(19)日本国特許庁(JP)　(12)公表特許公報(A)　(11)特許出願公表番号
特表2005-505055
(P2005-505055A)
(43)公表日　平成17年2月17日(2005.2.17)

(51)Int.Cl.[7]	F I		テーマコード（参考）
G06F 15/00	G06F	15/00　330A	5B085

審査請求 未請求　予備審査請求 有　（全 111 頁）

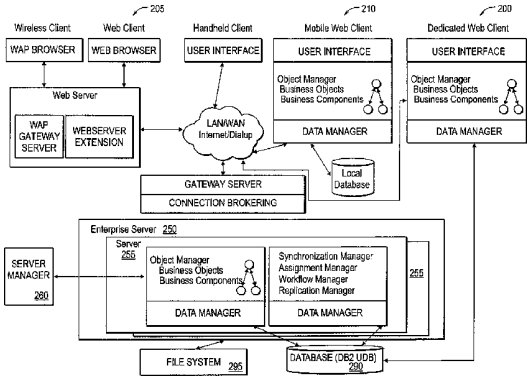
(21)出願番号	特願2003-533142(P2003-533142)
(86)(22)出願日	平成14年9月27日(2002.9.27)
(85)翻訳文提出日	平成16年3月29日(2004.3.29)
(86)国際出願番号	PCT/US2002/030885
(87)国際公開番号	WO2003/030005
(87)国際公開日	平成15年4月10日(2003.4.10)
(31)優先権主張番号	60/326,741
(32)優先日	平成13年9月29日(2001.9.29)
(33)優先権主張国	米国(US)
(31)優先権主張番号	10/254,384
(32)優先日	平成14年9月24日(2002.9.24)
(33)優先権主張国	米国(US)

(71)出願人　503353988
ジーベル　システムズ　インコーポレイテッド
アメリカ合衆国　カリフォルニア州　９４４０４　サン　マテオ　ブリッジポイント　パークウェイ　２２０７
(74)代理人　100082005
弁理士　熊倉　禎男
(74)代理人　100067013
弁理士　大塚　文昭
(74)代理人　100074228
弁理士　今城　俊夫
(74)代理人　100086771
弁理士　西島　孝喜

最終頁に続く

(54)【発明の名称】モバイルウェブクライアントに対する方法、装置及びシステム

(57)【要約】
モバイルウェブ（210）のための方法、システム及び装置を提供する。1つの実施例において、本発明は、ウェブアプリケーションを動作させる方法である。前記方法は、クライアント（210）からのリクエストを受信することを含んでいる。さらに、前記方法は、ウェブエミュレータを用いて前記リクエストを処理することを含んでいる。また、前記方法は、利用できるデータから前記リクエストを提供することを含んでいる。



【特許請求の範囲】
【請求項１】
クライアントウェブブラウザからのリクエストを受信する段階と、
ウェブエミュレータを用いて前記リクエストを処理する段階と、
利用できるデータから前記リクエストを提供する段階と、を備えているウェブアプリケーションを動作させる方法。
【請求項２】
前記利用できるデータが、ネットワークを介して利用できることを特徴とする請求項１に記載の方法。
【請求項３】
前記利用できるデータが、ネットワークに対するアクセスなしにローカルに利用できることを特徴とする請求項１に記載の方法。
【請求項４】
ネットワークを介して利用できるデータに対するアクセスを試行する段階をさらに備えている請求項１に記載の方法。
【請求項５】
前記ネットワークを介して成功したアクセスにおいて、前記リクエストを提供する段階が、前記ネットワークから利用できるデータを利用することを特徴とする請求項４に記載の方法。
【請求項６】
前記ネットワークを介して失敗したアクセスにおいて、前記リクエストを提供する段階が、前記ローカルデータからの利用できるデータを利用することを備えている請求項４に記載の方法。
【請求項７】
ネットワークを介して利用できるデータに対するアクセスを試行する段階を備え、
前記リクエストを提供する段階が、成功したアクセスにおいて前記ネットワークからの利用できるデータを利用し、失敗したアクセスにおいてローカルデータからの利用できるデータを利用することをさらに備えている請求項１に記載の方法。
【請求項８】
前記ウェブエミュレータがウェブサーバに対するプラグインであり、前記プラグインが前記リクエストをウェブエンジンに通過させ、前記ウェブエンジンがデータマネージャに関連して前記リクエストを提供することを特徴とする請求項１に記載の方法。
【請求項９】
前記利用できるデータがネットワークを介して利用できることを特徴とする請求項８に記載の方法。
【請求項１０】
前記利用できるデータがネットワークに対するアクセスなしにローカルに利用できることを特徴とする請求項８に記載の方法。
【請求項１１】
前記データマネージャが、ネットワークを介して利用できるデータに対するアクセスを試行することを備え、前記データマネージャが、成功したアクセスにおいて前記ネットワークからの利用できるデータを利用し、前記リクエストを提供するのに失敗したアクセスにおいてローカルデータから利用できるデータを利用することをさらに特徴とする請求項８に記載の方法。
【請求項１２】
前記リクエストがクライアントから前記ネットワークにデータを送信する段階を含むことを特徴とする請求項１に記載の方法。
【請求項１３】
前記リクエストが前記ネットワークからのデータを受信する段階を含むことを特徴とする請求項１に記載の方法。

【請求項１４】
前記リクエストがデータを送受信する段階をさらに含むことを特徴とする請求項１に記載の方法。
【請求項１５】
クライアントからのリクエストを受信する段階と、
ウェブエミュレータを用いて前記リクエストを処理する段階と、
データベースにおいて利用できるデータから前記リクエストを提供する段階と、を備えているデータベースに関連して利用するウェブアプリケーションを動作させる方法。
【請求項１６】
前記リクエストがクライアントウェブブラウザから生じることを特徴とする請求項１５に記載の方法。
【請求項１７】
前記利用できるデータがメインデータベースにおいてネットワークを介して利用できることを特徴とする請求項１５に記載の方法。
【請求項１８】
前記利用できるデータがネットワークに対するアクセスなしにローカルデータベースにおいてローカルに利用できることを特徴とする請求項１５に記載の方法。
【請求項１９】
ネットワークを介してメインデータベースにおいて利用できるデータに対するアクセスを試行することをさらに備えている請求項１５に記載の方法。
【請求項２０】
前記ネットワークを介して成功したアクセスにおいて、前記リクエストを提供する段階が、前記ネットワークを介して前記メインデータベースからの利用できるデータを利用することを特徴とする請求項１９に記載の方法。
【請求項２１】
前記ネットワークを介して失敗したアクセスにおいて、前記リクエストを提供する段階が、ローカルデータベースにおいてローカルデータからの利用できるデータを利用することを特徴とする請求項１９に記載の方法。
【請求項２２】
クライアントウェブブラウザからのリクエストを受信する手段と、
前記リクエストを処理する手段と、
利用できるデータからの前記リクエストを提供する手段と、を備えているウェブアプリケーションを動作させる装置。
【請求項２３】
プロセッサによって実行される場合に、前記プロセッサが、
クライアントウェブブラウザからのリクエストを受信する段階と、
ウェブエミュレータを用いて前記リクエストを処理する段階と、
利用できるデータから前記リクエストを提供する段階と、を備えている方法を実現できる、命令を組み入れているマシン読み取り可能な媒体。
【請求項２４】
前記プロセッサによって実行される場合に、前記プロセッサが、前記利用できるデータがネットワークを介して利用できる前記方法を実現できる、命令をさらに組み入れている請求項２３に記載のマシン読み取り可能な媒体。
【請求項２５】
前記プロセッサによって実行される場合に、前記プロセッサが、前記利用できるデータがネットワークに対するアクセスなしにローカルに利用できる前記方法を実現できる、命令をさらに組み入れている請求項２３に記載のマシン読み取り可能な媒体。
【請求項２６】
前記プロセッサによって実行される場合に、前記プロセッサが、
ネットワークを介して利用できるデータに対するアクセスを試行する段階を備える前記方

法であって、
前記リクエストを提供する段階が、成功したアクセスにおいて前記ネットワークから利用できるデータを利用し、失敗したアクセスにおいてローカルデータから利用できるデータを利用することをさらに備える前記方法を実現できる、命令をさらに組み入れている請求項２３に記載のマシン読み取り可能な媒体。
【請求項２７】
プロセッサによって実行される場合に、前記プロセッサが、
クライアントウェブブラウザからのリクエストを受信する段階と、
ウェブエミュレータを用いて前記リクエストを処理する段階と、
データベースにおいて利用できるデータから前記リクエストを提供する段階と、を備えている方法を実現できる、命令を組み入れているマシン読み取り可能な媒体。
【請求項２８】
前記プロセッサによって実行される場合に、前記プロセッサが、前記利用できるデータがメインデータベースにおいてネットワークを介して利用できる前記方法を実現できる、命令をさらに取り入れている請求項２７に記載のマシン読み取り可能な媒体。
【請求項２９】
前記プロセッサによって実行される場合に、前記プロセッサが、前記利用できるデータがネットワークに対するアクセスなしにローカルデータベースにおいてローカルに利用できる前記方法を実現できる、命令をさらに取り入れている請求項２７に記載のマシン読み取り可能な媒体。
【請求項３０】
前記プロセッサによって実行される場合に、前記プロセッサが、ネットワークを介してメインデータベースにおいて利用できるデータに対するアクセスを試行する段階をさらに備えている前記方法を実現できる、命令をさらに取り入れている請求項２７に記載のマシン読み取り可能な媒体。
【請求項３１】
前記プロセッサによって実行される場合に、前記プロセッサが、前記ネットワークを介して成功したアクセスにおいて前記リクエストを提供する段階が前記ネットワークを介して前記メインデータベースから利用できるデータを利用する前記方法を実現できる、命令をさらに取り入れている請求項３０に記載のマシン読み取り可能な媒体。
【請求項３２】
前記プロセッサによって実行される場合に、前記プロセッサが、前記ネットワークを介して失敗したアクセスにおいて前記リクエストを提供する段階がローカルデータベースにおいてローカルデータから利用できるデータを利用することをさらに備えている前記方法を実現できる、命令をさらに取り入れている請求項３０に記載のマシン読み取り可能な媒体。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、概して、データ処理の分野に関する。より詳細には、本発明は、モバイルウェブクライアントに対する方法、装置及びシステムに関する。
【背景技術】
【０００２】
本出願は、2001年9月29日に提出された“METHOD, APPARATUS AND SYSTEM FOR A MOBILE WEB CLIENT”の名称にかかる特許番号60/326,741を有する仮出願に対する優先権を主張する。
【０００３】
技術が進歩し続け、ビジネス環境がますます複雑かつ多様になるにつれて、より多くの会社が、様々な顧客関係管理（CRM）ソフトウェア及びeビジネスアプリケーションに依存してきており、それらの企業ビジネスの様々な側面を実行管理している。一般に、eビジネ

スアプリケーションは、その顧客、パートナー、供給者、分配者、従業員等を有する相互に作用するネットワーク（例えば、インターネット、イントラネット、エクストラネット等）を介して会社又は企業がそのビジネスを遂行できるように設計される。eビジネスアプリケーションは、コアビジネスプロセス、サプライチェーン、バックオフィスオペレーション、及びCRM機能を含んでいる。一般に、CRMは、会社が販売及び/又はサービスに対してその顧客と共に有する相互作用の様々な形勢を含んでいる。高いレベルでは、顧客関係管理は、顧客の要求を理解し、この理解を機能させて、売上を伸ばしサービスを改善することに集束される。CRMアプリケーション及びソフトウェアは、一般に、販売及びサービスの間の有効かつ効率のよい相互作用を提供するように設計され、顧客の周囲の会社の作業を統一し、顧客の満足を通して顧客共有及び顧客保有を増大させる。

【０００４】

典型的には、CRM導入戦略は、以下を考慮する必要がある。

【０００５】

知識管理：有効なCRM導入の重要な要素の１つは、顧客情報の獲得、その分析、共有及び追跡である。また、競争の優位性のために知識利用に対して絶対必要なのは、この知識の結果としてどのような行動が取られるかを従業員が理解することである。

【０００６】

データベース整理統合：有効かつ能率のよいCRMソリューションは、単一データベースでの顧客情報の整理統合及び顧客の周囲のビジネスプロセス改良である。ここで、目的は製造、マーケティング、販売及び顧客サポート作業を駆動させるために１つの場所に記憶される顧客と全ての相互作用を有することである。

【０００７】

チャネル及びシステムの統合：CRMｓアプリケーション/ソフトウェアが、それがeメール、電話、ウェブベースのユーザインターフェース等であろうとなかろうと、首尾一貫した顧客に対する応答能力、及びそれらの選択チャネルを介して高い品質の手法を提供することは非常に重要である。これは、顧客又は企業データベースを有する様々な通信チャネルのシームレスな統合を必要とする。また、それは、会社のビジネスシステム及びアプリケーションの他の部分とのCRMの統合を必要とする。

【０００８】

技術及びインフラストラクチャー：顧客サービスを強化するために、CRMアプリケーション/ソフトウェアは、オンライン顧客サービスを自動化して合理化する様々なツールを含んでいる。例えば、典型的には、自立モデルがツールの結合を利用して実現され得る（例えば、直感的サーチ能力を有する知識ベース、エージェント技術又は自動化されたeメール等）。

【０００９】

一般に、eビジネスアプリケーションは、組織が、ウェブ、コールセンター、フィールド、転売、小売り、及びディーラーネットワークを含む多数チャネルを介して顧客に販売し、購入し、供給するのをより簡単にする単一ソースの顧客情報を作ることができるように設計される。進歩したeビジネスは、典型的には、構成要素に基づいた構成で確立され、ウェブベースで設計され、モバイルクライアント、接続クライアント、シン・クライアント、及びハンドヘルドクライアント等を含む多数の計算プラットフォームの様々なタイプのクライアントに対してサポートするように設計される。

【００１０】

近年のウェブが急増している状況では、ウェブベース環境のeビジネスアプリケーションの機能性を提供することが望まれる。さらに、ユーザが既に熟知しているデスクトップベースのeビジネスアプリケーションのルックアンドフィールを保持することが、ウェブベース環境で機能するeビジネスアプリケーションに対して望まれる。

【００１１】

また、ウェブアプリケーションの出現で、前記アプリケーションの機能に対してインターネット又はワールドワイドウェブのようなネットワークを利用する依存状態に到達した。

しかしながら、ネットワーク環境又は状況と対応する非ネットワーク環境又は状況との両方を動作させる確固とした解決策は、ウェブアプリケーションのモバイルユーザに対して有用であることが評価されるであろう。前記ウェブアプリケーションは、すぐには利用できない。

【００１２】

本発明は、添付図の制限なしに例を通して説明される。

【００１３】

モバイルウェブクライアントに対する方法、装置及びシステムを描写する。以下の描写では、説明の目的で、多数の特定の詳細を記載し、本発明の完全な理解を提供する。しかしながら、当業者にとっては、本発明がこれらの特定の詳細なしに実行され得ることは明らかである。他の例としては、構造及び装置を、本発明を不明瞭にすることを避けるためにブロック図の形態で示している。

【００１４】

“１つの実施例”又は“ある実施例”に対する明細書における基準は、実施例と関連して描写される特定の特徴、構造、又は特性が、本発明の少なくとも１つの実施例に含まれていることを意味する。明細書中の様々な場所の“１つの実施例における”というフレーズの出現は、全てが必ずしも同じ実施例に対して言及していることではなく、他の実施例を相互に両立しない分離した又は択一的な実施例ということでもない。

【発明の開示】

【課題を解決するための手段】

【００１５】

システム概観及び全体構成

１つの実施例において、本発明の教示が実現されるシステムは、図１に示すようなマルチレイヤー構成として論理的に構築され得る。１つの実施例において、図１に示すような論理的なマルチレイヤー構成は、共通サービスに対するプラットフォームを提供し、様々なアプリケーションをサポートする。これらのサービスは、ユーザインターフェースレイヤー110、オブジェクトマネージャレイヤー120、データマネージャ130、及びデータ交換レイヤー140を含んでいる。

【００１６】

１つの実施例において、ユーザインターフェースレイヤー110は、アプレット、表示、チャート及びレポート等を提供し、1つ又は複数のアプリケーションと協働する。1つの実施例において、様々なタイプのクライアントは、ユーザインターフェースレイヤー110を介してサポートされ得る。これらのタイプのクライアントは、従来の接続クライアント、遠隔クライアント、イントラネットを介したシン・クライアント、Java（登録商標）シン・クライアント又は非ウインドウズ（登録商標）ベースのオペレーティングシステム、及びインターネットを介したHTMLクライアント等を含んでいる。

【００１７】

１つの実施例において、オブジェクトマネージャレイヤー120は、1つ又は複数のアプリケーションと協働する1つ又は複数のセットのビジネスルール又はビジネスコンセプトを管理するように設計され、ユーザインターフェースレイヤー110とデータマネージャレイヤー130の間のインターフェースを提供している。１つの実施例において、ビジネスルール又はコンセプトは、ビジネスオブジェクトとして表され得る。１つの実施例において、ビジネスオブジェクトは、アカウント、コンタクト、オポチュニティ、サービスリクエスト、ソリューション等のような様々なビジネスルール又はコンセプトを代表する設定可能なソフトウェアとして設計される。

【００１８】

1つの実施例において、データマネージャレイヤー130は、下位データの論理的表示を維持し、オブジェクトマネージャが、データが記憶される下位データの構造又はテーブルについて独立して機能するように設計されている。また、1つの実施例において、データマネージャ130は、データにアクセスするためにリアルタイムで構造化照会言語（structure q

uery language（SQL））の発生のような一定のデータベース照会機能を提供する。１つの実施例において、データマネージャ130は、データベーススキーマを定義するリポジトリファイル160のオブジェクト定義を動作させるように設計されている。１つの実施例において、データ記憶装置170は、１つ又は複数のアプリケーションと協働するデータモデルに対するデータ記憶を提供する。

【００１９】

1つの実施例において、データ交換レイヤーは、1つ又は複数の特定のターゲットデータベースと関連して処理し、データマネージャレイヤー130と下位データソースの間のインターフェースを提供するように設計されている。

【００２０】

図２は、本発明の教示が実現される１つの具体的なシステム配置のブロック図を示す。

【００２１】

1つの実施例において、マルチレイヤー構成により、1つ又は複数のソフトウェアレイヤーが異なるマシンに存在可能になる。例えば、１つの実施例において、ユーザインターフェース、オブジェクトマネージャ、及びデータマネージャは、全て、専用ウェブクライアントに存在し得る。ワイヤレスクライアントのような他のタイプのクライアントに対して、1つの実施例では、オブジェクトマネージャ及びデータマネージャは、システムサーバに存在し得る。図２に示すシステム配置は例証して説明する目的のものであり、本発明が教示する特定の実装及びアプリケーション次第で変化することは、当業者によって評価され、理解されるべきである。

【００２２】

1つの実施例において、図２に例証されるシステム環境は、2つ以上のデータベース290を含んでいる。1つ又は複数のデータベース290の部分は、複製マネージャによって作り出されるか又は複製される。加えて、モバイルウェブクライアントは、付加的な遠隔データベース（ローカルデータベースとも言う）を有し得る。1つの実施例において、モバイルウェブクライアントと協働する遠隔又はローカルなデータベースが、リードオンリーデータベースとして定義されない場合であって、各モバイルウェブクライアントがシステムサーバと同期する場合、これらのモバイルウェブクライアントは、最後にメインデータベースにまで伝搬されるデータをローカルに作り出し更新することができる。

【００２３】

1つの実施例において、データベース290は、定義済みデータスキーマ（例えば、テーブルオブジェクト、インデックスオブジェクト等）、リポジトリオブジェクト（例えば、ビジネスオブジェクト及びコンポーネント、表示定義、及びビジビリティルール等）、及びユーザの又は顧客のデータを含む様々なタイプのデータを記憶するように設計されている。1つの実施例において、専用ウェブクライアント及びサーバコンポーネントは、他のタイプのクライアントに関連して動作するものを含んでおり、直接、データベース290に接続し、リアルタイムで変更することができる。1つの実施例において、モバイルウェブクライアントは、一部のサーバのデータをダウンロードし、ローカルに利用でき、システムサーバを介して周期的にサーバデータベースと同期して、ローカル及びサーバデータベースの両方を更新することができる。

【００２４】

1つの実施例において、データベース290に含まれる様々なテーブルは、以下のタイプ：データテーブル、インターフェーステーブル、及びリポジトリテーブル等に論理的に組織化される。

【００２５】

1つの実施例において、データテーブルは、ユーザビジネスデータ、事務管理データ、シードデータ、及びトランザクションデータ等を記憶するのに用いられる。

【００２６】

1つの実施例において、これらのデータテーブルは、様々なアプリケーション及びプロセスを介して取り込まれ、更新される。1つの実施例において、データテーブルは、ベース

テーブル及びインターセクションテーブル等を含んでいる。1つの実施例において、ベーステーブルは、様々なアプリケーションによって定義されて利用される列を含んでいる。1つの実施例において、ベーステーブルは、ビジネスコンポーネントのテーブル特性において特定されるビジネスコンポーネントに対する列を提供するように設計されている。1つの実施例において、インターセクションテーブルは、2つのビジネスコンポーネント間の多対多の関係を実現するのに用いられるテーブルである。

【００２７】

また、それらはインターセクションデータ列を保持し、各関連付けに適する情報を記憶する。1つの実施例において、インターセクションテーブルは、データ構造に関連アプレットを提供する。

【００２８】

1つの実施例において、インターフェーステーブルは、ベーステーブルのグループを単一テーブルに正規化しないようにするために利用され、外部プログラムはそれにインターフェースで連結できる。１つの実施例において、それらは、データのエクスポート及びインポートに対する準備のための場所として利用される。

【００２９】

1つの実施例において、リポジトリテーブルは、以下を考慮する1つ又は複数のアプリケーションを特定するオブジェクト定義を含んでいる。

【００３０】

・クライアントアプリケーション構成
・データをインポート及びエクスポートするために用いられるマッピング
・モバイルクライアントに対する転送データのためのルール

【００３１】

1つの実施例において、ファイルシステム295は、アプリケーションサーバに位置し得るネットワークアクセス可能なディレクトリである。１つの実施例において、ファイルシステム295は、第三者のテキスト編集者によって作られるファイルのような様々なアプリケーションによって作り出される物理的ファイルを記憶し、他のデータはデータベース290に記憶されない。1つの実施例において、ファイルシステム295に記憶された物理的ファイルは、様々な命名規則の下で圧縮され、記憶され得る。1つの実施例において、専用ウェブクライアントは、ファイルシステム295へ及びから直接的にファイルを読み出しと書き込みを行うことができる。１つの実施例において、モバイルウェブクライアントは、ローカルファイルシステムを有し、それらは、周期的にサーバベースのファイルシステム290と同期する。１つの実施例において、ワイヤレスクライアント及びウェブクライアントのような他のタイプのクライアントは、システムサーバを介してファイルシステム290にアクセスすることができる。

【００３２】

1つの実施例において、企業サーバ250は、共通テーブル所有者又はデータベースを共有し、共通ゲートウェイサーバを指示し、サーバマネージャ260を利用するグループとして管理され得る、システムサーバ255の論理グルーピングである。1つの実施例において、ゲートウェイサーバに対する接続は、TCP/IPを介して確立され得る。1つの実施例において、企業サーバ250は、企業サーバ250の多数のシステムサーバ255を配置することによって効果的に拡張され、これにより、アプリケーションの中間層の高い拡張性を提供する。

【００３３】

1つの実施例において、サーバ255は、1つ又は多数のサーバプログラムを実行する。それは、入力処理リクエストを処理し、サーバの全プロセスの状態をモニターする。1つの実施例において、サーバプログラムは、データをインポート及びエクスポートすることを含み、データベースを構成し、ワークフロー及び処理自動化を遂行し、モバイルウェブクライアントをサポートするように処理し、ビジネスルールを実施する、1つ又は複数の特定の機能又はジョブ等を実現するために設計され構成される。1つの実施例において、サーバ255は、（ウィンドウズNTオペレーティングシステム下の）NTサーバ又はUNIX（登録商

標）オペレーティングシステム下のデーモン（例えば、バックグラウンドシェルプロセス）であり得る。1つの実施例において、サーバ255は、マルチプロセス及びマルチスレッドのコンポーネントの両方をサポートし、バッチ、サービス、及びインタラクティブモードのコンポーネントを動作させ得る。

【００３４】

1つの実施例において、サーバマネージャ260は、サーバ255及び企業サーバ250に対して異なるプログラムにわたる共通制御、管理、モニタリングを許可するユーティリティーとして構成される。1つの実施例において、サーバマネージャ260は、以下のタスク：スタート、ストップ、ポーズを実現するのに用いられ、サーバ255のコンポーネント及びタスクを再開し、ステータスをモニターし、多数のタスク、コンポーネント及び企業サーバ内のサーバに対する統計表を収集し、企業サーバ、個々のサーバ、個々のコンポーネント及びタスク等を構成する。

【００３５】

1つの実施例において、ゲートウェイサーバは、サーバにアクセスするための単一エントリポイントとして供する論理的エンティティとして構成され得る。それは、1つの実施例において、企業サーバにわたり強化された拡張性、ロードバランシング及び高稼働率を提供するのに用いられる。1つの実施例において、ゲートウェイサーバは、ネームサーバ及び接続ブローカコンポーネントを含んでいる。1つの実施例において、ネームサーバは、サーバと協働するパラメータのトラックを保持するように構成される。例えば、サーバと協働する稼働率及び接続性情報は、ネームサーバに記憶され得る。システムの様々なコンポーネントは、サーバの稼働率及び接続性を考慮している様々な情報をネームサーバに問い合わせることができる。ウィンドウズNT環境では、ネームサーバは、NTサービスとして実行され得る。UNIX環境では、ネームサーバは、デーモンプロセスとして実行され得る。1つの実施例において、接続ブローカコンポーネントは、クライアント接続リクエストを適当なサーバ（例えば、最も使用度の低いサーバ）に誘導するようにロードバランシング機能を実現するのに用いられる。

【００３６】

1つの実施例において、図２に例証するように、システムによってサポートされ得る様々なタイプのクライアントは、次のクライアント：専用ウェブクライアント、モバイルウェブクライアント、ウェブクライアント、ワイヤレスクライアント、及びハンドヘルドクライアント等を含んでいる。

【００３７】

1つの実施例において、専用ウェブクライアント（また、接続クライアントとも呼ばれる）は、LAN又はWAN接続を介したデータアクセスのためにデータベースサーバに直接的に接続される。1つの実施例において、これらの接続又は専用ウェブクライアントは、ローカルにデータを記憶しない。また、これらのウェブクライアントは、直接的にファイルシステムにアクセスできる。1つの実施例において、マルチレイヤー構成のユーザインターフェース、オブジェクトマネージャ、及びデータマネージャ層は、専用ウェブクライアントに存在する。

【００３８】

1つの実施例において、モバイルウェブクライアントは、ローカルデータアクセスに対して設計され、構成されており、それら自体のローカルデータベース及び/又はローカルファイルシステムを有する。1つの実施例において、モバイルウェブクライアントは、ゲートウェイサーバを介してシステム内の他のコンポーネントに相互に作用し得る。同期化を通して、ローカルデータベース及びサーバデータベースからの変更が、交換され得る。モバイルウェブクライアントについて、以下により詳細に描写する。

【００３９】

1つの実施例において、ウェブクライアントは、クライアントのマシンから標準ブラウザフォーマットで実行する。1つの実施例において、ウェブクライアントは、ウェブサーバを介してシステムサーバ255に接続し得る。1つの実施例において、システムサーバ255は

、データベース290及びファイルシステム295からのビジネスロジック及びアクセスデータを実行するように設計及び構成されている。1つの実施例において、この中で描写されるウェブクライアントは、本発明の教示に従って設計及び構成されており、インタラクティブモードで動作する。1つの実施例において、この中で描写されるように、相互に作用するウェブクライアント構成は、ブラウザ側のJavaスクリプトで実現される、動的に作られるオブジェクトを利用しており、サーバ側のオブジェクトに対応している。1つの実施例において、ブラウザ側のこれらの動的に作られるオブジェクトは、最新の表示及びその対応するアプレットである、最新のビジネスオブジェクト及びその対応するビジネスコンポーネントなどを含んでいる。ウェブクライアントについて、以下により詳細に描写する。

【００４０】

1つの実施例において、ワイヤレスクライアントは、本質的には、ワイヤレス装置において可能であるシン・クライアントである。ワイヤレスクライアントは、ユーザインターフェースに基づいて、システムサーバで情報/データを通信及び交換するワイヤレスアプリケーションプロトコル（WAP）を利用できる。

【００４１】

図2に例証するシステム構成について、説明例のような様々な構造、データベース、テーブル、ファイルシステム等を参照して、以下に、より詳細に例証する。

【００４２】

図３は、アプリケーションが本発明の教示に従って確立され得る、マルチレイヤー構成の別の論理表現を例証するブロック図を示す。再度、図３に例証されるようなマルチレイヤー構成は、様々なアプリケーションをサポートするように設計及び構成される様々な共通サービスをプラットフォームに提供する。1つの実施例において、これらの様々なサービスは、アプレットマネージャ及びユーザインターフェース層310に対応するプレゼンテーションサービス論理層315、オブジェクトマネージャ（OM）層320及びデータマネージャ（DM）層330に対応するアプリケーションサービス論理層325、及びデータベース層340に対応するデータサービス論理層345を含んでいる。

【００４３】

1つの実施例において、プレゼンテーションサービス315は、様々なタイプのクライアントをサポートするように設計及び構成されており、それらにユーザインターフェースアプレット、表示、チャート、及びレポート等を提供する。上述したように、ワイヤレスクライアント、ハンドヘルドクライアント、ウェブクライアント、モバイルウェブクライアント、及び専用（接続された）クライアント等を含めて、非常に多様なクライアントがサポートされる。

【００４４】

1つの実施例において、アプリケーションサービス325は、ビジネスロジックサービス及びデータベースインタラクションサービスを含んでいる。1つの実施例において、ビジネスロジックサービスは、ビジネスオブジェクト及びビジネスコンポーネントのクラス及びビヘイビアを提供する。1つの実施例において、データベースインタラクションサービスは、ビジネスコンポーネントからのデータに対するユーザインターフェース（UI）リクエストを受け取り、リクエストを満足させるのに必要なデータベースコマンド（例えば、SQL照会等）を発生させる。例えば、データインタラクションサービスは、DBMSの特定SQLステートメントに対するリクエストデータを翻訳するのに用いられる。

