(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第3628698号
(P3628698)
(45)発行日　平成17年3月16日(2005.3.16)　(24)登録日　平成16年12月17日(2004.12.17)

(51)Int.Cl.[7]　F I
G06F　9/46　G06F　9/46　360B

請求項の数 5　(全 32 頁)

(21)出願番号　特願平7-527457
(86)(22)出願日　平成7年4月21日(1995.4.21)
(65)公表番号　特表平9-512359
(43)公表日　平成9年12月9日(1997.12.9)
(86)国際出願番号　PCT/GB1995/000905
(87)国際公開番号　WO1995/029439
(87)国際公開日　平成7年11月2日(1995.11.2)
審査請求日　平成14年4月22日(2002.4.22)
(31)優先権主張番号　9407971.2
(32)優先日　平成6年4月21日(1994.4.21)
(33)優先権主張国　英国(GB)

(73)特許権者
ブリテイッシュ・テレコミュニケーションズ・パブリック・リミテッド・カンパニー
イギリス国、イーシー１エー・７エージェイ、ロンドン、ニューゲート・ストリート　８１
(74)代理人
弁理士　鈴江　武彦
(74)代理人
弁理士　村松　貞男
(74)代理人
弁理士　坪井　淳
(74)代理人
弁理士　橋本　良郎

最終頁に続く

(54)【発明の名称】メッセージ配送システム

(57)【特許請求の範囲】
【請求項１】
呼び出しプログラムと呼び出されるプログラムとの間でメッセージを配送するためのメッセージ配送システムであって、すべてのメッセージは所定のメッセージブロック内でかつメッセージインタフェースを介して配送され、
前記メッセージインタフェースは、各呼び出されるプログラムに対して、識別子とプログラムインタフェースとを記憶するための記憶手段を有し、前記メッセージブロックは、固定されたフォーマットを有する制御ブロックと、フリーフォーマットを有するデータエリアとを具備し、前記制御ブロックは、データエンティティキーワードを保持するための第１の部分と、データアイテムの特性を保持するための第２の部分とを具備し、前記第１の部分は、メッセージブロックが前記メッセージインタフェースと呼び出しプログラムまたは呼び出されるプログラムとの間で配送されるときに変更されず、前記第２の部分は、前記メッセージブロックの送信先に応じて変更され、
使用時において、前記メッセージインタフェースは、
ｉ）　識別子を含む呼び出しプログラムからのメッセージブロックを受信し、
ii）　前記識別子を特定の呼び出されるプログラムと特定のプログラムインタフェースとに関連付け、
iii）　前記特定のプログラムインタフェースを使用して、メッセージブロックを前記特定の呼び出されるプログラムに送信するメッセージ配送システム。
【請求項２】

前記呼び出されるプログラムの各々はデータエンティティに関連し、各データエンティティは複数のデータアイテムと独自のデータエンティティキーワードとを有し、前記メッセージインタフェースの記憶手段が各データエンティティキーワードを記憶する請求の範囲第１項に記載のメッセージ配送システム。

【請求項３】

前記制御ブロックの前記第２の部分に保持された前記データアイテムの特性は前記データアイテムの位置と長さを具備し、前記制御ブロックの前記第１の部分には、各データアイテムのための独自のデータアイテムタグが保持されている請求の範囲第１または第２項に記載のメッセージ配送システム。

【請求項４】

前記データアイテムは、前記メッセージブロックのデータエリア内で前記呼び出しプログラムと呼び出されるプログラムとの間に保持されている請求の範囲第１、２、３のいずれか１つに記載のメッセージ配送システム。

【請求項５】

前記データアイテムは、一度に１回前記呼び出しプログラムから前記呼び出されるプログラムに転送される請求の範囲第１乃至第４項のいずれか１つに記載のメッセージ配送システム。

【発明の詳細な説明】

本発明は呼び出しプログラムと呼び出されるプログラムとの間でメッセージを渡すメッセージ配送システムに関し、特に、すべてのメッセージが所定のメッセージブロック内で、かつ、メッセージインタフェースを介して渡されるメッセージ配送システムに関する。

大型コンピュータシステムは概して、多数のコンポーネントまたは呼び出しを介して互いに作用しあう多数のプログラムを具備する。大型コンピュータシステムを多数のプログラムに分割することは多くの点で有利である。例えば、多数のコンピュータプログラムがある程度の独立性をもってシステムに作用することを可能にし、コンピュータシステムをより容易に変更または強化することを可能にし、システム内のプログラムを他のコンピュータシステムにおいて再利用することを可能にする。

読み出しプログラムが呼び出されるプログラムを読み出すことができるように、コンピュータシステム内の種々のプログラム間のインタフェースを規定する必要がある。既知のコンピュータシステムにおいて、プログラム間のインタフェースは、呼び出しプログラム及び呼び出されるプログラムからなるコンピュータコードによって実現され、インタフェースに対する変更を行なうためにコンピュータシステム内の呼び出しプログラムが変更された場合は、変更されたプログラムを呼び出しているすべての他のプログラムは新しいインタフェース要件に一致させるべく変更されねばならない。

アプリケーションとコンピュータネットワークに設置された種々のコンピュータとの間のインタフェースを提供するシステムは、欧州特許出願EP－Ａ－0414624に記載されている。

例えばコンピュータシステムの発展によって、あるプログラムが初期の物理的ハードウェア、例えばメインフレームから、他の物理的システムに移行されるときに新たな問題が発生し、この移行を可能にするためにはインタフェースが強化されねばならない。この強化は、リモートの物理的システムへ移行されたプログラムに加えて、初期のメインフレームに駐在しているすべての呼び出しプログラムに対して行われなければならない。

本発明によれば、

すべてのメッセージが所定のメッセージブロック内でかつメッセージインタフェースを介して渡される呼び出しプログラムと呼び出されるプログラムとの間でメッセージを渡すためのメッセージ配送システムであって、

前記メッセージインタフェースは各呼び出されるプログラムに対して識別子とプログラムインタフェースとを記憶するためのメモリ手段を有し、前記メッセージブロックは、固定されたフォーマットを有する制御ブロックと、フリーフォーマットを有するデータエリアとを具備し、前記制御ブロックは、データエンティティキーワードを保持するための第１

の部分と、データアイテムの特性を保持するための第２の部分とを具備し、前記第１の部分は、メッセージブロックが前記メッセージインタフェースと呼び出しプログラムまたは呼び出されるプログラムとの間で配送されるときに変更されず、前記第２の部分は、前記メッセージブロックの送信先に応じて変更され、
使用時において、前記メッセージインタフェースは以下の処理を行なう。
ｉ）　識別子を含む呼び出しプログラムからのメッセージブロックを受信する。
ii）　識別子を特定の呼び出されるプログラムと特定のプログラムインタフェースに関連させる。
iii）　特定のプログラムインタフェースを用いして前記特定の呼び出されるプログラムにメッセージブロックを送る。
本発明の実施形態は、プログラムインタフェース詳細の中央記憶部（central repository）を提供することによって、プログラム自身によって保持するのに必要なインタフェース情報の量を減らす。すなわち、例えば、呼び出しプログラムは、読み出されるプログラムと直接通信するのではなく、所定のメッセージブロック内の識別子をメッセージインタフェースに送る。メッセージインタフェースはその後、例えば、呼び出されるプログラムの位置とインタフェース要件とを決定する。呼び出されるプログラムは同じ物理的機械上に設けられている場合は、比較的簡単な通信プロトコルが用いられる。しかしながら、呼び出されるプログラムがリモートのコンピュータシステム上に駐在しているときは、呼び出しプログラムからのメッセージは、適宜パッケージ化され、適当な通信プロトコルを介して、リモートシステムに駐在している呼び出されるプログラムに送信される。
本発明の実施形態はさらに、データ管理システムとともに使用するのに適しており、タグを各データエンティティに関連付けるとともに、所定のメッセージブロックとメッセージインタフェースとを用いることにより、以下の記載からわかるように、呼び出し及び呼び出されるプログラム間の高度のモジュール性を実現する。
以下に、添付の図面を参照して例を用いて本発明の実施形態を説明する。
図１は、本発明の実施形態によるメッセージ配送システムと、データ管理システムの他のコンポーネントとの間の関係を示すブロック図であり、
図２は、本発明の実施形態のコンポーネントを示すブロック図であり、
図３は、本発明の実施形態によって用いられる論理オブジェクト間の関係を示すブロック図であり、
図４は、本発明の実施形態のメモリ手段（ODICT）のコンポーネント間の論理関係を示すブロック図であり、
図５は、本発明の実施形態によって用いられる制御ブロック構造間の論理関係を示すブロック図であり、
図６は、本発明のメッセージ配送システムの実施形態を用いるトランザクションプログラムからのカプセルプログラムに対する呼び出しの一例を示すブロック図である。
本発明の実施形態は、複雑なコンピュータシステムの部品、特に、データ管理システムを、他の部品との取り決めに対する変更なしに、変更できる。この柔軟性はコンピュータシステムの部品を物理的に離れたコンピュータから分離することを可能にする。
データ管理システムとともに用いられる本発明の実施形態を以下に説明する。本実施形態のメッセージインタフェースは、ODICT（オブジェクト辞書）と呼ばれるメモリ手段と、XCALLと呼ばれるメッセージインタフェースとを具備する。図１において、データ管理システムはCICS（IBMの商標）と呼ばれるトランザクション処理環境１と、（コボルで書かれた）多数のトランザクション２と、本発明の実施形態によるメッセージ配送システム３と、多数のカプセル４と、DB2/（IBMの商標）データベース５とを具備する。すなわち、本実施形態においては、トランザクションプログラムが読み出しプログラムであり、カプセルプログラムが呼び出されるプログラムである。データ管理システムが使用中であるときは、呼び出しは呼び出しプログラム（トランザクション２）から呼び出されるプログラム（カプセル４）へ直接渡されない。呼び出しはまずXCALL3へ渡され、次にそこから呼び出されるプログラム、カプセル４に渡される。XCALL3は、呼び出されるプログラムを呼出者

