・クラス内の所望のメソッドが呼び出される（ステップ８１６）。呼び出されたアダプタ・クラス・メソッド内のコードは、ＣＯＲＢＡプロキシ内のその対応するメソッドを呼び出す（ステップ８１８）。一旦メソッドがＣＯＲＢＡプロキシ内で実行を開始すると、ＣＯＲＢＡインフラストラクチャが、ＣＯＲＢＡサーバ上のＣＯＲＢＡプロキシに対応するリモート・オブジェクトの呼び出しを処理する（ステップ８２０）。

【００６４】

図９の方法は、図８に示されるオブジェクト指向環境のために一般化され得る。ＣＯＲＢＡプロキシを通じてＥＪＢのメソッドを呼び出すのではなく、ソース・オブジェクト及びターゲット・オブジェクトがサーバ・オブジェクトのメソッドに一般化され、そのオブジェクト参照を通じて呼び出される。本発明の一般化された方法によれば、サーバ・オブジェクトに対するオブジェクト参照がアダプタによりラップされ、クライアント・オブジェクトによるサーバ・オブジェクト内のメソッドの呼び出しがアダプタにより透過的に処理される。クライアント・オブジェクトがサーバ・オブジェクトのメソッドを呼び出そうとするとき、メソッドは実際、アダプタ・オブジェクト内で呼び出される。アダプタ・クラスは本質的に、オブジェクト参照がクライアント・コードから分離されるように、オブジェクト参照をラップする。クライアントはオブジェクト参照について何も知らない。クライアントはただ、アダプタ・コードに"話しかける（talk to）"だけであり、他方、サーバ上のスケルトン・コードに"話しかける"オブジェクト参照は、クライアント・ベースのアダプタに関して何も知らない。

【００６５】

図１０乃至図１３を参照すると、ＪａｖａクライアントがＣＯＲＢＡサーバ内で実行されるEnterprise JavaBeanから、リモート・ビジネス・メソッドを呼び出す分散アプリケーションを記述するＪａｖａプログラミング言語ステートメントの例が示される。図１０では、Ｊａｖａプログラム内の標準のリモート・メソッド呼び出し（ＲＭＩ）技術の例が示される。CustomerImplクラスのリモート・オブジェクトは、インタフェース"CustomerInterface"により表されるビジネス・メソッドのセットを有し得る。CustomerInterface内に含まれるメソッドは、Ｊａｖａクライアントから遠隔的に呼び出され得る。この場合、ク

ライアント・コードはステートメント９０２乃至９０６に類似すると思われる。ステートメント９０２は、リモート・オブジェクトのオブジェクト参照を獲得するための命名サービスの使用を示す。オブジェクト"obj"は、CustomerInterfaceを実装するリモートＪａｖａオブジェクトのＲＭＩプロキシである。ステートメント９０２でオブジェクト参照を獲得後、ステートメント９０４で、オブジェクトを適切なオブジェクト・タイプに固定することによりオブジェクトが制限される。ステートメント９０６では、Ｊａｖａクライアント・コードがプロキシ・オブジェクト上のビジネス・メソッドをあたかもローカル・オブジェクトのように呼び出す。クライアントは、呼び出しがオブジェクト・リクエスト・ブローカ（ＯＲＢ）を用いて実施されることを知らない。プロキシ・オブジェクトはＣＯＲＢＡ規格により定義されるように、メソッド呼び出しをリモート・オブジェクトに転送する。

【００６６】

本発明の方法によれば、リモート・オブジェクトを実行するために、Ｊａｖａ内のＲＭＩ振舞いが、ＣＯＲＢＡ機構を用いてシミュレートされる。特殊なNamingContext及びアダプタ・クラスのセットが、図１１乃至図１３に示されるように実装される。特殊なNamingContextクラス内のルックアップ・メソッドが、図１１に示されるように実装される。ステートメント９１２は、特殊なルックアップ・メソッドの定義を示す。ステートメント９１４は、ＣＯＲＢＡ命名サービス内のルックアップ・メソッドがステートメント９１４内の所望のリモート・オブジェクトに対応するＣＯＲＢＡプロキシを見いだすために使用されることを示す。一旦ＣＯＲＢＡプロキシが見いだされると、ステートメント９１６が新たなアダプタ・クラスを作成する。ステートメント９１８は、ＣＯＲＢＡプロキシが適切なアダプタ・クラスによりラップされることを示す。ステートメント９２０は、ルックアップ・メソッドからの戻り値として、オブジェクト参照の返却を示す。