【００４５】

1つの実施例において、データ記憶サービス345は、様々なアプリケーションの基礎として提供する下位データモデルにデータ記憶を提供するように設計及び構成されている。例えば、データモデルは、eファイナンス、e保険、eコミュニケーション、及びeヘルスケア等のような様々な垂直産業生成及びアプリケーションと同様である、コールセンター、販売、サービス、及びマーケティング等を含む様々なソフトウェア生成及びアプリケーションをサポートするように設計及び構成されている。

【００４６】

図４は、本発明の教示が実現されるアプリケーション構成の1つの実施例のブロック図を説明する。図４に示されるように、アプリケーション構成は、様々なタイプのサービス及び様々なタイプのツールの様々なロジックグルーピングを含んでおり、ビジネスニーズ及び環境に基づいた特定アプリケーションを設計及び構成するのに用いられ得る。

【００４７】

1つの実施例において、核となるサービスは、アプリケーションが実行する構成を提供するように設計及び構成されている。1つの実施例において、核となるサービスは、以下を含んでいる。

【００４８】

・中間層アプリケーションサーバである企業サーバ
・これらの部分の全てを共にリンクするネットワーク
・様々なアプリケーションと他の外部アプリケーションの間と同様に様々なアプリケーションの多数インストールの間でデータを共有することを許可するイベント管理及びデータ複製のような機能
・認証及びアクセス制御であるセキュリティ機能

【００４９】

1つの実施例において、アプリケーション統合サービスは、様々なアプリケーションがこの構成に従って確立され得るように設計及び構成され、外部の世界と通信する。1つの実施例において、この論理グルーピングの様々なタイプのサービスは、リアルタイム、ニアリアルタイム、外部アプリケーションを有するバッチ統合を提供するように設計及び構成されている。例えば、これらの統合サービスは、利用できる方法、技術、及びソフトウェア製品を用いて外部アプリケーションと内部アプリケーションの間での通信を可能にするために利用される。1つの実施例において、アプリケーション統合サービスにより、システム又はアプリケーションは、他の外部企業アプリケーションを有するデータを共有し複製することができる。従って、これらのサービスにより、特定のアプリケーション又はシステムは、情報をリクエストするクライアントとそれからリクエストされる情報を有するサーバの両方であり得る。

【００５０】

1つの実施例において、ビジネスプロセスサービスは、クライアントがアプリケーションを介してビジネスプロセスを自動化できるように設計及び構成されている。1つの実施例において、これらの様々なビジネスプロセスサービスは、以下を含んでいる。

【００５１】

・アサインマネージャを介したタスクの割当て
・ワークフローマネージャを介したビジネス実務の施行
・ビジネスサービスを介したカスタムビジネスロジックの再使用
・適正プロダクト構成の保証、及びプロダクトコンフィギュレータ及びプライシングコンフィギュレーターを介した評価

【００５２】

1つの実施例において、これらのビジネスプロセスの創造は、パーソナリゼーションデザイナー、ワークフローデザイナー、スマートスクリプトデザイナー、アサインメントアドミニストレーション表示、及びモデルビルダー等のような実行時間ツールを介して行われ得る。

【００５３】

1つの実施例において、統合サービスは、クライアントにユーザインターフェース及びシン・クライアントサポートを提供するように設計及び構成されている。1つの実施例において、これらは、ウェブベースのアプリケーションを確立し維持する可能性を含んでおり、高度なスマートスクリプト等と同様に、ユーザプロフィール管理、コラボレーションサービス、及びEメールとファックスサービスのようなウェブサポート機能を提供する。

【００５４】

1つの実施例において、設計時ツールは、カスタマイズして設計し、統合ポイントを提供

し、アプリケーションを維持するサービスを提供するように設計及び構成されている。これらの様々なツールは、アプリケーションを定義するために1つの共通な場所を提供する。

【００５５】

1つの実施例において、管理サービスは、アプリケーション環境をモニターし、管理する1つの場所を提供するように設計及び構成されている。1つの実施例において、これらのサービスにより、ユーザは、グラフィックユーザインターフェース（GUI）又はコマンドライン等のいずれかを介してアプリケーションを管理することができる。

【００５６】

システム構成又はインフラストラクチャー

図５Ａは、本発明の1つの実施例に従う、インタラクティブウェブクライアント205及び図２のモバイルウェブクライアント210をサポートするための典型的なシステム構成又はインフラストラクチャー500を例証する。図５Ｂは、図５Ａに示した典型的なシステム構成又はインフラストラクチャー500の択一的な概観を例証する。

【００５７】

構成又はインフラストラクチャー500は、（図２に示した）インタラクティブウェブクライアント205及び（図２にも示した）モバイルウェブクライアント210をサポートしており、ウェブクライアント及びモバイルウェブクライアントの対話性及び性能を向上させて、共通する動作に対してページ再生数を低減させるような一定基準を満たし得る。

【００５８】

構成又はインフラストラクチャー500は、オブジェクトマネージャによって管理される、対応するオブジェクト504に良く似たブラウザ上に動的に作られるオブジェクト502を含み得る。1つの実施例において、オブジェクトマネージャ（OM）によって管理されるオブジェクト504は、オブジェクト指向パラダイムをサポートするC++のようなプログラミング言語を利用して確立され得る。

【００５９】

図５Ａ及び図５Ｂに示すように、OMによって管理される典型的なオブジェクト504は、CSSWEView506と表示しているオブジェクト506を含み得る。表示は、一般に、アプレットの特定の配置を構成するディスプレイパネルである。1つの実施例において、1つのアクティブな表示は、任意の所定時間に表示され得る。OMによって管理される別の典型的なオブジェクトは、CSSWEApplet508のアプレットを表すオブジェクト508であり得る。アプレットは、一般に、表示部分としてスクリーン上に現れるビジュアルアプリケーションユニットである。OMによって管理される他の典型的なオブジェクトは、ビジネスコンポーネント（CSSBusComp510）を表すオブジェクト510、ビジネスオブジェクト（CSSBusObj512）を表すオブジェクト512、フレーム（CSSWEFrame514）を表すオブジェクト514を含み得る。1つの実施例において、ビジネスオブジェクトは、アカウント、コンタクト、オポチュニティ、サービスリクエスト、ソリューション等のような様々なビジネスルール又はコンセプトの設定変更可能なソフトウェア表現として設計される。この実施例において、ビジネスコンポーネントは、典型的には、テーブルを重ねるレイヤーを提供し、それは下位テーブルよりもむしろアプレットリファレンスビジネスコンポーネントである。加えて、フレームは、一般に、表示のサブコンポーネントであり、1つ又は複数のアプレットから成る。

【００６０】

1つの実施例において、ブラウザ上のオブジェクト502は、Javaスクリプトを使って確立され得る。図５Ａ及び５Ｂに示すように、ブラウザ側の典型的なオブジェクト502は、ミラーCSSBusObj512、CSSBusComp510、CSSWEView506、及びCSSWEApplet508に対して、それぞれJSSBusObj516、JSSBusComp518、JSSView520、及びJSSApplet522を含んでおり、OMによって管理されるオブジェクト504である。

【００６１】

ブラウザ上のオブジェクト502及びOMによって管理されるオブジェクト504は、1つのコンピュータ計算装置又は多数のコンピュータ計算装置上に存在して動作するように構成され

得る。図６Ａは、ブラウザ上のオブジェクト502及びOMによって管理されるオブジェクト504が、クライアント602及びサーバ604を含む多数のコンピュータ計算装置602、604上に存在して動作する典型的な構成600を例証する。図６Ｂは、ブラウザ上のオブジェクト502及びOMによって管理されるオブジェクト504が、1つのコンピュータ計算装置652上に存在して動作する典型的な構成650を例証する。

【００６２】

図５Ａ及び５Ｂに戻ると、ブラウザ上のオブジェクト502は、一般に、OMによって管理される、対応又は反映されているオブジェクト504と同期する。同期化は、遠隔手続き呼び出し（RPC）メカニズム528及び通知メカニズム530を介して成し遂げられ得る。RPCメカニズム528及び通知メカニズム530については、以下により詳細に描写する。

【００６３】

ブラウザのオブジェクト502について、JSSApplicationオブジェクト524は、典型的にはユーザセッションにわたり存在する。JSSApplicationオブジェクト524は、ユーザがアプリケーションをスタートする際に最初にロードされるべきである。アプリケーションは、一般に、ユーザがデスクトップのアイコンから又はスタートメニューからアプリケーションの一部を起動する際に開始される。JSSApplicationオブジェクト524は、一般に、CSSModelオブジェクト534と類似する役割を実現する。CSSModelオブジェクト534は、一般に、メモリ上の最新のビジネスオブジェクトインスタンス、最新ビジネスオブジェクトとそれに含まれるビジネスコンポーネントの間の関係、及びユーザのグローバルステート情報を利用するリポジトリオブジェクトに対するアクセスを提供するグローバルセッションオブジェクトである。CSSModelオブジェクト534は、一般に、リポジトリ532にアクセスし、必要とされる情報を得る。リポジトリ532は、一般に、１組のオブジェクト定義であり、1つのアプリケーション又はひと続きのアプリケーションを定義するのに用いられる。しかしながら、JSSApplicationオブジェクト524は、一般に、1つの表示、トラックされる表示に関連するアプレット、1つのビジネスオブジェクト、及び表示に利用するビジネスコンポーネントをトラックするために縮小される。

【００６４】

JSSApplicationオブジェクト524と異なり、JSSViewオブジェクト520、JSSAppletオブジェクト522、JSSBusObjオブジェクト516及びJSSBusCompオブジェクト518は、典型的には、一時的又は永続しないエンティティであって、一般には、ページ再生が起こる場合に置き換えられる。例えば、新しい表示をナビゲートするリクエストは、ブラウザ上で実行するために作られることになる新しい組のJSSView520、JSSApplet522、JSSBusObj516及びJSSBusComp518オブジェクトを生じさせる。

【００６５】

従って、ブラウザ上のオブジェクト502は、一般には、OMによって管理される反映される又は対応するオブジェクト504の軽量表現として描写され得る。ブラウザ上の各オブジェクト502は、典型的には、OMによって管理される対応するオブジェクト504に含まれる機能性の一部を含んでいる。例えば、JSSViewオブジェクト520は、CSSViewオブジェクト506に類似し、一般には、アプレットの収集を表す。JSSBusObjオブジェクト516は、CSSBusObjオブジェクト512に類似しており、一般には、アクティブなビジネスコンポーネント間の様々な一対多対応の関係を管理し、その結果、これらのアクティブなビジネスコンポーネントが照会を介して取り込まれる場合に正確な関係が使用される。JSSBusOBjオブジェクト516は、一般に、最新表示が終わるまでブラウザ上に存在し、対応するCSSBusObjオブジェクト512との同期を保持すべきである。

【００６６】

1つの実施例において、ブラウザは、ウェブエンジン526に対する新しい表示にナビゲートするためにリクエストを提出する場合、ウェブエンジン526は、データが全くない表示レイアウトを含むレスポンスを送信する。その後、ウェブエンジン526は、表示を取り込むためにデータの文字列を含むレスポンスを送信する。

【００６７】

JSSApplicationオブジェクト524は、一般には、ブラウザ上のオブジェクトへ及びから流れる通信を管理する。1つの実施例において、ブラウザのオブジェクトで起動される方法は、起動方法がOMによって管理されるオブジェクト504に再び向けられるべきである場合、典型的には、JSSApplicationオブジェクト524に誘導される。JSSApplicationオブジェクト524は、一般には、ウェブエンジン526からOMによって管理される適切なオブジェクト504に起動方法を経路指定するためにRPCメカニズム528を利用する。ウェブエンジン526は、典型的には、OMによって管理されるオブジェクト504からブラウザ上のオブジェクト502に戻り通知及びデータを送信するのに使用される。ウェブエンジン526は、一般には、JSSApplicationオブジェクト524からブラウザ上のオブジェクト502に通知及びデータを経路指定するために通知メカニズム530を利用する。

【００６８】

ブラウザオブジェクト502は、一般には、OMによって管理されるオブジェクト504における方法を起動するために遠隔手続き呼び出し528を利用する。これらの遠隔手続き呼び出し528は、一般には、HTTPリクエストとして実装される。OMによって管理されるオブジェクト504からのレスポンスは、通知及び関連ステータス情報とデータを含むHTTPレスポンスとして実装される。1つの実施例において、遠隔手続き呼び出しは、ブロッキング動作可能な状態でなされ、ブラウザ上のオブジェクト502とOMによって管理されるオブジェクト504の間の同期を保証する。ブロッキング動作可能である場合、典型的には、制御により、呼び出された遠隔手続きが実行終了するまで、呼び出しコードに対して逆向きに通過しない。

【００６９】

遠隔手続き呼び出し（RPC）

RPCモデルは、一般に、従来のプログラムと同様の手続き抽象化を利用するが、手続き呼び出しが、2つのコンピュータ間の境界に及ぶことを許可する。図７は、RPCパラダイムが、どのようにプログラム700を個別のコンピュータ計算装置702、704上で実行される部分に分割するのに用いられるかについての例を示す。この図は、一般には、多数の手続きを有する分散プログラムを示す。Main()706，proc_1()708，proc_2()710，proc_3()712，proc_5()714，proc_6()716，及びproc_7()718は、第1のコンピュータ計算装置又はクライアント702に存在して動作し、proc_4()720，proc_8()722は、第2のコンピュータ計算装置又はサーバ704に存在し動作する。手続きnから手続きmへの実線は、nからmへの呼び出しを示す。破線726は、制御が、遠隔手続き呼び出し中に1つのコンピュータ計算装置から別のコンピュータ計算装置へどのように通過させるのかを示す。

【００７０】

図８は、遠隔手続き呼び出しで用いられる実行の典型的なモデル800を例証する。この図において、実線724は、一般には、コンピュータ計算装置内のフロー制御を示すのに用いられ、破線726は、一般には、制御が、遠隔手続き呼び出し中に1つのコンピュータ計算装置から別のコンピュータ計算装置にどのように通過させるかを示すのに用いられている。

【００７１】

上記のように、遠隔手続き呼び出しは、一般には、呼び出しコードから分離したアドレス空間に位置する手続きを実行する。RPCモデルは、一般には、ローカルな手続き呼び出しのプログラミングモデルから導出され、全ての手続きが手続き宣言を含むという事実を利用する。手続き宣言は、呼び出しコードと呼び出される手続きの間のインターフェースを定義する。手続き宣言は、呼び出しシンタックス及び手続きのパラメータを定義する。手続きに対する呼び出しは、典型的には、手続き宣言に従う。

【００７２】

遠隔手続き呼び出しを利用するアプリケーションは、ローカルアプリケーションに非常に類似した動作をする。しかしながら、RPCアプリケーションは、2つの部分：1つ又は複数の組の遠隔手続きを提供するサーバと、RPCサーバに対して遠隔手続き呼び出しをするクライアントに分割される。サーバ及びそのクライアントは、一般には、分離したシステムに存在し、ネットワークを介して通信する。RPCアプリケーションは、RPC実行時間ライブ

ラリに依存し、それらに対するネットワーク通信を制御する。RPC実行時間ライブラリは、一般には、クライアントに対してサーバを見つけ、サーバを管理するような追加のタスクをサポートする。

【００７３】

分散アプリケーションは、一般には、中央処理装置（CPU）、データベース、装置、及びサービスのような分散コンピュータ計算リソースを利用する。以下のアプリケーションは、分散アプリケーションの例であって、

権限のあるユーザが他のユーザの個人のカレンダーにアクセスできる、カレンダー管理アプリケーションと、

CPUのデータを処理し、ワークステーションに結果を表示するグラフィックスアプリケーションと、

異なるコンピュータに位置する設計、在庫、スケジューリング、及び財務会計プログラムの間のアセンブリ構成要素についての情報を共有する製造アプリケーションと、を示す。

【００７４】

RPCソフトウェアは、一般には、

適切なサーバを認定するクライアントと、

異種の環境で動作させるためのデータ変換と、

ネットワーク通信と、を含んでいる分散アプリケーションの基本的な必要性を満たさなければならない。

【００７５】

分散アプリケーションは、通信を管理し、サーバを認定し、セキュリティを提供する等ようなタスクを含んでいる。独立型の分散アプリケーションは、全てのこれらのタスク自体を実現する必要がある。これらの分散コンピュータ計算タスクに対する適切なメカニズムなしに、分散アプリケーションを書き込むことは、難しく、高価であり、エラーを生じ易い。

【００７６】

RPCソフトウェアは、典型的には、呼び出されたRPCスタブ、及びアプリケーションに対する分散コンピュータ計算タスクを実現するタスクRPC実行時間ライブラリであるコードを提供する。RPCスタブ及びRPC実行時間ライブラリは、RPCアプリケーションを形成するクライアント及びサーバアプリケーションコードとリンクされるべきである。

【００７７】

表１は、一般には、分散アプリケーションのクライアント及びサーバに対する基本的なタスクを示す。手続きを呼び出すこと、及び遠隔手続きを実行することは、強調された文字で示され、（ローカルアプリケーションのような）アプリケーションコードによって実現されるが、それらは、ここでは、クライアント及びサーバアドレス空間に存在する。他のタスクはどうかといえば、幾つかは、スタブ及びRPC実行時間ライブラリによって自動的に実現され、一方で、他の幾つかは、アプリケーションコードのAPI呼び出しを介してRPC実行時間ライブラリによって実現される。

【００７８】

表１：RPCアプリケーションの基本的なタスク

クライアントタスク		サーバタスク	
		1.	ネットワークプロトコルを選択する
		2.	RPCインターフェースを登録する
		3.	エンドポイントマップのエンドポイントを登録する
		4.	ネームスペースのRPCインターフェース及びオブジェクトを通知する
		5.	呼び出しに従う
6.	手続きを提供する、整合するサーバを見つける		
7.	遠隔手続きを呼び出す		
8.	サーバとの結合を確立する		
9.	入力アーギュメントをネットワークデータに変換する		
10.	サーバの実行時間に対するアーギュメントを送信する		
		11.	呼び出しを受信する
		12.	ネットワークデータを分解し、入力アーギュメントをローカルデータに変換する
		13.	呼び出された手続きを検出し、起動する
		14.	遠隔手続きを実行する
		15.	出力アーギュメントを変換し、値をネットワークデータに返す
		16.	クライアントの実行時間に対する結果を送信する
17.	結果を受信する		
18.	ネットワークデータを分解し、出力アーギュメントをローカルデータに変換する		
19.	呼び出しコードに結果及び制御を返す		

【００７９】

図９は、一般には、RPCの典型的なパーティション900について、アプリケーションコードセグメント904及び914、RPCインターフェース906及び916、クライアント及びサーバスタブ908及び918、及びRPCクライアント902及びRPCサーバ904におけるRPC実行時間ライブラリ910及び920を示す。

【００８０】

RPCクライアント902又はRPCサーバ912は、典型的には、RPCアプリケーションコードセグメント904及び914、RPCインターフェース906及び916、スタブ908及び918、及びRPC実行時間ライブラリ910及び920を含んでいる。RPCアプリケーションコードセグメント904、914は、一般には、アプリケーション開発者によって特定のRPCアプリケーションに対して書かれたコードである。RPCアプリケーションコードセグメント904、914は、一般には、遠隔手続きを実現し、呼び出し、また、RPC実行時間ライブラリにおいて必要とされるルーチン又は手続きを呼び出す。RPCスタブ908、918は、アーギュメントを通過させて受信するためにRPCインターフェース906、916を利用するインターフェース特定コードモジュー

ルである。クライアント902及びサーバ912は、典型的には、各共有RPCインターフェース906、916に対して補足的なRPCスタブ906、916を含んでいる。RPC実行時間ライブラリ910、920は、一般には、RPCアプリケーションに対する通信を管理する。加えて、RPC実行時間ライブラリ910、920は、RPCアプリケーションが、RPCアプリケーションコードによって利用されるアプリケーションプログラミングインターフェース（API）をサポートし、それらの通信をセットアップし、サーバについて情報を操作し、遠隔にサーバを管理してセキュリティ情報にアクセスするようなオプションタスクを実現することができるようにする。

【００８１】

RPCアプリケーションコードセグメント904、914は、通常、クライアント及びサーバに対して異なる。サーバ912におけるRPCアプリケーションコード914は、典型的には、1つのRPCインターフェースを実現する遠隔手続きを含んでいる。対応するクライアント902におけるRPCアプリケーションコード904は、典型的には、それらの遠隔手続きに対する呼び出しを含んでいる。

【００８２】

RPCスタブ908、918は、一般には、遠隔手続き呼び出しに対する基本的なサポート機能を実現する。例えば、RPCスタブ908、918は、異なる形態のデータ表現を有するシステム間の送信のための入力及び出力アーギュメントを備えている。RPCスタブ908、918は、RPC実行時間ライブラリ910、920を利用して、クライアント902とサーバ904の間の送信を処理する。また、クライアント902におけるRPCスタブ908は、ローカルRPC実行時間ライブラリ910を利用して、クライアント902に対して適切なサーバを見つける。

【００８３】

図１０は、一般には、クライアントデータ構造とサーバデータ構造の間の典型的なマーシャリング及びアンマーシャリングを示す。クライアントRPCアプリケーションコードが遠隔手続きを呼び出す場合、クライアントスタブ908は、送信のために入力アーギュメント1002を準備すべきである。送信のためにアーギュメントを準備する処理は、“マーシャリング”として知られている。

【００８４】

マーシャリング1004は、一般には、入力又は呼び出しアーギュメント1002をバイトストリームフォーマットに変換し、送信のためにそれらを実装する。呼び出しアーギュメントを受信した場合、サーバRPCスタブ918は、それらをアンマーシャリングする（1014）。アンマーシャリング1014は、一般には、スタブが入力ネットワークデータを分解し、それをローカルシステムが認識するフォーマットを用いてアプリケーションデータに変換する。マーシャリング1004、1016及びアンマーシャリング1014、1006は、両方とも、各遠隔手続き呼び出しに対して2倍発生する。クライアントRPCスタブ908は、入力アーギュメント1002をマーシャリングし（1004）、出力アーギュメント1008をマーシャリングする（1006）。サーバRPCスタブ918は、入力アーギュメント1006をアンマーシャリングし（1014）、出力アーギュメント1008をマーシャリングする（1016）。マーシャリング及びアンマーシャリングは、クライアント及びサーバシステムが、同値のデータに対する異なるデータ表現を利用できるようにする。例えば、クライアントシステムは、ASCIIデータ1002、1008を利用でき、サーバシステムは、図１０に示すようなユニコードデータ1018を利用できる。

【００８５】

IDLコンパイラ（アプリケーション開発のためのツール）は、アプリケーション開発者によって記述されたRPCインターフェース定義をコンパイルすることによってスタブを発生させる。コンパイラは、プラットフォーム独立型IDLデータタイプに対するマーシャリング及びアンマーシャリングルーチンを発生させる。RPCアプリケーションに対するクライアントを確立するために、開発者は、クライアントアプリケーションコードをアプリケーションが利用する全てのRPCインターフェースのクライアントスタブとリンクさせる。サーバを確立するために、開発者は、サーバアプリケーションコードを対応するサーバスタブとリンクさせる。

【００８６】
1つ又は複数のRPCスタブに加えて、各RPCサーバ及びRPCクライアントは、RPC実行時間ライブラリのコピーとリンクされる。RPC実行時間ライブラリは、一般に、クライアントとサーバの間の通信を制御し、リクエストに関してクライアントに対してサーバを見つけるような実行時間オペレーションを提供する。クライアント及びサーバのRPCスタブは、典型的には、クライアント及びサーバに対してそれぞれローカルであるRPC実行時間ライブラリを介してアーギュメントを交換する。クライアントのRPC実行時間ライブラリは、典型的には、遠隔手続き呼び出しをサーバに送信する。サーバのRPC実行時間ライブラリは、一般には、遠隔手続き呼び出しをクライアントから受信し、各呼び出しをサーバの適切なRPCスタブにタスク指名する。RPC実行時間ライブラリは、その後、各呼び出しの結果をクライアントのRPC実行時間ライブラリに送信する。
【００８７】
また、サーバのRPCアプリケーションコードは、サーバがスタートアップ及びシャットダウンしている場合には、サーバにおけるRPC実行時間ライブラリのルーチンを呼び出すサーバ初期化コードを含んでいなければならない。また、クライアントのRPCアプリケーションコードは、初期化のために、RPC実行時間ライブラリルーチンを呼び出すことができる。また、さらに、サーバのRPCアプリケーションコード及びクライアントのRPCアプリケーションコードは、RPCスタブサポートルーチンに対する呼び出しを含み得る。RPCスタブサポートルーチンにより、一般には、アプリケーションはメモリの割当て及び開放をするようなプログラミングタスクを管理することができる。
【００８８】
図１１は、一般には、遠隔手続き呼び出し期間のRPCアプリケーションコードセグメント904及び914、RPCインターフェース906及び916、RPCスタブ908及び918、及びRPC実行時間ライブラリ910及び920の典型的な役割を示す。クライアントのアプリケーションコード又は呼び出しコード908は、遠隔手続き呼び出しを起動し、クライアントのRPCインターフェース906からクライアントスタブ908に入力アーギュメント1002を通過させる。クライアントスタブ908は、入力アーギュメント1002をマーシャリングし、呼び出しをクライアントのRPC実行時間ライブラリ910にタスク指名する。クライアントのRPC実行時間ライブラリ910は、入力アーギュメント1002をサーバのRPC実行時間ライブラリ920に送信し、呼び出された手続きのRPCインターフェース916のために、呼び出しをサーバスタブ918にタスク指名する。サーバのスタブ918は、入力アーギュメント1002をアンマーシャリングし、それらを呼び出された遠隔手続き914に通過させる。サーバのアプリケーションコード又は遠隔手続き914は、実行し、その後、任意の出力アーギュメント1008をサーバスタブ918に返す。サーバスタブ918は、出力アーギュメント1008をマーシャリングし、それらをサーバのRPC実行時間ライブラリ920に返す。サーバのRPC実行時間ライブラリ920は、出力アーギュメント1008をクライアントのRPC実行時間ライブラリ910に送信し、それらをクライアントスタブ901にタスク指名する。クライアントのスタブ908は、出力アーギュメント1008をアンマーシャリングし、それらを呼び出しコード904に返す。
【００８９】
1つの実施例において、遠隔手続き呼び出しは、ブロッキング動作可能な状態で、ブラウザのオブジェクト502と（図５Ａ及び５Ｂに示される）OMによって管理されるオブジェクト504の間の同期化を保証する。ブロッキング動作可能な状態で、制御は、典型的には、呼び出された遠隔手続きが実行を終えるまで、呼び出しコードに逆向きに通過しない。
【００９０】
図１２は、一般には、分散RPCアプリケーションを確立する典型的なプロセス1200の概略を記す。プロセスは、一般には、以下の基本タスク：
アプリケーションを設計し、どの手続きが必要とされるか、必要とされる手続きのどれが遠隔手続きであるかを決定し、遠隔手続きがRPCインターフェースにグループ化される方法を決定するタスク（ブロック1205）と、
ユニバーサルユニーク識別子（UUID）発生器を利用して、RPCインターフェースの各々に

対するUUIDを発生させるタスク（ブロック1210）と、
インターフェース定義言語（IDL）を利用して、計画データタイプ及び遠隔手続きに対するRPCインターフェースを記述するタスク（ブロック1215）と、
IDLコンパイラを用いて前記IDL記述をコンパイルすることによってクライアント及びサーバスタブを発生させるタスク（ブロック1220）と、
RPCスタブに適合するプログラミング言語を利用してアプリケーションコードを書き込むか又は変更し、その結果、アプリケーションコードが前記スタブを処理するタスク（ブロック1225）と、
オブジェクトコードをアプリケーションコードから発生させるタスク（ブロック1230）と、
前記アプリケーションコードから発生させられたローカルRPC実行時間ライブラリ及びオブジェクトコードをリンクさせて、実行可能なコードを発生させるタスク（1235）と、を含んでいる。

【００９１】

図１３は、一般には、ローカルRPC実行時間ライブラリのリンク及びアプリケーションコードから発生するオブジェクトコードを例証する。クライアントに対しては、クライアントスタブ908、クライアントアプリケーションコード又は呼び出しコード904、及びクライアントのRPC実行時間ライブラリ910のオブジェクトコードが、リンカー1302を用いてリンクされ、クライアント実行可能コード1304を発生させる。サーバに対しては、サーバスタブ918、サーバの初期化コード1308、サーバのアプリケーションコード又は遠隔手続き914、及びサーバのRPC実行時間ライブラリ916に対するオブジェクトコードが、リンカー1302を用いてリンクされ、サーバ実行可能コード1306を発生させる。

【００９２】

従来、呼び出しコード及び呼び出される手続きは、同じアドレス空間を共有している。RPCアプリケーションにおいて、呼び出しコード及び呼び出される遠隔手続きは、リンクされず、むしろ、それらは、間接的に、RPCインターフェースを介して通信する。RPCインターフェースは、一般には、オペレーション、データタイプ、1組の遠隔手続きを短縮するのに供する定数のロジカルグルーピングである。RPCインターフェースは、典型的には、インターフェース定義言語（IDL）を用いてアプリケーション開発者によって書き込まれた正式なインターフェース定義からコンパイルされる。

【００９３】

分散アプリケーションを開発する際、インターフェース定義は、IDLで定義されるべきである。IDLコンパイラは、一般には、インターフェース定義を利用し、ヘッダファイル、クライアントスタブファイル、及びサーバスタブファイルを発生させる。IDLコンパイラは、標準プログラミング言語でヘッダファイルを生成し、ソースファイルとして又はオブジェクトファイルとしてスタブを発生し得る。幾つかのアプリケーションに対しては、インターフェース定義に付随する属性構成ファイル（ACF）が、定義される。ACFが存在する場合、IDLコンパイラは、それがインターフェース定義をコンパイルする時にACFを解釈する。ACFの情報は、コンパイラが発生するコードを変更するのに用いられる。