から実質的に隠す、あるいは呼出者を呼び出されるプログラムから隠すいくつかのサービスを提供する。
図２において、メッセージインタフェース（XCALL）３は、ODICT（オブジェクト辞書）と呼ばれるメモリ手段６における呼び出される及び呼び出しプログラムの外で定義され、インタフェース定義の符号化されたバージョンはラン時にどのデータアイテムがプログラム間で渡されているかを識別するために使用される。プログラムがコンパイルされるとき、他のプログラムへの呼び出しの設定及び／または他のプログラムからの呼び出しの受信に要求される定義が辞書から複写される。
コンパイルされている特定の呼び出しに必要なコンポーネントのみが呼び出しプログラムに複写される。すべての呼び出しによって要求されるコンポーネントが呼び出されるプログラムがコンパイルされるときに複写される。
呼び出されるプログラムが、いくつかの呼出者のための新たなまたは強化されたサービスを提供すべく変更されるとき、XCALLの隠蔽機能は他の呼出者が影響されないことを意味する。
XCALLは一度に１属性だけ呼び出しプログラムのデータエリアから呼び出されるプログラムのデータエリアへ及び／または呼び出されるプログラムのデータエリアから呼び出しプログラムのデータエリアへデータを移動する。XCALLは、プログラムが複写されるときにODICTから複写されるデータエンティティ数、カラム数、カラム長さに関して、インタフェースの符号化された記述を使用する。このプロセスは、２つのプログラムが同じコンピュータ上に駐在しているなら、あるいは、別のマシン上に位置しているならば、同じように適用することができる。
以下に本実施形態で用いられる術語を定義する。　カプセル:XCALLの基本単位。この名前はオブジェクト指向に関する技術分野において、データ及びプロセスを外界から隠蔽する技術であるカプセル化に由来している。カプセルはすべてのデータ定義と所定のENTITYに対するMETHODとを具備する。アプリケーションプログラムはメソッドを読みだした後、XCALLはODICTから構成されたLDSテーブルを用いて、メソッド名を適当なカプセルに関連付ける。あるエンティティは検索と更新について１つずつのカプセルを有するが、概してエンティティごとに１つのカプセルが存在する。まれにはエンティティは１つ以上のカプセルによって更新される。呼び出しは、静的（リンクエディット時）ではなく動的（ラン時）であり、これによって、用いられるプログラムを再コンパイルあるいは再テストを行なうことなしに、それを含むコードに対して変更が行なわれる。コードとデータが同じ位置に存在することも問題の識別とインパクト解析を行なうときの補助となる。
推論によるフィールド：メソッドによって提供されるがデータベースには直接保持されないフィールド。この方法は値を隠蔽し、例えば、コントラクトスタートデートとコントラクトタームが与えられたとき、コントラクトエンドデートを返すことは有用である。トランザクションはそれがどのように引き出されるかについて何等考慮することなしにこのようなフィールドを使用できる。同様にして逆の場合も使用できる。すなわち、コントラクトスタート及びエンドデートが与えられたとき、更新メソッドはタームを計算できる。他の使用は、ステータスフラグ、入力フィールド列をとるリスプタイプのメソッドのためのエラーフラグを含む。リターン時、入力フィールドの各々に対してデータが見いだされたか否かを決定するためにいくつかの機構が必要となる。
エンティティ：データベース上に保持された識別可能なタイプ。これはDB2テーブルかあるいは（IBMの商標）レコードタイプであり、例えばプロダクトを表す。または複合構造であり、例えば、関連するポストコード、ポストタウン、スルースフェア、ローカリティディテールなどを有するアドレスレコードタイプであり、そのようなエンティティはDB2テーブル数からのデータを具備する。エンティティ名は通常DB2テーブル名（これが適当ならば）であるが、18文字に制限される。各エンティティは独自の識別キーワードまたはエンティティ数と、付加的に４つの文字エンティティ略字とを有する。
エンティティブロック：エンティティに対してプレコンパイラによって生成されるデータエリア。２つのタイプが存在する。１つはメソッド（メソッドコールエンティティブロッ

ク）に対する呼び出しによって生成されるエンティティブロックであり、当該メソッドと他のメソッドに対する入力及び出力として必要なフィールドである。もう１つはカプセル内で生成されるフルエンティティブロック（カプセルエンティティブロック）であり、エンティティ内のすべてのフィールドを含む。エンティティブロック内の名前は、ODICTに保持されたエンティティ略字（その後にDB2カラム名が続く）と、ハイフンによって置き換えられるアンダースコア及びハイフンによって分離される２つのパーツを具備する。カプセルエンティティブロックの場合は、カプセル（”内部”メソッドコールを含む）内への呼び出しのために１を前に付加するかあるいは、他のカプセル内のメソッドに対する呼び出しの結果として組み込まれるエンティティのための"C"を前に付加する。エンティティブロックのサイズはMAPオプションセットを用いてコボルコンピレーションリストによって決定される。

メソッド：エンティティ上での独自のアクションを実行するためのプログラムによって呼び出されるプロシージャ。DB2術語に従えば、これはテーブル上の新たなロー（row）を挿入するか、存在するローを検索、更新、削除することを意味する（ここで、メソッドは（IBMの商標）データベースあるいは他のデータ記憶システムについて動作するのに用いられる。メソッドは概してパラメータを有する。これらはINPUT及びOUTPUTデータ仕様を含み、エンティティフィールド名（概してデータベースカラム名）を呼び出しプログラムのデータフィールド（ワーキングストーリッジまたはリンケージ）にリンクする。フィールドは、（カプセルレベルではないメソッドで）マンデトリ（強制的）あるいはオプションとして記載される。他のパラメータは返されたエンティティインスタンスの数、供給されたテーブル内の入力データフィールドの数、などに関する。メソッドは目的と使用に応じて名前がつけられ、特に、処理（GET、INS、MOD、DELなど）のタイプをカテゴリー別にするために接頭辞を用いる。

ODICT:オブジェクト辞書。これはエンティティ、カプセル、メソッドの詳細を記憶する一組のDB2テーブルである。エンティティ情報はそれが含むフィールド、推論されたフィールドを含む。ODICTは、アプリケーションプログラムの開発者のための情報源であるとともに、XCALLステートメントを拡張するため、かつ、必要なデータエリアを挿入するためにプレコンパイラによってアクセスされる。要求されたメソッドから読み出しに対する適当なカプセルを引き出すために、構成されたLDSテーブルがラン時にXCALLによって使用される。

パーティション：種々のプラットフォームにわたる物理的配分を可能にするデータベースの論理的サブセクション。XCALLに特定されないが、データベースパティショニングは、パティショニングバウンダリを横断するDB2ジョインタイプの検索の設計に影響を与える。

プレコンパイラ:XCALL及びXCステートメントを含むソースを獲得し、適当なCOBOL CALLステートメントに変換するソフトウェアであり、ODICTから生成されるデータ定義を含む。これはコンパイルプロセスのある段階で呼び起こされる。

スーパカプセル：ベースカプセル（”通常”カプセル）と対向する。名前からわかるように、これは”ハイレベル”カプセルである。例えばDB2タイプジョインにおいて、１つ以上のエンティティにアクセスすることによって獲得できるデータが要求されたときに、スーパカプセルを用いてもっとも効率的に処理される。概して、スーパカプセルは、カラムが異なるエンティティから取り出されるリストスクリーンに対して必要となる。例えば、スクリーンヘッダあるいはパーティションを横切るデータベースナビゲーションを構築するなど、再利用が課題になっているときもスーパカプセルが考慮される。

トランザクション：いつかのビジネス上の目的を達成するユーザによって開始されるか、あるいは、ユーザに反応するオンライン処理。

バージョン：エンティティ及びメソッドに関する。エンティティは新たなDBカラムを追加したり、古いカラムを削除したり、他のテーブルからデータを獲得することによる強化が必要である。同様にしてプロセス要件も変更される。XCALL環境はメソッド及びエンティティがバージョン番号をもつことを可能にする。特定のメソッドバージョンについては、

関連するもっとも有効なエンティティバージョン番号はODICTに保持される。トランザクション時にメソッドバージョン番号が特定され（デフォールトではプレコンパイル時に最新のバージョンを取り上げる）、エンティティの適当なバージョンがアクセスされる。
XCIB:XCALLはブロックに対するインタフェースとなる。これはXCALLの種々のコンポーネント（トランザクション、モジュール、XCALLそれ自身、結果的にデータベースにアクセスするカプセル）間の通信手段である。
まず、機能上の術語についての本実施形態の一般的な説明を行ない、続いて、本実施形態についてのより詳細な説明を行なう。
ODICTはカプセルとして定義される一組の呼び出されるプログラムによって管理されるデータの詳細を含むデータベース（IBM Db2）である。このカプセルはメソッドとして定義されるサービスの数を提供する。ODICTはさらに、これらのメソッドと、要求されるかまたは／及びこのメソッドから利用できるデータとを含む。
データは多数のエンティティからなり、各エンティティはカラムとして定義される１つ以上の属性を有する。エンティティは名前と番号（エンティティがODICTに対して定義されるときはいつでも割り当てられる）とを有する。カラムは名前（エンティティ内で独自）と、データタイプと、長さ及び番号（アイテムがODICTに追加されたときはいつでも割り当てられる）とを有する。
さらには、エンティティは１つ以上のDb2テーブル及び／または（IBMの商標）レコードを有していてもよいが、これは必須なものではない。
XCALLにおけるメッセージの配送
ラン時におけるXCALLアーキテクチャの中心となるのは、呼び出しプログラムから呼び出されるプログラムへ、あるいは、呼び出されるプログラムから呼び出しプログラムへ送られる所定のメッセージブロックである。このメッセージブロックは固定フォーマット制御ブロックにおける制御情報と、XCIB（XCALLインタフェースブロック）と、オプションとしてのスリーフォーマットのデータエリアを、IRB（中間要求／応答ブロック）からなる。
XCIBの一部は呼び出しプログラムから渡されるデータアイテムのリストと、呼び出されるプログラムから要求されているデータアイテムのリストを含む。これらのリストは’エンティティ番号’及び’カラム番号’として表現される。これらのリストは'Map1'と呼ばれ、入力に対して１つ、出力に対して１つ存在する。
XCIBの他の部分は'Map2'として定義され、'Map1'に列挙されたデータアイテムの位置と長さについての記述を含む。Map2に対して２つのフォーマットが存在し、１つは呼び出しプログラムとXCALL間で用いられ、他の１つはXCALLと呼び出されるプログラム間で用いられる。
XCIBは２つの部分を有している。’グローバル'XCIBは読み出しプログラム、XCALL及び呼び出されるプログラムによって使用される。’ローカル'XCIBは、読み出しがXCALLに渡されるとき、あるいは、応答が呼出者に返されるときに一組の内容を有し、読み出しがXCALLから呼び出されるプログラムに渡されるとき、あるいは、応答が呼び出されるプログラムからXCALLに返されるときに他の組の内容をもつ。
データが呼び出しプログラムから渡されるとき、当該プログラムのデータエリアからIRBに移動される。XCALLは、入力されたMap1における各エントリの特定数のローに対して、一度に１カラムずつこの機能を実行する。データアイテムの長さと位置とはMap2から獲得される。位置はカラムの開始に対するローの初めからのバイトの数によって表現される。ローカルXCIBは用いられた各エントリブロックの第１バイトのアドレスを含む。
XCALLが要求を受けたとき、要求されたメソッドが存在してユーザに利用可能であるか否かがチェックされる。いくつかの制御情報が、要求されたメソッドを含むプログラム（カプセル）の身元を含むXCIBに追加される。
XCALLは読み出しをカプセルに渡す。プレコンパイラによって生成されたコードは現在のメソッドインタフェースの詳細によってMap2を更新し、XCALLサービスルーチンを呼び出す。このルーチンはカプセルに対するデータエリアを生成し、IRBからのデータをこの領