【００６７】

図１２は、図１０に示されるＲＭＩに類似のシミュレート化ＲＭＩのためのクライアント・コードの例を示す。ステートメント９２２は、リモート・オブジェクトの名前によるリモート・オブジェクトのルックアップすなわち探索を示し、図１１に示されるコードにより実装された新たなNamingContextクラスを使用する。一旦オブジェクト参照が返却されると、ステートメント９２４がオブジェクト参照がCostomerInterfaceクラスを用いて制限されるか、固定されなければならないことを示す。ステートメント９２６は、クライアント・コード内で呼び出される所望のビジネス・メソッドを示す。しかしながら、CustomerInterfaceクラスの"cust"インスタンス上で呼び出されるビジネス・メソッドは、実際には、新たに定義されたNamingCotextクラスにより返却されたアダプタ・オブジェクト上で呼び出される。

【００６８】

図１３は、ビジネス・メソッドのためのアダプタ・クラス内のコードを示す。ステートメント９３０はbusinessMethodメソッドの定義の開始を示す。ステートメント９３２は、アダプタ・コードが所望のメソッド呼び出しをＣＯＲＢＡプロキシに委託することを示す。この場合、プロキシはＣＯＲＢＡサーバ上のＥＪＢのスケルトンのための、ＣＯＲＢＡプロキシのＪａｖａクラス・インプリメンテーションである。

【００６９】

アダプタ・クラスは、ＣＯＲＢＡサーバ上のＥＪＢによりサポートされるビジネス・インタフェースを実装しなければならない。アダプタ・クラスは本来、ＣＯＲＢＡプロキシがＪａｖａクライアント・コードから分離されるように、ＣＯＲＢＡプロキシをラップする。重要な点は、ＪａｖａクライアントがＣＯＲＢＡプロキシに関して何も知らないことである。ＪａｖａクライアントはＪａｖａベースのアダプタ・コードにだけ話しかけ、他方、ＣＯＲＢＡサーバ上のスケルトン・コードに話しかけるＣＯＲＢＡプロキシは、Ｊａｖａベースのアダプタに関して何も知らない。

【００７０】

図１１乃至図１３に示される例では、リモート・オブジェクトのビジネス・メソッドのた

めの引き数が、単にアダプタからプロキシ・オブジェクトに受け渡される。引き数に対するデータ変換は示されない。ビジネス・メソッドがＥＪＢを引き数として受け取ったり、返却する場合、必要に応じて好適なアダプタにより、プロキシ・オブジェクトの適切なラップまたはラップ解除を実行することが、そのビジネス・メソッドのアダプタ・コードの義務である。図示の例では、戻り値を期待すること無く、ビジネス・メソッドがCustomer Interfaceクラスのために呼び出される。

【００７１】

ビジネス・メソッドが引き数として整数及びＥＪＢを受け取り、戻り値としてＥＪＢを返却する例では、ＥＪＢはラップ及びラップ解除されなければならない。

【００７２】

図１４乃至図１６を参照すると、Ｊａｖａプログラミング言語ステートメントの例が、Ｊａｖａメソッドへの引き数として使用されるＥＪＢをラップ及びラップ解除するプロセスを示す。図１４は、Ｊａｖａビジネス・メソッドの宣言の例であり、これは整数及びタイプ"Employee"のＥＪＢを引き数として受け取り、タイプ"Customer"のＥＪＢを戻り値として返却する。

【００７３】

図１５は、リモートＥＪＢ内のビジネス・メソッドに対するシミュレート化ＲＭＩの例を示す。図１１乃至図１３に示されるリモートＥＪＢの場合同様、図１５は前述のように、ビジネス・メソッドの引き数をラップ及びラップ解除する追加のステップを示す以外は、プロキシをアダプタによりラップ及びラップ解除する本発明の方法を使用する。ステートメント１０１０は"EmployeeName"の探索、及びオブジェクト"obj1"へのその割当てを示し、ステートメント１０１２は、オブジェクト"obj1"から、タイプ"Employee"のオブジェクト"ee"への制限を示す。ステートメント１０１４は"CustomerName"の探索を示し、ステートメント１０１６は、Customerクラス内のビジネス・メソッドの呼び出しを示し、ＥＪＢ引き数"ee"及び新たな顧客"newCust"の戻り値を有する。

【００７４】

EmployeeのＥＪＢを引き数として受け渡すことにより、ビジネス・メソッドがＪａｖａクライアントによりアダプタ上で呼び出されるとき、"Employee"インタフェースを実現するのはアダプタ・オブジェクトであり、アダプタ・コードがアダプタを"Employee ee"からラップ解除し、その内部のＣＯＲＢＡプロキシをＯＲＢを介して受け渡す。同様に、ＯＲＢを介して返却される結果の戻り値が実際、CustomerのＥＪＢのＣＯＲＢＡプロキシを含む。この場合、アダプタ・コードは返却されたプロキシをクライアントに返送する前に、それを適切なアダプタによりラップする。

