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

特許第4583375号
(P4583375)

(45)発行日　平成22年11月17日(2010.11.17)　(24)登録日　平成22年9月10日(2010.9.10)

(51)Int.Cl.　　F I
G06F 12/00 (2006.01)　　G06F 12/00 533J
G06F 17/30 (2006.01)　　G06F 12/00 513A
　　G06F 12/00 547A
　　G06F 17/30 140

請求項の数 18　(全 93 頁)

(21)出願番号　特願2006-523856 (P2006-523856)
(86)(22)出願日　平成16年7月29日 (2004.7.29)
(65)公表番号　特表2007-503049 (P2007-503049A)
(43)公表日　平成19年2月15日 (2007.2.15)
(86)国際出願番号　PCT/US2004/024287
(87)国際公開番号　WO2005/024626
(87)国際公開日　平成17年3月17日 (2005.3.17)
審査請求日　平成19年7月30日 (2007.7.30)
(31)優先権主張番号　10/646,632
(32)優先日　平成15年8月21日 (2003.8.21)
(33)優先権主張国　米国(US)
(31)優先権主張番号　PCT/US03/26144
(32)優先日　平成15年8月21日 (2003.8.21)
(33)優先権主張国　米国(US)

(73)特許権者　500046438
マイクロソフト　コーポレーション
アメリカ合衆国　ワシントン州　９８０５２－６３９９　レッドモンド　ワン　マイクロソフト　ウェイ
(74)代理人　100077481
弁理士　谷　義一
(74)代理人　100088915
弁理士　阿部　和夫
(72)発明者　アシシュ　シャー
アメリカ合衆国　９８０５２　ワシントン州　レッドモンド　ワン　マイクロソフト　ウェイ　マイクロソフト　コーポレーション内

最終頁に続く

(54)【発明の名称】同期スキーマの実装のためのシステム

(57)【特許請求の範囲】
【請求項１】
ハードウェア／ソフトウェアインターフェースシステム用ストレージプラットフォームの同期をとる方法であって、
第１のリレーショナルデータベースを第１のコンピューティングシステムが格納するステップであって、前記第１のリレーショナルデータベースはアイテムのローカルバージョンおよびフォルダアイテムを含み、前記アイテムのローカルバージョンは変更ユニットおよび第１のバージョン番号を含み、前記フォルダアイテムはコミュニティフォルダへマッピングされる、格納するステップと、
前記アイテムのローカルバージョンの変更ユニットにおける要素の値の変更毎に前記第１のバージョン番号を前記第１のコンピューティングシステムが１増加させるステップと、
第２のコンピューティングシステムから前記アイテムのリモートバージョンを前記第１のコンピューティングシステムが受信するステップであって、前記第２のコンピューティングシステムは第２のリレーショナルデータベースを格納し、前記第２のリレーショナルデータベースは前記アイテムのリモートバージョンを含み、前記アイテムのリモートバージョンは変更ユニットおよび第２のバージョン番号を含み、前記第２のコンピューティングシステムは前記アイテムのリモートバージョンの変更ユニットにおける要素の値の変更毎に前記第２のバージョン番号を１増加させるよう構成される、受信するステップと、
前記第２のバージョン番号が前記第１のバージョン番号よりも新しい場合、

前記コミュニティフォルダからのアイテムのリモートバージョンをローカルバージョンへ前記第１のコンピューティングシステムが変換するステップと、
前記アイテムのリモートバージョンの変換後、前記アイテムのローカルバージョンの変更ユニットにおける要素の値を前記アイテムのリモートバージョンの変更ユニットにおける要素の値により前記第１のコンピューティングシステムが取り替えるステップと
を備えたことを特徴とする方法。
【請求項２】
前記変更ユニットは、Ｉｔｅｍであることを特徴とする請求項１に記載の方法。
【請求項３】
前記変更ユニットは、Ｐｒｏｐｅｒｔｙであることを特徴とする請求項１に記載の方法。
【請求項４】
前記変更ユニットは、Ｎｅｓｔｅｄ　ＥｌｅｍｅｎｔのＰｒｏｐｅｒｔｙを含まないことを特徴とする請求項１に記載の方法。
【請求項５】
前記ストレージプラットフォームの複数のインスタンスは、マルチマスター同期コミュニティを含むことを特徴とする請求項１に記載の方法。
【請求項６】
前記アイテムのローカルバージョンの変更ユニットにおける要素の値と前記アイテムのリモートバージョンの変更ユニットにおける要素の値との間の競合を前記第１のコンピューティングシステムが検出するステップと、
前記アイテムのローカルバージョンの変更ユニットにおける要素の値と前記アイテムのリモートバージョンの変更ユニットにおける要素の値との間の競合を前記第１のコンピューティングシステムが解決するステップと
をさらに備えたことを特徴とする請求項１に記載の方法。
【請求項７】
ハードウェア／ソフトウェアインターフェースシステム用ストレージプラットフォームの同期をとるコンピューティングシステムであって、
アイテムのローカルバージョンおよびフォルダアイテムを含む第１のリレーショナルデータベースであって、前記アイテムのローカルバージョンは変更ユニットおよび第１のバージョン番号を含み、前記フォルダアイテムはコミュニティフォルダへマッピングされる、第１のリレーショナルデータベースと、
前記アイテムのローカルバージョンの変更ユニットにおける要素の値の変更毎に前記第１のバージョン番号を１増加させる手段と、
別のコンピューティングシステムから前記アイテムのリモートバージョンを受信する手段であって、前記別のコンピューティングシステムは第２のリレーショナルデータベースを格納し、前記第２のリレーショナルデータベースは前記アイテムのリモートバージョンを含み、前記アイテムのリモートバージョンは変更ユニットおよび第２のバージョン番号を含み、前記別のコンピューティングシステムは前記アイテムのリモートバージョンの変更ユニットにおける要素の値の変更毎に前記第２のバージョン番号を１増加させるよう構成される、受信する手段と、
前記第２のバージョン番号が前記第１のバージョン番号よりも新しい場合、
前記コミュニティフォルダからのアイテムのリモートバージョンをローカルバージョンへ変換する手段と、
前記アイテムのリモートバージョンの変換後、前記アイテムのローカルバージョンの変更ユニットにおける要素の値を前記アイテムのリモートバージョンの変更ユニットにおける要素の値により取り替える手段と
を備えたことを特徴とするシステム。
【請求項８】
前記変更ユニットは、Ｉｔｅｍであることを特徴とする請求項７に記載のシステム。