域に移動してXCIBにおけるエンティティブロックにポインタを設定する。カプセルのデータエリアはEBB（エンティティブロックブロック）として知られ、１つ以上のエンティティブロックを含む。
エンティティブロックはカプセルによって使用され、呼出者に返されるべき応答を含む。XCALLがカプセルから制御を取り戻したとき、XCIBは呼出者に返すべきEBBにデータが存在するか否かを検出するために検査される。存在するときは、前のMap2プロセスは出力Map1を用いて反転されて、IRBを上書きする。
このデータを呼び出しプログラムに返すために、XCALLは呼出者によって最初に渡されたMap2を再記憶し、Map2と出力Map1とを用いて、IRBからのデータアイテムを呼出者のエンティティブロックに移動する。
図２のプレコンパイラ７はソースプログラム８におけるXCALLステートメントを獲得して、ODICT6の内容を用いてこれよりソースステートメントを生成する。
このようにして生成された呼び出しプログラムにおいて、コードはMap1及Map2を含むXCIBにおける制御情報を設定し、XCALLランタイムプログラムを呼ぶ。
呼び出されたプログラム（カプセル）において、プレコンパイラ７はXCALLから受信した読み出しを有効にするコードを生成し、Map2を設定し、呼出者から渡されたデータを獲得するためのXCALLサービスルーチンを呼び出して、特定のメソッドに好適するカプセルの一部に制御を渡す。
プレコンパイラは当該プログラムにおいて用いられる’エンティティ’のためのデータエリアに対する定義を生成する。呼び出しプログラムにおいて、この定義は、当該プログラムからの読み出しにおいて特に使用されるエンティティ及びカラムのみを含む。呼び出されるプログラムにおいて、定義はすべてのエンティティとカプセルがサービスを提供するためのカラムとを含む。いずれの場合においても、データは各エンティティのためのテーブルとして表現される。テーブルは１つ以上のカラムのローの１回以上の発生からなる。これらのテーブルは’エンティティブロック’として定義される。
読み出しは、呼び出されているプログラムあるいは見いだされるべきシステムではなく、要求されたサービス（すなわち、メソッド名）のみを特定する。
バージョンの独立性
エンティティとメソッドとはODICT内で定義される。要求された、及び／または利用可能な一組のメソッドが変更されるときにメソッドは変化する。同様にして、１つ以上のカラムはエンティティに追加、あるいはエンティティから削除される。いずれの場合においても、このような一群の変更は新たなバージョンを構成する。
エンティティ番号及びカラム番号として定義された、データタイプとカラムの長さとはどのような場合であっても変更されない。カラムが長さあるいはフォーマットを変更するときは、新たなカラム番号が割り当てられる。もとのカラム名が新たなカラムに対して再利用され、古いフォーマットには新たな名前が割り当てられる。
上記の特徴は、呼び出しプログラムにおける最適化されたエンティティブロックとともに、呼び出しプログラムと呼び出されるプログラムとの結合を最小に抑えることができる。メソッドインタフェースの最新の’公衆’バージョンは、呼び出しプログラムがコンパイルされるときに使用される。カプセルがコンパイルされるとき、当該メソッドを定義するXCALLステートメントと有効なエンティティバージョンの組合せがバリデーションコードを生成するために用いられる。カプセルが支持されていないエンティティあるいはメソッドバージョンに対する要求を受信するとき、この生成されたコードによって拒絶される。
古い呼び出しプログラムが再コンパイルされるとき、使用する最新の公衆バージョンのエンティティを取り出す。すなわち、以前に７つの文字であった'POSTCODE'と呼ばれるカラムが現在８であるときは、当該プログラムは、次にコンパイルされるときに最新のデータフォーマットを使用するために、変更する必要がない。もし可能なら、カプセルは古いカラム番号に対する要求を認識して一部除去された７つの文字郵便コードを渡す。このようにして、呼び出し及び呼び出されるプログラムの両方を同時に変更する必要性をなくすことができる。

配分／スケーラビリティ
呼び出しプログラムと呼び出されるプログラムとの間の中間手段としてIRBを用いている理由の１つは、このことがシステム間に通信リンクを挿入する構造の都合の良い部分であるからである。
カプセルがそれを呼び出しているプログラムから物理的に離れている場合は、XCALLはグローバルXCIBとIRBとをリモートシステムにおけるXCALLの他のインスタンスに渡す。リモートXCALLはMap2などを使用して、上記と同様の方法でカプセルにデータを渡したり、カプセルからの応答を受信する。グローバルXCIBとIRBは’ホーム'XCALLに戻って、ローカルの場合とまったく同様の方法で応答を呼出者に渡す。
これは、以前に同じ物理的システムに存在した呼び出し及び呼び出されるプログラムはどのようなときでも変更することなしに分離できることを意味している。
以下に本実施形態の一面をより詳細に説明する。上記したように、本実施形態は図２に示すように、メモリ手段と、ODICT6と、メッセージインタフェースと、XCALL3と、プレコンパイラ７とを具備している。
使用中のODICT
XCALLを支持するために、ODICTは２つの段階で使用される。コンパイル時、プレコンパイラは埋め込まれたXCALLステートメントを有するCOBOLソースを獲得して、適当なデータフィールドを含むCOBOL　CALLに拡張する。メインデータエリアは、XCALL自身の使用のための通信エリアであるXCALLインタフェースブロック（XCIB）と、アプリケーションによってアクセスされる各エンティティにデータを渡すかあるいは、各エンティティからデータを受信するエンティティブロックである。
２つのタイプのエンティティブロックが存在する。１つは全エンティティブロックであり、カプセルに対して呼び出されてモジュールコールのためのXC　COPYステートメントを使用するときに含まれる。他の１つは最適化されたエンティティブロックであり、トランザクション内あるいはXCALLメソッドステートメントを発行する他のコード内で生成される。後者の場合は、入力及び出力節で特定されたフィールドのみが含まれ、これによってラン時に必要とされるデータエリアのサイズを減らすことができる。同じエンティティにいくつかのメソッドが作用しているとき、あるいは、異なる入力及び出力要件を有する同一のメソッドに対して複数の読み出しが存在するときは、プレコンパイラは参照されるすべてのフィールドを具備する複合エンティティブロックを生成する。
２つのエンティティブロック間の混乱を避けるために、前者は概してカプセルエンティティブロックあるいはフルエンティティブロック、後者は概してエンティティブロックあるいはメソッドコールエンティティブロックと呼ばれる。
カプセル、メソッド、エンティティについての情報がリニアデータベース内に抽出されるとき、第２のXCALL　ODICTアクセスが毎日の初めに行われる。これらは、メソッドコールとカプセル間に最終的な取り決めを可能にするためにXCALLのラン時に用いられる。ODICTを直接用いることも可能であるが、パフォーマンス上のオーバヘッドが発生する。ODICTをラン時から分離することもODICT更新実行時の密接な制御を可能にする。
ODICT　XCALL支持構造
論理的オブジェクト
ODICTはXCALLによって用いられる論理的オブジェクトを記述する。メインオブジェクトはカプセル、メソッド、エンティティであり、図３を用いて以下に説明する。
エンティティ９
これはデータベースに保持される識別可能なタイプである。これは、例えばプロダクトを表すDB2テーブルあるいは、（IBMの商標）レコードタイプ、あるいは複合構造であり、例えば関連するポストコード、ポストタウン、スルースウェア、ローカリティデテールなどを有するアドレスレコードタイプからなり、このようなエンティティは多数のDB2テーブルからのデータを具備する。各エンティティは多数のフィールド（カラム）からなり、概してDB2カラムに対応する。フィールドは推論される記述、すなわち、物理的データとして保持されているのでなく計算によって得られる。

ODICTは各エンティティを記述し、エンティティ番号によって識別する。各エンティティは、アプリケーション内のグループフィールド名を生成するのに用いられる18文字までの名前を有する。独自でない名前でコラムの前に文字を付加するための省略文字（これはまた、このエンティティを処理するメソッド名を構成するのにも用いられる）と、名前の衝突を解決するための２つの接頭辞とをさらに有する。

メソッド10

エンティティに関して機能を実行するためにメソッドがCICSまたはバッチプログラムによって呼び出される。DB2の術語でいうと、これはテーブルに関して新たなローを挿入するか、あるいは、現在のローを検索、更新、または削除することを意味する（ここでメソッドは（IBMの商標）データベースまたはデータ記憶システムに作用するためにも用いられる）。

呼び出しプログラムはパラメータをメソッドコールに供給することでメソッドと通信する。これらはインプット及びアウトプットデータ仕様を含み、エンティティフィールド名を、（ワーキングストーリッジまたはリンケージにおける）呼び出しプログラムのデータフィールドにリンクさせる。

他のプログラムは返されたエンティティインスタンスの数と、供給されたテーブルにおける入力データフィールドの数などに関連する。

メソッドはメソッド名、メソッドコールにおいて使用される30の文字フィールドによって独自に識別される。メソッドは目的と使用、特に、処理（GET、INS、MOD、DELなど）のタイプをカテゴリー化するために、（アクションタイプとして知られている）接頭辞の使用を考慮して名前が付けられる。

カプセル11

カプセルは概して所定のエンティティにアクセスするメソッドをグループ化する。スーパカプセルと呼ばれる特定のタイプのカプセルは、関連する組のエンティティにアクセスするメソッドをグループ化し、その後、XCALLは当該メソッド名を適当なカプセルに関連付けるためにODICTデータを使用する。

各カプセルはカプセルネームによって識別する１つのエントリを有し、５つの文字のうちの最初の２つは関連するアプリケーションシステムを示す。保持された他の詳細は使用可能性インジケータ及び所有者ユーザIDである。