【００７５】

図１６は、CutomerAdapterクラス内のアダプタ・コードの例を示し、これはＥＪＢ引き数を必要に応じてラップ及びラップ解除するためのステップを示す以外は、図１３に示される例と類似である。ステートメント１０２０は、アダプタ・クラス内のビジネス・メソッドの宣言を示し、整数"Ｉ"及び"Employee ee"を引き数として受け取り、Customerを戻り値として返却する。ステートメント１０２２は、引き数"ee"がタイプ"eeAdapter"として書き直されることを示す。ステートメント１０２４は、EmployeeAdapterからＣＯＲＢＡプロキシを獲得するために、"eeAdapter"がラップ解除されることを示す。ステートメント１０２６は、ビジネス・メソッドが"eeProxy"を引き数として、プロキシ・オブジェクト上で呼び出されることを示す。なぜなら、ＣＯＲＢＡプロキシ・オブジェクトだけがＯＲＢを介して受け渡されるべきであるからである。ステートメント１０２８は、適切なアダプタにより、返却されたプロキシ"custProxy"をラップし、"custAdapter"を獲得することを示す。ステートメント１０３０は、"custAdapter"がビジネス・メソッドの呼び出しからの戻り値として、呼び出し側クライアント・オブジェクトに返却されることを示す。このように、アダプタ・コードは、メソッド呼び出しをＣＯＲＢＡプロキシに委託することに加え、必要に応じてデータ変換を実行する。この場合、アダプタ・コードは、ビジネス・メソッド内で引き数として渡されるＥＪＢをラップ及びラップ解除する。更に、アダ

プタは、ＯＲＢインプリメンテーションによりサポートされないデータ・タイプも実行する。

【００７６】

図１７を参照すると、本発明の好適な方法に従い、適切なアダプタによりＥＪＢ引き数をラップ及びラップ解除するプロセスのフロー図が示される。プロセスは、クライアント・オブジェクトがビジネス・メソッドを呼び出し、様々な引き数をビジネス・メソッドに受け渡すとき開始する（ステップ１１０２）。この場合、クライアントは、ＣＯＲＢＡサーバ上のリモートＥＪＢのビジネス・メソッドのためのインタフェースを実現する、アダプタ・クラス内のメソッドを呼び出す。アダプタはビジネス・メソッドのために引き数のリストを解析し（ステップ１１０４）、必要に応じて引き数に対してデータ変換を実行する（ステップ１１０６）。引き数がＥＪＢを表すか否か、この場合、アダプタによりラップされたＥＪＢのためのＣＯＲＢＡプロキシを表すか否かが判断される（ステップ１１０８）。肯定の場合、アダプタ・コードがアダプタをＥＪＢ引き数からラップ解除し、ＣＯＲＢＡプロキシを獲得する（ステップ１１１０）。

【００７７】

引き数がＥＪＢでない場合、プロセスはこのメソッド呼び出しにおいて、処理されるべき追加の引き数が存在するか否かを判断する（ステップ１１１２）。存在する場合、プロセスはステップ１１０４に戻り、次の引き数を獲得する。もはや引き数が存在しない場合、アダプタはメソッド呼び出しをＥＪＢを表すＣＯＲＢＡプロキシにＯＲＢを介して委託する（ステップ１１１４）。

【００７８】

呼び出されたＣＯＲＢＡプロキシに対して、戻り値が存在するか否かが判断される（ステップ１１１６）。存在しない場合、アダプタは実行を完了し、制御フローをクライアント・コードに返却する（ステップ１１２４）。戻り値が存在する場合、戻り値がＥＪＢプロキシか否かが判断される（ステップ１１１８）。そうでない場合、戻り値がクライアントへの結果値として返却される（ステップ１１２２）。戻り値がＥＪＢプロキシの場合、アダプタは結果をクライアントに返却する前に、返却されたプロキシを適切なアダプタによりラップする（ステップ１１２０）。プロセスは次に完了し、クライアント・コードがその実行を継続する（ステップ１１２４）。

【００７９】

本発明の利点は、図面に関する前述の詳細な説明から明かであろう。Ｊａｖａは、Ｊａｖａクライアントがリモート・メソッド呼び出し（ＲＭＩ）と呼ばれる方法を使用して、別のプロセスで実行されるＪａｖａサーバ上のメソッドを呼び出す機構を提供する。しかしながら、ＣＯＲＢＡ準拠のサーバなど、サーバがＪａｖａ環境で実行されていない場合、Ｊａｖａクライアントは、サーバ上のメソッドに対するメソッド呼び出しを発行できない。なぜなら、ＪａｖａはＣＯＲＢＡオブジェクトと通信する固有の機構を提供しないからである。ＪａｖａクライアントとＣＯＲＢＡサーバ間で、データ整備、すなわち異なるタイプのデータのためのデータ変換を実行する標準的な機構は存在しない。これらの方法は、他のサーバ上で実行される他のＥＪＢに参照を受け渡すステップを含む。Ｊａｖａクライアントが、ＣＯＲＢＡサーバなどの非Ｊａｖａ環境で実行されるＥＪＢなどの別のＪａｖａアプリケーションと通信するための標準的な機構は存在しない。