【００９４】

各RPCインターフェースのヘッダは、典型的には、ユニバーサルユニーク識別子（UUID）を含み、唯一の結果が出るようにエンティティを識別する16進数である。RPCインターフェースを識別するUUIDは、一般には、インターフェースUUIDとして知られている。インターフェースUUIDは、インターフェースが唯一の結果が出るように全ての可能性のあるネットワーク構成にわたって識別され得ることを保証する。インターフェースUUIDに加えて、各RPCインターフェースは、大小のバージョンナンバーを含んでいる。同時に、インターフェースUUID及びバージョンナンバーは、インターフェース識別子を形成し、時間の経過と共にシステムを横切るRPCインターフェースのインスタンスを識別する。

【００９５】

通知

図５Ａ及び５Ｂに振り返ると、ブラウザのオブジェクト502は、一般には、OMによって管理される、対応又は反映されているオブジェクト504と同期しており、変化が反映され得る。同期化は、遠隔手続き呼び出し（RPC）メカニズム528及び通知メカニズム530を介して実現され得る。

【００９６】

データ又はステータスがOMによって管理される対応するオブジェクト504（例えば、CSSBusComp 510）に変化する場合、通知メカニズム530は、一般には、ブラウザ上のオブジェクト502のデータ（例えば、JSSBusComp 518）が更新され得る手段を提供する。1つの実施例において、CSSSWEViewオブジェクト506は、1つ又は複数の通知を収集し、それらを表示サイクルの終端においてブラウザ上のオブジェクト502に送信する。

【００９７】

1つの実施例において、以下の典型的又は例示的通知が送信される。以下に列挙される通知の幾つかは、パラメータを必要とすることに注目すべきである。他の場合には、構成は、その最新状態を単純に認識することによってこれらの通知に対するコンテキストを理解し得る。

【００９８】

NotifyBeginNotifysは、1組の通知の開始を指示する。

【００９９】

NotifyEndNotifysは、1組の通知の終了を指示する。

【０１００】

NotifyStateChangedは、状態が変化していることを指示する。状態変化が生じる典型的なシナリオは、システムが照会状態に入る（ユーザが照会状態に入ることができる場合に照会を実行する）場合であり、その後、（ユーザがデータ値を更新できる場合に）引き渡し保留状態に入る。

【０１０１】

NotifyBeginQueryは、ビジネスコンポーネントが照会状態を受け入れる準備ができた状態にあることを指示する。

【０１０２】

NotifyExecuteは、ビジネスコンポーネントが実行される（すなわち、データベースの照会を実行し、それ自体を最新データと共に再度取り込む）ことを指示する。この通知は、照会明細を提供するオプションパラメータを含む。

【０１０３】

NotifyEndQueryは、照会が完了したことを指示する。

【０１０４】

NotifyDeleteRecordは、記録がデータベースから削除されたことを指示する。

【０１０５】

NotifyDeleteWorkSetは、記録がワーキングセットから取り除かれたことを指示する。通知は、取り除かれるのに必要とされるワーキングセット列のインデックスを提供するパラメータを含む。

【０１０６】

NotifyInsertWorkSetは、記録が最新のワーキングセットに追加されたことを指示する。この通知は、ワーキングセットの新しい記録のインデックスを提供するパラメータを含み得る。

【０１０７】

NotifyInsertWSFieldValsは、一定値がワーキングセットにおけるフィールドに追加されるのに必要であることを指示する。

【０１０８】

NotifyNewASctiveFieldは、新しいフィールドがアクティブであるか又は最新であることを指示する。

【０１０９】

NotifyNewActiveRowは、新しい列（すなわち記録）がアクティブであるか又は最新であることを指示する。

【０１１０】

NotifyNewDataは、データの変化があったことを指示する。

【０１１１】

NotifyNewDataWSは、ワーキングセットのデータの変化があったことを指示する。

【０１１２】

NotifyNewFieldDataは、特定のフィールドが新しいデータを有することを指示する。通知は、フィールド名を提供するパラメータを含む。

【０１１３】

NotifyNewFieldQuerySpecは、特定のフィールドが新しい照会又は探索明細を有することを指示する。

【０１１４】

NotifyNewPrimaryは、新しい記録が最初の記録になったことを指示する。

【０１１５】

NotifyNewRecordは、新しい記録が作成されたことを指示する。

【０１１６】

NotifyNewRecordDataは、新たに作成された記録が新しいデータを有することを指示する。

【０１１７】

NotifyNewRecordWSは、新たに作成された記録がワーキングセットに取り込まれる新しいデータを有することを指示する。

【０１１８】

NotifyNewSelectionは、記録の選定及び再選定を指示する。

【０１１９】

NotifyNewSellsは、多重記録の選定を指示する。

【０１２０】

NotifyPageRefreshは、UIが再生されるのに必要であることを指示する。この通知は、UIにおいて単純な更新データよりもむしろ一般に実質的な変化が必要とされる場合に、典型的には用いられる。この通知は、サーバトリップが新しいページを取り出し、それをブラウザに表示することを生じさせ得る。

【０１２１】

NotifyScrollDataは、記録が、スクロールアップ又はダウンされるのに必要であることを指示する。リストアプレットのような一定のユーザインターフェースオブジェクトは、この通知を利用して、異なる列を示すためにスクロールアップ及びダウンすることができる。

【０１２２】

NotifyChangeSelectionは、記録選定の変化があった（すなわち、最新列が選定又は再選定された）ことを指示する。

【０１２３】

NotifySelModeChangeは、選定モードの変化を指示する。1つの実施例において、（ｉ）ある時間における1つの記録の選定、及び（ii）同時に多重記録の選定（例えば、多重記録を1つのコマンドと共に削除すること）を含んでいる、選定の2つのモードが存在し得る。

【０１２４】

NotifyTotalsChangedは、合計値が変更される必要があることを指示する。多重記録が表示される幾つかの場合、幾つかのフィールドはまた、全ての記録における値の和を表示する。この通知は、合計値が変更されたことを指示する。

【０１２５】

NotifyLongOpProgressは、長い（すなわち時間消費）動作が進行していることを指示する。この通知は、ユーザインターフェースによって利用され、ユーザにフィードバックを提

供し、タスクのほとんどがどのように容易に完了するかを示している進行又はステータスバーを示している。

【０１２６】

NotifyGeneric-これは、幾つかの特定状況のユーザインターフェースオブジェクトを通知するのに用いられる汎用通知であり、上記で列挙され描写された通知のセットによって保護されない。各タイプの汎用通知は、汎用通知に対する名称を提供するパラメータを含み、あるタイプの汎用通知は、別のタイプの汎用仕様と識別され得る。各タイプの汎用通知は、その特定パラメータを含み得る。

【０１２７】

一般の通信プロセス

上述したように、ブラウザ上のオブジェクト及びOMによって管理されるオブジェクトは、1つ又は多数のコンピュータ計算装置上に存在して動作するように構成され得る。上記で示したように、図６Ａは、典型的な構成600を例証しており、ブラウザ上のオブジェクト502及びOMによって管理されるオブジェクト504は、クライアント602及びサーバ604を含む多数のコンピュータ計算装置602及び604上に存在して動作する。図６Ｂは、典型的な構成650を例証しており、ブラウザ上のオブジェクト502及びOMによって管理されるオブジェクト504は、1つのコンピュータ計算装置652上に存在して動作する。

【０１２８】

図１４は、一般には、図６Ａに示す多数の装置構成600で動作している、ブラウザ側又はクライアント側のオブジェクト502とサーバ側オブジェクト504の間の通信の典型的プロセス1400を例証する。図１４の通信の典型的プロセス1400は、典型的には、ユーザ入力によって、ブラウザ側のアプレットJSSAppletで最初に処理される（ブロック1405）。ユーザがリンク又は他のユーザインターフェース機構においてクリックする場合、起動手段が発生する。JSSAppletは、典型的には、発生した起動手段を受信するための第1のオブジェクトである（ブロック1410）。JSSAppletは、その後、JSSApplicationを介して遠隔手続き呼び出しを発行し、サーバ側のアプレットCSSWEAppletに対する起動手段を再度ターゲットとする（ブロック1415）。ターゲットとなるサーバ側のアプレットCSSWEAppletは、一般には、ステータスフラグを設定することによって、ブラウザ側のアプレットJSSAppletからのRPC起動手段に応答し得る（ブロック1420）。1つの実施例において、ステータスフラグは、次の値の1つに設定され得る。

【０１２９】

Continue-この値は、一般には、サーバ側のアプレットCSSWEAppletが、起動手段の処理のその共有を実現したこと（又は実行する役割を有しないこと）、及びブラウザ上のJSSAppletが動作を完了するのに必要となることを指示する。通知は、応答において提供されるが、多くの場合は無意味である。

【０１３０】

Completed-この値は、一般には、サーバ側のアプレットが起動手段の処理を完了したこと、及びブラウザが応答で提供される通知に対する応答に対するもの以外にさらなる動作を実現する必要がないことを指示する。

【０１３１】

NewPage-この値は、一般には、起動リクエスト又は他のコマンドに対する応答が、ブラウザ側の全ての一時的なオブジェクトの再発生を含む、ブラウザのページ再生を必要とすることを指示する。URLは、ブラウザに送信され、新しいページが得られる。しかしながら、通知は存在しない。この値は、典型的には、異なる表示に対するドリルダウンが要求されるような場合に設定される。

【０１３２】

Error-この値は、一般には、起動手段リクエストが失敗したことを指示する。エラーステータスを受信する際には、JSSAppletは、典型的には、エラーページを表示する。

【０１３３】

ブラウザ側のアプレットであるJSSAppletがRPCを介してサーバを呼び出す場合、ブラウザ

側のアプレットは、典型的には、応答のステータスフラグを見て、その後、それを処理する（ブロック1425）。戻りステータスがエラーである場合、ブラウザ側のアプレットは、エラーページを示している。戻りステータスが完了する場合、サーバは、一般には、それがもう既に起動手段を処理し、ブラウザに対して行うことが残されていないことを指示している。戻りステータスが存続している場合、サーバは、一般には、それが起動手段を処理していないことを指示している。ブラウザ側のアプレットは、一般には、手段を直接的にJSSBusCompオブジェクト上で起動することによって手段をJSSBusCompオブジェクトに再度誘導することにより、存続の戻りステータスに対して応答する。JSSBusCompオブジェクトは、起動手段リクエストを満足させることができ、JSSApplicationを介してその対応するサーバ側のビジネスコンポーネントにそのRPC呼び出しを送信しなければならない。

【０１３４】

図１５は、ブラウザ側のアプレットがJSSBusCompオブジェクトにおいて直接的に手段を起動する通信の典型的なプロセス1500を示す。図１５のプロセス1500は、プロセスが図６Ａに示した多数の装置構成において発生したかのように、以下に示されていることに注目すべきである。ブロック1505において、ブラウザ側のアプレットJSSAppletは、手段をJSSBusCompオブジェクトに再度誘導する。ブロック1510において、クライアント側のビジネスコンポーネントオブジェクトJSSBusCompが、JSSApplicationオブジェクトを介してサーバ側のビジネスコンポーネントCSSBusCompに遠隔手続き呼び出しを発行する。サーバ側のビジネスコンポーネントは、一般には、RPC手段呼び出しを処理し、ステータスを設定して返信し、また、適当な場合に1組の通知を戻す（ブロック1515）。1つの実施例において、ステータスフラグは、次の値を有する。

【０１３５】

Completed-この値は、一般には、サーバ側のビジネスコンポーネントCSSBusCompが、一般には、起動手段を首尾よく処理したことを指示する。通知は、典型的には生じる。

【０１３６】

Error-この値は、一般には、サーバ側のビジネスコンポーネントCSSBusCompが、首尾よく起動手段呼び出しを処理したことを指示する。エラーの戻りステータスを受信する際に、ブラウザは、典型的にはエラーページを表示する。

【０１３７】

ブロック1520において、クライアント側のビジネスコンポーネント（JSSBusComp）は、戻りステータスフラグを調査し、適切に応答する。起動手段呼び出しが、ブラウザとサーバのビジネスコンポーネントの間の同期を必要とする動作を実現する場合に、サーバは交信されることに注目すべきである。さらに、JSSBusCompオブジェクトが、サーバと交信することなしに、ローカルにリクエストを処理することができる状況があり得ることに注目すべきである。前記状況の例は、ユーザが次の記録動作を実現し、その後、任意データを変更することなしに事前の記録動作を実現する場合である。

【０１３８】

択一的なクライアント-サーバ実施例

以下の描写の幾つかの部分は、コンポーネントによって与えられ、システム又は装置のコンポーネント、コードのオブジェクト、コードの他の一部、プログラムの一部、又は全体のうちの他の部分であるように理解され、その部分の間の幾つかが独立した場合、集合的に機能することが期待される。

【０１３９】

図１６Ａは、ウェブアプリケーションの実施例を説明する。クライアント1610は、第1のマシン1605において動作する（実行される）。クライアント1610は、第1のマシン1605と第2のマシン1615の間のインターフェースを利用して、プラグイン1630に結合する。典型的には、第1のマシン1605と第2のマシン1615の間のインターフェースは、ワールドワイドウェブ、インターネット、又はマシンの接続に適当なネットワークの幾つかの他の形態である。ウェブサーバ1620は、第2のマシン1615において動作し、第1のマシンとのインターフェースからのリクエストを提供し、プラグイン1630は、クライアント1610からのリクエ

ストを処理するのに適当であるウェブサーバ1620の一部である。プラグイン1630が、第1に又はもっぱらクライアント1610を有して利用するために設計され、ウェブサーバ1620を供給するように期待されたエンティティ以外のエンティティによって提供される特定のコンポーネントであることに注目すべきである。選択肢として、プラグイン1630は、クライアント1610からのリクエストを処理するのに適したウェブサーバ1620の一部であり得る。

【０１４０】

プラグイン1630は、クライアント1610からのリクエストを認識し、（例えば、別のワールドワイドウェブ又はインターネットインターフェースのような）第2のマシン1615と第3のマシン1645の間のインターフェースを介してそれらのリクエストをSWE1660に再度誘導する。前記再誘導は、問題のリクエストを形成しているメッセージを送るか、変換するか、又は別の方法で操作するデータを含み、又は、それは単純に、変更されていないSWE1660に対するリクエストを通すことを含んでいる。SWE1660は、データマネージャ1670と共にオブジェクトマネージャ1650の一部であり、その全ては、第3のマシン1645上で動作する。典型的には、クライアント1610からのリクエストは、SQLリポジトリ（データベース）1680におけるデータのアクセスを結果として生じ、第3のマシン1645によって受け入れ可能である。ウェブサーバ1620及びオブジェクトマネージャ1630は、同様の又は同一の論理関係を維持している一方で、2つの分離したマシンよりもむしろ、単一マシンで動作することに注目すべきである。

【０１４１】

図１６Ｂは、ウェブアプリケーションの択一的な実施例を説明する。第1のマシン1605は、第1のマシンは、独立型動作、又は第3のマシン1645に対する接続と関連した動作のどちらかに対して適しており、同様のマシンは、SQLリポジトリ1680に対するアクセスを許可する。クライアント1610は、オブジェクトマネージャ1650と共に第1のマシン1605上で動作する。プラグイン1630は、オブジェクトマネージャ1650のコンポーネント又は一部として統合され、介在ネットワーク接続を必要とすることなしに、図１６Ａで例証されるクライアント1610を有する同様のインターフェースを許可する。従って、図１６Ｂのプラグイン1630は、同様の又は実質的に同様のコード又は図１６Ａに例証した実施例に利用したような他の実装を利用する。

【０１４２】

プラグイン1630は、データマネージャ1670と相互に作用する、SWE1660に対するリクエストに関して通過させる。データマネージャ1670は、第3のマシン1645に対する接続が与えられるかどうかをよくある（潜在的に近く連続的な又は択一的に規則的な）基準で決定する。前記接続が与えられる場合、データマネージャ1670は、リクエストを処理する目的で第3のマシン1645におけるSQLデータベース1680にアクセスする。前記接続が与えられない場合、データマネージャ1670は、リクエストを処理する目的で第1のマシン1605におけるローカルSQLデータベース1675にアクセスする。

【０１４３】

1つの実施例において、ローカルSQLデータベース1675は、縮小するか又はデータベース1680のコピーを制限される。択一的な実施例において、ローカルデータベース1675は、データベース1680のコピーである。いずれかの実施例において、ローカルデータベース1675とデータベース1680の同期化は、様々な良く知られている手段を用いて処理される。例えば、データマネージャ1670は、1つの実施例において、データベース1675とデータベース1680の同期化を処理する。前記同期化は、リンクが第1のマシン1605と第3のマシン1645の間に与えられる場合にのみ必ず生じる。加えて、幾つかの実施例がデータベース1680とローカルデータベース1675の間に同一のスキーマを有する一方で、択一的な実施例は、データベース1680に対する第1のスキーマ及びデータベース1675に対する第2のスキーマを有する。

【０１４４】

図１７は、ウェブアプリケーションの別の択一的な実施例を説明する。クライアント1710及びローカルウェブサーバ1740の両方が、第1のマシンで動作する。ローカルウェブサー

バ1740は、クライアント1710の表示ポイントから遠隔ウェブサーバをエミュレートするのに適したウェブエミュレーションコンポーネント1720を含んでいる。また、ローカルウェブサーバ1740は、クライアント1710からのリクエストを処理する際に利用されるデータにアクセスするのに適したサーバエミュレーションコンポーネント1730を含んでいる。前記データは、独立型の第1のマシンにおける（例えばキャッシュ又はディレクトリのような）ローカルデータ1750から利用できる。第1のマシンが第2のマシンに結合される場合、ネットワークデータ1775は、クライアント1710からのリクエストを処理するのに適したデータのアクセスのために利用できる。

【０１４５】

評価されるように、クライアント1710及びローカルサーバ1740は、同様の全コンポーネントの一部であるか、又は択一的な実施例における分離した又は互いに異なるコンポーネントである。さらに、サーバエミュレーションコンポーネント1730及びウェブエミュレーションコンポーネント1720は、幾つかの実施例においてローカルサーバコンポーネント1740を含むことを必要とすることなく、分離したコンポーネントとして実現される。望ましくは、クライアント1710は、特定のウェブアプリケーションにおける利用のために適した幾つかの専用化を有する、ユーザによる利用ためのウェブブラウザ又は類似するインターフェースである。しかしながら、クライアント1710は、ウェブベースのアプリケーションを提供するウェブブラウザのような様々なユーザインターフェースと共に利用するのに適したコンポーネントである。

【０１４６】

図１８は、データベースに対してウェブアプリケーションのリクエストを提供するプロセスの実施例を説明する。ブロック1810において、ウェブアプリケーションリクエストは、例えばクライアントから受信される。ブロック1820において、決定は、サービスエージェント又はコンポーネントとメインデータベースの間の最新の接続が与えられるかどうかに関してなされる。接続がされない場合、ブロック1830において、リクエストが、サービスエージェントによって受け入れ可能なローカルSQLデータベースにおいて与えられるデータに基づいて提供される（又はエラー出力する）。接続が与えられる場合、リクエストが、メインSQLデータベースにおいて与えられるデータに基づいてブロック1840において提供される（又はエラー出力する）。メインSQLデータベースは、信頼性を変化させる接続を有することはないネットワーク接続又は幾つかの他のインターフェースを介してサービスエージェントによって受け入れ可能であることに注目すべきである。さらに、サービスエージェントは、1つの実施例において、ウェブアプリケーションリクエストが生ずるクライアントと同様のマシン上で動作するローカルウェブサーバコンポーネント又はプロセスである。

【０１４７】

図１９は、ウェブアプリケーションにおけるサービスリクエストのプロセスの実施例を説明する。ブロック1910において、ウェブアプリケーションリクエストは、例えばクライアントから受信される。ブロック1920において、決定は、サービスエージェント又はコンポーネントとネットワークの間の最新の接続が与えられるかどうかに関してなされる。接続される場合、ブロック1940において、リクエストが、ネットワーク接続を介して受け入れ可能なデータに基づいて（可能な範囲で）提供される。接続されない場合、ブロック1930において、リクエストが、サービスエージェントによって受け入れ可能なデータのローカルキャッシュ又は他のソースで与えられるデータに基づいて（可能な範囲で）提供される。

【０１４８】

ネットワーク接続又は幾つかの他のインターフェースが信頼性を変化させる接続を有しないことに注目すべきである。さらに、接続の存在又は不存在の決定は、あらゆるリクエストに対してなされる必要はなく、その決定は、ネットワーク接続を介してリクエストを試みることに基づいてなされることに注目すべきであり、その後、結果、その欠乏、又は例えばタイムアウトのある形態を調査する。ウェブアプリケーションのサービスリクエスト

は、データの回復を含むか、既にローカルに与えらデータのローカル処理を含むか、又は遠隔データのローカル又は遠隔処理を含む。

【０１４９】

図２０は、データベースに対してウェブアプリケーションを動作させるプロセスの実施例を説明する。ブロック2010において、ウェブアプリケーションは、ローカルモードで動作し、ローカルデータベースを利用しているローカルウェブサーバを介してリクエストを提供する。ブロック2020において、メイン又は第1のデータベースに対するネットワーク接続が認識される。ブロック2030において、ローカルデータベースとメインデータベースの間の同期化が生じる。ブロック2040において、ウェブアプリケーションは、ネットワークモードにおいてメインデータベースのデータからのサービスリクエストを動作させる。ブロック2050において、ネットワーク接続の欠乏又はネットワーク接続における故障についての認識が生じる。ブロック2060において、ウェブアプリケーションは、ローカルモードのオペレーションに復帰する。

【０１５０】

ネットワークとの切断は、幾つかの例において適切な方法で処理され、同期化は切断より前に生じることに注目すべきである。さらに、同期は一切生じる必要はないことに注目すべきである。択一的に、メインデータベースとローカルデータベースの間の同期化は、ネットワークモードにおいて動作する一方で、裏プロセスとして実現される。同様に、ネットワークモードで動作中のメインデータベースにおけるデータの任意変化は、本質的には、ローカルデータベースにおいて同時になされるか、又はメインデータベースに対する前記変化をコミットすることが成功した指示が戻る際になされる。

【０１５１】

図２１は、データベースに対してウェブアプリケーションを動作させるプロセスの択一的な実施例を説明する。ブロック2110において、システムは起動し、システム自体を初期化し、その構成を決定するか又はパラメータを動作させる。ブロック2020において、ネットワーク接続が認識され、ネットワークとの通信が可能であることを指示する。ブロック2030において、ローカルデータベースとメインデータベースの間の同期化は、ネットワーク接続を介して生じる。ブロック2040において、ウェブアプリケーションは、ネットワークモードにおいて動作し、メインデータベースにおけるデータからのリクエストを提供する。ブロック2150において、システム動作は停止する。

【０１５２】

図２２は、データベースに対するウェブアプリケーションを動作させるプロセスの別の択一的な実施例を説明する。ブロック2210において、システムは起動し、システム自体を初期化し、その構成を決定するか又はパラメータを動作させる。ブロック2050において、ネットワーク接続の欠乏又はネットワーク接続における故障の認識がなされる。ブロック2060において、ウェブアプリケーションは、ローカルモードにおいて動作し、ローカルデータベースを利用しているローカルウェブサーバを介してリクエストを提供する。ブロック2270において、システム動作は停止する。

【０１５３】

図２３は、データベースに対してウェブアプリケーションにおけるリクエストを提供するプロセスの択一的な実施例を説明する。ブロック2300において、手段は、ステータスの初期化及び認識で始まる。ブロック2310において、決定は、ネットワーク接続の最新ステータスに関してなされる。ネットワーク接続されない場合、動作はブロック2320におけるネットワークなしに進行する。ブロック2320において、ウェブアプリケーションリクエストが受信される。ブロック2330において、ブロック2320において受信されたリクエストが、ローカルデータベースの利用するか又はローカルデータを用いて提供される。ブロック2340において、決定は、動作が終了するかどうかに関してなされる。終了しない場合、プロセスは、ブロック2320に戻り、次のリクエストを待つ。動作が終了すべき場合、ブロック2390において、動作が停止する。

【０１５４】

ネットワーク接続される場合、プロセスは、ブロック2310からブロック2350に進行する。ブロック2350において、ウェブアプリケーションリクエストが受信される。ブロック2360において、リクエストが、メインデータベースからのデータを用いるか又はネットワーク接続を介して利用できる他のデータを介して提供される。ブロック2370において、決定は、動作が終了するかどうかに関してなされる。終了しない場合、プロセスは、ブロック2350に戻り、別のリクエストを待つ。動作が終了する場合、プロセスは、ブロック2390に進行し、停止する。

【０１５５】

図２４は、ウェブアプリケーションにおいてリクエストを提供するプロセスの択一的な実施例を説明する。ブロック2410において、手段は、ステータスの初期化及び認識で始まる。ブロック2415において、決定は、ネットワーク接続の最新ステータスに関してなされる。接続される（又は動作可能な）場合、ブロック2020において、メイン又は第1のデータベースに対するネットワーク接続が認識される。ブロック2030において、ローカルデータベースとメインデータベースの間の同期化が生じる。ブロック2040において、ウェブアプリケーションは、ネットワークモードにおいて動作し、ネットワーク接続を介してメインデータベースにおけるデータからのリクエストを提供する。動作が完了する場合、ブロック2490において、プロセスが停止する。

【０１５６】

ネットワーク接続がされていないか又は動作可能でない場合、プロセスはブロック2050に進行する。ブロック2050において、ネットワーク接続の欠乏又はネットワーク接続における故障の認識がなされる。ブロック2060において、ウェブアプリケーションはローカルモードで動作し、ローカルデータベースを利用しているローカルウェブサーバを介してリクエストを提供する。最後に、動作が完了する場合、ブロック2490において、プロセスが停止する。

【０１５７】

上記詳細説明の幾つかの部分は、コンピュータメモリ内のデータビット動作のアルゴリズム及び抽象的表現に関して与えられている。これらのアルゴリズム説明及び表現は、データ処理技術における当業者によって利用されている手段であり、最も効果的にその動作の実体を他の当業者に伝えるものである。アルゴリズムは、ここでは、一般に、自己矛盾のない連続ステップであると理解され、所望の結果を導く。ステップは、それらの必要とされる物理量の物理的操作である。通常、必ずしも必要ではないが、これらの量は、記憶、転送、結合、比較、及び別の方法で操作することが可能な電気的又は磁気的信号の形態をとる。主として共通の使用という理由により、これらの信号をビット、値、構成要素、記号、特性、用語、数、又はその類似のものとして言及することは、時には便利であることを立証した。

【０１５８】

しかしながら、これらの全て及び類似する用語が、適切な物理量に関連することとなり、これらの量を適用した単に便利なラベルであることは、精神として伝えられるべきである。以下の議論から明らかなように別な方法で特別に述べられない限り、説明にわたり、“処理”又は“コンピュータ計算”又は“計算”又は“決定”又は“表示”又はその類似のもののような用語を利用する議論は、コンピュータシステムの動作及びプロセス、又は類似する電子コンピュータ計算装置について言及しており、コンピュータシステムのレジスタ及びメモリ内の物理（電子）量として表現されるデータを、コンピュータシステムのメモリ又はレジスタ又は他の類似する情報記憶、伝送、又は表示装置内の物理量として同様に表現される他のデータに操作及び変換させる。

【０１５９】

また、本発明は、この中で動作を実行するための装置に関する。この装置は、特に、必要な目的に対して構成されるか、又は、それはコンピュータ上で記憶されるコンピュータプログラムによって選択的に動作し、又は再構成される、一般の目的のコンピュータを備えている。前記コンピュータプログラムは、コンピュータ読み取り可能な記憶媒体に記憶さ

れているが、例えば、フロッピー（登録商標）ディスク、光ディスク、CD-ROM、磁気光ディスク、リード-オンリーメモリ（ROM）、ランダムアクセスメモリ（RAM）、EPROM、EEPROM、磁気又は光カードを含む任意タイプのディスク、又は電子式命令を記憶するのに適した任意タイプの媒体に限定されておらず、各々はコンピュータシステムバスに接続されている。

【０１６０】

この中に与えられるアルゴリズム及び表示は、本質的に任意の特定のコンピュータ又は他の装置に関連するものではない。様々な一般目的のシステムは、この中の教示に従うプログラムと共に利用され、それは、必要な手段ステップを実現するより詳細な装置を構成するのに便利であることがわかる。様々なこれらのシステムに対して必要な構造は、描写から現れる。加えて、本発明は、任意の特定のプログラミング言語に関係して述べてはいない。様々なプログラミング言語がこの中で描写した発明の教示を実現するのに用いられる。

【０１６１】

図に示される、及び図に付随する文で説明される機能要素は、ソフトウェアコードセグメントを用いて実現され得る。前記機能要素がソフトウェアコードセグメントを利用して実現される場合、それは、さらに、これらのコードセグメントが、フロッピーディスク、ハードドライブ、CD-ROM、DVD、テープ、メモリ、又はコンピュータ計算マシンによってアクセス可能な任意の記憶装置のようなマシン読み取り可能な媒体に記憶され得ることが強調されるべきである。

【０１６２】

モバイルウェブクライアントに対する方法、システム、及び装置が提供される。1つの実施例において、発明は、ウェブアプリケーションを動作させる方法である。方法は、クライアントからのリクエストを受信することを含む。さらに、方法は、利用できるデータからのリクエストを提供することを含む。

【０１６３】

幾つかの実施例において、さらに、方法は、ウェブエミュレータとしてのウェブサーバに対するプラグインを用いることを含み、プラグインは、ウェブエンジンにリクエストを通し、ウェブエンジンは、データマネージャに関連してリクエストを提供する。幾つかの実施例において、さらに、方法は、ネットワークを介して利用できるデータに対するアクセスを試行するデータマネージャを含む。加えて、方法は、成功したアクセスにおいてネットワークから利用できるデータを利用し、リクエストを提供するのに失敗したアクセスにおいてローカルデータからの利用できるデータを利用するデータマネージャを含む。さらに、方法は、クライアントからネットワークにデータを送信するリクエストを含み、リクエストはネットワークからデータを受信する。

【０１６４】

択一的な実施例において、本発明は、データベースと共に利用するためのウェブアプリケーションを動作させる方法である。方法は、クライアントからのリクエストを受信すること、ウェブエミュレータを有するリクエストを処理すること、及びデータベースにおいて利用できるデータからのリクエストを提供することを含む。