テーブル12

ODICTは各テーブルの各カラムに対するエントリを有するDB2テーブルの構造を記憶する。各エンティティはゼロ、１つ以上のDB2テーブルによって支持され、各テーブルは１つ以上のエンティティを支持する。

カラム13

ODICTはDB2であろうと推論されたものであろうと、各エンティティの各カラムに対して１つのエントリを有する。これは削除されたカラムを含む（例えばフォーマットが変更されたときカラム名を再利用するための機構は存在するが、カラム番号は再利用されない）。カラムはエンティティ番号内のカラム番号によって識別され、タイプ及びサイズの詳細は各カラムに対して保持される。

エントリバージョン

エンティティは新たなDBカラムを追加、古いカラムを削除、他のテーブルからデータを獲得するか、あるいはカラムを改善することによって、強化する必要がある。各エンティティについて、ODICTは各新たなバージョンに対するエントリを保持する。概して、最新のバージョンが使用されるが、使用可能性インジケータは新たなバージョンを入力させ、プロダクションアプリケーションを混乱させることなしにテスト環境で用いられる。

メソッドバージョン

同様にして処理要件は変化する。特に、メソッドに対するインタフェースが変化、例えば、入力データフィールドがマンデトリ（強制的）であるときあるいはフォーマットを変更したときは、新たなメソッドバージョンが必要になる。所定のメソッドバージョンについて、それに関連する最も有効なエンティティバージョン番号はODICTに保持される。トラ

ンザクションにおいて、メソッソバージョン番号が特定され（デフォールトではプレコンパイル時で最新のバージョンを取り出す）、エンティティの適当なバージョンがアクセスされる。

DB2テーブルの詳細

ODICTによって記憶されるDB2テーブルは図４を参照して記述される。

カプセル（OCAP）14

カプセルテーブル：カプセルごとに１つのローを含む。

カラム名	タイプ	記述
CAPSULE_NAME	Char(5)	プログラムＩＤに対してチェックされる
AVAIL_FOR_USE	Char(1)	'N','W','Y'
CAP_SECRTY_TYPE	Char(1)	
OWNER_USERID	Char(8)	テストハーネスにおけるルーティング コールのときに用いられる
UPDATED	timestamp	
UPDATED_USERID	Char(8)	
CAPSULE_DBMS	Char(8)	
CAPSULE_ROUTER	Char(8)	

エンティティ（OENT）15

各エンティティの各バージョンに対して１つのロー。１つのエンティティは１つ以上のDB2テーブルを参照することがある。Entity_Abbrev,Entity_version（独自）、及びEntity_Name,Entity_Version（独自）が以下に示される。

カラム名	タイプ	記述
ENTITY_NUMBER	Smallint	不変
ENTITY_VERSION	Smallint	
ENTITY_NAME	Char(18)	意味のある独自の名前、アプリケーション内のエンティティブロックにエンティティ名を生成する
ENTITY_ABBREV	Char(4)	異なるエンティティに起こる同一名のカラムカラムを区別したり、メソッド名の構築に用いられる。これはＯＤＩＣＴ内で独自である。
DB_PARTITION	Char(2)	有効なＤＢ分割コード
CAPSULE_PREFIX	Char(1)	内部メソッドコールに用いられる接頭辞、現在’Ｉ’
CAP2CAP_PREFIX	Char(1)	カプセルからカプセルへの呼び出に用いられる接頭辞
AVAIL_FOR_USE	Char(1)	’Ｎ’（初期時）、’Ｙ’
UPDATED	timestamp	
UPDATED_USERID	Char(8)	

メソッド（OMTH）16
ODICT同期により多くなるが、概してメソッドバージョンごとに１つのローが出力される。各メソッドは１つのカプセルに属する。一度メソッドが利用可能になれば、新たなバージョンがメソッドのどのような変更に対しても生成される。
Capsule_name,Method_Name,カプセルテーブル（OCAP）との関連のためのMethod_Versionが以下に示される。

カラム名	タイプ	記述
METHOD_NAME	Char(30)	メソッド名をつけるための標準に従って構成される。最初の部分（アクションタイプ）は'GET','GETM','MOD'などである。次の部分はエンティティ省略文字である。より詳細についてはXCALL Programmer's Guide Appendix Bを参照せよ。
METHOD_VERSION	Smallint	
METHOD_TYPE	Char(2)	アクションタイプに応じて'RR','RU','UU''RL'

METHOD_TYPE_STATS	Char(1)	メソッドタイプに応じて'M','I','D','R'
CAPSULE_NAME	Char(5)	OMAP外部キー
DEF_TUPLES_BACK	Smallint	戻されたアイテムのデフォールト番号 これはアクションタイプに依存する。すなわち、GETタイプのメソッドに対しては１、GETMまたはGETNに対しては＞１。
MAX_TUPLES_BACK	Smallint	戻されたアイテムの最大の番号 これはDef_Tuples_Back以下にはならない。
AVAIL_FOR_USE	Char(1)	'N'（初期時）、'Y' 'Y'はメソッドの削除を禁止し、新たなバージョンの入力を許可する。'N'は推論されたフィールドの入力を可能にする。
METHOD_DESCRIPTION	V240	叙述的概要
UPDATED	Timestamp	
UPDATED_USERID	Char(8)	

エンティティカラム（OCOL）17
DB2であり推論されたものであれ現在あるいは前のエンティティバージョンで使用された各カラムのための１つのロー。各カラムの属性を定義する。

カラム名	タイプ	記述
COLUMN_NUMBER	Smallint	エンティティ内のカラム番号。これは、下部のテーブル構造が変化しても変化しない。削除されたカラムは後で再利用される。
COLUMN_TYPE	Char(8)	CHAR,DATE,DECIMAL,INTEGER,SMALLINT,TIME,TIMESTMP,VARCHAR
COLUMN_LENGTH	Smallint	エンティティブロックのオフセットサイズを計算するのに使用される。
COLUMN_SCALE	Smallint	デシマル位置の番号
COLUMN_NULLS	Char(1)	'N','Y'
DELETED	Timestamp	許可されたナル値。
DELETED_USERID	Char(8)	許可されたナル値。

DB2テーブル（OXTB）18
DB2テーブルによるエンティティ実行を示す（エンティティは１つ以上のDB2テーブルによって実行される）。カプセルテーブル（OCAP）との関係のためのベースカプセルを以下に示す。所定のエンティティ、例えば（IBM商標）データに対するテーブルは存在しない。

カラム名	タイプ	記述
TABLE_NAME	Char(18)	
ENTITY_NUMBER	Smallint	OENT外部キー
ENTITY_VERSION	Smallint	OENT外部キー
BASE_CAPSULE	Char(5)	OCAP外部キー。ナル値が許される。常に現在のカプセルを参照する。
TABLE_PREFIX	Char(4)	
CREATOR_ID	Char(8)	システム識別子。例えば、COSMOSS, TRAINING,OMS

（ODRV）19

各推論されたフィールド（物理的にDB2テーブル上には存在しないが、計算によって求められるか他のDBMS例えば、TRADEMARK　OFIBMに保持されているフィールド）に対する１つのロー。この情報はOCOL上のインジケータカラムとして保持されるが、あとでより多くの情報を保持することが必要である。各ローはエンティティカラム（OCOL）上に均等なローを有する。

カラム名	タイプ	記述
ENTITY_NUMBER	Smallint	OCOL外部キー
COLUMN_NUMBER	Smallint	OCOL外部キー

インタフェース（OINT）20

各メソッドバージョン、すなわち、当該バージョンによって使用されたカラムに対するインタフェースブロックを定義するとともに、入力に対して強制なのかあるいは出力に対して利用可能なのかを定義する。これはエンティティカラムとメソッドバージョン間の多対多の関係である。ENTITY_NUMBER,エンティティカラムテーブル（OCOL）との関係のためのCOLUMN_NUMBERが以下に示される。METHOD_NAME,METHOD_VERSIONはメソッドテーブル（OMTH）との関係の'Many'終端（end）に対して使用される。

カラム名	タイプ	記述
METHOD_NAME	Char(30)	OMTH外部キー
METHOD_VERSION	Smallint	OMTH外部キー
TYPE_IFACE	Char(1)	フィールドがこのメソッドにおいて強制的かどうか。'A'-出力(A)vailable,'M'-入力(M)andatory, '0'-入力(O)ptional
ENTITY_NUMBER	Smallint	OCOL外部キー
COLUMN_NUMBER	Smallint	OCOL外部キー

DB2テーブルカラム（OSHD）21
DB2テーブルの現在のバージョンにおけるリストカラム。

カラム名	タイプ	記述
TABLE_NAME	Char(18)	DB2テーブル名
COLUMN_NAME	Char(16)	（ＯＸＴＢテーブルとの関係を介して供給された）エンティティ内のカラム名。
DB2_COLUMN_NO	Smallint	

エンティティマッピングに対するエンティティカラム（OMAP）
各エンティティの各バージョンによって支持された名前によってカラムを列挙して、エンティティとエンティティカラム間の多対多リンクを実行する。もはや使用されないが古いバージョンコールによって要求されるカラムを含むエンティティバージョンとともに使用すべく定義されたすべてのカラムに対する完全な組のローを保持する。ENTITY_NUMBER及びCOLUMN_NUMBERはカラムの詳細を保持しているOCOLローを指し示す。LAST_ENT_VER_USEDはこのカラムを用いて最後のエントリバージョンのNETITY_VERSIONを保持する。
このテーブルは、所定のエンティティバージョンに対するエンティティブロックを生成するためにXCALLプレコンパイラによって使用される。アプリケーションにおいて用いるためのフィールド名を構成するのに使用されるカラムのための名前をも提供する。

ENTITY_NUMBER,エンティティカラムテーブル（OCOL）との関係のためのCOLUMN_NUMBER,ENTITY_NUMBER,エンティティテーブル（OENT）との関係の'Many'終端に対して使用されるENTITY_VERSIONは以下に示される。

カラム名	タイプ	記述
ENTITY_NUMBER	Smallint	OCOL外部キー、OENT外部キー
ENTITY_VERSION	Smallint	OENT外部キー
COLUMN_NAME	Char(20)	
COLUMN_NUMBER	Smallint	OCOL外部キー。これは（ＤＢ２カラム番号とは対照的に）エンティティカラム番号である。これはＯＣＯＬに特定されている。

カプセルエンティティ（OXCE）
相互参照カプセル及びエンティティ。ENTITY_NUMBER,ENTITY_VERSIONは以下に示される。

カラム名	タイプ	記述
CAPSULE_NAME	Char(5)	OCAP外部キー
ENTITY_NUMBER	Smallint	OENT外部キー
ENTITY_VERSION	Smallint	OENT外部キー