【００８０】

本発明は、ＣＯＲＢＡサーバ内で実行されるEnterprise JavaBean（ＥＪＢ）のインタフェースを内観することにより生成されるアダプタを使用する。これらのアダプタはＪａｖａクライアント側に存在し、それらの内部にＥＪＢを実行するＣＯＲＢＡサーバのリモート・プロキシを保持する。アダプタは、ビジネス・メソッドを呼び出すためにＥＪＢにより指定されるインタフェースを実現するＪａｖａクラスである。アダプタはクライアントからの全てのビジネス・メソッド呼び出しをサーバ上のＣＯＲＢＡプロキシに委託し、ＪａｖａクライアントからＣＯＲＢＡプロキシへの、及びその逆のデータ整備を実行する。

【００８１】

クライアント・オブジェクトの見地から、アダプタは実際にＣＯＲＢＡサーバ上に存在するＥＪＢのレプリカである。クライアント・オブジェクトは、自身がオブジェクト・リクエスト・ブローカ（ＯＲＢ）を介してＣＯＲＢＡサーバ上のＥＪＢと通信することを意識しない。アダプタはそれ自身内にＥＪＢを実行するサーバのＣＯＲＢＡプロキシを記憶する。クライアントによりアダプタに発せられるあらゆるビジネス・メソッド呼び出しが適切なデータ変換の後に、アダプタによりＣＯＲＢＡプロキシに委託される。従って、アダプタは、Ｊａｖａクライアントとの間の透過的な接着剤として作用する。

【００８２】

重要な点は、本発明は完全機能型のデータ処理システムの状況において述べられてきたが、当業者であれば、本発明のプロセスが命令のコンピュータ読取り可能媒体の形態、及び様々な形態で配布され得、本発明が配布を実施するために実際に使用される特定のタイプの信号担持媒体に関係無しに、同様に当てはまることが理解できよう。コンピュータ読取り可能媒体の例には、フロッピー・ディスク、ハード・ディスク・ドライブ、ＲＡＭ、及びＣＤ－ＲＯＭなどの記録型媒体と、デジタル及びアナログ通信リンクなどの伝送型媒体とが含まれる。

【００８３】

まとめとして、本発明の構成に関して以下の事項を開示する。

【００８４】

（１）分散データ処理システム内の分散アプリケーションにおいて、サーバ・オブジェクトのメソッドを呼び出すプロセスであって、
前記サーバ・オブジェクトと異なるプログラミング・パラダイムで実装されるクライアント・オブジェクトを実行するステップと、
リモート・サーバ・オブジェクトのためのオブジェクト参照を獲得するステップと、
前記オブジェクト参照をアダプタ内でラップするステップと、
前記アダプタのメソッドを呼び出すステップと
を含む、プロセス。

（２）前記アダプタが前記オブジェクト参照を使用して、前記サーバ上のスケルトンのメソッドを呼び出す、前記（１）記載のプロセス。

（３）前記スケルトンが前記サーバ・オブジェクトのメソッドを呼び出す、前記（１）記載のプロセス。

（４）分散データ処理システムにおいて分散アプリケーションを実装する方法であって、
サーバ・オブジェクトのプロキシためのオブジェクト参照を獲得するステップと、
前記プロキシをアダプタ内でラップするステップと、
前記アダプタのメソッドを呼び出すステップと
を含む、方法。

（５）前記アダプタが、前記サーバ・オブジェクトによりサポートされるインタフェースを実装するＪａｖａクラスである、前記（４）記載の方法。

（６）前記サーバ・オブジェクトがEnterprise JavaBeanである、前記（４）記載の方法。

（７）前記オブジェクト参照が命名サービスから獲得される、前記（４）記載の方法。

（８）前記プロキシがＣＯＲＢＡプロキシである、前記（４）記載の方法。

（９）前記アダプタが前記ＣＯＲＢＡプロキシのメソッドを呼び出す、前記（８）記載の方法。

（１０）前記ＣＯＲＢＡプロキシがクライアント・コンピュータ上に存在するＪａｖａクラスである、前記（８）記載の方法。

（１１）前記ＣＯＲＢＡプロキシがメソッド要求をオブジェクト・リクエスト・ブローカに受け渡す、前記（８）記載の方法。

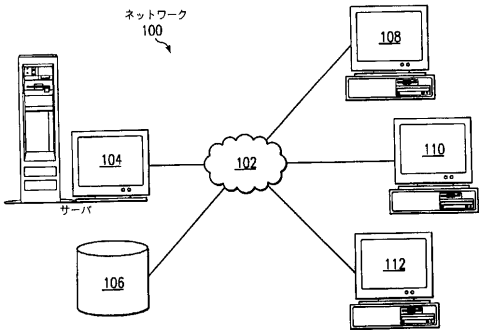
（１２）分散データ処理システム内の分散アプリケーションにおいて、サーバ・オブジェクトのメソッドを呼び出すデータ処理システムであって、