【０１６５】

択一的な実施例において、利用できるデータは、メインデータベースにおいてネットワークを介して利用できるか、又はネットワークにアクセスすることなしにローカルデータベースにおいてローカルに利用できる。同様に、択一的な実施例において、さらに、方法は、ネットワークを介してメインデータベースにおいて利用できるデータに対するアクセスを試行することを含む。また、さらに、方法は、ネットワークを介して成功したアクセスにおいて、リクエストを提供することが、ネットワークを介してメインデータベースから利用できるデータを利用することを含む。また同様に、方法は、さらに、ネットワークを介して失敗したアクセスにおいて、リクエストを提供することが、ローカルデータベースにおいてローカルデータから利用できるデータを利用することを含む。

【０１６６】
前述の詳細説明において、本発明の方法及び装置は、特定の典型的な実施例を参照して描写されている。しかしながら、様々な変更及び変化が、本発明の広範な精神及び範囲から出発することなしにそれに対してなされることが明らかである。特に、様々なブロック図の分離したブロックは、方法又は装置の機能ブロックを表し、必ずしも物理的又は論理的分離、又は本発明の精神及び範囲に内在する動作命令を指示するものではない。例えば、図１７の様々なブロックコンポーネントに統合されるか、又はコンポーネントに再分割される。同様に、図１８のブロックは、（例えば）幾つかの実施例において、再命令されるか、又は、線形又は段階的様式におけるよりもむしろ並列に作り上げられる方法の一部を表す。従って、存在する明細書及び図面は、限定的というよりもむしろ例示的であるとみなされることになる。

【図面の簡単な説明】

【０１６７】
【図１】本発明の教示が実現されるマルチレイヤーシステム構成を示す。
【図２】本発明の教示が実現される1つの具体的なシステム配置のブロック図を示す。
【図３】アプリケーションが本発明の教示に従って確立され得る、別の論理表現のマルチレイヤー構成を例証するブロック図を示す。
【図４】本発明の教示が実現される１つの具体的なアプリケーション構想のブロック図を例証する。
【図５Ａ】図２の相互作用するウェブクライアント及びモバイルウェブクライアントをサポートする典型的構想又はインフラストラクチャー500を例証する。
【図５Ｂ】図５Ａに示した典型的構想又はインフラストラクチャーの択一的表示を例証する。
【図６Ａ】ブラウザのオブジェクト及びオブジェクトマネージャ（OM）によって管理されるオブジェクトが、クライアント及びサーバを含む多数の計算機上に存在し機能する典型的な配置を例証する。
【図６Ｂ】ブラウザのオブジェクト及びOMによって管理されるオブジェクトが、１つの計算機上に存在し機能する典型的な配置を例証する。
【図７】遠隔手続き呼び出し（RPC）の典型が、分離した計算機上で実行され得るプログラムの一部分を分割するのにどのように用いられるかについての例を説明する。
【図８】遠隔手続き呼び出しを利用した実行モデルのうちの典型的なモデルを例証する。
【図９】RPCクライアント及びRPCサーバにおけるRPCアプリケーションコードセグメント、RPCインターフェース、クライアント及びサーバ・スタブ、及びRPC実行時間ライブラリの典型的な分割を概して示す。
【図１０】クライアントデータ構造とサーバデータ構造の間の典型的なマーシャリング及びアンマーシャリングを概して示す。
【図１１】遠隔手続き呼び出し中のRPCアプリケーションコードセグメント、RPCインターフェース、RPCスタブ、及びRPC実行時間ライブラリの典型的な役割を概して示す。
【図１２】分散RPCアプリケーションを確立する典型的な処理1200を概説する。
【図１３】ローカルRPC実行時間ライブラリとアプリケーションコードから発生したオブジェクトコードのリンクを概して例証する。
【図１４】図６Ａに示した多数装置の配置で実行する、ブラウザ側又はクライアント側オブジェクトとサーバ側オブジェクトの間の典型的な通信処理を概して例証する。
【図１５】ブラウザ側アプレットが、JSSBusCompオブジェクトで方法を直接起動する典型的な通信処理を示す。
【図１６Ａ】ウェブアプリケーションの具体例を説明する。
【図１６Ｂ】ウェブアプリケーションの択一的に具体例を説明する。
【図１７】ウェブアプリケーションの別の択一的な具体例を説明する。
【図１８】データベースに対するウェブアプリケーションのサービスリクエスト処理の具体例を説明する。

【図１９】ウェブアプリケーションのサービスリクエスト処理の具体例を説明する。
【図２０】データベースに対するウェブアプリケーション動作処理の具体例を説明する。
【図２１】データベースに対するウェブアプリケーション動作処理の択一的な具体例を説明する。
【図２２】データベースに対するウェブアプリケーション動作処理の別の択一的な具体例を説明する。
【図２３】データベースに対するウェブアプリケーションのサービスリクエスト処理の択一的な具体例を説明する。
【図２４】ウェブアプリケーションのサービスリクエスト処理の択一的な具体例を説明する。