エンティティメソッド（OXME）
相互参照メソッドバージョン及びエンティティバージョン
NBプライマリインデックスはENTITY_NUMBER,ENTITY_VERSION,METHOD_NAME,METHOD_VERSIONに関している。
METHOD_NAME,METHOD_VERSION,ENTITY_NUMBER,ENTITY_VERSIONは以下に示される。
メソッドは、エンティティが１つ以上のメソッドによって処理されるのと同様に、１つ以上のエンティティを処理可能である。

カラム名	タイプ	記述
METHOD_NAME	Char(30)	OMTH外部キー
METHOD_VERSION	Smallint	OMTH外部キー
ENTITY_NUMBER	Smallint	OENT外部キー
ENTITY_VERSION	Smallint	OENT外部キー

XCALLランタイムシステム
XCALLランタイムシステムの動作は以下のように要約される。
・トランザクションがメソッドコールを生成する。
・XCALLはトランザクションエンティティブロックからIRBデータを’収集’する。
・XCALLは呼び出すためのカプセルを見いだすためにリニアデータベースにアクセスする。
・カプセルに’提供された'IRBデータ
・カプセルは呼び出しを処理する。
・XCALLはカプセルのエンティティブロックからIRBデータを’収集’する。
・XCALLはデータをトランザクションブロックに返す。
以下にXCALLランタイムシステムについて詳細に説明する。
バックエンド　選択されたカプセルに対する呼び出しを担うXCALLランタイムの最低レベル層は、次の機能を有する。

カプセルに対する要求	ＸＣＡＬＬに対する要求
ＸＣＩＢと要求メッセージ	第１の呼び出しでセッションを
をカプセルへ渡す。	確立する。
リターン時にＸＣＩＢと	ＸＣＩＢと要求メッセージを送って
ＩＲＢ又は応答メッセージを	待機する。
制御及び連続へ返す	
	リターン時、ＸＣＩＢと
	応答メッセージ又はＩＲＢ
	を受信する。

カプセルコール　カプセルからカプセルへの読み出しとして知られる’レベル'2の呼び出し。カプセルが異なるメソッドを呼び出すときは、別個のXCIBにおける情報を設定する必要がある。名前の衝突を避けるためにこれはXC2Bと呼ばれ、通常のXCIBにおいてはXC接頭辞ではなくX2接頭辞のほうがファイル名を正確に識別する。さらに、より多くの情報を記

憶してさらなる呼び出しを監視することが必要である。
制御及び連続（C/C） XCALLランタイムの第二レベル層は以下の機能を有している。
呼び出されたカプセルを監視する。
現在の呼び出しのステータスを記録する。
DBMSスレッド／ランユニットを’所有する’
リモートのカプセルに対する適当なバックエンドを識別する。
要求をバックエンドに渡す。
明示的コミットリクエストを管理する－必要に応じてコミットをリモートのXC ALLに送る。
フロントエンド XCALLランタイムの第一レベル層は次の機能を有する。

Ａｐｐｌからの要求	ＸＣＡＬＬからの要求
メソッドを支持するカプセルを見いだす。	ＸＣＡＬＬ及び要求メッセージを受信する。
制御をＣ／Ｃに渡す。	これがＣ／Ｃに対するパスの正しい場所であることをチェックする。
リターン時、呼出者にデータを提供する。	リターン時、ＸＣＩＢ及び応答メッセージを呼び出しＸＣＡＬＬに送る。

エンティティブロックブロック これは、ラン時に割り当てられたカプセルエンティティブロックに対する集合名詞である。当該カプセル内のメソッドコールから戻される大量のデータを収納するために記憶領域が割り当てられ、そのサイズは、要求されるロー数によって乗算された当該カプセルによって管理されるすべてのカプセルエンティティブロックの全サイズである。
IRB 中間要求／応答ブロック。これはカプセルに渡されるかあるいはカプセルからのデータエリアである。スペースを最小にするために連結された入力または出力フィールドからなり、エンティティ構造についての記述はない。
ローカルコール ローカルノードのカプセルに対する’伝統的’コール:XCALLはバックエンド処理から直接カプセルを呼び出せる。
Map1、Map2及びMap3 入力及び出力データのIRBに対するマッピングについての詳細を保持するデータエリア。
リモートコール リモートカプセルに対する呼び出し。これはより複雑な処理と制御を必要とする（リリース3.0から）。
トランザクションコール コントロールとしてのXCIBフィールドを使用する標準の呼び出し。プレコンパイラは呼び出しを適当なエントリポイントに挿入する。さらに、IRBサイズを計算する。（要求されたロー＊出力ロー長さ）かあるいは、（供給された入力ロー＊入力ロー長さ）のうち大きい方。結果として得られた値はXC－IRB－SIZEに入力される。Map1及びMap2Aデータは呼び出しが発生するまでにあらかじめ設定される。
XCCB 制御及び連続ブロック
XCCT XCALLカプセルテーブル。これはLDSテーブルからロードされ、カプセルステータス、所有者名、位置を含む各カプセルについての詳細を保持する。
XCIB XCALLインタフェースブロック。これは種々のXCALLランタイムコンポーネントと呼び出しプロセスとの間の通信手段である。呼び出すべきカプセルとメソッドについての情

報、監視情報、エンティティブロックのアドレス、IRBからエンティティブロックへのマッピングについての詳細などの情報を含む。
XCMT　XCALLメソッドテーブル。これはLDSテーブルからロードされ、ラン時、XCALLはこれをメソッド名をキーとして取り出し、それを含むカプセルを決定する。さらに、XCMTはメソッドに対してステータス及びローが要求された／戻された詳細を維持する。
XCNT　XCALLノードテーブル。これはLDSテーブルからロードされ、各ノードについての詳細を保持する。
XCALLにおいて用いられる制御ブロック構造が図５に示されている。
XCALL構造
エントリポイント
呼び出しが発生する環境によって種々のエントリポイントがある。これらは、トランザクションコール／カプセルコール（カプセルからカプセル）、CICSフォアグラウンド/CICSバックグラウンド／バッチ、ローカル／リモートの順列からなる。
３つの層、フロントエンド。連続＆制御、バックエンド
各場合において、エントリポイントは、要求されたカプセルが実際に呼び出される前に、フロントエンド、制御及び連続、バックエンドの３層の処理によって支持される。
呼出者処理
プレコンパイラによって生成されたコードは、呼出者エンティティブロックデータをIRBにマッピングするためのMap1、Map2Aデータをロードして、IRBのサイズを計算し、これを記憶するとともにXCIB内の情報を監視する。エントリポイントを呼び出す前に、入力データは呼出者エンティティブロックフィールドに転送される。
XCALLに対する各呼び出しの後に、アプリケーションはリターンコードをチェックして適切に動作しなければならない。制御が戻された後、監視情報は更新され、要求されたデータは指定の出力フィールドに入力される。
エントリポイント
エントリポイントの主な機能は、初めの呼び出しに対するパラメータとして渡されたエンティティブロックのアドレスを獲得することである。渡された意味のある（significant）パラメータの数に基づいて、XCIB（XC－EM－ADDR）におけるエントリはこれらのアドレスに設定される。さらに、環境の発生などについてもチェックする。
フロントエンド
これはアプリケーションプログラム（及びリモートアプリケーションについてはリモートシステム）からの要求を受信する。適当なセキュリティ情報を獲得して、受信した呼び出しの形跡を維持する。さらに、IRBを呼出者エンティティブロックにマッピングする処理を行なう。XCALLが読み出されたとき、フロントエンドは、CICS　GETMAINあるいは、バッチと同等のアセンブラーを用いて、IRBのためのGETMANを実行して、オペレーティングシステムからの記憶エリアを獲得する。獲得されたスペースのサイズはXC－CURRENT－IRBSIZEに保持される。
次の呼び出しにおいて、呼出者におけるプレコンパイラ挿入コードによってXC－IRB－SIZEが再び設定される。フロントエンドはこれをXC－CURRENT－IRB－SIZEと比較する。現在のサイズが要求されたサイズに等しいかあるいは大きい場合はこれ以上の処理は行われず、FREEMAINについての現在のスペースが解放されて、新たなIRBがGETMAINによって獲得される。
以下の点を考慮して種々の管理上及びセキュリティ上の処理が実行される。
・XCIB追跡ポインタをインクリメントして現在の要求情報を新たな追跡エントリに移動する。
・XCIB呼出者のノードとタスク番号フィールドを設定する。
・XCMTを用いてメソッド名が有効が否かをチェックし、（テストハーネスの使用を考慮して）カプセル名を獲得し、（ローカルのものであれば）そのカプセルが要求のタイプに対して利用できるかどうかをチェックする（できない場合は、適当なSWEコードを設定して呼出者に戻る）。