前記サーバ・オブジェクトと異なるプログラミング・パラダイムで実装されるクライアント・オブジェクトを実行する実行手段と、
リモート・サーバ・オブジェクトのためのオブジェクト参照を獲得する獲得手段と、
前記オブジェクト参照をアダプタ内でラップするラッピング手段と、
前記アダプタのメソッドを呼び出す呼び出し手段と
を含む、データ処理システム。
（１３）前記アダプタが前記オブジェクト参照を使用して、前記サーバ上のスケルトンのメソッドを呼び出す、前記（１２）記載のデータ処理システム。
（１４）前記スケルトンが前記サーバ・オブジェクトのメソッドを呼び出す、前記（１２）記載のデータ処理システム。
（１５）分散データ処理システムにおいて分散アプリケーションを実装するデータ処理システムであって、
サーバ・オブジェクトのプロキシためのオブジェクト参照を獲得する獲得手段と、
前記プロキシをアダプタ内でラップするラッピング手段と、
前記アダプタのメソッドを呼び出す呼び出し手段と
を含む、データ処理システム。
（１６）前記アダプタが、前記サーバ・オブジェクトによりサポートされるインタフェースを実装するＪａｖａクラスである、前記（１５）記載のデータ処理システム。
（１７）前記サーバ・オブジェクトがEnterprise JavaBeanである、前記（１５）記載のデータ処理システム。
（１８）前記オブジェクト参照が命名サービスから獲得される、前記（１５）記載のデータ処理システム。
（１９）前記プロキシがＣＯＲＢＡプロキシである、前記（１５）記載のデータ処理システム。
（２０）前記アダプタが前記ＣＯＲＢＡプロキシのメソッドを呼び出す、前記（１９）記載のデータ処理システム。
（２１）前記ＣＯＲＢＡプロキシがクライアント・コンピュータ上に存在するＪａｖａクラスである、前記（１９）記載のデータ処理システム。
（２２）前記ＣＯＲＢＡプロキシがメソッド要求をオブジェクト・リクエスト・ブローカに受け渡す、前記（１９）記載のデータ処理システム。
（２３）分散データ処理システム内の分散アプリケーションにおいて、サーバ・オブジェクトのメソッドを呼び出すためにデータ処理システム内で使用されるコンピュータ・プログラムを記憶したコンピュータ読取り可能記憶媒体であって、前記コンピュータ・プログラムは、
前記サーバ・オブジェクトと異なるプログラミング・パラダイムで実装されるクライアント・オブジェクトを実行する第１の命令と、
リモート・サーバ・オブジェクトのためのオブジェクト参照を獲得する第２の命令と、
前記オブジェクト参照をアダプタ内でラップする第３の命令と、
前記アダプタのメソッドを呼び出す第４の命令と
を含む、記憶媒体。
（２４）分散データ処理システムにおいて分散アプリケーションを実装するために、データ処理システム内で使用されるコンピュータ・プログラムを記憶したコンピュータ読取り可能記憶媒体であって、前記コンピュータ・プログラムは、サーバ・オブジェクトのプロキシのためのオブジェクト参照を獲得する第１の命令と、
前記プロキシをアダプタ内でラップする第２の命令と、
前記アダプタのメソッドを呼び出す第３の命令と
を含む、記憶媒体。
【図面の簡単な説明】
【図１】本発明が実装される分散データ処理システムを示す図である。
【図２】サーバとして実装され得るデータ処理システムを示すブロック図である。