【図１】

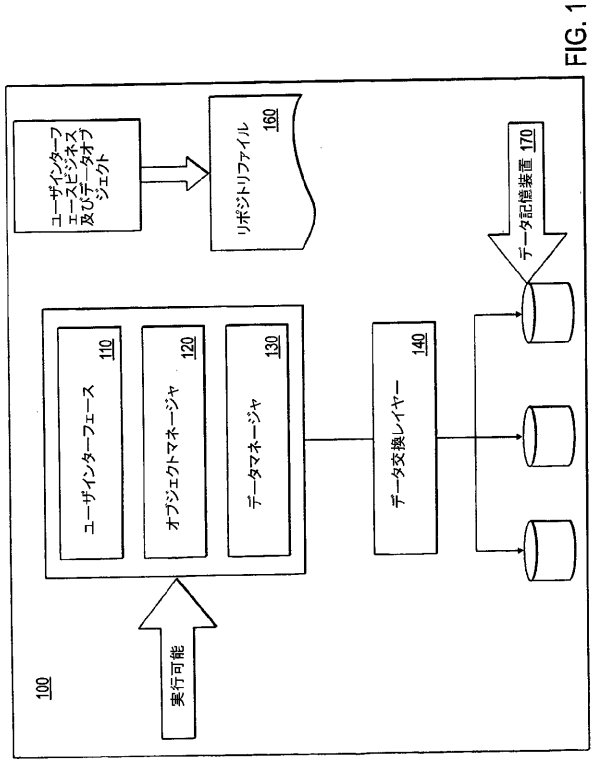


FIG. 1

【図２】

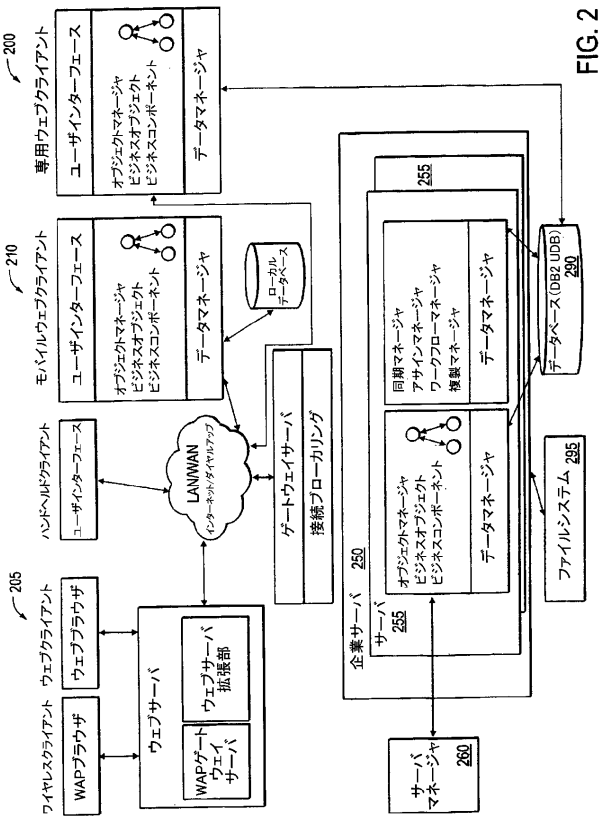


FIG. 2

【図３】

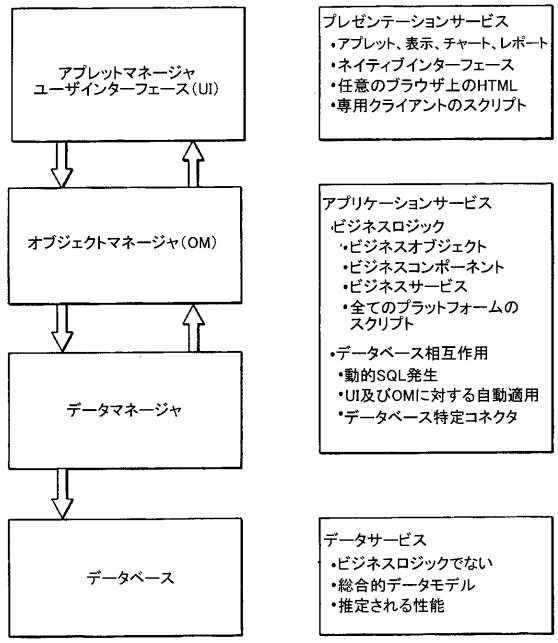


Fig. 3

【図４】

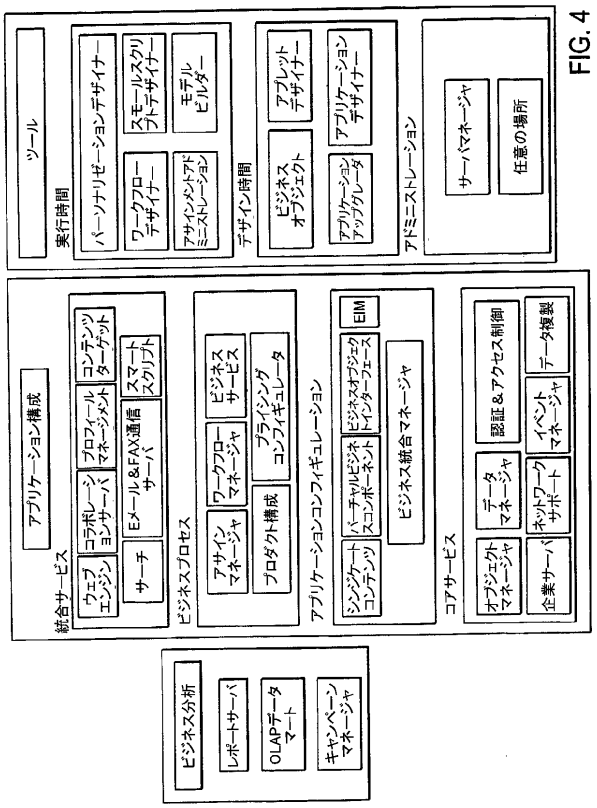


FIG. 4

【図５Ａ】

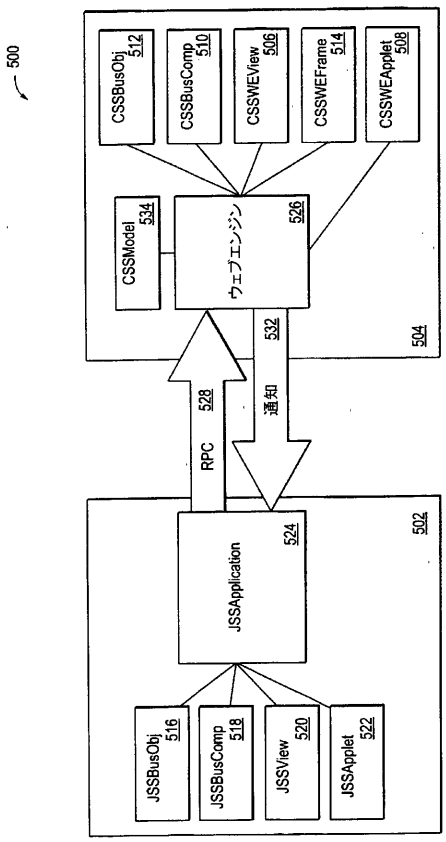


FIG. 5A

【図５Ｂ】

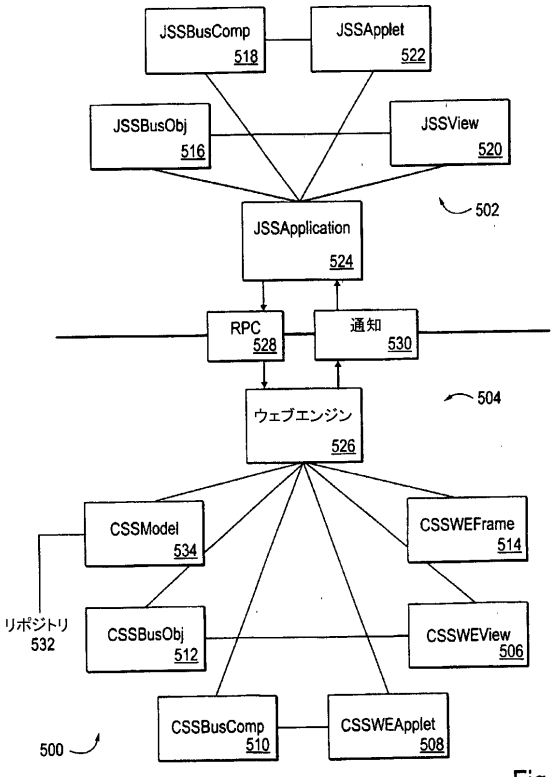


Fig. 5B

【図６Ａ】

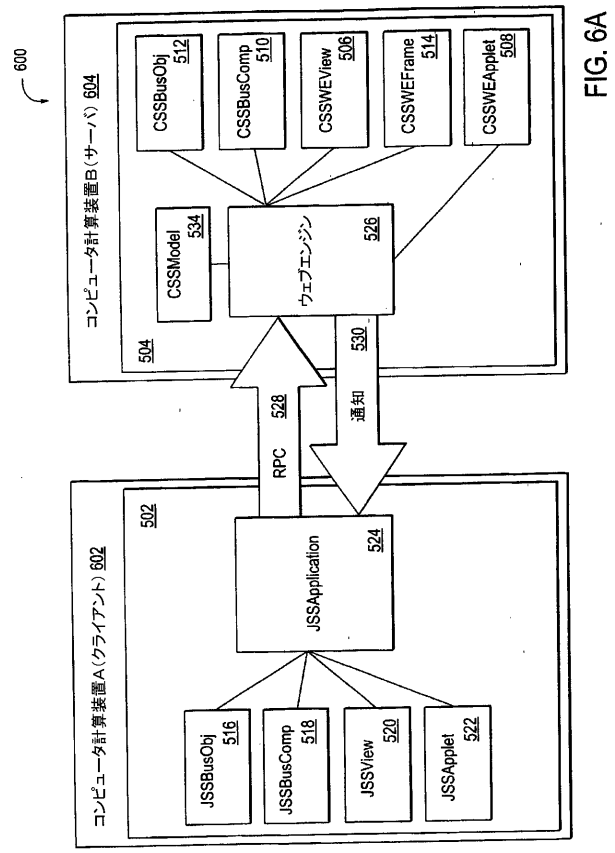


FIG. 6A

【図６Ｂ】

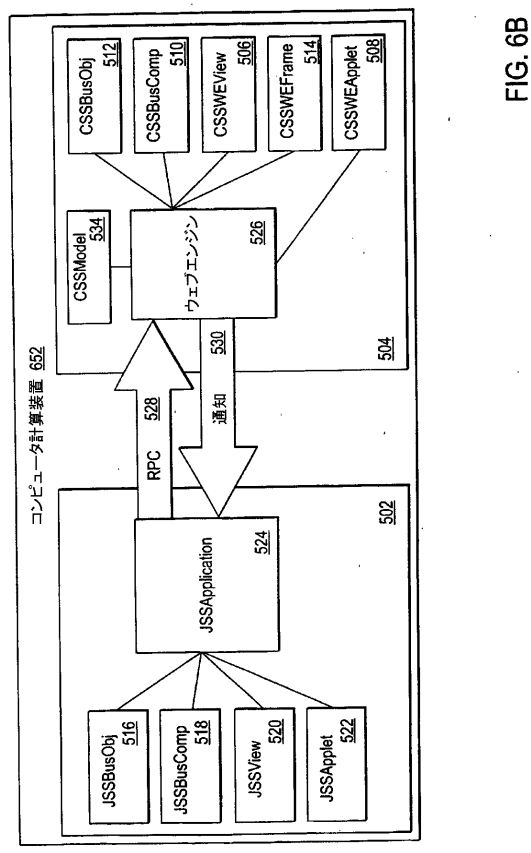


FIG. 6B

【図７】

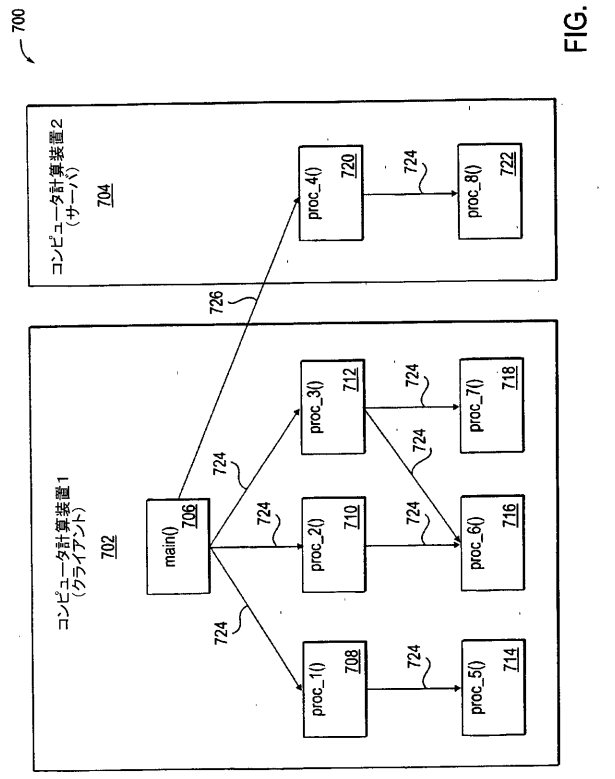


FIG. 7

【図８】

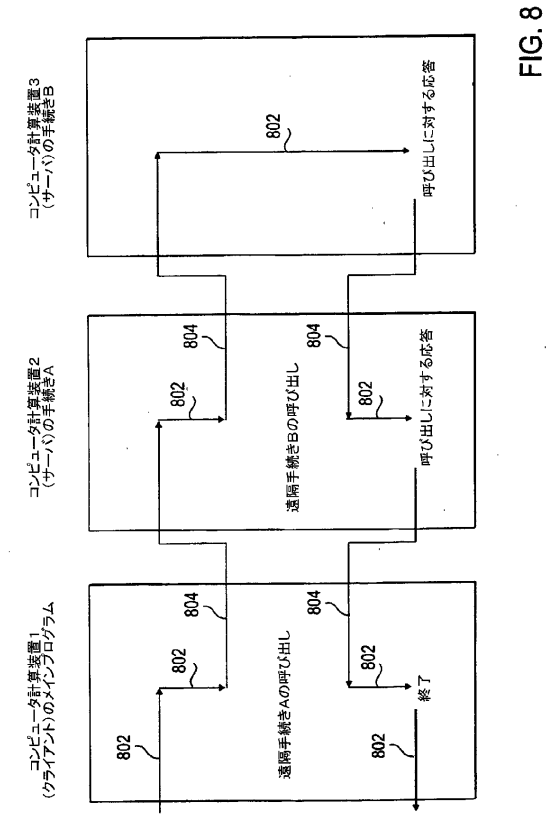


FIG. 8

【図９】

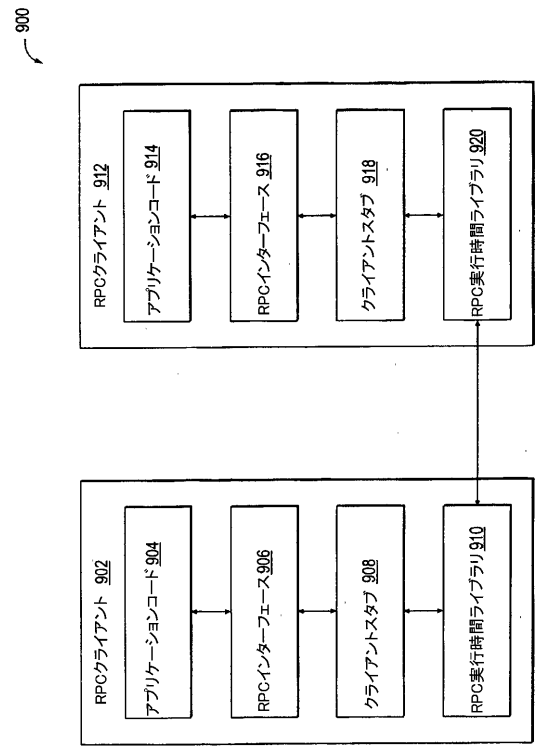


FIG. 9

【図１０】

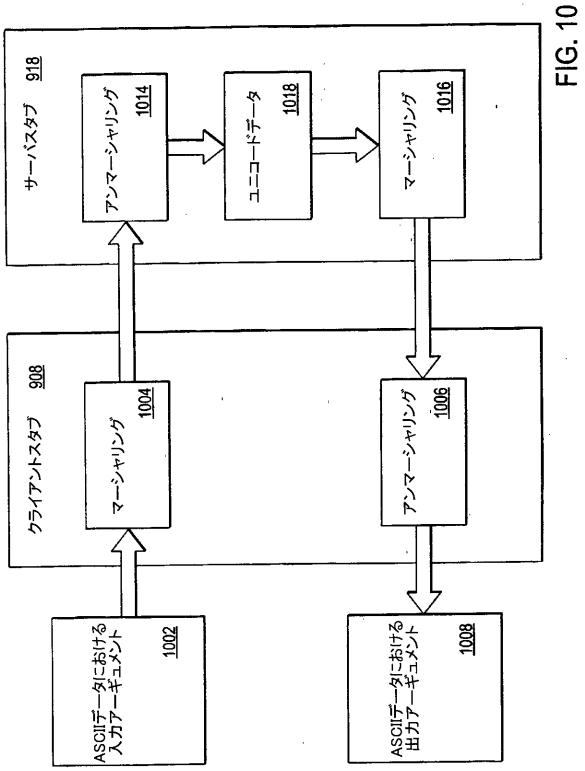


FIG. 10

【図１１】

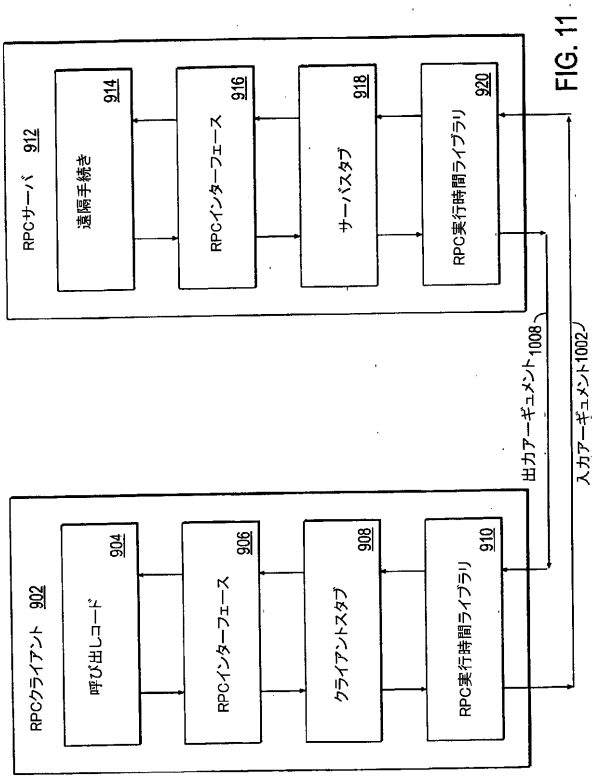


FIG. 11

【図１２】

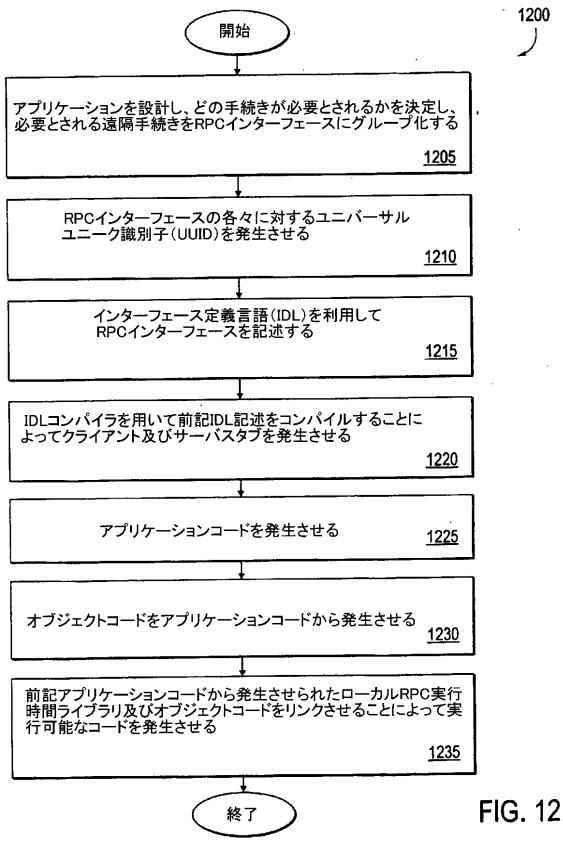


FIG. 12

【図１３】

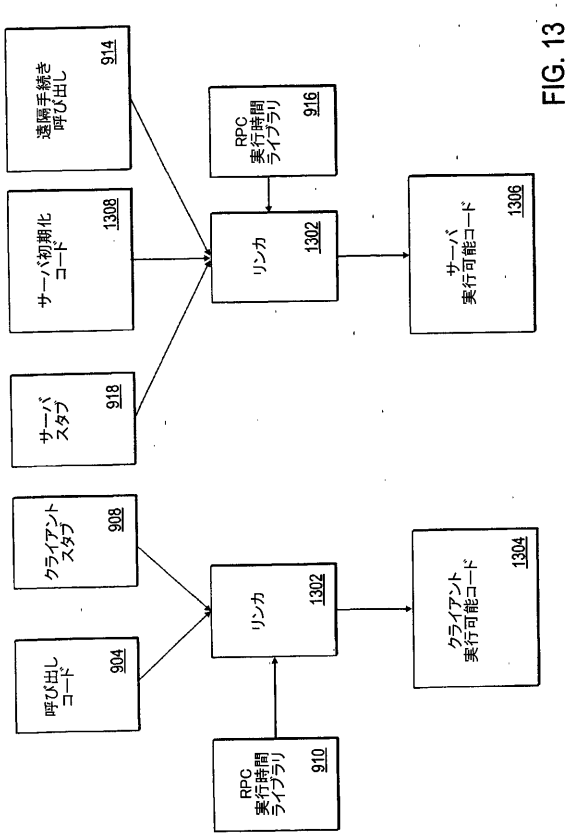


FIG. 13

【図１４】

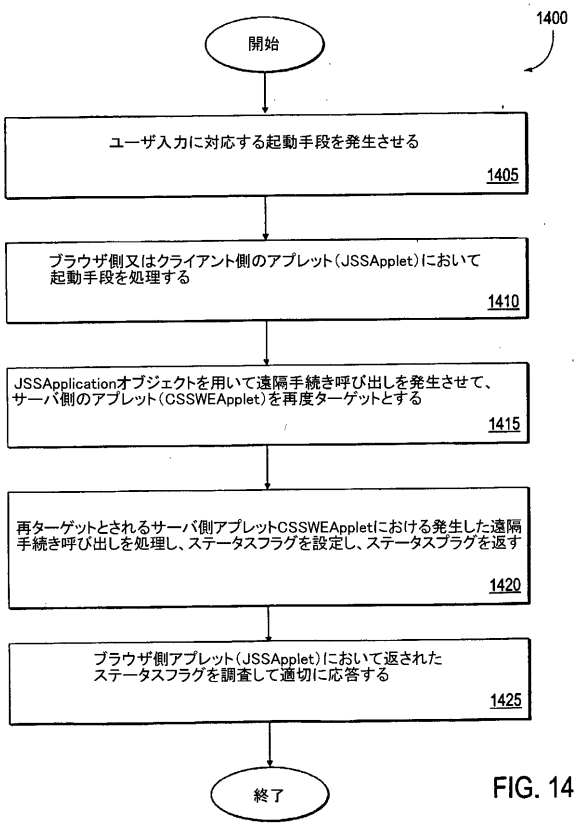


FIG. 14

【図１５】

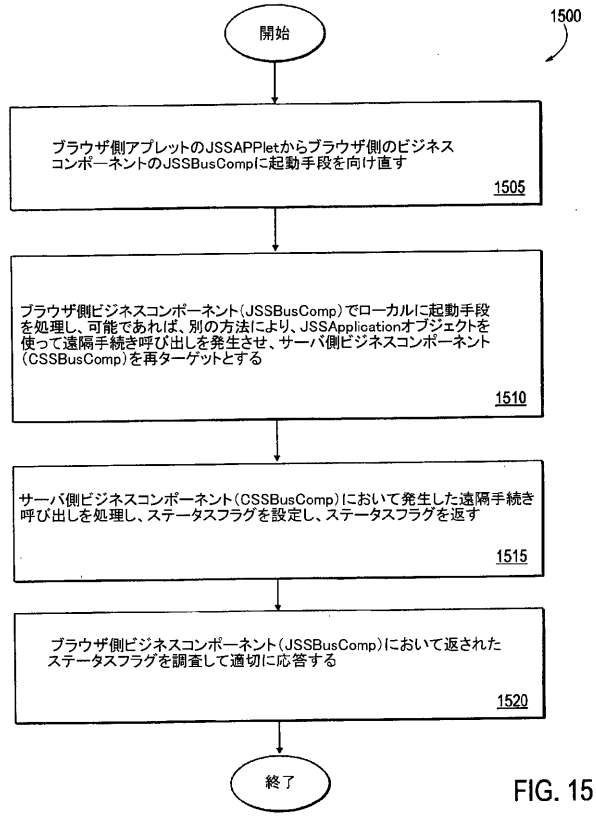


FIG. 15

【図１６Ａ】

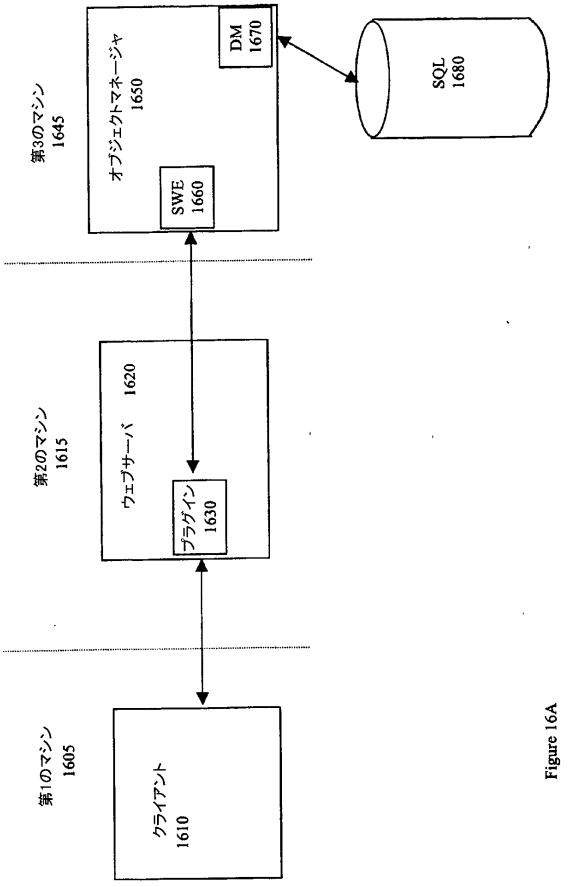


Figure 16A

【図１６Ｂ】

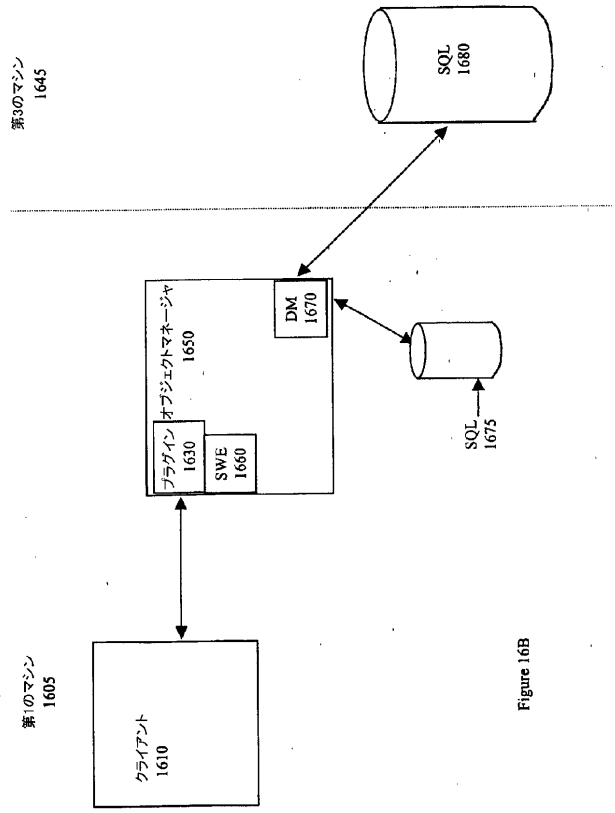


Figure 16B

【図１７】

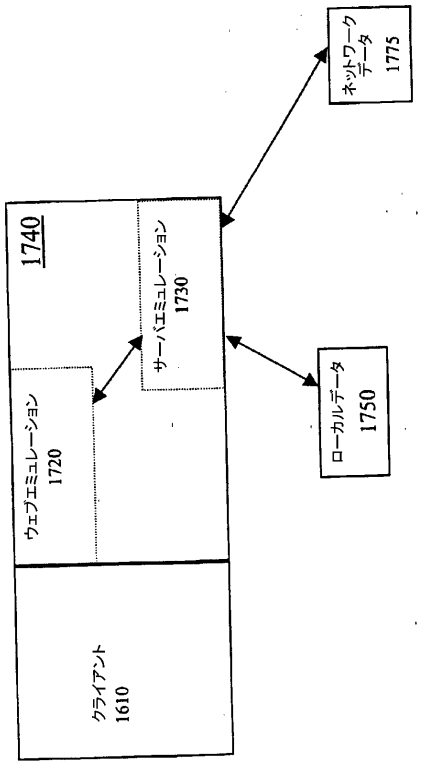


Figure 17

【図１８】

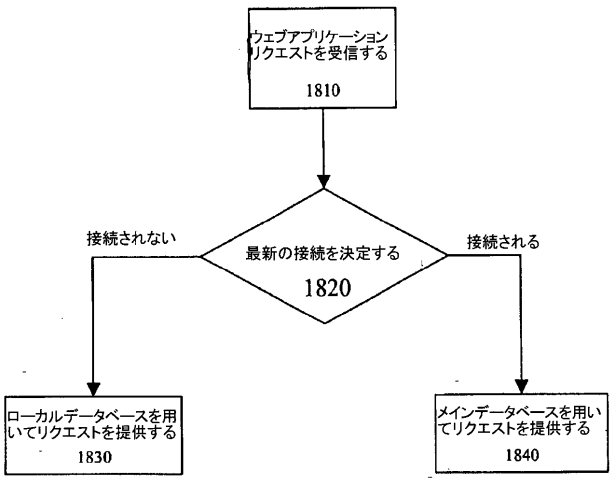


Figure 18

【図１９】

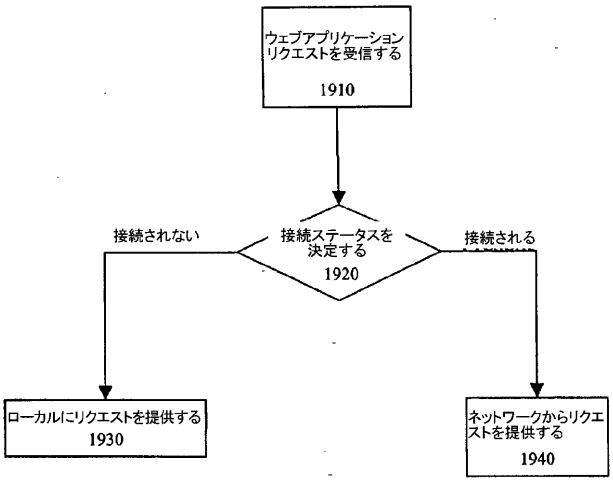


Figure 19

【図２０】

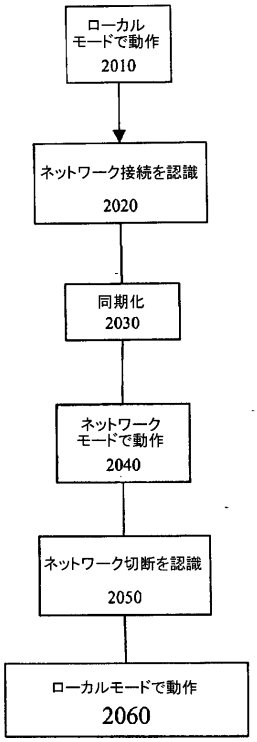


Figure 20

【図２１】

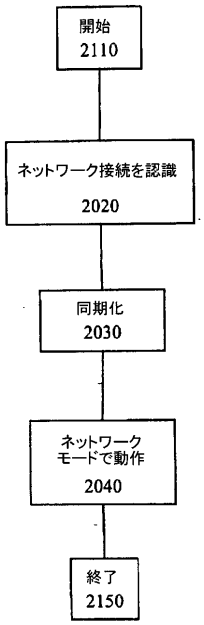


Figure 21

【図２２】

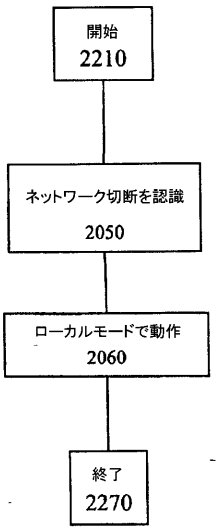


Figure 22

【図２３】

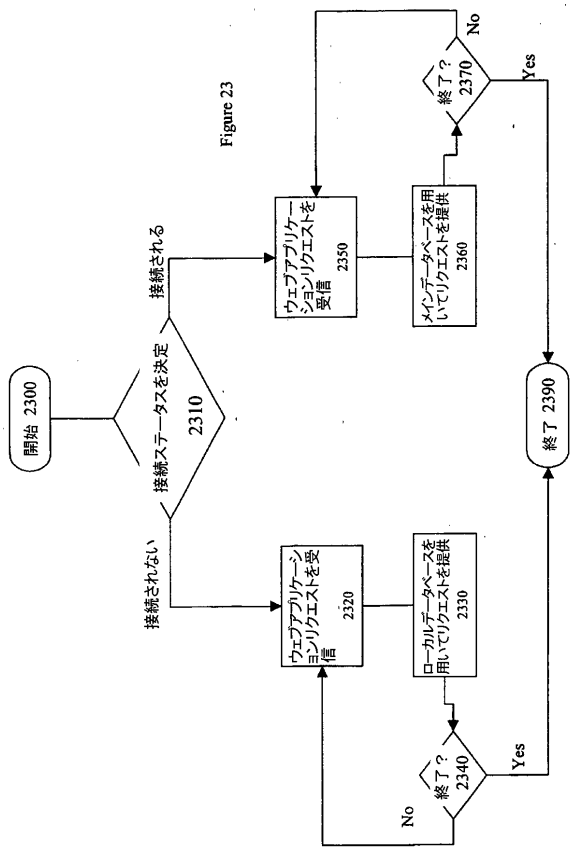


Figure 23

【図２４】

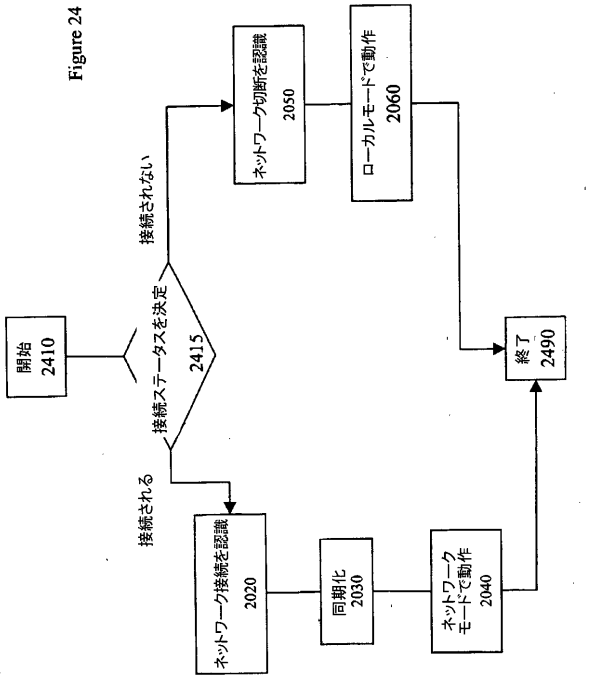


Figure 24
開始
2410
接続ステータスを決定
2415
接続されない
ネットワーク切断を認識
2050
ローカルモードで動作
2060
接続される
ネットワーク接続を認識
2020
同期化
2030
ネットワーク
モードで動作
2040
終了
2490

【国際公開パンフレット】

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
10 April 2003 (10.04.2003)

PCT

(10) International Publication Number
WO 03/030005 A1

(51) International Patent Classification[7]: G06F 15/16, 15/173

(21) International Application Number: PCT/US02/30885

(22) International Filing Date:
27 September 2002 (27.09.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/326,741 29 September 2001 (29.09.2001) US
10/254,384 24 September 2002 (24.09.2002) US

(71) Applicant *(for all designated States except US)*: **SIEBEL SYSTEMS, INC.** [US/US]; 2207 Bridgepoint Parkway, San Mateo, CA 94404 (US).

(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: **COKER, John** [US/US]; 723 Chateau Drive, Hillsborough, CA 94010 (US).

(74) Agents: **MALLIE, Michael, J.** et al.; Blakely, Sokoloff, Taylor & Zafman LLP, 12400 Wilshire Boulevard, 7th Floor, Los Angeles, CA 90025 (US).

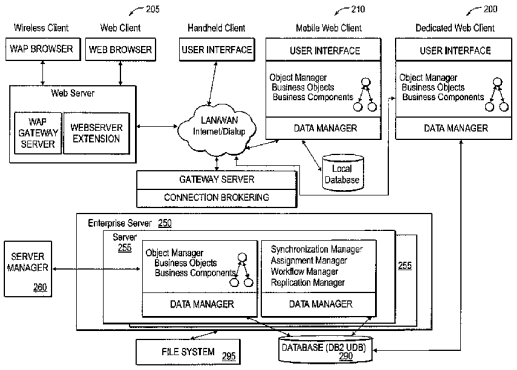
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

[Continued on next page]

(54) **Title:** METHOD, APPARATUS AND SYSTEM FOR A MOBILE WEB CLIENT



(57) **Abstract:** A method, system and apparatus for a mobile web (210)is presented. In one embodiment, the invention is a method of operating a web application.The method includes receiving a request from a client (210). The method further includes processing the request with a web emulator. The method also includes servicing the request from available data.

WO 03/030005 A1

WO 03/030005 A1

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

METHOD, APPARATUS AND SYSTEM FOR A MOBILE WEB CLIENT

CLAIM OF DOMESTIC PRIORITY

[001] This application claims priority to a previously filed provisional application having serial number 60/326,741, entitled "METHOD, APPARATUS AND SYSTEM FOR A MOBILE WEB CLIENT" and filed on September 29, 2001.

BACKGROUND OF THE INVENTION

Field of the Invention

[001] The present invention relates generally to the field of data processing. More specifically, the present invention relates to a method, apparatus, and system for a mobile web client.

Description of the Related Art

[002] As technology continues to advance and the business environments have become increasingly complex and diverse, more and more companies have relied on various customer relationship management (CRM) software and eBusiness applications to conduct and manage various aspects of their enterprise business. In general, eBusiness applications are designed to enable a company or enterprise to conduct its business over an interactive network (e.g., Internet, Intranet, Extranet, etc.) with its customers, partners, suppliers, distributors, employees, etc. eBusiness applications may include core business processes, supply chain, back-office operations, and CRM functions. CRM generally includes various aspects of interaction a company has with its customers, relating to sales and/or services. At a high level, customer relationship management is focused on understanding the customer's needs and leveraging this knowledge to increase sales and improve service. CRM application and software is generally designed to provide effective and efficient interactions between sales and service, and unify a company's activities around the customer in order to increase customer share and customer retention through customer satisfaction.

[003] Typically, CRM implementation strategy needs to consider the following:

[004] Knowledge Management: one of the important factors of an effective CRM implementation is the acquisition of information about a customer, its analysis, sharing and tracking. Also integral to the use of knowledge for competitive advantage is for employees to know what actions to take as a result of this knowledge.

[005] Database Consolidation: another important aspect of an effective and efficient CRM solution is the consolidation of customer information in a single database and the re-engineering of business processes around the customer. The goal here is to have all interactions with a customer recorded in one place to drive production, marketing, sales and customer support activities.

[006] Integration of Channels and Systems: it is very important for a CRM application/software to provide the capability to respond to customers in a consistent and high-quality manner through their channel of choice, whether that is the e-mail, the phone, web-based user interfaces, etc. This may require the seamless integration of various communication channels with the customer or enterprise database. It also may require the integration of CRM with other parts of a company's business systems and applications.

[007] Technology and Infrastructure: to enhance customer services, a CRM application/software may include various tools to automate and streamline online customer service. For example, a self-help model typically can be implemented using a combination of tools (e.g. knowledge bases with an intuitive search capability, agent technology or automated email, etc.).

[008] Generally, *e*Business applications are designed to allow organizations to create a single source of customer information that makes it easier to sell to, market to, and service customers across multiple channels, including the Web, call centers, field, resellers, retail, and dealer networks. Advanced eBusiness applications are typically built on a component-based architecture and are designed to be Web-based and to deliver support for various types of clients on multiple computing platforms including mobile clients, connected clients, thin clients, and handheld clients, etc.

[009] With the recent proliferation of the Web, it is desirable to provide the functionalities of the eBusiness applications in a Web-based environment. Furthermore, it is desirable for the eBusiness applications operating in a Web-based environment to

retain the look-and-feel of desktop-based eBusiness applications with which the users are already familiar.

[010] With the advent of web applications, dependency on availability of a network such as an internet or the world wide web for operation of such applications also arrived. However, it will be appreciated that a robust solution for working both in a networked environment or situation and a corresponding non-networked environment or situation may be useful to mobile users of a web application. Such a web application is not immediately available.

BRIEF DESCRIPTION OF THE DRAWINGS

[011] The present invention is illustrated by way of example and not limitation in the accompanying figures.

[012] Figure 1 shows a multi-layered system architecture in which the teachings of the present invention are implemented.

[013] Figure 2 shows a block diagram of one embodiment of a system configuration in which the teachings of the present invention are implemented.

[014] Figure 3 shows a block diagram illustrating another logical representation of a multi-layered architecture in which applications can be built in accordance with the teachings of the present invention.

[015] Figure 4 illustrates a block diagram of one embodiment of an application framework in which the teachings of the present invention may be implemented.

[016] Figure 5A illustrates an exemplary framework or infrastructure 500 to support an interactive web client and an mobile web client of figure 2.

[017] Figure 5B illustrates an alternative view of the exemplary framework or infrastructure shown in figure 5A.

[018] Figure 6A illustrates an exemplary configuration in which objects on the browser and objects managed by the object manager (OM) reside and operate on multiple computing devices, including a client and a server.

[019] Figure 6B illustrates an exemplary configuration in which objects on the browser and objects managed by the OM reside and operate on one computing device.

[020] Figure 7 illustrates an example of how the remote procedure call (RPC) paradigm can be used to divide a program into pieces that can be executed on separate computing devices.

[021] Figure 8 illustrates an exemplary model of execution used with remote procedure calls.

[022] Figure 9 generally shows an exemplary partitioning of RPC application code segments, RPC interfaces, client and server stubs, and the RPC runtime libraries in the RPC client and the RPC server.

[023] Figure 10 generally shows an exemplary marshalling and unmarshalling between client data structures and server data structures.

[024] Figure 11 generally shows exemplary roles of RPC application code segments, RPC interfaces, RPC stubs, and RPC runtime libraries during a remote procedure call.

[025] Figure 12 generally outlines an exemplary process 1200 of building a distributed RPC application.

[026] Figure 13 generally illustrates the linking of the local RPC runtime library and the object code generated from the application code.

[027] Figure 14 generally illustrates an exemplary process of communication between the browser-side or client-side objects and server-side objects running on a multiple-devices configuration shown in figure 6A.

[028] Figure 15 shows an exemplary process of communication in which the browser-side applet invokes the method directly on the JSSBusComp object.

[029] Figure 16A illustrates an embodiment of a web application.

[030] Figure 16B illustrates an alternate embodiment of a web application.

[031] Figure 17 illustrates another alternate embodiment of a web application.

[032] Figure 18 illustrates an embodiment of a process of servicing requests in a web application for databases.

[033] Figure 19 illustrates an embodiment of a process of servicing requests in a web application.

[034] Figure 20 illustrates an embodiment of a process of operating a web application for databases.

[035] Figure 21 illustrates an alternate embodiment of a process of operating a web application for databases.

[036] Figure 22 illustrates another alternate embodiment of a process of operating a web application for databases.

[037] Figure 23 illustrates an alternate embodiment of a process of servicing requests in a web application for databases.

[038] Figure 24 illustrates an alternate embodiment of a process of servicing requests in a web application.

DETAILED DESCRIPTION

[039] A method, apparatus and system for a mobile web client is described. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the invention. It will be apparent, however, to one skilled in the art that the invention can be practiced without these specific details. In other instances, structures and devices are shown in block diagram form in order to avoid obscuring the invention.

[040] Reference in the specification to "one embodiment" or "an embodiment" means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the invention. The appearances of the phrase "in one embodiment" in various places in the specification are not necessarily all referring to the same embodiment, nor are separate or alternative embodiments mutually exclusive of other embodiments.

[041] System overview and overall architecture

[042] In one embodiment, a system in which the teachings of the present invention are implemented can be logically structured as a multi-layered architecture as shown in Figure 1. In one embodiment, the logical multi-layered architecture as shown in Figure 1 provides a platform for common services to support the various applications. These services may include a user interface layer 110, an object manager layer 120, a data manager layer 130, and a data exchange layer 140.

[043] In one embodiment, the user Interface layer 110 may provide the applets, views, charts and reports, etc. associated with one or more applications. In one embodiment, various types of clients can be supported via the user interface layer 110. These various types of clients may include traditional connected clients, remote clients, thin clients over an intranet, Java thin clients or non-Windows-based operating systems, and HTML clients over the Internet, etc.

[044] In one embodiment, the object manager layer 120 is designed to manage one or more sets of business rules or business concepts associated with one or more applications and to provide the interface between the user interface layer 110 and the data manager layer 130. In one embodiment, the business rules or concepts can be represented as business objects. In one embodiment, the business objects may be designed as configurable software representations of the various business rules or concepts such as accounts, contacts, opportunities, service requests, solutions, etc.

[045] In one embodiment, the data manager layer 130 is designed to maintain logical views of the underlying data and to allow the object manager to function independently of underlying data structures or tables in which data are stored. In one embodiment, the data manager 130 may also provide certain database query functions such as generation of structure query language (SQL) in real time to access the data. In one embodiment, the data manager 130 is designed to operate on object definitions in a repository file 160 that define the database schema. In one embodiment, the data storage services 170 provide the data storage for the data model associated with one or more applications.

[046] In one embodiment, the data exchange layer is designed to handle the interactions with one or more specific target databases and provide the interface between the data manager layer 130 and the underlying data sources.

[047] Figure 2 shows a block diagram of one embodiment of a system configuration in which the teachings of the present invention are implemented.

[048] In one embodiment, the multi-layered architecture allows one or more software layers to reside on different machines. For example, in one embodiment, the user interface, the object manager, and the data manager can all reside on the dedicated web clients. For other types of clients such as the wireless clients, in one embodiment,

the object manager and data manager can reside on a system server. It should be appreciated and understood by one skilled in the art that the system configuration shown in Figure 2 is for illustrative and explanative purposes and may vary depending upon the particular implementations and applications of the teachings of the present invention.

[049] In one embodiment, the system environment illustrated in Figure 2 may include more than one database 290. One or more subsets of the database 290 can be created or replicated by a replication manager. In addition, mobile web clients can have additional remote databases (also called local databases). In one embodiment, unless the remote or local databases associated with the mobile web clients are defined as read-only databases, these mobile web clients can create and update data locally that will be ultimately propagated up to the primary database when each mobile web client synchronizes with the system server.

[050] In one embodiment, the database 290 is designed to store various types of data including predefined data schema (e.g., table objects, index objects, etc.), repository objects (e.g., business objects and components, view definitions and visibility rules, etc.), and user's or customer's data. In one embodiment, dedicated web clients and server components, including those that operate in conjunction with the other types of clients, can connect directly to the database 290 and make changes in real time. In one embodiment, mobile web clients can download a subset of the server's data to use locally, and periodically synchronize with the server database through the system server to update both the local and the server database.

[051] In one embodiment, various tables included in the database 290 may be logically organized into the following types: data tables, interface tables, and repository tables, etc.

[052] In one embodiment, data tables may be used to store user business data, administrative data, seed data, and

[053] transaction data, etc. In one embodiment, these data tables may be populated and updated through the various applications and processes. In one embodiment, data tables may include the base tables and the intersection tables, etc. In one embodiment, base tables may contain columns that are defined and used by the various applications. In one embodiment, the base tables are designed to provide the

columns for a business component specified in the table property of that business component. In one embodiment, intersection tables are tables that are used to implement a many-to-many

[054] relationship between two business components. They may also hold intersection data columns, which store information pertaining to each association. In one embodiment, intersection tables provide the data structures for association applets.

[055] In one embodiment, interface tables are used to denormalize a group of base tables into a single table that external programs can interface to. In one embodiment, they may be used as a staging area for exporting and importing of data.

[056] In one embodiment, repository tables contain the object definitions that specify one or more applications regarding:

[057] • The client application configuration

[058] • The mapping used for importing and exporting data

[059] • Rules for transferring data to mobile clients

[060] In one embodiment, the file system 295 is a network-accessible directory that can be located on an application server. In one embodiment, the file system 295 stores the physical files created by various applications, such as files created by third-party text editors, and other data that is not stored in the database 290. In one embodiment, physical files stored in the file system 295 can be compressed and stored under various naming conventions. In one embodiment, dedicated web clients can read and write files directly to and from the file system 295. In one embodiment, mobile web clients can have a local file system, which they synchronize with the server-based file system 290 periodically. In one embodiment, other types of client such as the wireless clients and the web clients can access the file system 290 via the system server.

[061] In one embodiment, the enterprise server 250 is a logical grouping of the system servers 255 that share a common table owner or a database, point to a common gateway Server, and can be administered as a group using server manager 260. In one embodiment, the connection to the gateway server can be established via TCP/IP. In one embodiment, the enterprise server 250 can be scaled effectively by deploying multiple system servers 255 in the enterprise server 250, thus providing a high degree of scalability in the middle tier of applications.

[062] In one embodiment, the server 255 runs one or multiple server programs. It handles the incoming processing requests and monitors the state of all processes on the server. In one embodiment, server programs are designed and configured to perform one or more specific functions or jobs including importing and exporting data, configuring the database, executing workflow and process automation, processing to support mobile web clients for data synchronization and replication, and enforcing business rules, etc. In one embodiment, the server 255 can be an NT Service (under Windows NT operating system) or a daemon (e.g., a background shell process) under UNIX operating system. In one embodiment, the server 255 supports both multi-process and multi-threaded components and can operate components in batch, service, and interactive modes.

[063] In one embodiment, the server manager 260 is configured as a utility that allows common control, administration and monitoring across disparate programs for the servers 255 and the enterprise server 250. In one embodiment, the server manager 260 can be used to perform the following tasks: start, stop, pause, and resume servers 255, components, and tasks; monitor status and collect statistics for multiple tasks, components, and servers within an enterprise server; and configure the enterprise server, individual servers individual components, and tasks, etc.

[064] In one embodiment, the gateway server can be configured as a logical entity that serves as a single entry point for accessing servers. In one embodiment, it can be used to provide enhanced scalability, load balancing and high availability across the enterprise server. In one embodiment, the gateway server may include a name server and a connection brokering component. In one embodiment, the name server is configured to keep track of the parameters associated with the servers. For example, the availability and connectivity information associated with the servers can be stored in the name server. The various components in the system can query the name server for various information regarding the servers' availability and connectivity. In a Windows NT environment, the name server can be run as a NT service. In a UNIX environment, the name server can run as a daemon process. In one embodiment, the connection brokering component is used to perform load balancing function such as directing client connection requests to an appropriate server (e.g., the least-busy server).

[065] In one embodiment, as illustrated in Figure 2, the various types of clients that can be supported by the system may include the following clients: dedicated web clients, mobile web clients, web clients, wireless clients, and handheld clients, etc.

[066] In one embodiment, dedicated web clients (also called connected clients) are connected directly to a database server for data access via a LAN or WAN connection. In one embodiment, these connected or dedicated web clients do not store data locally. These dedicated web clients can also access the file system directly. In one embodiment, the user interface, the object manager, and the data manager layers of the multi-layered architecture reside on the dedicated web client.

[067] In one embodiment, the mobile web clients are designed and configured for local data access and thus can have their own local database and/or local file system. In one embodiment, mobile web clients can interact with other components within the system via the gateway server. Through synchronization, the modifications from the local database and the server database can be exchanged. Mobile web clients are described in more detail below.

[068] In one embodiment, a web client runs in a standard browser format from the client's machine. In one embodiment, the web client can connect to a system server 255 through a web server. In one embodiment, the system server 255 is designed and configured to execute business logic and access data from the database 290 and file system 295. In one embodiment, the web client described herein is designed and configured in accordance with the teachings of the present invention to operate in an interactive mode. In one embodiment, the interactive web client framework as described herein utilizes dynamically created objects implemented in JavaScript on the browser side that correspond to objects on the server side. In one embodiment, these dynamically created objects on the browser side may include the current view and its corresponding applets, the current business object and the corresponding business components, etc. The web client is described in more details below.

[069] In one embodiment, wireless clients are essentially thin clients enabled on wireless devices. The wireless clients can use a wireless application protocol (WAP)-based user interface to communicate and exchange information/data with the system server.

[070] The system configuration illustrated in Figure 2 is described in more details below with references to various structures, databases, tables, file systems, etc. as illustrating examples.

[071] Figure 3 shows a block diagram illustrating another logical representation of a multi-layered architecture in which applications can be built in accordance with the teachings of the present invention. Again, the multi-layered architecture as illustrated in Figure 3 provides the platform for various common services designed and configured to support the various applications. In one embodiment, these various services may include presentation services logic layer 315 which corresponds to an applet manager and user interface layer 310, application services logical layer 325 which corresponds to an object manager (OM)layer 320 and a data manager (DM) layer 330, and data services logical layer 345 which corresponds to a database layer 340.

[072] In one embodiment, the presentation services 315 may be designed and configured to support various types of clients and may provide them with user interface applets, views, charts, and reports, etc. As described above, a large variety of clients may be supported including wireless clients, handheld clients, web clients, mobile web clients, and dedicated (connected) clients, etc.

[073] In one embodiment, the application services 325 may include business logic services and database interaction services. In one embodiment, business logic services provide the class and behaviors of business objects and business components. In one embodiment, database interaction services may be designed and configured to take the user interface (UI) request for data from a business component and generate the database commands (e.g.SQL queries, etc.) necessary to satisfy the request. For example, the data interaction services may be used to translate a call for data into DBMS-specific SQL statements.

[074] In one embodiment, data storage services 345 may be designed and configured to provide the data storage for the underlying data model which serves as the basis of the various applications. For example, the data model may be designed and configured to support various software products and applications including call center, sales, services, and marketing, etc., as well as various industry vertical products and applications such as eFinance, eInsurance, eCommunications, and eHealthcare, etc.