【請求項９】
前記変更ユニットは、Ｐｒｏｐｅｒｔｙであることを特徴とする請求項７に記載のシステム。
【請求項１０】
前記変更ユニットは、Ｎｅｓｔｅｄ　ＥｌｅｍｅｎｔのＰｒｏｐｅｒｔｙを含まないことを特徴とする請求項７に記載のシステム。
【請求項１１】
前記ストレージプラットフォームの複数のインスタンスは、マルチマスター同期コミュニティを含むことを特徴とする請求項７に記載のシステム。
【請求項１２】
前記アイテムのローカルバージョンの変更ユニットにおける要素の値と前記アイテムのリモートバージョンの変更ユニットにおける要素の値との間の競合を検出する手段と、
前記アイテムのローカルバージョンの変更ユニットにおける要素の値と前記アイテムのリモートバージョンの変更ユニットにおける要素の値との間の競合を解決する手段と
をさらに備えたことを特徴とする請求項７に記載のシステム。
【請求項１３】
コンピュータに、ハードウェア／ソフトウェアインターフェースシステム用ストレージプラットフォームの同期を実行させるためのプログラムを記録したコンピュータ可読記録媒体であって、
第１のリレーショナルデータベースを格納する手順であって、前記第１のリレーショナルデータベースはアイテムのローカルバージョンおよびフォルダアイテムを含み、前記アイテムのローカルバージョンは変更ユニットおよび第１のバージョン番号を含み、前記フォルダアイテムはコミュニティフォルダへマッピングされる、格納する手順と、
前記アイテムのローカルバージョンの変更ユニットにおける要素の値の変更毎に前記第１のバージョン番号を１増加させる手順と、
別のコンピューティングシステムから前記アイテムのリモートバージョンを受信する手順であって、前記別のコンピューティングシステムは第２のリレーショナルデータベースを格納し、前記第２のリレーショナルデータベースは前記アイテムのリモートバージョンを含み、前記アイテムのリモートバージョンは変更ユニットおよび第２のバージョン番号を含み、前記別のコンピューティングシステムは前記アイテムのリモートバージョンの変更ユニットにおける要素の値の変更毎に前記第２のバージョン番号を１増加させるよう構成される、受信する手順と、
前記第２のバージョン番号が前記第１のバージョン番号よりも新しい場合、
前記コミュニティフォルダからのアイテムのリモートバージョンをローカルバージョンへ変換する手順と、
前記アイテムのリモートバージョンの変換後、前記アイテムのローカルバージョンの変更ユニットにおける要素の値を前記アイテムのリモートバージョンの変更ユニットにおける要素の値により取り替える手順と
を実行させるプログラムを記録したことを特徴とするコンピュータ可読記録媒体。
【請求項１４】
前記変更ユニットは、Ｉｔｅｍであることを特徴とする請求項１３に記載のコンピュータ可読記録媒体。
【請求項１５】
前記変更ユニットは、Ｐｒｏｐｅｒｔｙであることを特徴とする請求項１３に記載のコンピュータ可読記録媒体。
【請求項１６】
前記変更ユニットは、Ｎｅｓｔｅｄ　ＥｌｅｍｅｎｔのＰｒｏｐｅｒｔｙを含まないことを特徴とする請求項１３に記載のコンピュータ可読記録媒体。
【請求項１７】
前記ストレージプラットフォームの複数のインスタンスは、マルチマスター同期コミュ