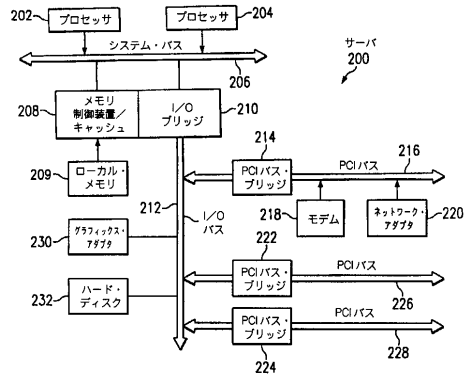
【図３】本発明が実装され得るデータ処理システムを示すブロック図である。
【図４】従来の分散アプリケーションを示す図である。
【図５】ＣＯＲＢＡ規格を使用する従来の分散アプリケーションを示す図である。
【図６】ＣＯＲＢＡを用いて相互運用性機能を提供する従来の分散アプリケーション内のコンポーネントを示す図である。
【図７】ＣＯＲＢＡサーバ内で実行されるＥＪＢのリモート・ビジネス・メソッドを呼び出す方法を実装するために使用されるコンポーネントを示すブロック図である。
【図８】サーバ・オブジェクトのリモート・メソッドを呼び出す方法を実装するために使用されるコンポーネントを示すブロック図である。
【図９】ＣＯＲＢＡサーバ内で実行されるＥＪＢ上のリモート・ビジネス・メソッドを呼び出す方法を示すフロー図である。
【図１０】ＪａｖａクライアントがＣＯＲＢＡサーバ内で実行されるＥＪＢからリモート・ビジネス・メソッドを呼び出すための分散アプリケーションを記述する、Ｊａｖａプログラミング言語文の例を示す図である。
【図１１】ＪａｖａクライアントがＣＯＲＢＡサーバ内で実行されるＥＪＢからリモート・ビジネス・メソッドを呼び出すための分散アプリケーションを記述する、Ｊａｖａプログラミング言語文の例を示す図である。
【図１２】ＪａｖａクライアントがＣＯＲＢＡサーバ内で実行されるＥＪＢからリモート・ビジネス・メソッドを呼び出すための分散アプリケーションを記述する、Ｊａｖａプログラミング言語文の例を示す図である。
【図１３】ＪａｖａクライアントがＣＯＲＢＡサーバ内で実行されるＥＪＢからリモート・ビジネス・メソッドを呼び出すための分散アプリケーションを記述する、Ｊａｖａプログラミング言語文の例を示す図である。
【図１４】Ｊａｖａメソッドへの引き数として使用されるＥＪＢをラップ及びラップ解除するプロセスを記述するＪａｖａプログラミング言語文の例を示す図である。
【図１５】Ｊａｖａメソッドへの引き数として使用されるＥＪＢをラップ及びラップ解除するプロセスを記述するＪａｖａプログラミング言語文の例を示す図である。
【図１６】Ｊａｖａメソッドへの引き数として使用されるＥＪＢをラップ及びラップ解除するプロセスを記述するＪａｖａプログラミング言語文の例を示す図である。
【図１７】本発明の方法に従う好適なアダプタによる、ＥＪＢ引き数のラップ及びラップ解除のフロー図である。
【符号の説明】
１００、２００、３００　分散データ処理システム
１０２　ネットワーク
１０４、２００、６１４、７２６、７７２　サーバ
１０６　記憶ユニット
１０８、１１０、１１２、６０２、７００、７５０　クライアント
２０２、２０４、３０２　プロセッサ
２０６　システム・バス
２０８　メモリ制御装置／キャッシュ
２０９　ローカル・メモリ
２１０　Ｉ／Ｏバス・ブリッジ
２１２　Ｉ／Ｏバス
２１４　周辺コンポーネント相互接続（ＰＣＩ）バス・ブリッジ
２１６、３０６　ＰＣＩローカル・バス
２１８、３２２　モデム
２２０　ネットワーク・アダプタ
２２２、２２４　ＰＣＩバス・ブリッジ
２２６、２２８　ＰＣＩバス
２３０　メモリマップド・グラフィックス・アダプタ

２３２　ハード・ディスク
３０４　主メモリ
３０８　ＰＣＩブリッジ
３１０　ローカル・エリア・ネットワーク（ＬＡＮ）・アダプタ
３１２　ＳＣＳＩホスト・バス・アダプタ
３１４　拡張バス・インタフェース
３１６　音声アダプタ
３１８　グラフィックス・アダプタ
３１９　スマートカード・アダプタ
３２０　キーボード及びマウス・アダプタ
３２４　メモリ
３２６　ハード・ディスク・ドライブ
３２８　テープ・ドライブ
３３０　ＣＤ－ＲＯＭドライブ
４００、５００、７５２　クライアント・オブジェクト
４０２、７７０　サーバ・オブジェクト
４０４　データベース
４０６　Ｊａｖａリモート・メソッド呼び出し（ＲＭＩ）プロトコル
４０８　レガシ・アプリケーション
４１０　現企業アプリケーション
５１０　通信リンク
５２０　ＣＯＲＢＡサーバ
５２１　JavaBean
５２２　Ｃ＋＋オブジェクト
５２３　Enterprise JavaBean（ＥＪＢ）
５２４　ＣＯＢＯＬオブジェクト
６０６、６１０　オブジェクト・リクエスト・ブローカ
６１２　スケルトン・コード
６１６　オブジェクト参照
６１８　メソッド要求
６２０　サーバント
７０２　Ｊａｖａ仮想マシンＪＶＭ
７０４　Ｊａｖａクライアント・オブジェクト
７０６　アダプタ
７０８　ＥＪＢインタフェース
７１０　ＣＯＲＢＡプロキシ
７１４、７１８　オブジェクト・リクエスト・ブローカＯＲＢ
７１６　ＩＩＯＰ
７２０　ＥＪＢスケルトン
７２２　ＪＮＩ
７２４　ＪＶＭ
７２８　リモートＥＪＢ
７５６　オブジェクト参照
７５８　データ整備モジュール
７６０、７６４　オブジェクト・ディスパッチャ
７６２　オブジェクト通信リンク
７６６　データ整備モジュール
７６８　リモート呼び出しスケルトン