[075] Figure 4 illustrates a block diagram of one embodiment of an application framework in which the teachings of the present invention may be implemented. As illustrated in Figure 4, the application framework may include various logical groupings of various types of services and various types of tools that can be used to design and configure particular applications based on business needs and environments.

[076] In one embodiment, the core services are are designed and configured to provide the framework in which the applications execute. In one embodiment, the core services

[077] may include the following:

[078] • The enterprise server, which is the middle-tier application server

[079] • The networks that link all of these pieces together

[080] • Facilities like event manager and data replication, which allow sharing data between multiple installations of various applications as well as between the various applications and other external applications

[081] • The authentication and access control, the security facilities.

[082] In one embodiment, application integration services may be designed and configured to allow the various applications built in accordance with this framework to communicate with the external world. In one embodiment, the various types of services in this logical grouping may be designed and configured to provide for real-time, near-real-time, and batch integration with external applications. For example, these integration services may be used to enable communications between external applications and the internal applications using available methods, technologies, and software products. In one embodiment, application integration services allow the systems or applications to share and replicate data with other external enterprise applications. Accordingly, these services allow a particular application or system to be both a client requesting information, and a server having information requested from it.

[083] In one embodiment, business processes services are designed and configured to allow the client to automate business processes through the application. In one embodiment, these various business process services may include the following:

[084] • Assignment of tasks through Assignment Manager

[085] • Enforcement of business practices through Workflow Manager

[086] • Reuse of custom business logic through Business Services

[087] • Ensuring proper product configuration and pricing through the Product Configurator and Pricing Configurator

[088] In one embodiment, creation of these business processes can be done through Run-Time tools such as Personalization Designer, Workflow Designer, SmartScript Designer, Assignment Administration Views, and the Model Builder, etc.

[089] In one embodiment, integration services may be designed and configured to provide the client with user interface and thin client support. In one embodiment, these may include capabilities for building and maintaining web-based applications, providing web support facilities such as user Profile Management, Collaboration Services and Email and Fax services, as well as advanced Smart Scripting, etc.

[090] In one embodiment, design time tools may be designed and configured to provide the services to customize, design, provide integration points, and maintain the application. These various tools provide one common place to define the application.

[091] In one embodiment, admin services are designed and configured provide one place to monitor and administer the application environment. In one embodiment, these services allow the user to administer the application either through a graphic user interface (GUI) or from a command line, etc.

[092] System Framework or Infrastructure

[093] Figure 5A illustrates an exemplary system framework or infrastructure 500 to support an interactive web client 205 and a mobile web client 210 of figure 2 in accordance with one embodiment of the present invention. Figure 5B illustrates an alternative view of the exemplary system framework or infrastructure 500 shown in figure 5A.

[094] The framework or infrastructure 500 can support the interactive web client 205 (shown in figure 2) and the mobile web client 210 (also shown in figure 2), and is capable of meeting certain criteria, such as increasing the interactivity and performance of the web client and the mobile web client, and reducing the number of page refreshes for common actions.

[095] The framework or infrastructure 500 can include objects 502 that can be dynamically created on the browser to mimic corresponding objects 504 managed by the object-manager. In one embodiment, the objects 504 managed by the object manager (OM) can be built using a programming language, such as C++, supporting the object-oriented paradigm.

[096] As shown in figures 5A and 5B, exemplary objects 504 managed by the OM can include an object 506 representing a view, CSSWEView 506. A view is generally a display panel consisting of a particular arrangement of applets. In one embodiment, one active view can be displayed at any given time. Another exemplary object managed by the OM can be an object 508 representing an applet, CSSWEApplet 508. An applet is generally a visual application unit that appears on the screen as part of a view. Other exemplary objects managed by the OM can include an object 510 representing a business component (CSSBusComp 510), an object 512 representing a business object (CSSBusObj 512), and an object 514 representing a frame (CSSWEFrame 514). In one embodiment, the business object may be designed as configurable software representations of the various business rules or concepts such as accounts, contacts, opportunities, service requests, solutions, etc. In this embodiment, the business components typically provide a layer of wrapping over tables, and the applets reference business components rather than the underlying tables. In addition, a frame is generally a sub-component of a view and may comprise of one or more applets.

[097] In one embodiment, objects 502 on the browser can be built using JavaScript. As shown in Figures 5A and 5B, exemplary objects 502 on the browser side may include JSSBusObj 516, JSSBusComp 518, JSSView 520, and JSSApplet 522 to respectively mirror CSSBusObj 512, CSSBusComp 510, CSSWEView 506, and CSSWEApplet 508, which are objects 504 managed by the OM.

[098] Objects 502 on the browser and objects 504 managed by the OM can be configured to reside and operate on one computing device or multiple computing devices. Figure 6A illustrates an exemplary configuration 600 in which objects 502 on the browser and objects 504 managed by the OM reside and operate on multiple computing devices 602,604, including a client 602 and a server 604. Figure 6B

illustrates an exemplary configuration 650 in which objects 502 on the browser and objects 504 managed by the OM reside and operate on one computing device 652.

[099] Returning to figures 5A and 5B, objects 502 on the browser are generally synchronized with corresponding or mirrored objects 504 managed by the OM. Synchronization can be accomplished through a remote procedure call (RPC) mechanism 528 and a notification mechanism 530. The RPC mechanism 528 and the notification mechanism 530 will be described below in more details.

[0100] Of the objects 502 on the browser, the JSSApplication object 524 typically exists throughout a user-session. The JSSApplication object 524 should be initially loaded initially when the user starts an application. An application would generally be started when the user invokes a subset of the application from an icon on the desktop or from the Start menu. The JSSApplication object 524 generally performs a role similar to that of the CSSModel object 534. The CSSModel object 534 is generally a global session object that provides access to repository objects that are in use, the current business object instance in memory, the relationships between the current business object and the business components contained in it, and the user's global state information. The CSSModel object 534 generally accesses a repository 532 to obtain needed information. The repository 532 is generally a set of object definitions used to define an application or a suite of applications. However, the JSSApplication object 524 is generally scaled down to track one view, applets associated to the tracked view, one business object, and the business components that are in use in the view.

[0101] Unlike the JSSApplication object 524, the JSSView object 520, the JSSApplet object 522, the JSSBusObj object 516 and the JSSBusComp object 518 are typically temporary or impermanent entities, and are generally replaced when a page refresh occurs. For example, a request to navigate to a new view may cause a new set of JSSView 520, JSSApplet 522, JSSBusObj 516, and JSSBusComp 518 objects to be created to run on the browser.

[0102] Accordingly, objects 502 on the browser can be generally described as lightweight representations of mirrored or corresponding objects 504 managed by the OM. Each object 502 on the browser would typically include a subset of the functionalities included in corresponding objects 504 managed by the OM. For example,

the JSSView object 520, similar to a CSSView object 506, generally represents a collection of applets. The JSSBusObj object 516, similar to a CSSBusObj object 512, generally manages the various one-to-many relationships between active business components so that correct relationships are employed when these active business components are populated via queries. The JSSBusObj object 516 generally exists on the browser for the life of the current view, and should be kept in sync with the corresponding CSSBusObj object 512.

[0103] In one embodiment, when the browser submits a request to navigate to a new view to the web engine 526, the web engine 526 would send a response containing the view layout that is devoid of data. Then the web engine 526 would send a response containing a string of data to populate the view.

[0104] The JSSApplication object 524 generally manages communications flowing into and out from objects on the browser. In one embodiment, a method invoked on an object on the browser would typically be directed to the JSSApplication object 524 if the invoked method should be retargeted to an object 504 managed by the OM. The JSSApplication object 524 would generally use the RPC mechanism 528 to route the invoked method through the web engine 526 to the appropriate object 504 managed by the OM. The web engine 526 would typically be employed to send return notifications and data from objects 504 managed by the OM to objects 502 on the browser. The web engine 526 would generally use the notification mechanism 530 to route notifications and data through the JSSApplication object 524 to objects 502 on the browser.

[0105] The browser objects 502 generally use the remote procedure calls 528 to invoke methods on the objects 504 managed by the OM. These remote procedure calls 528 are generally packaged as HTTP requests. Responses from the objects 504 managed by the OM are packaged as HTTP responses containing notifications and associated status information and data. In one embodiment, remote procedure calls are made with blocking enabled to ensure synchronization between the objects 502 on the browser and the objects 504 managed by the OM. With blocking enabled, control would typically not be passed back to the calling code until the called remote procedure finishes executing.

[0106] Remote Procedure Call (RPC)

[0107] The RPC model generally uses the same procedural abstraction as a conventional program, but allows a procedure call to span the boundary between two computers. Figure 7 illustrates an example of how the RPC paradigm can be used to divide a program 700 into pieces that can be executed on separate computing devices 702,704. This figure generally shows a distributed program having multiple procedures. Main() 706, proc_1() 708, proc_2() 710, proc_3() 712, proc_5() 714, proc_6() 716, and proc_7()718 reside and operate in the first computing device or the client 702; and proc_4() 720 and proc_8() 722 reside and operate in the second computing device or the server 704. A solid line 724 from procedure n to procedure m denotes a call from n to m. A dashed line 726 shows how control passes from one computing device to another computing device during a remote procedure call.

[0108] Figure 8 illustrates an exemplary model 800 of execution used with remote procedure calls. In this figure, solid lines 724 are generally used to denote flow control within a computing device; and dashed lines 726 are generally used to show how control passes from one computing device to another computing device during a remote procedure call.

[0109] As such, a remote procedure call generally executes a procedure located in a separate address space from the calling code. The RPC model is generally derived from the programming model of local procedure calls and takes advantage of the fact that every procedure contains a procedure declaration. The procedure declaration defines the interface between the calling code and the called procedure. The procedure declaration defines the call syntax and parameters of the procedure. Calls to a procedure should typically conform to the procedure declaration.

[0110] Applications that use remote procedure calls look and behave much like local applications. However, an RPC application is divided into two parts: a server, which offers one or more sets of remote procedures, and a client, which makes remote procedure calls to RPC servers. A server and its client(s) generally reside on separate systems and communicate over a network. RPC applications depend on the RPC runtime

library to control network communications for them. The RPC runtime library generally supports additional tasks, such as finding servers for clients and managing servers.

[0111] A distributed application generally uses dispersed computing resources such as central processing units (CPU), databases, devices, and services. The following applications are illustrative examples of distributed applications:

[0112] A calendar-management application that allows authorized users to access the personal calendars of other users;

[0113] A graphics application that processes data on CPUs and displays the results on workstations; and

[0114] A manufacturing application that shares information about assembly components among design, inventory, scheduling, and accounting programs located on different computers.

[0115] RPC software should generally meets the basic requirements of a distributed application including:

[0116] Clients finding the appropriate servers;

[0117] Data conversion for operating in a heterogeneous environment; and

[0118] Network communications

[0119] Distributed applications include tasks such as managing communications, finding servers, providing security, and so forth. A standalone distributed application needs to perform all of these tasks itself. Without a convenient mechanism for these distributed computing tasks, writing distributed applications is difficult, expensive, and error-prone.

[0120] RPC software typically provides the code, called RPC stubs, and the RPC runtime library that performs distributed computing tasks for applications. The RPC stubs and the RPC runtime library should be linked with client and server application code to form an RPC application.

[0121] Table 1 generally shows the basic tasks for the client and server of a distributed application. Calling the procedure and executing the remote procedure, shown in italicized text, are performed by the application code (just as in a local application) but here they are in the client and server address spaces. As for the other

tasks, some are performed automatically by the stubs and RPC runtime library, while others are performed by the RPC runtime library via API calls in the application code.

Table 1: Basic Tasks of an RPC Application

Client Tasks		Server Tasks	
		1.	Select network protocols
		2.	Register RPC interfaces
		3.	Register endpoints in endpoint map
		4.	Advertise RPC interfaces and objects in the namespace
		5.	Listen for calls
6.	Find compatible servers that offer the procedures		
7.	Call the remote procedure		
8.	Establish a binding with the server		
9.	Convert input arguments into network data		
10.	Transmit arguments to the server's runtime		
		11.	Receive call
		12.	Disassemble network data and convert input arguments into local data
		13.	Locate and invoke the called procedure
		14.	Execute the remote procedure
		15.	Convert the output arguments and return value into network data
		16.	Transmit results to the client's runtime
17.	Receive results		

18.	Disassemble network data and convert output arguments into local data		
19.	Return results and control to calling code		

[0122] Figure 9 generally shows an exemplary partitioning 900 of RPC application code segments 904 and 914, RPC interfaces 906 and 916, client and server stubs 908 and 918, and the RPC runtime libraries 910 and 920 in the RPC client 902 and the RPC server 904.

[0123] The RPC client 902 or the RPC server 912 typically contains RPC application code segments 904 and 914, RPC interfaces 906 and 916, stubs 908 and 918, and the RPC runtime libraries 910 and 920. The RPC application code segments 904,914 are generally the code written for a specific RPC application by the application developer. The RPC application code segments 904,914 generally implement and call remote procedures, and also calls needed routines or procedures in the RPC runtime library. An RPC stub 908,918 is generally an interface-specific code module that uses an RPC interface 906,916 to pass and receive arguments. A client 902 and a server 912 typically contain complementary RPC stubs 906,916 for each shared RPC interface 906,916. The RPC runtime library 910,920 generally manages communications for RPC applications. In addition, the RPC runtime library 910,920 should support an Application Programming Interface (API) used by RPC application code to enable RPC applications to set up their communications, manipulate information about servers, and perform optional tasks such as remotely managing servers and accessing security information.

[0124] RPC application code segments 904,914 usually differ for clients and servers. RPC application code 914 on the server 912 typically contains the remote procedures that implement one RPC interface. RPC application code 904 on the corresponding client 902 typically contains calls to those remote procedures.

[0125] RPC stubs 908,918 generally perform basic support functions for remote procedure calls. For instance, RPC stubs 908,918 prepare input and output arguments for transmission between systems with different forms of data representation. RPC stubs

908,918 use the RPC runtime library 910,920 to handle the transmission between the client 902 and server 904. RPC stubs 908 on the client 902 can also use the local RPC runtime library 910 to find appropriate servers for the client 902.

[0126] Figure 10 generally shows an exemplary marshalling and unmarshalling between client data structures and server data structures. When the client RPC application code calls a remote procedure, the client RPC stub 908 should prepare the input arguments 1002 for transmission. The process for preparing arguments for transmission is known as ``marshalling."

[0127] Marshalling 1004 generally converts input or call arguments 1002 into a byte-stream format and packages them for transmission. Upon receiving call arguments, a server RPC stub 918 unmarshalls 1014 them. Unmarshalling 1014 is generally the process by which a stub disassembles incoming network data and converts it into application data using a format that the local system understands. Marshalling 1004,1016 and unmarshalling 1014,1006 both occur twice for each remote procedure call. The client RPC stub 908 marshalls 1004 input arguments 1002 and unmarshalls 1006 output arguments 1008. The server RPC stub 918 unmarshalls 1014 input arguments 1006 and marshalls 1016 output arguments 1008. Marshalling and unmarshalling permit client and server systems to use different data representations for equivalent data. For example, the client system can use ASCII data 1002,1008 and the server system can use Unicode data 1018 as shown in figure 10.

[0128] The IDL compiler (a tool for application development) generates stubs by compiling an RPC interface definition written by application developers. The compiler generates marshalling and unmarshalling routines for platform-independent IDL data types. To build the client for an RPC application, a developer links client application code with the client stubs of all the RPC interfaces the application uses. To build the server, the developer links the server application code with the corresponding server stubs.

[0129] In addition to one or more RPC stubs, each RPC server and RPC client should be linked with a copy of the RPC runtime library. The RPC runtime library generally provides runtime operations such as controlling communications between clients and servers and finding servers for clients on request. RPC stubs in the client and

the server typically exchange arguments through the RPC runtime library that is respectively local to the client and the server. The RPC runtime library on the client typically transmits remote procedure calls to the server. The RPC runtime library on the server generally receives the remote procedure calls from the client and dispatches each call to the appropriate RPC stub on the server. The RPC runtime library then sends the results of each call to the RPC runtime library on the client.

[0130] RPC application code on the server must also contain server initialization code that calls routines in the RPC runtime library on the server when the server is starting up and shutting down. RPC application code on the client can also call RPC runtime library routines for initialization purposes. Furthermore, RPC application code on the server and RPC application code on the client can also contain calls to RPC stub-support routines. RPC stub-support routines generally allow applications to manage programming tasks such as allocating and freeing memory.

[0131] Figure 11 generally shows exemplary roles of RPC application code segments 904 and 914, RPC interfaces 906 and 916, RPC stubs 908 and 918, and RPC runtime libraries 910 and 920 during a remote procedure call. The client's application code or calling code 908 invokes a remote procedure call, passing the input arguments 1002 through the client's RPC interface 906 to the client stub 908. The client stub 908 marshalls the input arguments 1002 and dispatches the call to the client's RPC runtime library 910. The client's RPC runtime library 910 transmits the input arguments 1002 to the server's RPC runtime library 920, which dispatches the call to the server stub 918 for the RPC interface 916 of the called procedure. The server's stub 918 unmarshalls the input arguments 1002 and passes them to the called remote procedure 914. The server's application code or remote procedure 914 executes and then returns any output arguments 1008 to the server stub 918. The server stub 918 marshalls the output arguments 1008 and returns them to the server's RPC runtime library 920. The server's RPC runtime library 920 transmits the output arguments 1008 to the client's RPC runtime library 910, which dispatches them to the client stub 901. The client's stub 908 unmarshalls output arguments 1008 and returns them to the calling code 904.

[0132] In one embodiment, remote procedure calls are made with blocking enabled to ensure synchronization between the objects 502 on the browser and the

objects 504 managed by the OM (shown in figures 5A and 5B). With blocking enabled, control would typically not be passed back to the calling code until the called remote procedure finishes executing.

[0133] Figure 12 generally outlines an exemplary process 1200 of building a distributed RPC application. The process generally includes the following basic tasks:

[0134] Designing the application, deciding what procedures are needed and which of the needed procedures will be remote procedures, and deciding how the remote procedures will be grouped into RPC interfaces (block 1205);

[0135] Using the Universal Unique Identifier (UUID) generator to generate a UUID for each of the RPC interfaces (block 1210);

[0136] Using the Interface Definition Language (IDL) to describe the RPC interfaces for planned data types and remote procedures (block 1215);

[0137] Generating the client and server stubs by compiling the IDL description using an IDL compiler (block 1220);

[0138] Writing or modifying application code using a programming language that is compatible with the RPC stubs, so that the application code works with the stubs (block 1225);

[0139] Generating object code from application code (block 1230); and

[0140] Linking the local RPC runtime library and the object code generated from the application code to generate executable code (block 1235).

[0141] Figure 13 generally illustrates the linking of the local RPC runtime library and the object code generated from the application code. For the client, object code of the client stub 908, the client application code or calling code 904, and the client's RPC runtime library 910 are linked using a linker 1302 to generate the client executable code 1304. For the server, object code for the server stub 918, the server's initialization code 1308, the server's application code or remote procedures 914, and the server's RPC runtime library 916 are linked using the linker 1302 to generate the server executable code 1306.

[0142] Traditionally, calling code and called procedures share the same address space. In an RPC application, the calling code and the called remote procedures are not linked; rather, they communicate indirectly through an RPC interface. An RPC interface

is generally a logical grouping of operations, data types, and constants that serves as a contract for a set of remote procedures. RPC interfaces are typically compiled from formal interface definitions written by application developers using the Interface Definition Language (IDL).

[0143] In developing a distributed application, an interface definition should be defined in IDL. The IDL compiler generally uses the interface definition to generate a header file, a client stub file, and a server stub file. The IDL compiler can produce header files in a standard programming language, and stubs as source files or as object file. For some applications, an Attribute Configuration File (ACF) accompanying the interface definition may be defined. If an ACF exists, the IDL compiler interprets the ACF when it compiles the interface definition. Information in the ACF is used to modify the code that the compiler generates.

[0144] The header of each RPC interface typically contains a Universal Unique Identifier (UUID), which is a hexadecimal number that uniquely identifies an entity. A UUID that identifies an RPC interface is generally known as an interface UUID. The interface UUID ensures that the interface can be uniquely identified across all possible network configurations. In addition to an interface UUID, each RPC interface contains major and minor version numbers. Together, the interface UUID and version numbers form an interface identifier that identifies an instance of an RPC interface across systems and through time.

[0145] Notifications

[0146] Returning to figures 5A and 5B, objects 502 on the browser are generally synchronized with corresponding or mirrored objects 504 managed by the OM so that changes can be reflected. Synchronization can be accomplished through a remote procedure call (RPC) mechanism 528 and a notification mechanism 530.

[0147] The notification mechanism 530 generally provides the means by which data in an object 502 on the browser (e.g., JSSBusComp 518) can be updated when data or status is changed in a corresponding object 504 managed by the OM (e.g., CSSBusComp 510). In one embodiment, the CSSSWEView object 506 would collect

one or more notifications, and send them to the objects 502 on the browser at the end of a view show cycle.

[0148] In one embodiment, the following exemplary or illustrative notifications can be transmitted. It should be noted that some of the notifications listed below require parameters. In other cases, the framework can understand the context for these notifications by simply knowing its current state.

[0149] *NotifyBeginNotifys* - Indicates the start of a set of notifications.

[0150] *NotifyEndNotifys* - Indicates the end of a set of notifications

[0151] *NotifyStateChanged* - Indicates that there has been a change of state. An exemplary scenario in which a change of state may occur is when the system goes into a query state (where the user can enter query conditions execute a query), and then goes into a commit pending state (where the user can update the data values).

[0152] *NotifyBeginQuery* - Indicates that the business component is in a state ready to accept query conditions

[0153] *NotifyExecute* - Indicates that the business component has been executed (i.e., has executed a query on the database and repopulated itself with fresh data). This notification can include an optional parameter to provide query specifications.

[0154] *NotifyEndQuery* - Indicates that a query has been completed.

[0155] *NotifyDeleteRecord* - Indicates that a record has been deleted from the database.

[0156] *NotifyDeleteWorkSet* - Indicates that a record has been removed from the working set. This notification can include a parameter that provides the index of the working set row that needs to be removed.

[0157] *NotifyInsertWorkSet* - Indicates that a record has been added to the current working set. This notification can include parameters to provide the index of the new record in the working set.

[0158] *NotifyInsertWSFieldVals* - Indicates that certain value(s) need to be added to a field in the working set.

[0159] *NotifyNewActiveField* - Indicates that a new field is active or current.

[0160] *NotifyNewActiveRow* - Indicates that a new row (i.e., record) is active or current.

[0161] *NotifyNewData* - Indicates that there has been change(s) in the data.

[0162] *NotifyNewDataWS* - Indicates that there has been change(s) in the data in the working set.

[0163] *NotifyNewFieldData* - Indicates that a particular field has new data. This notification includes parameter that provides the name of the field.

[0164] *NotifyNewFieldQuerySpec* - Indicates that a particular field has a new query or search specification.

[0165] *NotifyNewPrimary* - Indicates that a new record has become the primary record.

[0166] *NotifyNewRecord* - Indicates that a new record has been created.

[0167] *NotifyNewRecordData* - Indicates that a newly created record has new data.

[0168] *NotifyNewRecordDataWS* - Indicates that a newly created record has new data that to be populated into the working set.

[0169] *NotifyNewSelection* - Indicates the selection and de-selection of a record.

[0170] *NotifyNewSelIds* - Indicates the selection of multiple records.

[0171] *NotifyPageRefresh* - Indicates that the UI needs to be refreshed. This notification is typically used when a generally substantial change in the UI is required rather than just updating data. This notification can cause a server trip to fetch a new page and display it in the browser.

[0172] *NotifyScrollData* - Indicates that the records need to be scrolled up or down. Certain User Interface objects, such as list applets, can use this notification to scroll up and down to show different rows.

[0173] *NotifyChangeSelection* - Indicates that there has been a change in record selection (i.e., the current row has either been selected or deselected).

[0174] *NotifySelModeChange* - Indicates a change in the selection mode. In one embodiment, there can be two modes of selection, including (i) selection of one

record at a time and (ii) selection of multiple records simultaneously (e.g., deleting multiple records with one command).

[0175] *NotifyTotalsChanged* - Indicates that total values need to be changed. In some cases when multiple records are displayed, some fields may also display the summation of values in all the records. This notification indicates that total value has changed.

[0176] *NotifyLongOpProgress* - Indicates that a long (i.e. time consuming) action is in progress. This notification is used by the User Interface to provide feedback to the user, such as showing a progress or status bar showing how much of the task is currently complete.

[0177] *NotifyGeneric* – This is a generic notification used to notify the User Interface object of some special conditions that are not covered by set of notifications listed and described above. Each type of generic notification can include a parameter providing a name for the generic notification so that one type of generic notification can be distinguished from another type of generic specification. Each type of generic notification can include its unique parameters.

[0178] General Communication Processes

[0179] As stated above, objects on the browser and objects managed by the OM can be configured to reside and operate on one or multiple computing devices. As shown above, figure 6A illustrates an exemplary configuration 600 in which objects 502 on the browser and objects 504 managed by the OM reside and operate on multiple computing devices 602 and 604, including a client 602 and a server 604. Figure 6B illustrates an exemplary configuration 650 in which objects 502 on the browser and objects 504 managed by the OM reside and operate on one computing device 652.

[0180] Figure 14 generally illustrates an exemplary process 1400 of communication between the browser-side or client-side objects 502 and server-side objects 504 running on a multiple-device configuration 600 shown in figure 6A. The exemplary process 1400 of communication of figure 14 is typically by user input and is handled first at browser-side applet, JSSApplet (block 1405). When the user clicks on a

link or other user interface feature, an invoke method is generated. The JSSApplet is typically the first object to receive the generated invoke method (block 1410). The JSSApplet can then issue a remote procedure call, by way of the JSSApplication, to retarget the invoke method to the server-side applet, CSSWEApplet (block 1415). The targeted server-side applet, CSSWEApplet, can generally respond to an RPC invoke method from the browser-side applet, JSSApplet, by setting a status flag (block 1420). In one embodiment, the status flag can be set to one the following values:

[0181] • *Continue* – This value generally indicate that the server-side applet, CSSWEApplet, has performed its share of handling the invoke method (or has no role to perform), and that the JSSApplet on the browser needs to complete the action. Notifications are provided in the response, but are often empty.

[0182] • *Completed* – This value generally indicates that the server-side applet has completed the handling of the invoke method, and that the browser needs to perform no further action other than to respond to notifications provided in the response.

[0183] • *NewPage* – This value generally indicates that the response to the invoke request or other command requires a page refresh on the browser, including re-a generation all the temporary browser-side objects. A URL is sent to the browser, so that the new page can be obtained. However, there will no notifications. This value is typically set in cases such as when a drilldown to a different view is requested.

[0184] • *Error* – This value generally indicates that the invoke method request failed. Upon receiving an Error status, the JSSApplet would typically display an error page.

[0185] When a browser-side applet, JSSApplet, calls the server through the RPC, the browser-side applet typically looks at the status flag in the response and then handles it (block 1425). If the returned status is Error, the browser-side applet would show an error page. If the returned status is Completed, the server is generally indicating that it had already handled the invoke method and that there's nothing left for the browser to do. If the returned status is Continue, the server is generally indicating that it is not handling the invoke method. The browser-side applet would generally respond to a returned status of Continue by redirecting the method to a JSSBusComp object by invoking the method directly on a JSSBusComp object. The JSSBusComp object may be able to satisfy the

invoke method request, or may have to send its own RPC call through the JSSApplication to its corresponding server-side business component.

[0186] Figure 15 shows an exemplary process 1500 of communication in which the browser-side applet invokes the method directly on the JSSBusComp object. It should be noted that the process 1500 of figure 15 will be described below as though the process had occurred in a multiple-device configuration shown in figure 6A. In block 1505, the browser-side applet, JSSApplet, redirects the method to the JSSBusComp object. In block 1510, the client-side business component object, JSSBusComp, would then issue a remote procedure call, through JSSApplication object, to a server-side business component, CSSBusComp. The server-side business component generally processes the RPC method call, sets and sends back a status, and also returns a set of notifications in appropriate cases (block 1515). In one embodiment, the status flag can have the following values:

[0187] • *Completed* – This value generally indicates the server-side business component, CSSBusComp, had generally processed the invoke method successfully. Notifications will typically occur.

[0188] • *Error* – This value generally indicates that the server-side business component, CSSBusComp, had unsuccessfully processed the invoke method call. Upon receiving a returned status of Error, the browser would typically display an error page.

[0189] In block 1520, the client-side business component (JSSBusComp) examines the returned status flag and responds appropriately. It should be noted that the server will be contacted if an invoke method call performs actions that would require synchronization between the browser and server business components. It should be further noted that there could be circumstances where the JSSBusComp object may be able to handle the request locally without needing to contact the server. An example of such circumstances is when the user performs a next record operation and then a previous record operation without changing any data.

[0190] Alternate Client-Server Embodiments

[0191] Some portions of the following description are presented in terms of components, which may be understood to be components of a system or apparatus, objects of code, other portions of code, portions of a program, or other pieces of a whole which may be expected to function collectively, in some cases with some independence between the pieces.

[0192] Figure 16A illustrates an embodiment of a web application. Client 1610 runs (is executed) on first machine 1605. Client 1610 utilizes an interface between the first machine 1605 and the second machine 1615 to couple to plug-in 1630. Typically, the interface between first machine 1605 and second machine 1615 will be the world wide web, an internet, or some other form of network suitable for connection of machines. Web server 1620 runs on second machine 1615 to service requests from the interface with first machine 1605, and plug-in 1630 is a portion of web server 1620 which is suitable for handling requests from client 1610. Note that plug-in 1630 may be a specialized component designed for use primarily or exclusively with client 1610 and supplied by an entity other than the entity expected to supply the web server 1620. Alternatively, plug-in 1630 may be a portion of web server 1620 which is suitable for handling requests from client 1610.