ニティを含むことを特徴とする請求項１３に記載のコンピュータ可読記録媒体。
【請求項１８】
前記第１のコンピューティングシステムによって、前記アイテムのローカルバージョンの変更ユニットにおける要素の値と前記アイテムのリモートバージョンの変更ユニットにおける要素の値との間の競合を検出する手段と、
前記第１のコンピューティングシステムによって、前記アイテムのローカルバージョンの変更ユニットにおける要素の値と前記アイテムのリモートバージョンの変更ユニットにおける要素の値との間の競合を解決する手段と
を実行させるプログラムをさらに記録したことを特徴とする請求項１３に記載のコンピュータ可読記録媒体。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、一般には、情報の格納および取り出しの分野に関係し、より詳細には、コンピュータ化システムにおけるさまざまな型のデータの編成、検索、および共有のためのアクティブストレージプラットフォームに関係するとともに、データストアまたはその部分集合の複数のインスタンスの同期にも関係する。
【背景技術】
【０００２】
（本出願は、参照によりその開示が本明細書に組み込まれている、２００３年１０月２４日に出願した米国出願第１０／６９３，３６２号明細書（整理番号ＭＳＦＴ－２８４６）、「SYSTEMS AND METHODS FOR THE IMPLEMENTATION OF A CORE SCHEMA FOR PROVIDING A TOP-LEVEL STRUCTURE FOR ORGANIZING UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM」という表題の２００３年８月２１日に出願した米国特許出願第１０／６４６，６３２号明細書（整理番号ＭＳＦＴ－１７５１）、２００３年８月２１日に出願した国際出願ＰＣＴ／ＵＳ０３／２６１４４明細書の優先権を主張するものである。）
（本出願は、内容全体がこれにより本出願に組み込まれている（便宜上部分的に要約されている）、同一出願人による、「SYSTEMS AND METHODS FOR REPRESENTING UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM BUT INDEPENDENT OF PHYSICAL REPRESENTATION」という表題の２００３年８月２１日に出願した米国特許出願第１０／６４７，０５８号明細書（整理番号ＭＳＦＴ－１７４８）、「SYSTEMS AND METHODS FOR SEPARATING UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM FROM THEIR PHYSICAL ORGANIZATION」という表題の２００３年８月２１日に出願した米国特許出願第１０／６４６，９４１号明細書（整理番号ＭＳＦＴ－１７４９）、「SYSTEMS AND METHODS FOR THE IMPLEMENTATION OF A BASE SCHEMA FOR ORGANIZING UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM」という表題の２００３年８月２１日に出願した米国特許出願第１０／６４６，９４０号明細書（整理番号ＭＳＦＴ－１７５０）、「SYSTEMS AND METHOD FOR REPRESENTING RELATIONSHIPS BETWEEN UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM」という表題の２００３年８月２１日に出願した米国特許出願第１０／６４６，６４５号明細書（整理番号ＭＳＦＴ－１７５２）、「SYSTEMS AND METHODS FOR INTERFACING APPLICATION PROGRAMS WITH AN ITEM-BASED STORAGE PLATFORM」という表題の２００３年８月２１日に出願した、米国特許出願第１０／６４６，５７５号明細書（整理番号ＭＳＦＴ－２７３３）、「STORAGE PLATFORM FOR ORGANIZING, SEARCHING, AND SHARING DATA」という表題の２００３年８月２１日に出願した、米国特許出願第１０／６４６，６４６号明細書（整理番号ＭＳＦＴ－２７３４）、「SYSTEMS AND METHODS FOR DATA MODELING IN AN ITEM-BASED STORAGE PLATFORM」という表題の２００３年８月２１日に出願した、米国特許出願第１０／６４６，５８０号明細書（整理番号ＭＳＦＴ－２７３５）、「SYSTEMS AND METHODS FOR THE IMPLEMENTATION OF A DIGITAL IMAGES SCHEMA FOR ORGANIZING UNITS

OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM」という表題の２００３年１０月２４日に出願した米国特許出願第１０／６９２，７７９号明細書（整理番号ＭＳＦＴ－２８２９）、「SYSTEMS AND METHODS FOR PROVIDING SYNCHRONIZATION SERVICES FOR UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM」という表題の２００３年１０月２４日に出願した米国特許出願第１０／６９２，５１５号明細書（整理番号ＭＳＦＴ－２８４４）、「SYSTEMS AND METHODS FOR PROVIDING RELATIONAL AND HIERARCHICAL SYNCHRONIZATION SERVICES FOR UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM」という表題の２００３年１０月２４日に出願した米国特許出願第１０／６９２，５０８号明細書（整理番号ＭＳＦＴ－２８４５）、および「SYSTEMS AND METHODS FOR EXTENSIONS AND INHERITANCE FOR UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM」という表題の２００３年１０月２４日に出願した米国特許出願第１０／６９３，５７４号明細書（整理番号ＭＳＦＴ－２８４７）で開示されている発明の主題と関連している。）

【０００３】

個々のディスク容量は、最近１０年間では、年にほぼ７０％の割合で増大してきている。ムーアの法則は、長年にわたって続いてきた中央演算処理装置（ＣＰＵ）パワーの途方もない増大を正確に予測した。有線および無線技術は、驚異的な接続性および帯域幅をもたらした。現在のトレンドが続くと仮定すると、数年以内に、平均的なラップトップコンピュータでおおよそ１テラバイト（ＴＢ）のストレージを備え、数百万個のファイルを格納するようになり、５００ギガバイト（ＧＢ）ドライブがあたりまえになることであろう。

【０００４】

消費者は、コンピュータを主に通信と個人情報の編成に使用しており、これは、従来のスケジュール管理（ＰＩＭ）スタイルのデータであろうとデジタル音楽であるか写真などの媒体であるかに関係しない。デジタルコンテンツの量、および未加工のバイトを格納する能力は、途方もなく増大してきているが、消費者がこのようなデータの編成および統合に使用できる方法は追随できていない。知識労働者は、情報の管理および共有に膨大な時間を費やしており、ある調査では、知識労働者は非生産的情報関連の活動に全時間の１５～２５％を費やしていると推測している。また他の調査では、標準的な知識労働者は情報の検索に毎日約２．５時間を費やしていると推測している。

【０００５】

開発者および情報技術（ＩＴ）部門は、人々、場所、時間、および出来事などを表す共通ストレージ抽象化用に自データストアを構築することに対し相当の時間と資金を投資している。この結果、作業を重複させるだけでなく、そのようなデータの共通検索または共有のためのメカニズムのない共通データの孤島をいくつも生み出している。今日、Ｍｉｃｒｏｓｏｆｔ　Ｗｉｎｄｏｗｓ（登録商標）オペレーティングシステムが稼働するコンピュータ上にいったいいくつのアドレス帳が存在しうるか考えてみよう。電子メールクライアントおよびパーソナルファイナンスプログラムなどの多くのアプリケーションは、個々にアドレス帳を保存し、そのようなそれぞれのプログラムが個別に保持するアドレス帳データのアプリケーション間の共有はほとんどない。その結果、ファイナンスプログラム（Ｍｉｃｒｏｓｏｆｔ　Ｍｏｎｅｙのようなプログラム）は、受取人のアドレスを電子メール連絡先フォルダ（Ｍｉｃｒｏｓｏｆｔ　Ｏｕｔｌｏｏｋが備えているようなフォルダ）に保持されているアドレスと共有しない。実際、多くのユーザは、複数のデバイスを持ち、論理的に、そのパーソナルデータをそれら自体の間で同期させ、また携帯電話からＭＳＮおよびＡＯＬなどの商業サービスにいたるまで、様々な追加ソースにまたがって同期させなければならないが、共有されているドキュメントの協働は、電子メールメッセージにドキュメントを添付することにより大半が行われている、つまり、手作業で、しかも非効率的に行われている。