・XCCTからのノードを獲得してノードidをXCIBに設定する。
・セキュリティ情報（ミドルウェアからのユーザid）を獲得してセキュリティチェックフィールドを生成するためにチェックサムを行なう。
・タスクに対する第一の呼び出しについて主要なXCCBを生成する。
・XCCB詳細を記憶する。ランタイムテーブルなどのアドレス及び必要に応じてカプセルからカプセルへの呼出情報など。
フロントエンドは呼出者からの要求データを記憶してIRBに入力する。これを行なうためにアッセンブラルーチンXC700UASを呼び出す（詳細は以下のMap1、Map2処理の部分を参照せよ）。XCIBの詳細はXCCBに記憶される。
連続及び制御を呼びだした後、フロントエンドはカプセルからカプセルへの呼びだし処理を行ない、詳細を適当なXCIBフィールドに複写する。
次に、アッセンブラルーチンXC703UAS（実質的にはリバース時のXC700UAS）を呼び出して、戻された更新IRBからのデータを呼出者エンティティブロックに転送する。コミット処理が必要な場合はこのときに発生する。一方、制御がエントリポイントに渡され、さらには呼出者に渡される。
制御及び連続
これはアプリケーションが呼び出すカプセルの詳細を維持する。要求を適当なカプセルに渡すためにXCCTを使用する。呼出者によって明示的に要求されたバックアウト及びコミットあるいは、エラー処理要件によって発生したバックアウトを処理する作業を担う。
バックアウト／コミットコールは、要求されたアクションが起こる前に、これまでにこのトランザクションによって呼び出された各カプセルに対して、DBMSタイプのチェックを行なう。
当該呼び出しがバックアウトまたはコミットでないときは、Ｃ＆Ｃはメソッドコールを想定することを行なう。
呼び出されたカプセルがXCCBにないとき、カプセルが駐在しているシステムとリモートシステムとの会話のために用いられるバックエンド（通信アクセスメソッドまたはローカル）のタイプを探索するためにXCCTが使用される。
更新アクセスがチェックされて、呼び出し統計が更新される。呼び出しの上限を越えていなければ、Ｃ＆Ｃは制御をバックエンドに渡す。
バックエンドからのリターン時、Ｃ＆ＣはXCIBが破損したかどうかをチェックして、このカプセルに対するXCCBを更新する。リターンSWEコードが８より大きいときはバックアウトが発生する。
最後に制御がフロントエンドに戻される。
バックエンド
ステーブルバージョン、ワーキングコピー、スタブ（最後の２つはテスト時に用いられる）など適当なカプセルを呼び出す。XCIBのローカル部をXCCBのセーブエリアに記憶し、制御をカプセルに渡す。
リターン時、なにも破損しなかったことと、ローのリターン値が正常かどうかをチェックする。ローカルXCIBを再記憶する前に、アッセンブラルーチンXC702UAS（以下のMap1、Map処理を参照）を用いて、呼出者に返すべきデータによってIRBを生成する。
制御はＣ＆Ｃに戻る。
カプセル処理
プレコンパイラはカプセルの詳細をXCIBに記憶するためのコードを生成し、メソッドが有効であり、要求されたメッセージデータが渡されたか否かをチェックする。
次に、アッセンブラルーチンXC701UASを用いて提供されたIRBデータを解読する。これが最初であるときは、このカプセルが呼び出され、その後、XC704TCSがエンティティブロックブロックのための記憶エリアを獲得するために使用される。これは、必要なだけ発生した（XC－ROWS－REQUESTED）すべてのカプセルエンティティブロックからなる。
カプセルエンティティブロックアドレスをXCIBに設定した後、カプセルは要求されたメソッドを実行して、XCIBにおけるXC－SWE－CODE及びXC－ROWS－RETURNEDフィールドを更新

する。その後、制御はバックエンドに戻る。
MAP1及びMAP2処理
Map1及びMap2によって管理されるデータエリア
上記したようにデータは、（他のすべての送信もとあるいは送信先データフィールドをのぞく）３つのエリアを用いてXCALLの各部に渡される。
・この呼出者によって参照されたフィールドからなる最適化ブロックである呼出者エンティティブロック。これは呼出者ベースで構成される：エンティティブロックは、異なるフィールドを参照するいくつかの異なる呼び出しの結果発生したフィールドを含む。さらに、呼び出しが配列が必要であることを示したときは、OCCURS節はプレコンパイラによって見いだされたエンティティに対して最大となる。このようなときは、エンティティブロックは不要となる。
・データベースであれ推論によるものであれ、エンティティのこのバージョンに対して削除されないものとして定義された、すべてのフィールドからなる全エンティティブロックであるカプセルエンティティブロック。
・各個々の呼び出しに対してデータがパックあるいはアンパックされる中間要求／応答ブロック（IRB）。
要求された種々の移動の制御は送信もと及び送信先フィールドを識別及び探索するマップテーブルによる。このようなマップテーブルは３つある。
・MAP1　各部がそれぞれ入力及び出力データを識別するエントリを含む：望ましくは２つのリストがあり、１つは入力、他の１つは出力用である。プレコンパイラは、特定のXCALLコールにおけるINPUTあるいはOUTPUT節に記載された各フィールドに対するエンティティ及びフィールド番号を生成する。入力及び出力仕様の間には暗黙の関係はない：例えばただ１つの意味する入力エントリと多くの出力エントリが存在する場合がある。ゼロに設定されたエントリは各リストの終わりを示す。所定の呼び出し、すなわち、入力及び出力節において参照された各カラムに対して１つのMap1エントリがある。各リストはゼロによって終わりとなる。各発生における入力及び出力要素の間には関係はない：例えば、ただ１つの意味のある入力エントリといくつかの出力エントリとが存在する場合がある。この場合（及びMap2フォーマットＡ）における値はプレコンパイラ生成コードによってロードされる。
・MAP2バージョンＡ　IRBデータを呼出者エンティティブロックにリンクする。このマップ及びMap1のエントリ間には１対１の関係がある。Map1で示したように、各エントリは入力及び出力部分を有する。各リストの終わりを示す特別の表示は不要である：これはMap1から容易に推論可能である。詳細はプレコンパイラによってロードされ、XCALLは制御をカプセルに渡すまえに値をセーブして、IRBが戻されたデータによって再生成されるときに再記憶する。
MAP2バージョンＢ　同一の物理的データエリアを共有し、値は、カプセル内で実行されたプレコンパイラ生成コードによってロードされる。Map2Aは呼出者エンティティブロックに関しているのに対し、Map2BはIRBデータをカプセルエンティティブロックにリンクする。削除と記されていないエンティティバージョンにおける各フィールドに対して１つのエントリがある。Map1はまだ必要である:IRB及びカプセルエンティティブロック間のデータ移動は、Map2Bと同等なエントリに一致するMap1の意味のあるエントリ／フィールド番号の組合せの存在に依存する。
MAP1/MAP2処理の一例
この例は、入力及び出力の操作は、２つの入力キーを供給する代わりに、各々についてローを受け取る単一のメソッドコールに対して示される。単純であるが、各段階でのデータを詳細に検討することにより、IRB/エンティティブロック作用の詳細を見いだすことができる。
エンティティPARROTは以下のテーブルで示される属性を有し、辞書（ODICT）に記載されている。特定のメソッドコールは利用可能なカラムのサブセットのみを要求する：他のメソッドコールは例として含まれている。

エンティティの詳細（ODICTより）
エンティティ番号　500　バージョン１
エンティティ名　PARROT
エンティティの省略文字　PARR
カラムの詳細

カラム番号	名前	タイプ	長さ
1	コード	Char	4
2	名前	Char	２０
3	サイズ	Smallint	2
4	カラー	Char	8
5	フィード	Char	１０
6	重み	Smallint	2

ＮＢタイプ　Char = Character,Smallint = small integer (binary in the range 1-32767)
XCALLステートメント
XCALL　GETN－PARR
INPUT（PARR－CODE FROM WS－PARR－CODE（＊））
OUTPUT（PARR－CODE　INTO　WS－PARR－CODE（＊）；
PARR－NAME　INTO　WS－PARR－NAME（＊）；
PARR－WEIGHT　INTO　WS－PARR－WEIGHT（＊））
ITEM（２、２）
END－XCALL
データ（エンティティブロック）のカプセルビュー
エンティティ（カラム）の属性のすべてについての多数の生起（ロー）からなる。

カラム名	カラム タイプ	カラム 長さ
I-PARR-CODE	Char	4
I-PARR-NAME	Char	２０
I-PARR-SIZE	Smallint	2
I-PARR-COLOUR	Char	8
I-PARR-FEED	Char	8
I-WEIGHT	Smallint	2

（ローの全長さ：４４バイト）

データ（エンティティブロック）の呼出者ビュー
エンティティ（カラム）の属性のすべてについての多数の生起（ロー）からなる。

カラム名	カラム タイプ	カラム 長さ
I-PARR-CODE	Char	4
I-PARR-NAME	Char	２０
I-PARR-WEIGHT	Smallint	2

（ローの全長さ：２６バイト）

処理の詳細
XCALLコールが実行される前に更新されるテーブル
Ｍａｐｌ：

入力 ｴﾝﾃｨﾃｨ No	入力 ﾌｨｰﾙﾄﾞ No	出力 ｴﾝﾃｨﾃｨ No	出力 ﾌｨｰﾙﾄﾞ No
500	1	500	1
0	0	500	2
		500	6
		0	0

Ｍａｐ２Ａ：

入力ベース	入力 ﾌｨｰﾙﾄﾞ オフセット	入力 ﾌｨｰﾙﾄﾞ 長さ	出力 ベース	出力 ﾌｨｰﾙﾄﾞ オフセット	出力 ﾌｨｰﾙﾄﾞ 長さ
1	0	4	1	0	4
			1	4	20
			1	24	2

呼出者エンティティブロック（プログラムによって設定されたｗｓフィールドを仮定）

PARR-CODE	PARR-NAME	PARR-WEIGHT
MC01		
CK01		

処理

呼び出されたフロントエンド：メソッドEBからIRBにデータを移動すべく呼び出されたXC700UAS:Map1（ゼロで満たされたフィールドを見いだしたときは処理を停止）上の各入力フィールドをIRBに配列する。入力配列に対して、配列操作はローごとに行なう。算出で用いられたフィールドはオフセット、Map2Aに保持されている、呼出者エンティティブロック内の長さ、（最初にパラメータとしてXCALLに対する最初の呼び出しに渡された）エンティティブロックのアドレス、XCIBに保持されているエンティティブロックローの長さである。オフセットはローの初めから始まり、次のローのためにロー長さによってインクリメントされる。

ＩＲＢの中身

MC01	CK01			

XC701UASにより、Map1とMap1の詳細を述べているMap2Bを用いてカプセルエンティティブロックを更新するために使用されるIRBデータは入力データのエンティティとフィールド番号を含む。Map2Bデータから、各入力フィールドの長さを決定し、エンティティ及びフィールド番号をMap1エントリのものと一致させることが可能である。同じMap2Bエントリから、全エンティティブロック内のオフセットが利用可能である。全エンティティブロックのアドレスは主記憶部の獲得に続いて設定される。

PARR-CODE	PARR-NAME	PARR-SIZE	PARR-COLOUR	PARR-FEED	PARR-WEIGHT
MC01					
CK01					

カプセルは処理を行いエンティティブロック（すべてのフィールドが利用可能であるとする：このメソッドコールはサブセットのみを必要とする）を駐在させる。

PARR-CODE	PARR-NAME	PARR-SIZE	PARR-COLOUR	PARR-FEED	PARR-WEIGHT
MC01	MACAW	10	BLUE	NUTS	25
CK01	COCKATOO	8	WHITE	FINGERS	20

XCIBは２つのローが返される（XC－ROWS－RETURNEDは２に設定される）ことを示すために更新される。
XC702UASによって満たされたIRB:Map1はどの出力エンティティ／フィールド組合せが必要なのかを教示し、Map2Bはそれらがカプセルエンティティブロック内のどこにあるのかを教示する。