[0193] Plug-in 1630 recognizes requests from client 1610 and redirects those requests to SWE 1660 through an interface between second machine 1615 and third machine 1645 (such as another world wide web or internet interface for example). Such redirection may include massaging, transforming, or otherwise manipulating data forming the request in question, or it may simply include passing requests on to the SWE 1660 unchanged. SWE 1660 is part of object manager 1650 along with data manager 1670, all of which run on third machine 1645. Typically, requests from client 1610 will result in accesses of data in SQL repository (database) 1680, which is accessible by third machine 1645. Note that web server 1620 and object manager 1630 may be run on a single machine, rather than two separate machines, while still maintaining similar or identical logical relationships.

[0194] Figure 16B illustrates an alternate embodiment of a web application. First machine 1605 is suitable for either standalone operation or operation in conjunction with a connection to third machine 1645 or a similar machine allowing for access to SQL repository 1680. Client 1610 runs on first machine 1605, along with object manager 1650. Plug-in 1630 is integrated as a component or portion of object manager 1650, allowing for the same interface with client 1610 illustrated in Figure 16A, without the need for the intervening network connection. Thus, plug-in 1630 of Figure 16B may use the same or essentially the same code or other implementation as was used for the embodiment illustrated in Figure 16A.

[0195] Plug-in 1630 passes on requests to SWE 1660 which interacts with data manager 1670. Data manager 1670 determines on a frequent (potentially near continuous or alternatively regular) basis whether a connection to third machine 1645 is present. When such a connection is present, data manager 1670 accesses SQL database 1680 on third machine 1645 for purposes of handling requests. When such a connection is not present, data manager 1670 accesses local SQL database 1675 on first machine 1605 for purposes of handling requests.

[0196] In one embodiment, local SQL database 1675 is a scaled-down or limited copy of database 1680. In an alternate embodiment, local database 1675 is a copy of database 1680. Note that in either embodiment, synchronization of database 1680 with local database 1675 may be handled with a variety of well-known methods. For example, data manager 1670, in one embodiment, may handle synchronization of database 1680 with database 1675. Such synchronization will necessarily only occur when a link is present between first machine 1605 and third machine 1645. Additionally, while some embodiments may have identical schema between database 1680 and local database 1675, alternate embodiments may have a first schema for database 1680 and a second schema for database 1675.

[0197] Figure 17 illustrates another alternate embodiment of a web application. Client 1710 and local web server 1740 both run on a first machine. Local web server 1740 includes a web emulation component 1720 which is suitable for emulating a remote web server from the point of view of the client 1710. Local web server 1740 also includes a server emulation component 1730 which is suitable for accessing data which

may be utilized in handling a request from client 1710. Such data may be available from local data 1750 (such as a cache or local directory for example) on a standalone first machine. When the first machine is coupled to a second machine, network data 1775 may be available for access of data suitable for handling requests from client 1710.

[0198] As will be appreciated, client 1710 and local server 1740 may be portions of the same overall component in some embodiments, or may be separate or distinct components in alternate embodiments. Furthermore, server emulation component 1730 and web emulation component 1720 may be implemented as separate components without the need for an encompassing local server component 1740 in some embodiments. Preferably, the client 1710 is a web browser or similar interface for use by a user, with some customizations as appropriate for use in a particular web application. However, the client 1710 may be a component suitable for use with a variety of user interfaces such as web browsers to provide a web-based application.

[0199] Figure 18 illustrates an embodiment of a process of servicing requests in a web application for databases. At block 1810, a web application request is received, such as from a client. At block 1820, a determination is made as to whether a current connection between the servicing agent or component and a main database is present. If not, at block 1830, the request is serviced (or errors out) based on data present in a local SQL database accessible by the servicing agent. If the connection is present, the request is serviced (or errors out) at block 1840 based on data present in the main SQL database. Note that the main SQL database may be accessible by the servicing agent through a network connection or through some other interface, which may or may not have a connection of varying reliability. Furthermore, the servicing agent, in one embodiment, is a local web server component or process run on the same machine as a client from which a web application request originates.

[0200] Figure 19 illustrates an embodiment of a process of servicing requests in a web application. At block 1910, a web application request is received, such as from a client. At block 1920, a determination is made as to whether a current connection between the servicing agent or component and a network is present. If the connection is present, at block 1940, the request is serviced (to the extent possible) based on data accessible via a network connection. If not, at block 1930, the request is serviced (to the

extent possible) based on data present in a local cache or other source of data accessible by the servicing agent.

[0201] Note that the network connection or some other interface may or may not have a connection of varying reliability. Furthermore, it will be appreciated that the determination of presence or absence of a connection need not be made for every request, and that the determination may be made based on attempting a request via the network connection and then examining a result, lack thereof, or some form of timeout for example. Servicing requests of web applications may involve retrieval of data, may involve local processing of data already locally present, or may involve local or remote processing of remote data.

[0202] Figure 20 illustrates an embodiment of a process of operating a web application for databases. At block 2010, the web application operates in local mode, servicing requests through a local web server utilizing a local database. At block 2020, a network connection to a main or primary database is recognized. At block 2030, synchronization between the local database and the main database occurs. At block 2040, the web application operates in network mode, servicing requests from data in the main database. At block 2050, recognition of lack of a network connection or a failure in the network connection occurs. At block 2060, the web application reverts to operation in local mode.

[0203] Note that disconnection with the network may be handled in a graceful way in some instances, such that synchronization occurs prior to the disconnection. Furthermore, note that synchronization need not occur at all. Alternatively, synchronization between the main database and local database may be accomplished as a background process while operating in network mode. Similarly, any changes in data in the main database during operation in network mode may be effected at essentially the same time in the local database, or on return of an indication that committing such changes to the main database was successful.

[0204] Figure 21 illustrates an alternate embodiment of a process of operating a web application for databases. At block 2110, the system starts up, initializing itself and determining its configuration or operating parameters. At block 2020, a network connection is recognized, indicating that communication with a network will be possible.

At block 2030, synchronization between the local database and the main database occurs through the network connection. At block 2040, the web application operates in network mode, servicing requests from data in the main database. At block 2150, operation of the system stops.

[0205] Figure 22 illustrates another alternate embodiment of a process of operating a web application for databases. At block 2210, the system starts up, initializing itself and determining its configuration or operating parameters. At block 2050, recognition of lack of a network connection or a failure in the network connection occurs. At block 2060, the web application operates in local mode, servicing requests through a local web server utilizing a local database. At block 2270, operation of the system stops.

[0206] Figure 23 illustrates an alternate embodiment of a process of servicing requests in a web application for databases. At block 2300, the method begins with initialization and recognition of status. At block 2310, a determination is made as to current status of a network connection. If the network connection is absent, operation proceeds without the network at block 2320. At block 2320, a web application request is received. At block 2330, the request received at block 2320 is serviced through use of a local database or with local data. At block 2340, a determination is made as to whether operation should cease. If not, the process returns to block 2320 to await the next request. If operation should cease, at block 2390, operation stops.

[0207] If a network connection is present, the process proceeds from block 2310 to block 2350. At block 2350, a web application request is received. At block 2360, the request is serviced using data from a main database or through other data available through the network connection. At block 2370, a determination is made as to whether operation should cease. If not, the process returns to block 2350 and awaits another request. If operation should cease, the process proceeds to block 2390 and stops.

[0208] Figure 24 illustrates an alternate embodiment of a process of servicing requests in a web application. At block 2410, the method begins with initialization and recognition of status. At block 2415, a determination is made as to current status of a network connection. If the connection is present (or operational), at block 2020, a network connection to a main or primary database is recognized. At block 2030,

synchronization between the local database and the main database occurs. At block 2040, the web application operates in network mode, servicing requests from data in the main database through the network connection. When operation is completed, at block 2490, the process stops.

[0209] If the network connection is not present, or not operational, the process proceeds to block 2050. At block 2050, recognition of lack of a network connection or a failure in the network connection occurs. At block 2060, the web application operates in local mode, servicing requests through a local web server utilizing a local database. Finally, when operation is completed, at block 2490, the process stops.

[0210] Some portions of the previous detailed descriptions are presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

[0211] It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussion, it is appreciated that throughout the description, discussions utilizing terms such as "processing" or "computing" or "calculating" or "determining" or "displaying" or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

[0212] The present invention also relates to apparatus for performing the operations herein. This apparatus may be specially constructed for the required purposes, or it may comprise a general purpose computer selectively activated or reconfigured by a computer program stored in the computer. Such a computer program may be stored in a computer readable storage medium, such as, but is not limited to, any type of disk including floppy disks, optical disks, CD-ROMs, and magnetic-optical disks, read-only memories (ROMs), random access memories (RAMs), EPROMs, EEPROMs, magnetic or optical cards, or any type of media suitable for storing electronic instructions, and each coupled to a computer system bus.

[0213] The algorithms and displays presented herein are not inherently related to any particular computer or other apparatus. Various general purpose systems may be used with programs in accordance with the teachings herein, or it may prove convenient to construct more specialized apparatus to perform the required method steps. The required structure for a variety of these systems will appear from the description. In addition, the present invention is not described with reference to any particular programming language. It will be appreciated that a variety of programming languages may be used to implement the teachings of the invention as described herein.

[0214] It should be emphasized that functional components, as shown above in the figures and described in the text accompanying the figures, could be implemented using software code segments. If the aforementioned functional components are implemented using software code segments, it should be further emphasized that these code segments can be stored on a machine-readable medium, such as floppy disk, hard drive, CD-ROM, DVD, tape, memory, or any storage device that is accessible by a computing machine.

[0215] A method, system and apparatus for a mobile web client is presented. In one embodiment, the invention is a method of operating a web application. The method includes receiving a request from a client. The method further includes processing the request with a web emulator. The method also includes servicing the request from available data.

[0216] In some embodiments, the method may further include using a plug-in for a web server as the web emulator, the plug-in passing the request to a web engine, and

the web engine servicing the request in conjunction with a data manager. In some embodiments, the method may further include the data manager attempting access to available data over a network. Additionally, the method may include the data manager utilizing available data from the network upon successful access and utilizing available data from local data upon failed access to service the request. Furthermore, the method may involve the request including sending data from the client to the network and the request including receiving data from the network.

[0217] In an alternate embodiment, the invention is a method of operating a web application for use in conjunction with a database. The method includes receiving a request from a client, processing the request with a web emulator, and servicing the request from available data in a database.

[0218] In alternate embodiments the available data is available over a network in a main database or is available locally in a local database, without access to a network. Similarly, in alternate embodiments, the method may further include attempting access to available data in a main database over a network. The method may also further include: upon successful access over the network, servicing the request utilizes available data from the main database through the network. Similarly, the method may also further include: upon failed access over the network, servicing the request utilizes available data from local data in a local database.

[0219] In the foregoing detailed description, the method and apparatus of the present invention has been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the present invention. In particular, the separate blocks of the various block diagrams represent functional blocks of methods or apparatuses and are not necessarily indicative of physical or logical separations or of an order of operation inherent in the spirit and scope of the present invention. For example, the various blocks of Figure 17 may be integrated into components, or may be subdivided into components. Similarly, the blocks of Figure 18 (for example) represent portions of a method which, in some embodiments, may be reordered or may be organized in parallel rather than in a linear or step-wise fashion.

The present specification and figures are accordingly to be regarded as illustrative rather than restrictive.

CLAIMS

What is claimed is:

1. A method of operating a web application comprising:
receiving a request from a client web browser;
processing the request with a web emulator; and
servicing the request from available data.

2. The method of claim 1 wherein the available data is available over a network.

3. The method of claim 1 wherein the available data is available locally, without access to a network.

4. The method of claim 1 further comprising:
Attempting access to available data over a network.

5. The method of claim 4 wherein:
Upon successful access over the network, servicing the request utilizes available data from the network.

6. The method of claim 4 further comprising:
Upon failed access over the network, servicing the request utilizes available data from local data.

7. The method of claim 1 further comprising:
Attempting access to available data over a network;
And wherein servicing the request utilizes available data from the network upon successful access and utilizes available data from local data upon failed access.

8. The method of claim 1 wherein:

The web emulator is a plug-in for a web server, the plug-in passing the request to a web engine, the web engine servicing the request in conjunction with a data manager.

9. The method of claim 8 wherein the available data is available over a network.

10. The method of claim 8 wherein the available data is available locally, without access to a network.

11. The method of claim 8 further comprising:

the data manager attempting access to available data over a network;

And wherein the data manager utilizes available data from the network upon successful access and utilizes available data from local data upon failed access to service the request.

12. The method of claim 1 wherein the request includes sending data from the client to the network.

13. The method of claim 1 wherein the request includes receiving data from the network.

14. The method of claim 1 wherein the request includes sending and receiving data.

15. A method of operating a web application for use in conjunction with a database comprising:

receiving a request from a client;

processing the request with a web emulator; and

servicing the request from available data in a database.

16. The method of claim 15 wherein the request comes from a client web browser.

17. The method of claim 15 wherein the available data is available over a network in a main database.

18. The method of claim 15 wherein the available data is available locally in a local database, without access to a network.

19. The method of claim 15 further comprising:

Attempting access to available data in a main database over a network.

20. The method of claim 19 wherein:

Upon successful access over the network, servicing the request utilizes available data from the the main database through the network.

21. The method of claim 19 further comprising:

Upon failed access over the network, servicing the request utilizes available data from local data in a local database.

22. An apparatus for operating a web application comprising:

Means for receiving a request from a client web browser;

Means for processing the request; and

Means for servicing the request from available data.

23. A machine-readable medium embodying instructions, which, when executed by a processor, cause the processor to perform a method, the method comprising:

receiving a request from a client web browser;

processing the request with a web emulator; and

servicing the request from available data.

24. The machine-readable medium of claim 23 further embodying instructions, which, when executed by the processor, cause the processor to perform the method wherein the available data is available over a network.

25. The machine-readable medium of claim 23 further embodying instructions, which, when executed by the processor, cause the processor to perform the method wherein the available data is available locally, without access to a network.

26. The machine-readable medium of claim 23 further embodying instructions, which, when executed by the processor, cause the processor to perform the method which further comprises:

Attempting access to available data over a network;

And wherein servicing the request utilizes available data from the network upon successful access and utilizes available data from local data upon failed access.

27. A machine-readable medium embodying instructions, which, when executed by a processor, cause the processor to perform a method, the method comprising:

receiving a request from a client web browser;

processing the request with a web emulator; and

servicing the request from available data in a database.

28. The machine-readable medium of claim 27 further embodying instructions, which, when executed by the processor, cause the processor to perform the method wherein the available data is available over a network in a main database.

29. The machine-readable medium of claim 27 further embodying instructions, which, when executed by the processor, cause the processor to perform the method wherein the available data is available locally in a local database, without access to a network.

30. The machine-readable medium of claim 27 further embodying instructions, which, when executed by the processor, cause the processor to perform the method which further comprises:

Attempting access to available data in a main database over a network.

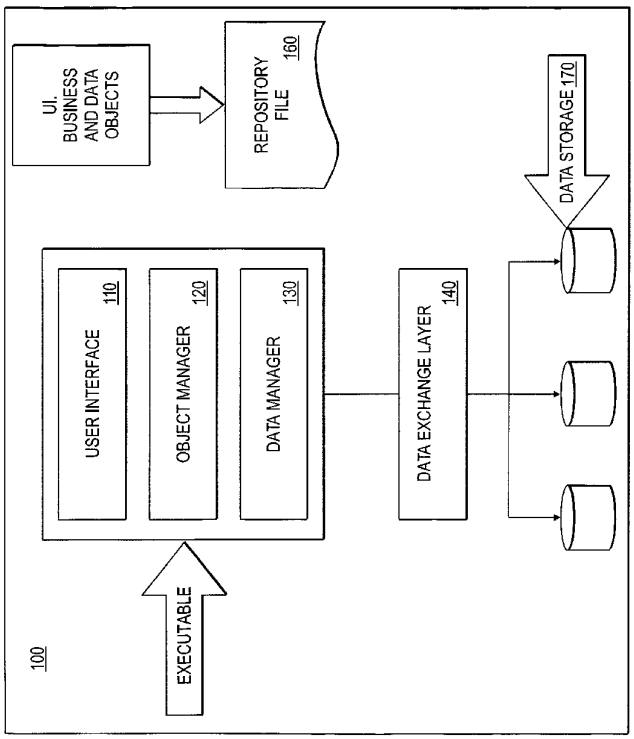
31. The machine-readable medium of claim 30 further embodying instructions, which, when executed by the processor, cause the processor to perform the method wherein:

Upon successful access over the network, servicing the request utilizes available data from the the main database through the network.

32. The machine-readable medium of claim 30 further embodying instructions, which, when executed by the processor, cause the processor to perform the method which further comprises:

Upon failed access over the network, servicing the request utilizes available data from local data in a local database.

FIG. 1


UI. BUSINESS AND DATA OBJECTS
REPOSITORY FILE 160
DATA STORAGE 170
USER INTERFACE 110
OBJECT MANAGER 120
DATA MANAGER 130
DATA EXCHANGE LAYER 140
EXECUTABLE
100

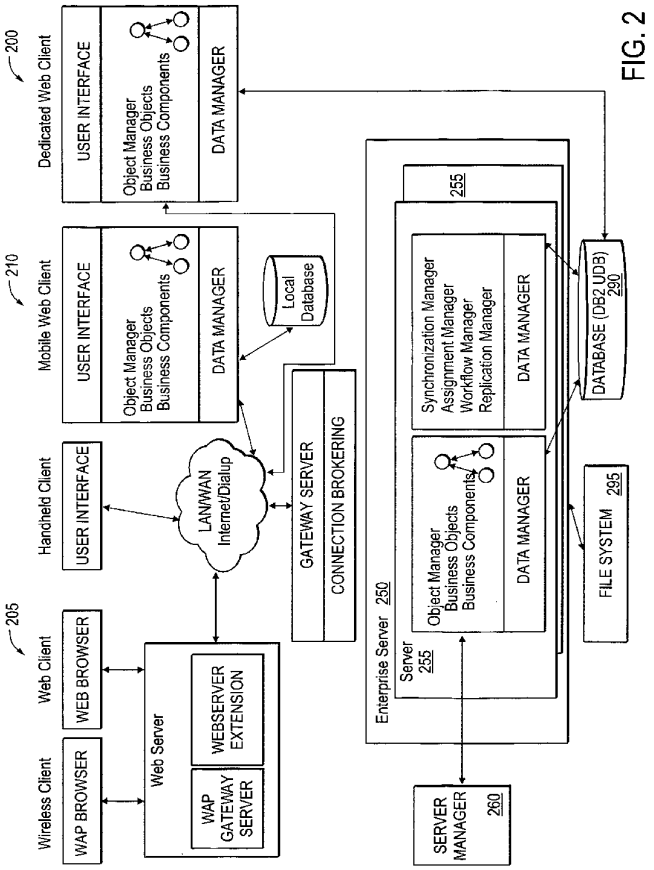

200
Dedicated Web Client
USER INTERFACE
Object Manager
Business Objects
Business Components
DATA MANAGER
210
Mobile Web Client
USER INTERFACE
Object Manager
Business Objects
Business Components
DATA MANAGER
Local Database
Handheld Client
USER INTERFACE
LAN/WAN
Internet/Dialup
GATEWAY SERVER
CONNECTION BROKERING
205
Web Client
WEB BROWSER
Wireless Client
WAP BROWSER
Web Server
WEBSERVER EXTENSION
WAP GATEWAY SERVER
Enterprise Server 250
Server 255
Object Manager
Business Objects
Business Components
DATA MANAGER
255
Synchronization Manager
Assignment Manager
Workflow Manager
Replication Manager
DATA MANAGER
DATABASE (DB2 UDB) 290
FILE SYSTEM 295
SERVER MANAGER 260

FIG. 2

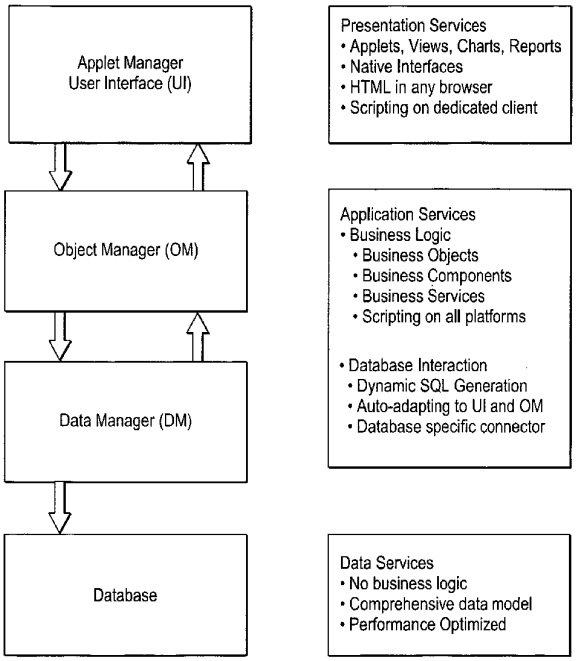

Applet Manager
User Interface (UI)
Object Manager (OM)
Data Manager (DM)
Database
Presentation Services
• Applets, Views, Charts, Reports
• Native Interfaces
• HTML in any browser
• Scripting on dedicated client
Application Services
• Business Logic
• Business Objects
• Business Components
• Business Services
• Scripting on all platforms
• Database Interaction
• Dynamic SQL Generation
• Auto-adapting to UI and OM
• Database specific connector
Data Services
• No business logic
• Comprehensive data model
• Performance Optimized

Fig. 3

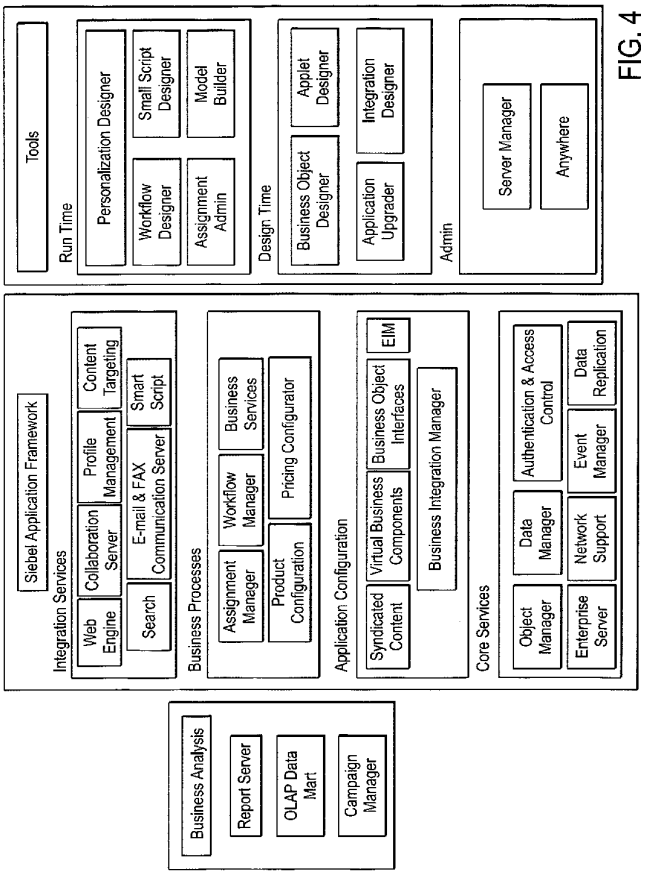

Tools
Run Time
Personalization Designer
Small Script Designer
Workflow Designer
Model Builder
Assignment Admin
Design Time
Business Object Designer
Applet Designer
Application Upgrader
Integration Designer
Admin
Server Manager
Anywhere
Siebel Application Framework
Integration Services
Web Engine
Collaboration Server
Profile Management
Content Targeting
Search
E-mail & FAX Communication Server
Smart Script
Business Processes
Assignment Manager
Workflow Manager
Business Services
Product Configuration
Pricing Configurator
Application Configuration
Syndicated Content
Virtual Business Components
Business Object Interfaces
EIM
Business Integration Manager
Core Services
Object Manager
Data Manager
Authentication & Access Control
Enterprise Server
Network Support
Event Manager
Data Replication
Business Analysis
Report Server
OLAP Data Mart
Campaign Manager

FIG. 4

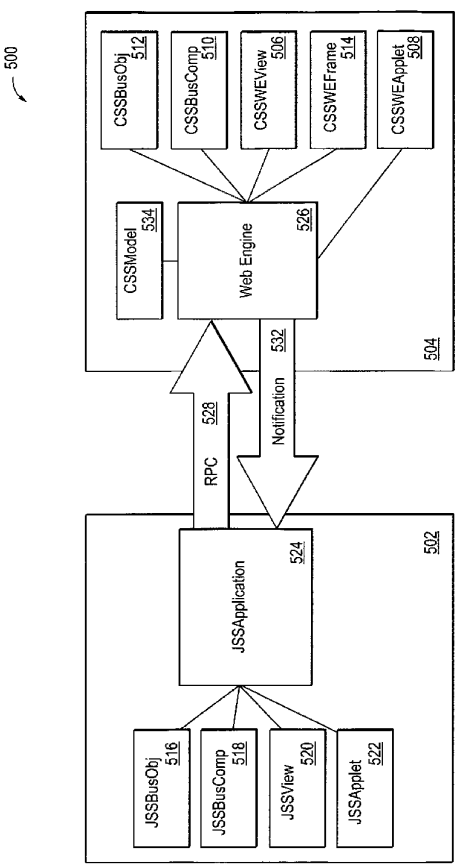

500
CSSBusObj 512
CSSBusComp 510
CSSWEView 506
CSSWEFrame 514
CSSWEApplet 508
CSSModel 534
Web Engine 526
504
532
Notification
528
RPC
502
JSSApplication 524
JSSBusObj 516
JSSBusComp 518
JSSView 520
JSSApplet 522

FIG. 5A

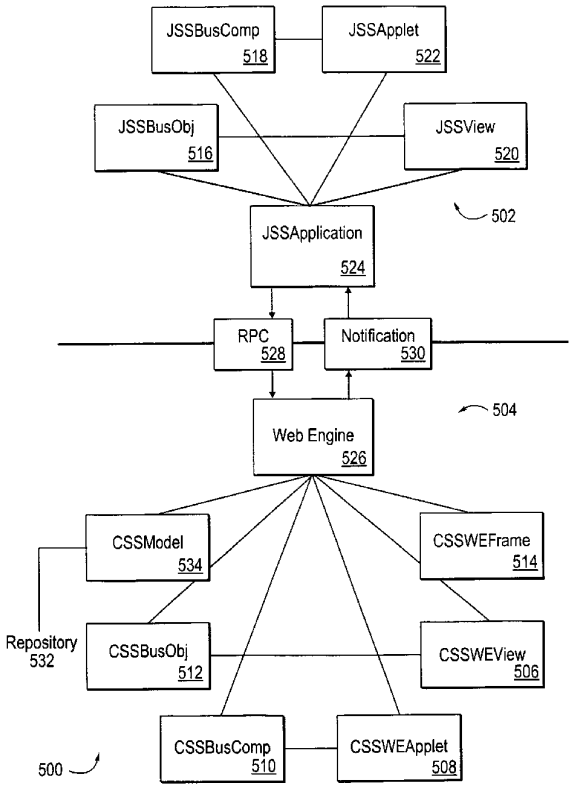

JSSBusComp
518
JSSApplet
522
JSSBusObj
516
JSSView
520
502
JSSApplication
524
RPC
528
Notification
530
504
Web Engine
526
CSSModel
534
CSSWEFrame
514
Repository
532
CSSBusObj
512
CSSWEView
506
CSSBusComp
510
CSSWEApplet
508
500

Fig. 5B

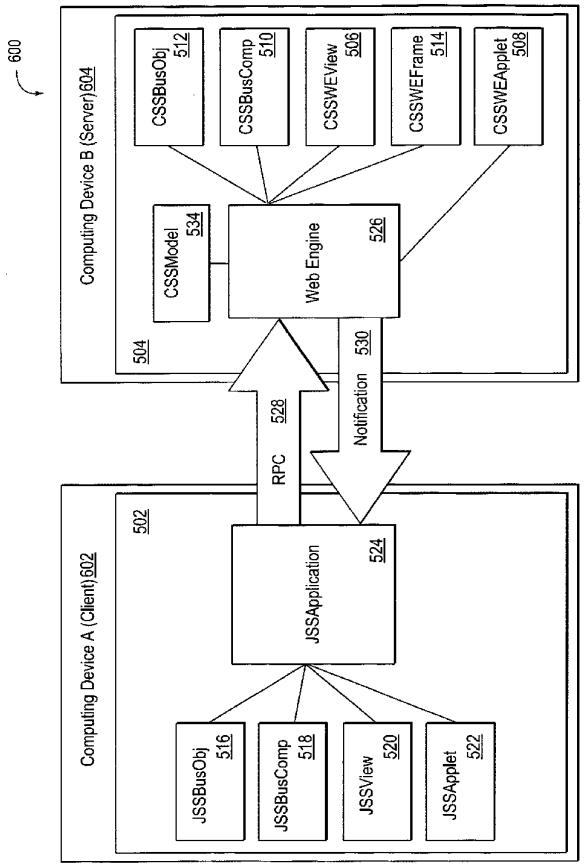

600
Computing Device B (Server) 604
CSSBusObj 512
CSSBusComp 510
CSSWEView 506
CSSWEFrame 514
CSSWEApplet 508
CSSModel 534
Web Engine 526
504
RPC 528
Notification 530
Computing Device A (Client) 602
502
JSSApplication 524
JSSBusObj 516
JSSBusComp 518
JSSView 520
JSSApplet 522

FIG. 6A

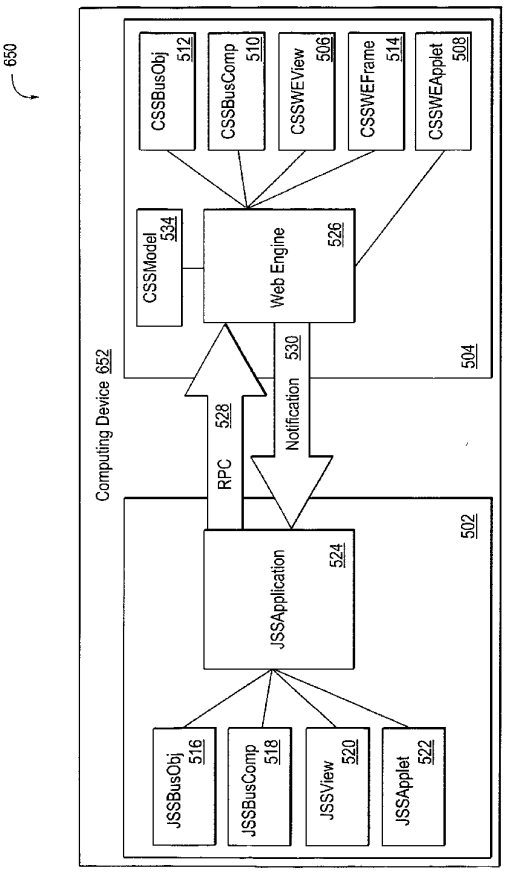

650
Computing Device 652
CSSBusObj 512
CSSBusComp 510
CSSWEView 506
CSSWEFrame 514
CSSWEApplet 508
CSSModel 534
Web Engine 526
504
530
Notification
528
RPC
502
JSSApplication 524
JSSBusObj 516
JSSBusComp 518
JSSView 520
JSSApplet 522