【０００６】

このようなコラボレーションの欠落の理由の１つは、コンピュータシステム内に情報を

編成する伝統的なアプローチは、ファイル－フォルダ／ディレクトリベースのシステム（「ファイルシステム」）を使用し、複数のファイルを格納するために使用される記憶媒体の物理的編成の抽象化に基づいて複数のファイルをフォルダのディレクトリ階層に編成することを中心に機能していることである。１９６０年代に開発されたＭｕｌｔｉｃｓオペレーティングシステムは、初めてファイル、フォルダ、およびディレクトリを使用してオペレーティングシステムレベルでデータの格納可能単位を管理したことで高い評価を得ている。特に、Ｍｕｌｔｉｃｓはファイルの階層内で記号アドレスを使用し（それによってファイルパスという概念を導入）、ファイルの物理的アドレスはユーザ（アプリケーションおよびエンドユーザ）に対し透過的でなかった。このファイルシステムは、全体として、どの個別ファイルのファイル形式についても意識しておらず、ファイル間の関係は、オペレーティングシステムレベル（つまり、階層内のファイルの場所以外）で重要でないとみなされていた。Ｍｕｌｔｉｃｓの出現以来、格納可能なデータは、これまで、オペレーティングシステムレベルでファイル、フォルダ、およびディレクトリに編成されてきた。これらのファイルは、一般に、ファイルシステムにより保持されている特殊ファイルで具現化されたファイル階層自体（「ディレクトリ」）を含む。このディレクトリは、次に、ディレクトリ内の他のファイルおよび階層内のそのようなファイルのノード位置のすべてに対応するエントリのリストを保持する（ここでは、フォルダと呼ばれる）。このようなものは、約４０年間最新であった。

【０００７】

しかし、コンピュータ物理的ストレージシステム内に置かれている情報の妥当な表現を実現しながら、それにもかかわらずファイルシステムは、その物理的ストレージシステムの抽象化であり、したがって、それらのファイルの利用にはユーザの操作内容（コンテキストを持つユニット、特徴、および他のユニットとの関係）とオペレーティングシステムが備える内容（ファイル、フォルダ、およびディレクトリ）との間のあるレベルの指示（解釈）を必要とする。したがって、ユーザ（アプリケーションおよび／またはエンドユーザ）には選択権がないが、そうすることが不効率であり、不整合であり、または他の何らかの形で望ましくない場合でも、情報のユニットをファイルシステム構造に強制する。さらに、既存のファイルシステムは、個別ファイルに格納されているデータの構造についてはほとんど関知せず、このため、情報のほとんどは、それらを書いたアプリケーションにしかアクセス（理解）できないファイル内にロックアップされたままとなる。その結果、情報の概要説明、および情報管理するためのメカニズムに対するこのような欠損により、個別の保管場所の間で共有するデータがほとんどないままデータの保管場所を作成することになる。例えば、多くのパーソナルコンピュータ（ＰＣ）ユーザは、ファイル編成はＰＣユーザにとってかなり難しい作業であるため、あるレベルで－例えば、Ｏｕｔｌｏｏｋ Ｃｏｎｔａｃｔｓ、オンライン口座アドレス、Ｗｉｎｄｏｗｓ（登録商標）Ａｄｄｒｅｓｓ Ｂｏｏｋ、Ｑｕｉｃｋｅｎ Ｐａｙｅｅｓ、およびインスタントメッセージング（ＩＭ）の仲間リスト－やり取りする人々に関する情報を格納する５つを超える異なるストアを持つ。ほとんどの既存のファイルシステムではファイルおよびフォルダの編成にネストされたフォルダメタファを使用しているため、ファイルの個数が増えると、柔軟で効率的な編成形式を維持するために要する労力は相当きついものとなる。このような状況では、単一ファイルの複数の分類があると非常に有益であるが、しかし、既存ファイルシステム内のハードまたはソフトリンクを使用することは厄介であり、維持も困難である。

【０００８】

過去には、ファイルシステムの欠点をなくそうとする試みが何回かあったがうまくいかなかった。それらの以前の試みのうちいくつかは、連想メモリを利用し、データを物理アドレスではなく内容でアクセスできるメカニズムを実現しようとした。しかし、連想メモリはキャッシュおよびメモリ管理ユニットなどのデバイスによる小規模な利用については有用であることが実証されたが、物理的記憶媒体などのデバイスでの大規模な利用は様々な理由からまだ可能でないため、こした努力は成功しておらず、したがって、このような解決策はただ単に存在していない。オブジェクト指向データベース（ＯＯＤＢ）システム