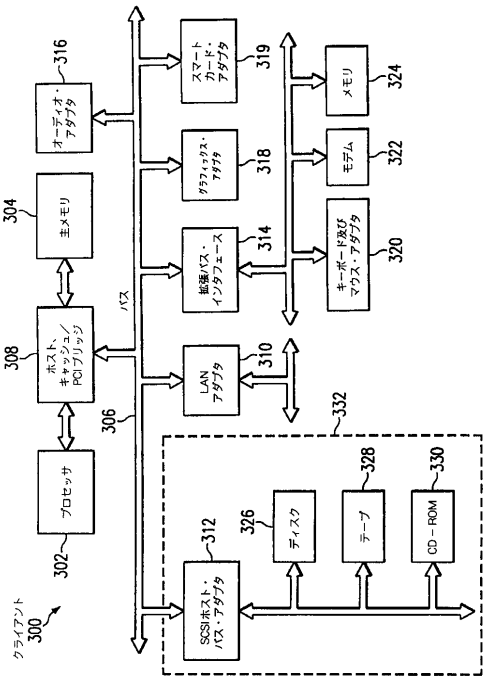
【図１】



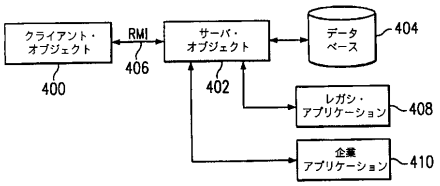
【図２】



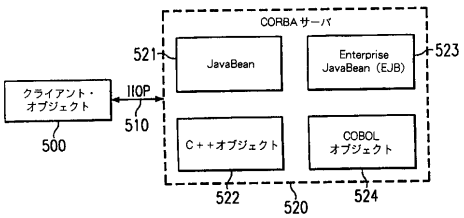
【図３】



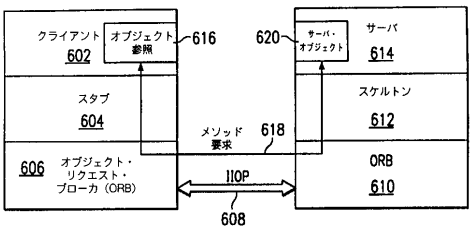
【図４】



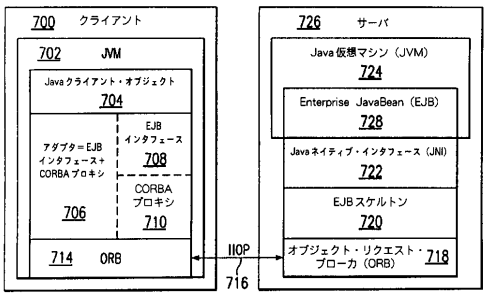
【図５】



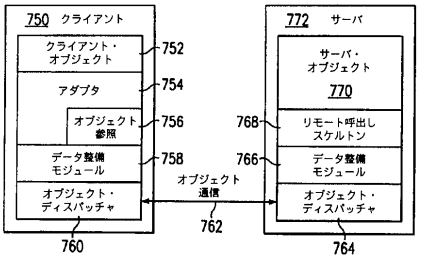
【図６】



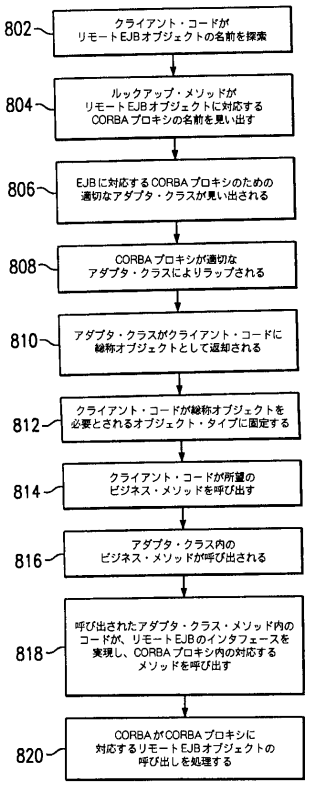
【図７】



【図８】



【図９】



【図１０】

```
    }
902 Object obj = JavaNamingContext.lookup ("RemoteObjectName");
904 CustomerInterface cast = (CustomerInterface) obj;
906 cust.businessMethod (args);
    }
```