FIG. 6B

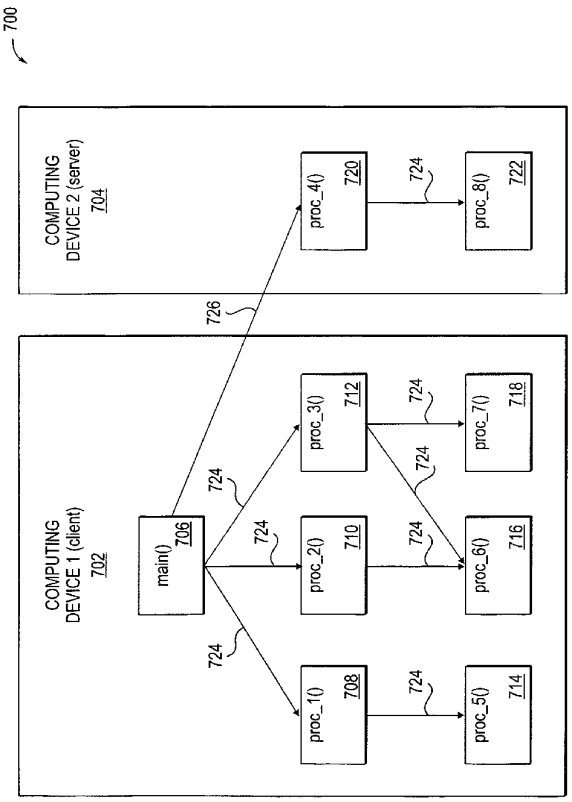

700
COMPUTING DEVICE 2 (server) 704
COMPUTING DEVICE 1 (client) 702
main() 706
proc_1() 708
proc_2() 710
proc_3() 712
proc_5() 714
proc_6() 716
proc_7() 718
proc_8() 722
proc_4() 720
724
726

FIG. 7

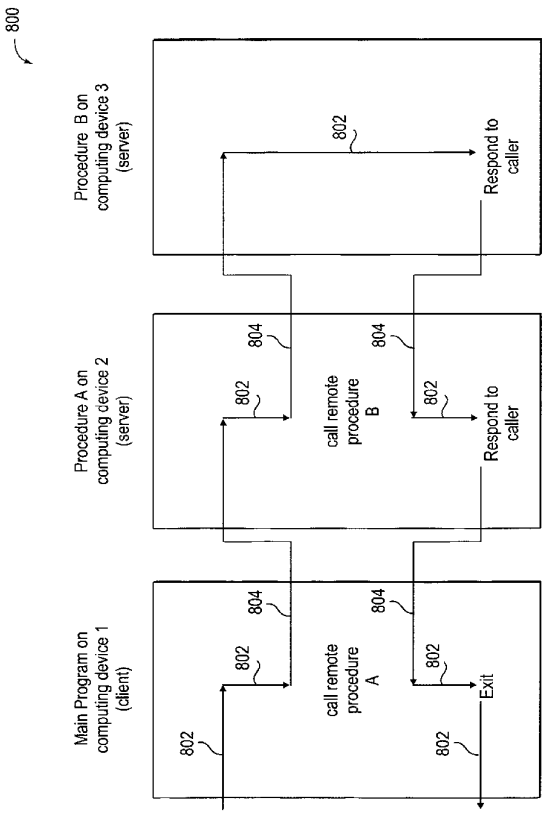

800
Procedure B on computing device 3 (server)
802
Respond to caller
Procedure A on computing device 2 (server)
804
802
call remote procedure B
804
802
Respond to caller
Main Program on computing device 1 (client)
804
802
call remote procedure A
804
802
Exit
802
802

FIG. 8

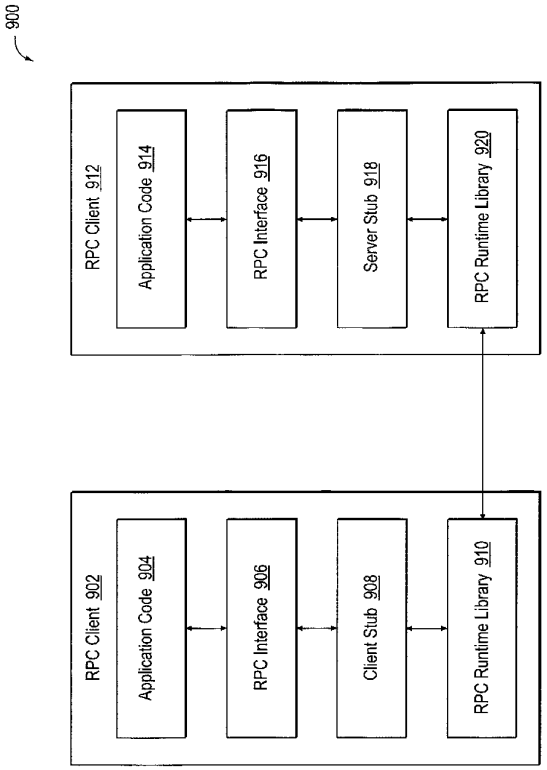

900
RPC Client 912
Application Code 914
RPC Interface 916
Server Stub 918
RPC Runtime Library 920
RPC Client 902
Application Code 904
RPC Interface 906
Client Stub 908
RPC Runtime Library 910

FIG. 9

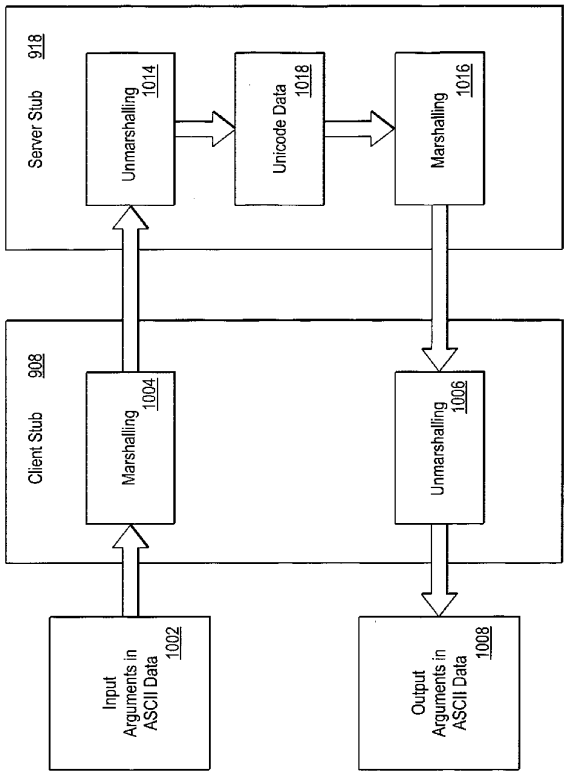

Server Stub 918
Unmarshalling 1014
Unicode Data 1018
Marshalling 1016
Client Stub 908
Marshalling 1004
Unmarshalling 1006
Input Arguments in ASCII Data 1002
Output Arguments in ASCII Data 1008

FIG. 10

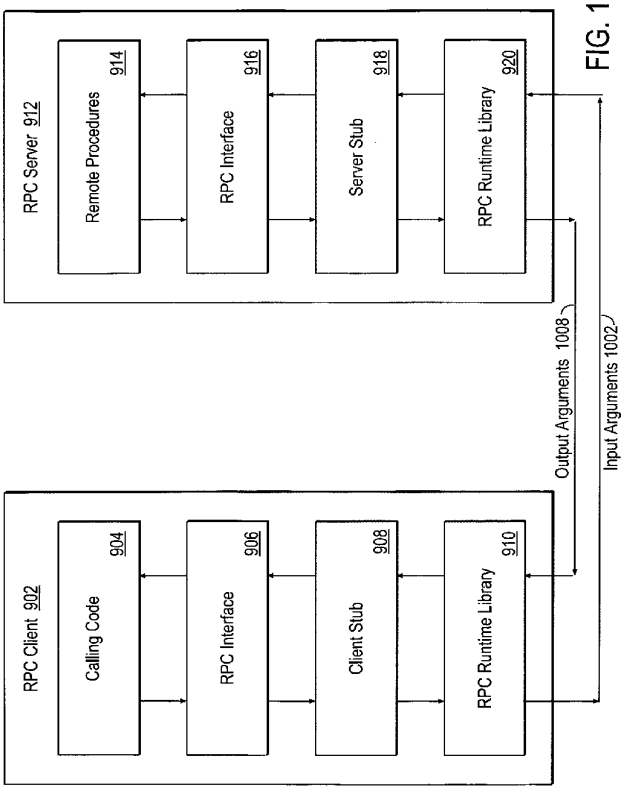

RPC Server 912
Remote Procedures 914
RPC Interface 916
Server Stub 918
RPC Runtime Library 920
Output Arguments 1008
Input Arguments 1002
RPC Client 902
Calling Code 904
RPC Interface 906
Client Stub 908
RPC Runtime Library 910

FIG. 11

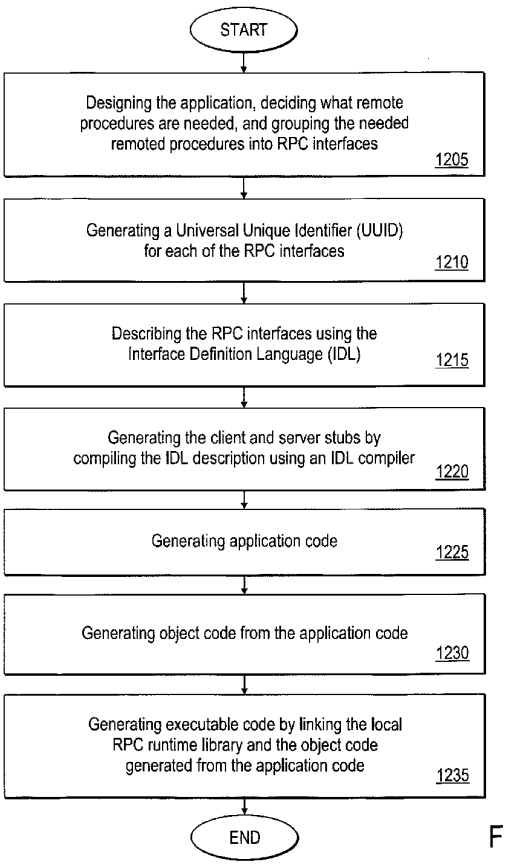

1200
START
Designing the application, deciding what remote procedures are needed, and grouping the needed remoted procedures into RPC interfaces
1205
Generating a Universal Unique Identifier (UUID) for each of the RPC interfaces
1210
Describing the RPC interfaces using the Interface Definition Language (IDL)
1215
Generating the client and server stubs by compiling the IDL description using an IDL compiler
1220
Generating application code
1225
Generating object code from the application code
1230
Generating executable code by linking the local RPC runtime library and the object code generated from the application code
1235
END

FIG. 12

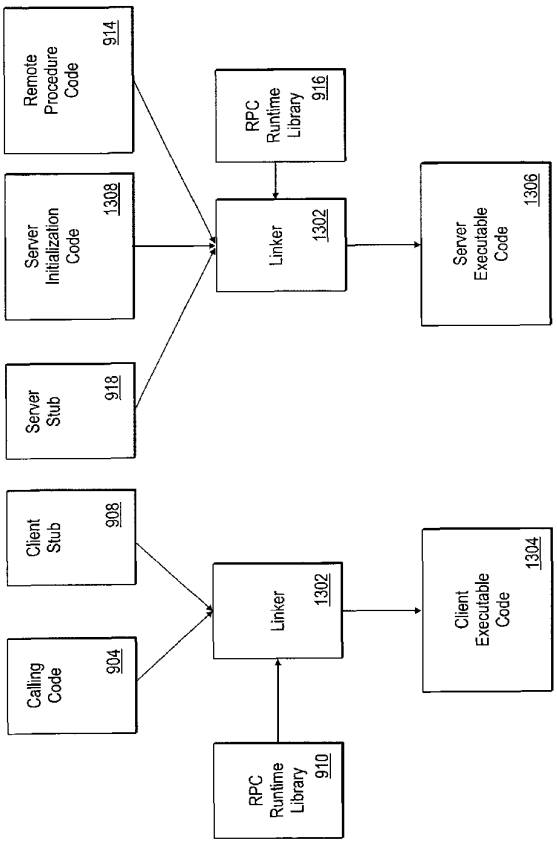

Remote Procedure Code 914
RPC Runtime Library 916
Server Initialization Code 1308
Linker 1302
Server Executable Code 1306
Server Stub 918
Client Stub 908
Linker 1302
Client Executable Code 1304
Calling Code 904
RPC Runtime Library 910

FIG. 13

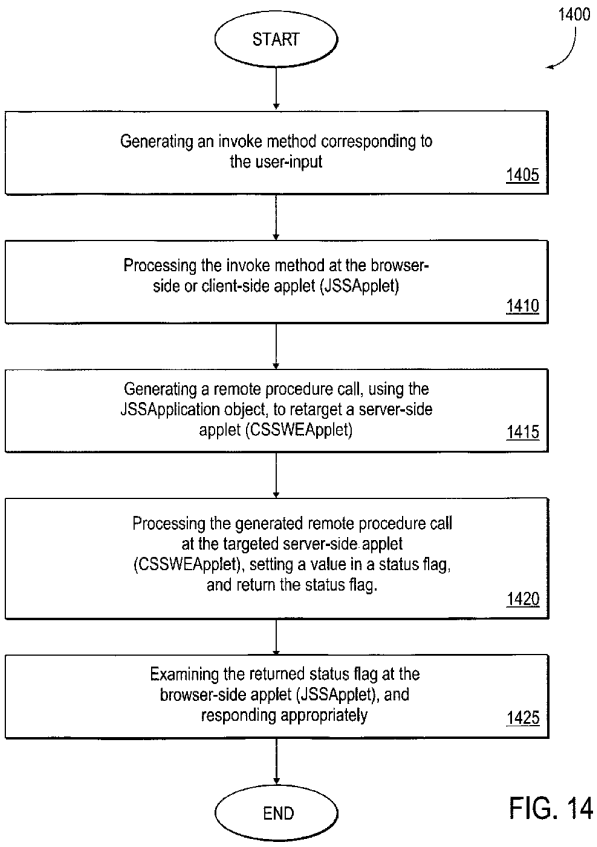

1400
START
Generating an invoke method corresponding to the user-input
1405
Processing the invoke method at the browser-side or client-side applet (JSSApplet)
1410
Generating a remote procedure call, using the JSSApplication object, to retarget a server-side applet (CSSWEApplet)
1415
Processing the generated remote procedure call at the targeted server-side applet (CSSWEApplet), setting a value in a status flag, and return the status flag.
1420
Examining the returned status flag at the browser-side applet (JSSApplet), and responding appropriately
1425
END

FIG. 14

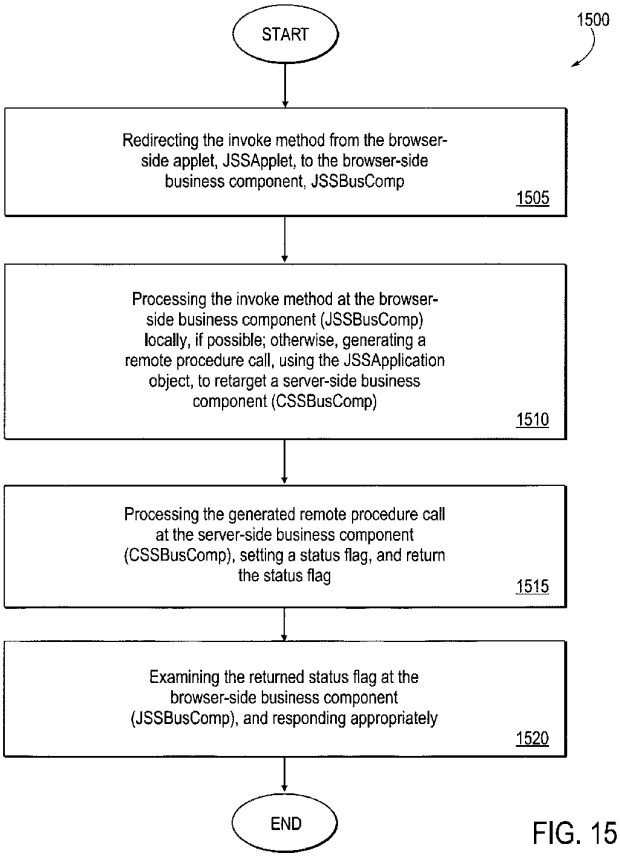

START
1500
Redirecting the invoke method from the browser-side applet, JSSApplet, to the browser-side business component, JSSBusComp
1505
Processing the invoke method at the browser-side business component (JSSBusComp) locally, if possible; otherwise, generating a remote procedure call, using the JSSApplication object, to retarget a server-side business component (CSSBusComp)
1510
Processing the generated remote procedure call at the server-side business component (CSSBusComp), setting a status flag, and return the status flag
1515
Examining the returned status flag at the browser-side business component (JSSBusComp), and responding appropriately
1520
END

FIG. 15

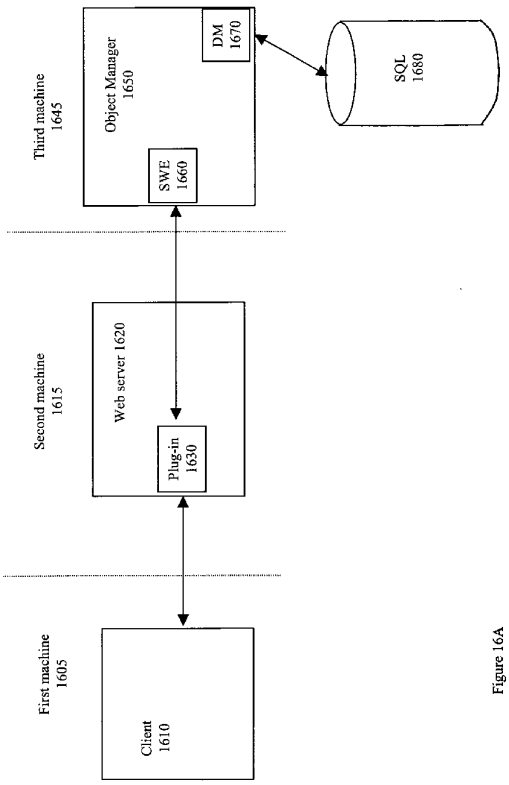

Third machine
1645
Object Manager
1650
DM
1670
SWE
1660
SQL
1680
Second machine
1615
Web server 1620
Plug-in
1630
First machine
1605
Client
1610

Figure 16A

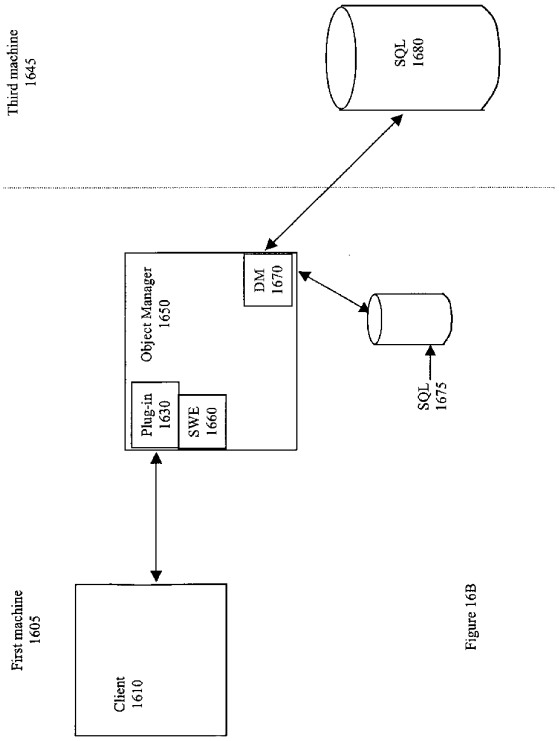

Third machine
1645
SQL
1680
Object Manager
1650
DM
1670
Plug-in
1630
SWE
1660
SQL
1675
First machine
1605
Client
1610

Figure 16B

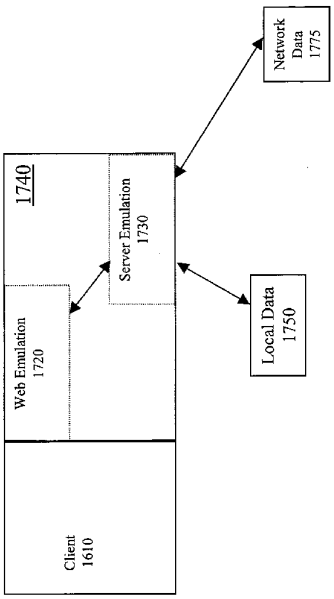

Network Data 1775
1740
Server Emulation 1730
Web Emulation 1720
Local Data 1750
Client 1610

Figure 17

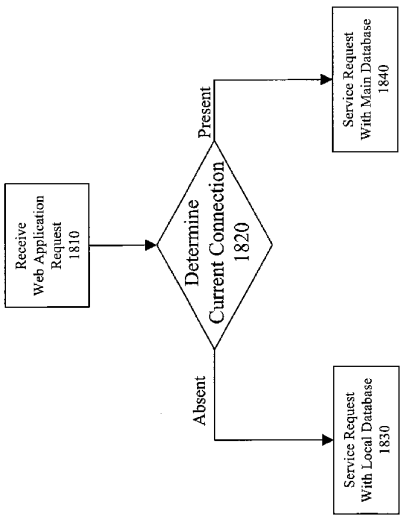

Receive
Web Application
Request
1810
Determine
Current Connection
1820
Present
Absent
Service Request
With Main Database
1840
Service Request
With Local Database
1830

Figure 18

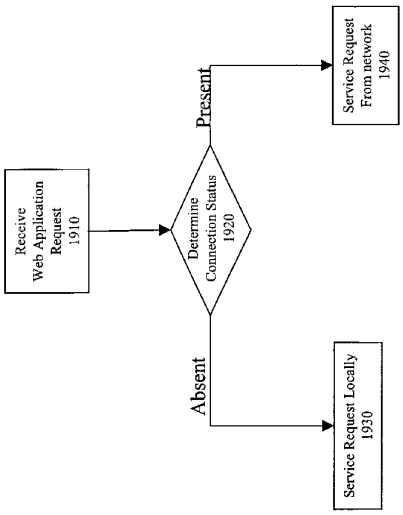

Receive
Web Application
Request
1910
Determine
Connection Status
1920
Present
Absent
Service Request
From network
1940
Service Request Locally
1930

Figure 19

Figure 20

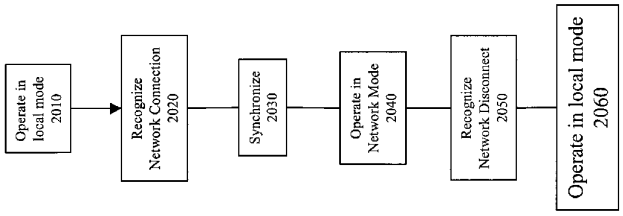

Operate in local mode 2010
Recognize Network Connection 2020
Synchronize 2030
Operate in Network Mode 2040
Recognize Network Disconnect 2050
Operate in local mode 2060

Figure 21

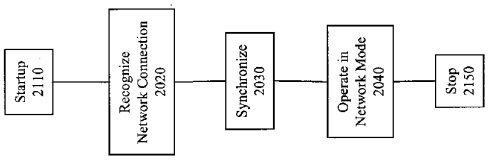

Startup
2110
Recognize
Network Connection
2020
Synchronize
2030
Operate in
Network Mode
2040
Stop
2150

Figure 22

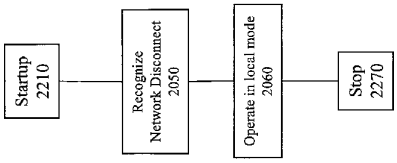

Startup
2210
Recognize
Network Disconnect
2050
Operate in local mode
2060
Stop
2270

Figure 23

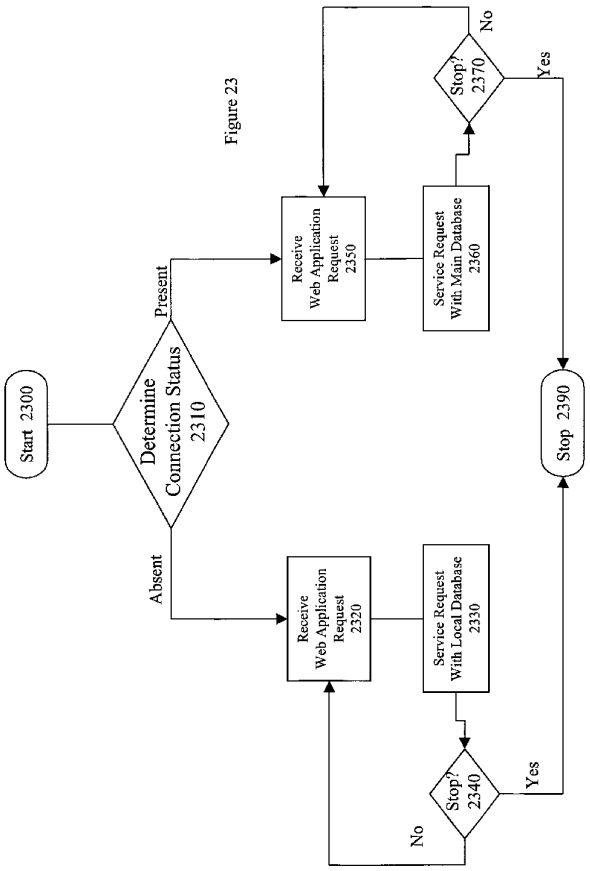
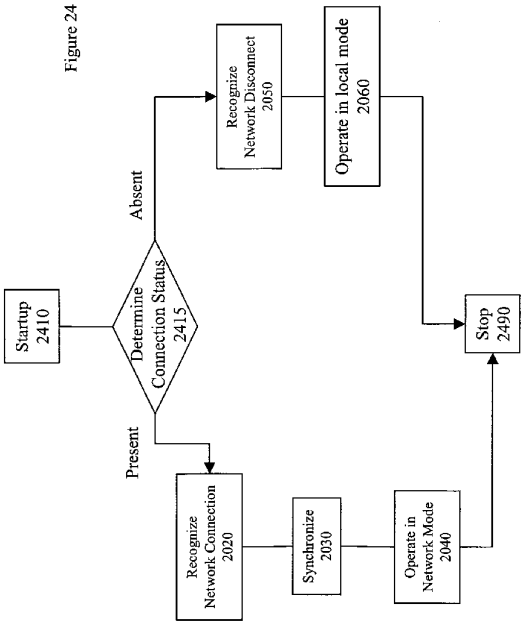

Start 2300
Determine Connection Status 2310
Present
Absent
Receive Web Application Request 2350
Service Request With Main Database 2360
Stop? 2370
No
Yes
Receive Web Application Request 2320
Service Request With Local Database 2330
Stop? 2340
No
Yes
Stop 2390

Figure 24


Startup
2410
Determine
Connection Status
2415
Absent
Present
Recognize
Network Disconnect
2050
Operate in local mode
2060
Recognize
Network Connection
2020
Synchronize
2030
Operate in
Network Mode
2040
Stop
2490

【国際調査報告】

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US02/30885

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) : G06F 15/16, 15/173
US CL : 709/217, 226
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 709/217, 226

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 6,016,318 A (TOMOIKE) 18 January 2000 (18.01.2000), see abstract, column 1, line 60 to column column 8, line 4.	1-32
A	US 6,233,541 B1 (BUTTS et al.) 15 May 2001 (15.05.2001), see the whole reference.	1-32
A	US 6,096,096 A (MURPHY et al.) 01 August 2000 (01.08.2000), see the whole reference.	1-32
A,P	US 6,374,207 B1 (LI et al.) 16 April 2002 (16.04.2002), see the whole reference.	1-32

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search
13 December 2002 (13.12.2002)

Date of mailing of the international search report
10 JAN 2003

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703)305-3230

Authorized officer
Meng Ai An
Telephone No. 703-305-3900

Form PCT/ISA/210 (second sheet) (July 1998)

フロントページの続き

(81)指定国 AP(GH,GM,KE,LS,MW,MZ,SD,SL,SZ,TZ,UG,ZM,ZW),EA(AM,AZ,BY,KG,KZ,MD,RU,TJ,TM),EP(AT,BE,BG,CH,CY,CZ,DE,DK,EE,ES,FI,FR,GB,GR,IE,IT,LU,MC,NL,PT,SE,SK,TR),OA(BF,BJ,CF,CG,CI,CM,GA,GN,GQ,GW,ML,MR,NE,SN,TD,TG),AE,AG,AL,AM,AT,AU,AZ,BA,BB,BG,BR,BY,BZ,CA,CH,CN,CO,CR,CU,CZ,DE,DK,DM,DZ,EC,EE,ES,FI,GB,GD,GE,GH,GM,HR,HU,ID,IL,IN,IS,JP,KE,KG,KP,KR,KZ,LC,LK,LR,LS,LT,LU,LV,MA,MD,MG,MK,MN,MW,MX,MZ,NO,NZ,OM,PH,PL,PT,RO,RU,SD,SE,SG,SI,SK,SL,TJ,TM,TN,TR,TT,TZ,UA,UG,US,UZ,VN,YU,ZA,ZM,ZW

(72)発明者 コッカー ジョン
アメリカ合衆国 カリフォルニア州 ９４０１０ ヒルズボロー シャトー ドライヴ ７２３

Ｆターム(参考) 5B085 AE00