を使用した他の試みもなされているが、これらの試みは、強いデータベース特性とよい非ファイル表現を特徴としており、ファイル表現の取り扱いでは効果的でなく、ハードウェア／ソフトウェアインターフェースのシステムレベルでのファイルおよびフォルダベースの階層構造の速度、効率、および簡潔さを再現することも可能でない。ＳｍａｌｌＴａｌｋ（およびその派生言語）を使用する試みなどの他の取り組みは、ファイルおよび非ファイル表現の取り扱いではきわめて効果的であることが実証されたが、様々なデータファイルの間に存在する関係を効率よく編成し、利用するために必要なデータベース機能を欠いており、したがって、そのようなシステムの全体的効率は許容できないものであった。さらにＢｅＯＳを使用する他の試み（および他のそのようなオペレーティングシステム研究）は、必要なデータベース機能をいくつか備えておりファイルを適切に表現できるにもかかわらず、非ファイル表現を取り扱うのには不適当である－従来のファイルシステムの中心的な欠点と同じ欠点－ことを実証した。

【０００９】

データベース技術は、同様の難題が存在するもう１つの技術分野である。例えば、リレーショナルデータベースモデルは大きな商業的成功を収めたが、実際には、独立系ソフトウェアベンダ（ＩＳＶ）は、一般に、リレーショナルデータベースのソフトウェア製品（Ｍｉｃｒｏｓｏｆｔ　ＳＱＬ　Ｓｅｒｖｅｒ）に用意されている機能のわずかな部分を利用している。その代わりに、そのような製品とのアプリケーションの相互作用のほとんどは、単純な「読み取る」と「書き込む」の形態である。これに対しすぐにわかる理由－プラットフォームまたはデータベースがわからないなど－が多数占めるが、気づかれないことが多い重要な理由の１つは、データベースは、必ずしも、大手ビジネスアプリケーションベンダが本当に必要とする的確な抽象化を行わないということである。例えば、現実世界には、「顧客」または「注文」（それら自身の中のアイテム、またそれら自身のアイテムとして注文の埋め込まれた「ラインアイテム」とともに）「アイテム」という概念があるが、リレーショナルデータベースはテーブルおよび行に関してしか認識しない。その結果、アプリケーションは、（例えば）アイテムレベルでの整合性、ロッキング、セキュリティ、および／またはトリガの側面を持つことが望ましいかもしれないが、一般的には、データベースはこれらの機能をテーブル／行のレベルでしか備えていない。これは、それぞれのアイテムがデータベースのあるテーブル内の単一行にマッピングされる場合にうまく機能する可能性があるが、複数のラインアイテムの注文の場合、アイテムが実際に複数のテーブルにマッピングされる理由があり得、それがその場合であれば、単純なリレーショナルデータベースシステムではまったく正しい抽象化を行わない。その結果、アプリケーションは、それらの基本的抽象化を実現するためにデータベースの上にロジックを構築しなければならない。つまり、基本的リレーショナルモデルは、基本的にリレーショナルモデルはアプリケーションとストレージシステムとの間のあるレベルの間接性を必要とするため、高水準のアプリケーションが容易に開発可能であるデータを格納するのに十分なプラットフォームを提供しない－データの意味構造は、いくつかの場合でのみアプリケーション内で可視とすることが可能である。いくつかのデータベースベンダが、高水準の機能をその製品に組み込んでいる－オブジェクトリレーショナル機能、新しい組織化可能モデルなどを備える－が、どれも、必要な種類の包括的ソリューションを実現していない、というのも、本当に包括的なソリューションとは、有用なドメイン抽象化（「Ｐｅｒｓｏｎｓ」、「Ｌｏｃａｔｉｏｎｓ」、「Ｅｖｅｎｔｓ」など）に対し両方とも有用なデータモデル抽象化（「Ｉｔｅｍｓ」、「Ｅｘｔｅｎｓｉｏｎｓ」、「Ｒｅｌａｔｉｏｎｓｈｉｐｓ」など）を実現するものだからである。

【００１０】

既存のデータストレージ（ｄａｔａ　ｓｔｏｒａｇｅ）およびデータベース技術における前記の欠点に関して、コンピュータシステム内のデータのすべての型を編成し、検索し、共有する能力を高めた新しいストレージプラットフォーム－既存のファイルシステムおよびデータベースシステムを超えてデータプラットフォームを拡張し拡大する、すべての型のデータを格納するように設計されているストレージプラットフォーム－が必要である

。本発明は、本明細書に参照により前の方に組み込まれている関連する発明とともに、この要求条件を満たす。

【００１１】

【非特許文献１】WinFS Sync Adapter API spec [SADP]

【非特許文献２】WinFS Sync Controller API [SCTRL] spec

【発明の開示】

【発明が解決しようとする課題】

【００１２】

以下の概要では、参照により前の方で本明細書に組み込まれている関連する発明（「関連する発明」）の背景状況において説明されている発明の様々な態様の概要を述べる。この概要は、本発明の重要な態様のすべてを網羅的に説明することも、また本発明の範囲を定めることも意図していない。むしろ、この概要は、以下の詳細な説明および図面の導入部として使用されることを意図している。

【００１３】

本発明は、関連する発明とともに、トータルとして、データの編成、検索、および共有のためのストレージプラットフォームを対象とする。本発明のストレージプラットフォームは、既存のファイルシステムおよびデータベースシステムを超えるデータストレージの概念を拡張し、広げるものであり、構造化、非構造化、または半構造化データを含むすべての型のデータ用のストアとなるように設計されている。