【図１１】

```
912 public Object lookup(NameToBeLookedUp) {
914 CorbaProxy proxy = CorbaNamingServices.lookup(NameToBeLookedUp);
916 Class adapterClass = findAdapterClassForCorbaProxy(proxy);
918 Adapter obj = new adapterClass(proxy);
920 return obj
    }
```

【図１２】

```
    }
922 Object obj = RINamingContext.lookup("RemoteObjectName");
924 CustomerInterface cust = (CustomerInterface) obj;
926 cust.businessMethod(args);
    }
```

【図１３】

```
930 public void businessMethod(args) {
932 proxy.businessMethod(args)
    }
```

【図１４】

```
1000 public Customer businessMethod(integer I, Employee ee);
```

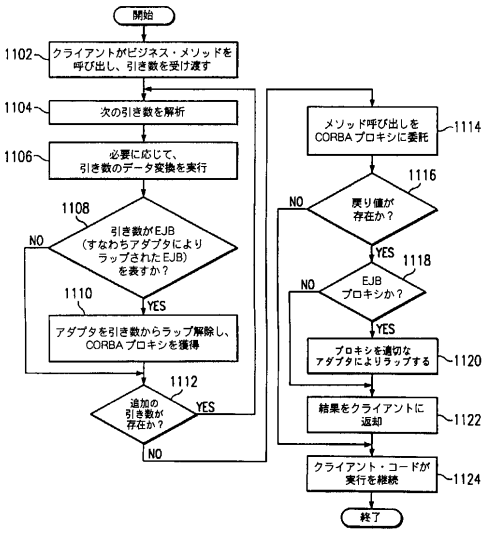
【図１５】

```
    {
1010 Object obj1= RINamingContext.lookup(EmployeeName);
1012 Employee ee= (Employee) obj1;
1014 Customer cust= (Customer)RINamingContext.lookup(CustomerName);
1016 Customer newCust= cust.businessMethod(1,ee);
    }
```

【図１６】

```
1020 public Customer businessMethod (integer I, Employee ee) {
1022 EmployeeAdapter eeAdapter = (EmployeeAdapter) ee;
1024 EmployeeProxy eeProxy = eeAdapter.unwrapAdapter();
1026 CustomerProxy custProxy = proxy.businessMethod(I,eeProxy);
1028 CustomerAdapter custAdapter = new CustomerAdapter(custProxy);
1030 return custAdapter;
    }
```

【図１７】



フロントページの続き

(72)発明者　アジャ・エイ・アプト
アメリカ合衆国７８７２８、テキサス州オースティン、ナンバー２３１５、ウエルス・ブランチ・パークウエイ　１８０１

合議体
審判長　吉岡　浩
審判官　野仲　松男
審判官　冨吉　伸弥

(56)参考文献　特開平９－５４６８５（ＪＰ，Ａ）
特表平１０－５０５６９３（ＪＰ，Ａ）
Ｊｉｍ　Ｆａｒｌｅｙ，小俣裕一監訳，”Ｊａｖａ分散コンピューティング”，株式会社オライリー・ジャパン，１９９８年9月２４日，ｐ．４７－８７
岩山知三郎，”ＩＢＭのＪａｖａソリューションの決定版　ＷｅｂＳｐｈｅｒｅ　Ａｐｐｌｉｃａｔｉｏｎ　Ｓｅｒｖｅｒ”，Ｃｏｍｐｕｔｏｐｉａ，株式会社コンピュータ・エージ社，１９９９年2月１９日（工業所有権総合情報館受入日），第３３巻，第３９０号，ｐ．３６－４３
岩山知三郎，”ビジネスオブジェクト推進協議会とOMGが提携国際標準に沿ったコンポーネント開発を推進”，Ｃｏｍｐｕｔｏｐｉａ，株式会社コンピュータ・エージ社，１９９９年2月１日，第３３巻，第３８９号，ｐ．６４－６６
齋藤眞人外，”ＦＯＲＥＦＲＯＮＴ　ｗｉｔｈ　Ｃｙｂｅｒｓｐａｃｅのシステムアーキテクチャ”，日立評論，１９９７年4月1日　，第７９巻，第4号，ｐ．９－１４
田中良明外，”異種分散ＰＤＭシステム間の設計製造情報共有方法”，電気学会研究会資料（産業システム情報化研究会），１９９８年１１月5日，第９８－４３／５２号，ｐ．１１－１６

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 15/16-15/177