MC01	MACAW	25	CK01	COCKATOO	20

値がバックエンド処理によって再記憶されたMap1及びMap2Aを用いて、XC703UASによって更新されるメソッドエンティティブロック。

PARR-CODE	PARR-NAME	PARR-WEIGHT
MC01	MACAW	25
CK01	COCKATOO	20

カプセルに対する単一の呼び出しの実行時に、種々のアプリケーション及びXCALL層によって実行される処理について以下に図６を参照して説明する。
まず、図６において１、２、３、４で示された４つのプロセスについて説明する。
XC700UAS（図６におけるプロセス１）は、アプリケーションエンティティブロックからIRBを生成する。
IRBヘッダを初期化する。
MAP1における第１入力フィールドで処理を開始し、ゼロのカラム番号が見つかるかあるいはエントリ255が処理されるまで続ける。各入力ローに対してプロセスを一回実行する。
以下の方法によって移動すべき位置を計算する。
ａ）XCIBからエンティティブロックの初めのアドレスを取り出す。エンティティは現在のMap2aによって示されている配列エントリを記載している。
ｂ）現在のローから１だけ減算し、それに（エンティティ詳細配列に示されている）ローの長さを乗算し、結果をすでに取り出したアドレスに加算する。
ｃ）結果にバイト数を加算し、（現在のMap2Aエントリに保持されている）フィールドの開始アドレスとする。
現在のMap2Aエントリにおけるフィールド長さで示されたバイト数を、上記の方法で計算された開始アドレスからIRBの次のフリーの位置に移動する。IRBにおける次のフリーの位置を移動したバイト数だけインクリメントする。
ここで、Map2AエントリｎはMap1エントリｎに対応する。
XC701UAS（図６におけるプロセス２）はIRBからEBBデータを生成する。
EBBを初期化する。
ゼロのエンティティ番号が見つかるまであるいはエントリ255が処理されるまで、各INPUT　Map1エントリを順に処理する。プロセスを入力ローの数だけ反復する。
現在のMap1　INPUTエントリに対するエンティティ番号及びカラム番号に一致するMap2Bにおけるエントリを探索する（両者はエンティティ番号順において上に向かうカラム番号である）。
以下の方法によって移動すべき位置を計算する。
ａ）XCIBからエンティティブロックの初めのアドレスを取り出す。エンティティは現在のMap2bエントリによって指示される配列エントリの詳細を示す。
ｂ）現在のローの数から１を減算し、それに、（エンティティ詳細配列において示された）ローの長さを乗算し、その結果に取り出したアドレスを加算する。

ｃ）その結果にバイト数を加算して、（現在のMap2Bに保持されている）フィールドの開始アドレスとする。
現在のMap2Bエントリにおけるフィールド長さで示されたバイト数を、IRB内の次に移動すべき位置から、カプセルエンティティブロックにおいて上記の方法で計算された開始アドレスに移動する。IRBから移動すべく次の位置を移動したバイト数だけインクリメントする。
XC702UAS（図６のプロセス３）はカプセルエンティティブロックからIRBを生成する。
これはXC701UASと同じ処理を行なうが、今度は逆の処理を行なう。Map1及びMap2Bプロセスはカプセルのエンティティブロック内から移動すべきアドレスを生成し、データは入力データをオーバライトするIRB内に移動される。
XC703UAS（図６のプロセス４）はIRBからアプリケーションエンティティブロックを生成する。
これはXC700UASと同じ処理を行なうが、今度は逆の処理を行なう。Map1及びMap2Aプロセスは移動すべきアドレスを生成し、データはアプリケーションのエンティティブロックの存在する内容をオーバライトするIRBから移動される。
アプリケーション－’通常’コード。
1.アプリケーションはメソッドにおくるべき入力データをXCALLステートメントにおけるINPUTの右側に記載されている変数に入力する。
アプリケーション－プレコンパイラ生成コード
2.XCALLステートメント（入力及び出力）の左側に記載されているカラムのエンティティ及びカラム（フィールド）番号によって、XCIBにおけるMap1を更新する。エンティティの詳細をXCIBに記憶する。供給された入力ローの数と、XCIBで要求された応答ローの数とを設定する。
3.エンティティサブスクリプト、オフセット、Map1で述べられたカラムの長さで、Map2Aを更新する。このメソッドコールのために要するIRBのサイズで、XCIBを更新する。
4.XCALLステートメントにおけるINPUTの右側に記載された変数を、XCALLステートメントの左側に記載されたフィールドを表すエンティティブロック変数に移動する。
5.XCIBを呼び出しを行っているプログラムの詳細で更新する。
6.このプログラムが実行している環境に関連するXCALLプログラムを呼び出す。XCIB、INPUTに要する全ブロック、OUTPUTフィールドを渡す。
XCALL－フロントエンド
7.渡された全ブロックの詳細でローカルXCIBを更新する。
8.メソッド、カプセル、ノード、それらの間の関係（ODICTに定義されたように）を定義する制御記憶テーブルにアクセスする。存在するメソッドを有効にし、XCIBをそのメソッドに対するカプセルidとそのカプセルに対するノードidで更新する。
9.これがこの’タスク’における最初の呼び出しの場合は、XCCBのための記憶エリアを獲得し、呼出タスクの詳細でXCIBを更新する。IRBが存在しないか、あるいは現在のものが小さすぎるときはIRBのための記憶エリアを獲得する。
10.XC700UAS
XCALL－XC700UAS
11.（グローバルXCIBにおけるMap1に列挙され、ローカルXCIBにおけるMap2Aに記載されたように）入力データを、呼出者エンティティブロックからIRBに移動する。
12.フロントエンドに戻る。
XCALL－フロントエンド
13.連続及び制御層を呼び出す。
XCALL－連続及び制御層
14.呼出者が要求されたレベル（リードオンリアクセスなど）に対するカプセルに対するアクセスを有するかどうかをチェックする。当該メソッド内の呼び出し統計とカプセル制御テーブルを更新する。
15.バックエンドを呼び出す。

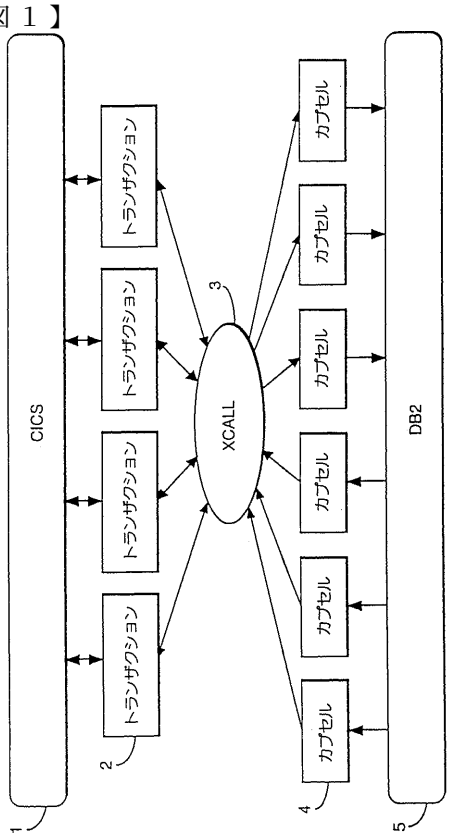
XCALL－バックエンド
16.ローカルXCIBをセーブする。
17.カプセルプログラムを呼び出す。
カプセル－プレコンパイラ生成コード（XC　ENTRYステートメントの拡張）
18.この呼び出しにおいて渡された、メソッド名、メソッド、バージョン、エンティティ番号、エンティティバージョンが有効であるか（カプセルがプレコンパイルされたときのODICT及びRECEIVEステートメントに示すように）否かをチェックする。
19.XC701UASを呼び出す。
XCALL－XC701UAS
20.現在のEBBサイズが現在の呼び出しに対して十分かどうかをチェックする。十分でないときは、XCCBを要求されたEBBのサイズで更新し、リターンコードを設定してカプセルに戻る（すなわち、以下の23に続く）。
21.ローカルXCIBにおけるEBB内のエンティティブロックのアドレスを設定する。
22.入力データ（グローバルXCIBにおけるMap1に列挙され、ローカルXCIBのMap2bに記載されているように）をIRBから移動する。
23.カプセルに戻る。
カプセル－プレコンパイラ生成コード（XC　ENTRYステートメントの拡張）
24.リターンコードをチェックする。EBBが十分大きくないときはXC704TCSを呼び出し、さもなければ、以下の28に続く）。
XCALL－XC704TCS
25.現存するEBBがあるときは記憶エリアを解放する。XCCBにおいて示されたサイズのEBBに対する記憶エリアを獲得する。
26.カプセルに戻る。
カプセル－プレコンパイラ生成コード（XC　ENTRYステートメントの拡張）
27.XC701UASを呼び出す。
上記した20に続く。
カプセル－プレコンパイラ生成コード（XC　ENTRYステートメントの拡張）
28.制御を特定のメソッドコードに渡す。
カプセル－メソッドコード
29.エンティティブロックにおける入力データを処理し、エンティティブロック内に応答（出力）データを生成する。XCIBを返されたローの数で更新する。
30.XCALLに戻る。
XCALL－バックエンド
31.カプセルがデータを返したときは、XC702UASを呼び出す。さもなければ、以下の34に続く。
XCALL－XC702UAS
32.出力データ（グローバルXCIBにおけるMap1に列挙され、ローカルXCIBにおけるMap2bに記載されているように）をIRBに移動する。
33.バックエンドにリターンする。
XCALL－バックエンド
34.上記した16でセーブしたコピーからローカルXCIBを再記憶する。
35.連続及び制御に戻る。
XCALL－連続及び制御
36.当該メソッドにおける応答統計とカプセル制御テーブルを更新する。
37.フロントエンドに戻る。
XCALL－フロントエンド
38.カプセルがデータを返したときはXC703UASを呼び出す。さもなければ、以下の41に続ける。
XCALL－XC703UAS
39.出力データ（グローバルXCIBにおけるMap1に列挙され、ローカルXCIBにおけるMap2aに