【課題を解決するための手段】

【００１４】

本発明のストレージプラットフォームは、データベースエンジン上に実装されたデータストアを含む。データベースエンジンは、オブジェクトリレーショナルの拡張機能（ｏｂｊｅｃｔ　ｒｅｌａｔｉｏｎａｌ　ｅｘｔｅｎｓｉｏｎｓ）を備えるリレーショナルデータベースエンジンを含む。データストアは、データの編成、検索、共有、同期、およびセキュリティをサポートするデータモデルを実装する。特定の型のデータがスキーマ内に記述されており、プラットフォームは新しい型のデータ（本質的に、スキーマにより実現される基本型の子型）を定義するためスキーマの集合を拡張するメカニズムを備える。同期処理機能は、ユーザまたはシステム間のデータの共有を円滑にする。ファイルシステム風の機能が用意され、これにより、そのような従来のファイルシステムの制限のない既存のファイルシステムとのデータストアの相互運用性を実現できる。変更追跡メカニズムは、データストアに対する変更追跡の機能を備える。ストレージプラットフォームは、さらに、アプリケーションからストレージプラットフォームの前記の機能のすべてにアクセスし、スキーマで記述されているデータにアクセスするための一組のアプリケーションプログラムインターフェースを備える。

【００１５】

データストアにより実装されるデータモデルは、アイテム、要素、および関係に関してデータストレージのユニットを定義する。アイテムは、データストアに格納可能なデータの１つのユニットであり、１つまたは複数の要素および関係を含むことができる。要素は、１つまたは複数のフィールドを含む型のインスタンスである（ここではプロパティとも呼ばれる）。関係は、２つのアイテムの間のリンクである。（本明細書で使用されているように、これらの用語および他の特定の用語は、極めて密接に使用される他の用語から分けるため原文の英語では先頭文字が大文字にされている場合があるが、何であれ英語で先頭が大文字の用語は例えば「Ｉｔｅｍ」とし、英語で大文字でないときの同じ用語、例えば、「ｉｔｅｍ」は日本語に訳してアイテムとするが、区別する意図はなく、そのような区別は想定されるべきでも、暗黙のうちに含まれるべきでもない）。

【００１６】

コンピュータシステムは、さらに、それぞれのＩｔｅｍがハードウェア／ソフトウェアインターフェースシステムにより操作することができる情報の離散的格納可能ユニットを構成する複数のＩｔｅｍｓ、前記Ｉｔｅｍｓに対する組織構造を構成する複数のＩｔｅｍ

Ｆｏｌｄｅｒｓ、およびそれぞれのＩｔｅｍが少なくとも１つのＩｔｅｍ Ｆｏｌｄｅｒに属し、複数のＩｔｅｍ Ｆｏｌｄｅｒｓに属す場合もある、複数のＩｔｅｍｓを操作するためのハードウェア／ソフトウェアインターフェースシステムを備える。

【００１７】

ＩｔｅｍまたはＩｔｅｍのプロパティ値のいくつかは、永続的ストアから派生されるのとは反対に動的に計算することができる。つまり、ハードウェア／ソフトウェアインターフェースシステムは、Ｉｔｅｍを格納する必要はなく、Ｉｔｅｍｓの現在の集合を列挙する機能またはストレージプラットフォームの識別子（アプリケーションプログラミングインターフェースつまりＡＰＩを説明するセッションでさらに詳しく説明する）が与えられた場合のＩｔｅｍを取り出す機能などのいくつかのオペレーションがサポートされる－例えば、Ｉｔｅｍは、携帯電話の現在位置または温度センサ上の温度読み取り値とすることが可能である。ハードウェア／ソフトウェアインターフェースシステムは、複数のＩｔｅｍｓを操作し、さらに、ハードウェア／ソフトウェアインターフェースシステムにより管理される複数のＲｅｌａｔｉｏｎｓｈｉｐｓにより相互接続される複数のＩｔｅｍｓを含むことができる。

【００１８】

コンピュータシステムのハードウェア／ソフトウェアインターフェースシステムは、さらに、前記ハードウェア／ソフトウェアインターフェースシステムが理解し、所定の予測可能な方法で直接処理することができるコアＩｔｅｍｓの集合を定義するコアスキーマを備える。複数のＩｔｅｍｓを操作するために、コンピュータシステムは、ハードウェア／ソフトウェアインターフェースシステムレベルで、前記複数のＩｔｅｍｓと複数のＲｅｌａｔｉｏｎｓｈｉｐｓとを相互接続し、前記Ｒｅｌａｔｉｏｎｓｈｉｐｓを管理する。

【００１９】

ストレージプラットフォームのＡＰＩは、ストレージプラットフォームスキーマの集合で定義されているそれぞれのアイテム、アイテム拡張、および関係に対するデータクラスを用意する。さらに、アプリケーションプログラミングインターフェースは、データクラスに対するビヘイビアの共通の集合を定義し、データクラスとともに、ストレージプラットフォームＡＰＩの基本プログラミングモデルを提供するフレームワーククラスの集合を実現する。ストレージプラットフォームＡＰＩは、アプリケーションプログラマを基礎のデータベースエンジンのクエリ言語の詳細から分離する形で、データストア内のアイテムの様々なプロパティに基づいてアプリケーションプログラマがクエリを形成するために使用できる簡略化されたクエリモデルを備える。ストレージプラットフォームＡＰＩは、さらに、アプリケーションプログラムによりアイテムに加えられた変更を集めてから、それらをデータストアが実装されているデータベースエンジン（または何らかの種類のストレージエンジン）により要求される正しい更新に編成する。これにより、アプリケーションプログラマは、ＡＰＩへのデータストア更新の複雑さを任せたまま、メモリ中のアイテムに変更を加えることができる。

【００２０】

本発明のストレージプラットフォームでは、共通のストレージ基盤および図式化されたデータを通じて、より効率的なアプリケーション開発を消費者、知識労働者、および企業向けに行うことができる。これは、データモデルに固有の機能を利用可能にするだけでなく、既存のファイルシステムおよびデータベースアクセス方法も包含し、拡張する機能豊富な拡張可能アプリケーションプログラミングインターフェースを備える。

【００２１】

相互関係のある発明のこの包括的構造の一部として（「発明を実施するための最良の形態」のセクションＩＩで詳しく説明する）、本発明は、特に、同期スキーマを対象とする（「発明を実施するための最良の形態」のセクションＩＩＩで詳しく説明する）。本発明の他の特徴および利点は、本発明の詳細な説明および付属の図面から明白になるであろう。

【００２２】

前述の概要は、本発明の以下の詳細な説明とともに、付属の図面を参照しつつ読むとよく理解できる。本発明を例示する目的のために、図面には、本発明の様々な態様の実施例が示されているが、本発明は、開示されている特定の方法および手段に限定されない。図面は以下で説明される。

【発明を実施するための最良の形態】

【００２３】

目次

９．更新
１０．変更追跡＆ツームストーン
ａ）変更追跡
（１）「マスター」検索ビューでの変更追跡
（２）「型付き」検索ビューでの変更追跡
ｂ）ツームストーン
（１）アイテムツームストーン
（２）拡張ツームストーン
（３）関係ツームストーン
（４）ツームストーンのクリーンアップ
１１．ヘルパＡＰＩおよび関数
ａ）関数［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．ＧｅｔＩｔｅｍ
ｂ）関数［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．ＧｅｔＥｘｔｅｎｓｉｏｎ
ｃ）関数［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．ＧｅｔＲｅｌａｔｉｏｎｓｈｉｐ
１２．メタデータ
ａ）スキーマメタデータ
ｂ）インスタンスメタデータ
Ｅ．セキュリティ
Ｆ．通知および変更追跡
Ｇ．従来のファイルシステムの相互運用性
Ｈ．ストレージプラットフォームＡＰＩ
ＩＩＩ．同期ＡＰＩ
Ａ．同期の概要
１．ストレージプラットフォームとストレージプラットフォームとの間の同期処理
ａ）同期（Ｓｙｎｃ）制御アプリケーション
ｂ）スキーマ注釈
ｃ）同期構成
（１）コミュニティフォルダーマッピング
（２）プロファイル
（３）スケジュール
ｄ）競合回避処理
（１）競合検出
（ａ）知識ベースの競合
（ｂ）制約ベースの競合
（２）競合処理
（ａ）自動競合解決
（ｂ）競合ログ作成
（ｃ）競合の検査および解決
（ｄ）レプリカの収束および競合解決の伝播
２．非ストレージプラットフォームデータストアの同期処理
ａ）同期サービス
（１）Ｃｈａｎｇｅ　Ｅｎｕｍｅｒａｔｉｏｎ
（２）Ｃｈａｎｇｅ　Ａｐｐｌｉｃａｔｉｏｎ
（３）Ｃｏｎｆｌｉｃｔ　Ｒｅｓｏｌｕｔｉｏｎ
ｂ）アダプタの実装
３．セキュリティ
４．管理可能性
Ｂ．同期処理ＡＰＩの概要
１．一般的用語
２．同期処理ＡＰＩの主要事項

Ｃ．同期処理ＡＰＩサービス
１．変更列挙
２．変更適用
３．サンプルコード
４．ＡＰＩ同期処理のメソッド
ＩＶ．結論
【００２４】
Ｉ．はじめに
本発明の主題は、法的要件を満たすように特異性とともに説明されている。しかし、説明自体は、本発明のスコープを制限することを意図されていない。むしろ、発明者は、主張されている主題は、他の現在または将来の技術とともに、本明細書で説明されている活動または要素と類似しているが異なるステップまたはステップの組合せを含むように、他の方法でも具現化されうることを考察した。さらに、「ステップ」という用語は、本明細書では、採用されている方法の異なる要素を暗示するために使用される場合があるが、この用語は、個々のステップの順序が明示的に説明されていない場合、かつ説明されている場合を除き、本明細書で開示されている様々なステップ間の特定の順序を暗示するものとして解釈すべきではない。
【００２５】
Ａ．コンピューティング環境例
本発明の数多くの実施形態は、コンピュータ上で実行することができる。図１および以下の説明は、本発明を実施できる適当なコンピューティング環境について簡潔に述べた一般的な説明である。必要というわけではないが、本発明の様々な態様は、クライアントワークステーションまたはサーバなどのコンピュータにより実行される、プログラムモジュールなどの、コンピュータ実行可能命令の一般的文脈において説明することができる。一般に、プログラムモジュールは、特定のタスクを実行する、または特定の抽象データ型を実装するルーチン、プログラム、オブジェクト、コンポーネント、データ構造などを含む。さらに、本発明は、ハンドヘルドデバイス、マルチプロセッサシステム、マイクロプロセッサベースのまたはプログラム可能な家電製品、ネットワークＰＣ、ミニコンピュータ、メインフレームコンピュータなどを含む、他のコンピュータシステム構成を使用して実施できる。また、本発明は、通信ネットワークを通じてリンクされているリモート処理デバイスによりタスクが実行される分散コンピューティング環境で実施することもできる。分散コンピューティング環境では、プログラムモジュールは、ローカルおよびリモートの両方のメモリ記憶デバイス内に配置されうる。
【００２６】
図１に示されているように、汎用コンピューティングシステム例は、処理ユニット２１、システムメモリ２２、およびシステムメモリを含む様々なシステムコンポーネントを処理ユニット２１に結合するシステムバス２３を備える、従来のパーソナルコンピュータ２０などを含む。システムバス２３は、メモリバスまたはメモリコントローラ、周辺機器バス、および様々なバスアーキテクチャを使用するローカルバスを含む数種類のバス構造のうちのいずれでもよい。システムメモリは、読み取り専用メモリ（ＲＯＭ）２４およびランダムアクセスメモリ（ＲＡＭ）２５を含む。起動時などにパーソナルコンピュータ２０内の要素間の情報伝送を助ける基本ルーチンを含む基本入出力システム２６（ＢＩＯＳ）は、ＲＯＭ　２４に格納される。パーソナルコンピュータ２０は、さらに、図に示されていないハードディスクへの読み書きを行うためのハードディスクドライブ２７、取り外し可能磁気ディスク２９への読み書きを行うための磁気ディスクドライブ２８、およびＣＤ－ＲＯＭまたはその他の光媒体などの取り外し可能光ディスク３１への読み書きを行うための光ディスクドライブ３０を備える。ハードディスクドライブ２７、磁気ディスクドライブ２８、および光ディスクドライブ３０は、ハードディスクドライブインターフェース３２、磁気ディスクドライブインターフェース３３、および光ドライブインターフェース３４によりそれぞれシステムバス２３に接続される。ドライブおよび関連コンピュータ可