記載されているように）をIRBから呼出者エンティティブロックに移動する。
40.フロントエンドに戻る。
XCALL－フロントエンド
41.アプリケーションに戻る。
アプリケーション－プレコンパイラ生成コード
42.カプセルがデータを返したときは、出力データをエンティティブロックからXCALLステートメントにおけるINPUTの右側に記載されている変数に移動する。
アプリケーション－’通常’コード
43.プロセス応答－さらなるXCALLなどを生成する。
この例は、XCALLメッセージ配送システムがどのようにしてトランザクションとカプセルとの間のバージョン独立性を実現したかについて示している。エンティティ１及び２におけるデータのトランザクションビューは、変数ＡからＤを有するエンティティ１と、変数ＥとＦとを有するエンティティ２に関している。データのカプセルビューは、エンティティ１は変数ＡからＤを有し、エンティティ２が変数ＥからＧを有していることである。例えばこのことは、付加的変数Ｇが、トランザクションによる認識なしにデータベース（カプセルを介してアクセスされる）に追加されたときに発生する。例えばこれは、トランザクションが書かれた後、変数Ｇがデータベースに追加されたときに発生する。トランザクションとカプセルとの間のデータに関する異なる視点を考慮すると、XCALLが使用されないとき、すなわち、トランザクションとカプセルとの間の通常のリンクが使用されたときは、エンティティ２がアクセスされたときに問題を引き起こす。このエンティティが読み出されて、E2に到達したとき、カプセルは実際には、G1ではなくE2に、かつ、E2ではなくF2にアクセスする。これによって、不正確なデータがトランザクションに渡されてしまう。上記したように、XCALLと中間IRBを利用することによって、トランザクションとカプセルとは各々同じエンティティについての独自のビュー（観点）をもち、同時に、XCALLに対するMAP2a及びMAP2bを介して、データについてのビューを交信することによって、一貫性をもって正確にアクセスすることができる。
MAP2bからXCALL
リモートコール処理
XCALLにおいて、制御及び連続層が、当該要求がカプセルコールを行なうべく標準バックエンドを呼び出すのではなく、リモートマシン上のカプセルに関していることを検出したときは、新たなバックエンドを呼び出す。新たなバックエンドは要求をリモートシステムに渡さなければならない。
リモートシステム上のカプセルを直接呼び出す代わりに、新たなバックエンドは、リモートシステム上の新たなXCALLサーバプログラムを呼び出す。これはまた、フロントエンド、制御及び連続層、バックエンドの３つのプログラムに対する静的リンクを生成する。この新たなサーバにおいては、フロントエンドのみが新規である。
制御がリモートシステムから受信されたときは、最初はXCIBのみが受信される（システムではない場合は少なくともXCALLによって）。ローカルで保持されているカプセルの詳細に基づいて、要求及び応答ブロックに対する記憶エリアを獲得し、XCIBのローカル部を更新する。制御が次の層に渡される前に以下のチェックが行われる。
ｉ）これが新たなローカルタスクであるならば、呼び出しタスク番号をセーブする。さもなければ、タスクが同じであることをチェックする。
ii）カプセルがローカルであり、さらに、キーによる経路選択がローカルルーティングテーブルと一致するかどうかをチェックする。
iii）リモートシステム及びユーザidに対してローカルセキュリティチェックを行なう。セキュリティチェックフィールドが正しいことをチェックする。
要求がリモートシステムからの場合は、フロントエンドが更新されたXCIBと適当なIRBと応答ブロックとを要求者のシステムに送る。応答ブロックのRLENフィールドは各々についてデータを送るかいなか、及びデータをどれだけ送るかを示すものである。すべての適当なデータが送信された後、ブロックが占有していた記憶エリア（XCIB及びXCCBとは別個に

）を解放し、次の要求があるまで待機する。

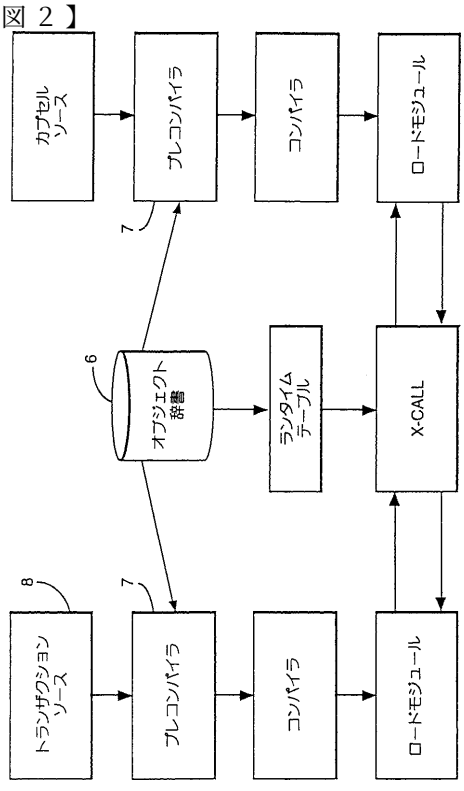
【図１】

Fig.1.



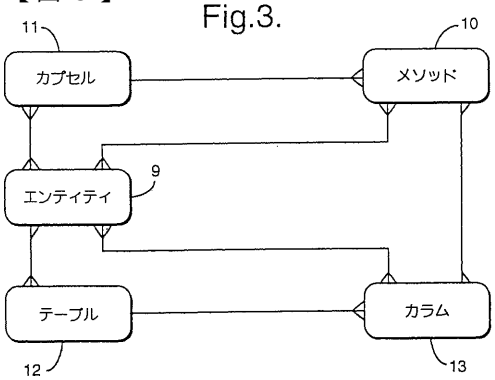
【図２】

Fig.2.



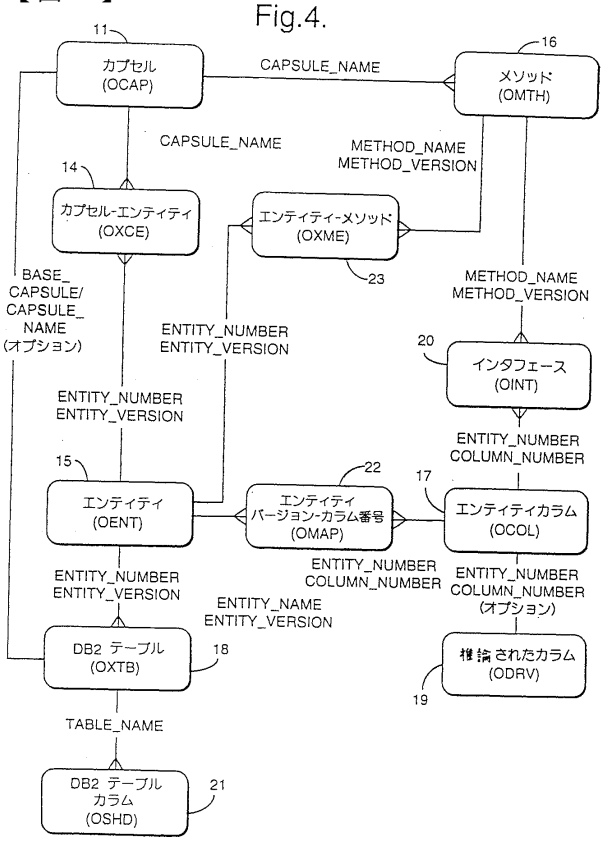
【図３】

Fig.3.



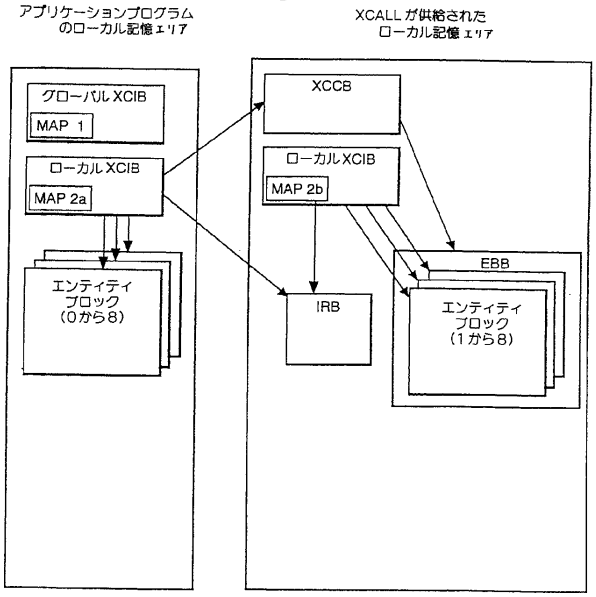
【図４】

Fig.4.



【図５】

Fig.5.



【図６】

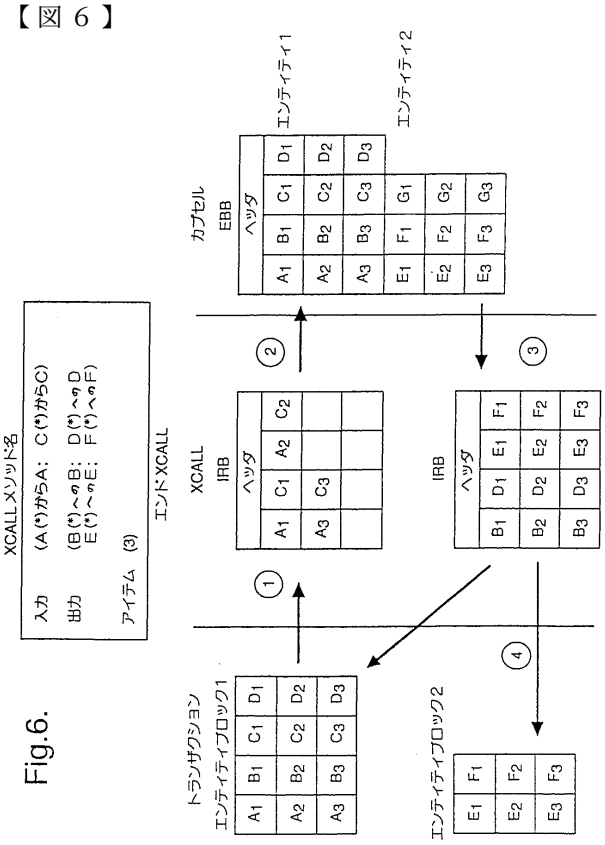


Fig.6.

フロントページの続き

(74)代理人
弁理士 白根 俊郎
(72)発明者 バグビー、ケニス・ジョン
イギリス国、エヌ５・１エーゼット、ロンドン、ハイベリー、エルフォート・ロード １８

審査官 殿川 雅也

(56)参考文献 特開平０４－０７５１３４（ＪＰ，Ａ）
米国特許第０５７９４０４２（ＵＳ，Ａ）
欧州特許出願公開第００４１４６２４（ＥＰ，Ａ１）
CHEN, P.P., The Entity-Relationship Model-Toward a Unified View of Data, ACM Transactions on Database Systems, 米国, Association for Computing Machinery, １９７６年 3月, Vol. 1, No. 1, pp.9-36
OBJECT DATA MANAGER, IBM TECHNICAL DISCLOSURE BULLETIN, 米国, IBM, １９９０年 3月, Vol.32,No. 10A, pp. 55-57

(58)調査した分野(Int.Cl.[7], ＤＢ名)
G06F 9/46