読媒体は、パーソナルコンピュータ２０用のコンピュータ可読命令、データ構造体、プログラムモジュール、およびその他のデータを格納する不揮発性記憶装置を実現する。本発明で説明されている環境例ではハードディスク、取り外し可能磁気ディスク２９、および取り外し可能光ディスク３１を採用しているが、当業者であれば、磁気カセット、フラッシュメモリカード、デジタルビデオディスク、ベルヌーイカートリッジ、ランダムアクセスメモリ（ＲＡＭ）、読み取り専用メモリ（ＲＯＭ）などのコンピュータからアクセス可能なデータを格納できる他のタイプのコンピュータ可読媒体もこの動作環境で使用できることを理解するであろう。同様に、この環境は、さらに、熱感知器およびセキュリティまたは火災警報装置などの多くのタイプの監視デバイス、およびその他の情報源を含むことができる。

【００２７】

オペレーティングシステム３５、１つまたは複数のアプリケーションプログラム３６、その他のプログラムモジュール３７、およびプログラムデータ３８を含む、多くのプログラムモジュールは、ハードディスク、磁気ディスク２９、光ディスク３１、ＲＯＭ　２４、またはＲＡＭ　２５に格納されることができる。ユーザはキーボード４０およびポインティングデバイス４２などの入力デバイスを通じてパーソナルコンピュータ２０にコマンドおよび情報を入力することができる。他の入力デバイス（図に示されていない）としては、マイク、ジョイスティック、ゲームパッド、衛星放送受信アンテナ、スキャナなどがある。これらの入力デバイスやその他の入力デバイスは、システムバスに結合されているシリアポートインターフェース４６を介して処理ユニット２１に接続されることが多いが、パラレルポート、ゲームポート、またはユニバーサルシリアルバス（ＵＳＢ）などの他のインターフェースにより接続されることもできる。モニタ４７またはその他の種類の表示デバイスも、ビデオアダプタ４８などのインターフェースを介してシステムバス２３に接続される。パーソナルコンピュータは、通常、モニタ４７の他に、スピーカおよびプリンタなど、他の周辺出力装置（図に示されていない）を備える。図１のシステム例は、さらに、ホストアダプタ５５、ＳＣＳＩ（Ｓｍａｌｌ　Ｃｏｍｐｕｔｅｒ　Ｓｙｓｔｅｍ　Ｉｎｔｅｒｆａｃｅ）バス５６、およびＳＣＳＩバス５６に接続された外部記憶デバイス６２も含む。

【００２８】

パーソナルコンピュータ２０は、リモートコンピュータ４９などの１つまたは複数のリモートコンピュータへの論理接続を使用してネットワーク接続環境で動作することができる。リモートコンピュータ４９は、他のパーソナルコンピュータ、サーバ、ルータ、ネットワークＰＣ、ピアデバイス、またはその他の共通ネットワークノードでもよく、通常は、パーソナルコンピュータ２０に関係する上述の要素の多くまたはすべてを含むが、メモリ記憶デバイス５０だけが図１に示されている。図１で説明されている論理接続は、ローカルエリアネットワーク（ＬＡＮ）５１とワイドエリアネットワーク（ＷＡＮ）５２を含む。このようなネットワーキング環境は、オフィス、企業全体にわたるコンピュータネットワーク、イントラネット、およびインターネットでは一般的である。

【００２９】

ＬＡＮネットワーキング環境で使用される場合、パーソナルコンピュータ２０は、ネットワークインターフェースまたはアダプタ５３を介してＬＡＮ　５１に接続される。ＷＡＮネットワーキング環境で使用される場合、パーソナルコンピュータ２０は、通常、モデム５４またはインターネットなどのワイドエリアネットワーク５２上で通信を確立するためのその他の手段を備える。モデム５４は、内蔵でも外付けでもよいが、シリアルポートインターフェース４６を介してシステムバス２３に接続される。ネットワーク接続環境では、パーソナルコンピュータ２０またはその一部に関して示されているプログラムモジュールは、リモートメモリ記憶デバイスに格納されうる。図に示されているネットワーク接続は実施例であり、コンピュータ間の通信リンクを確立するのに他の手段が使用可能であることは理解されるであろう。

【００３０】

図２のブロック図に例示されているように、コンピュータシステム２００は、大まかに言って、ハードウェアコンポーネント２０２、ハードウェア／ソフトウェアインターフェースシステムコンポーネント２０４、およびアプリケーションプログラムコンポーネント２０６（本明細書では状況に応じて「ユーザコンポーネント」または「ソフトウェアコンポーネント」とも呼ぶ）の３つのコンポーネントグループに分けることができる。

【００３１】

コンピュータシステム２００の様々な実施形態において、図１を再び参照すると、ハードウェアコンポーネント２０２は、中央演算処理装置（ＣＰＵ）２１、メモリ（ＲＯＭ　２４およびＲＡＭ　２５の両方）、基本入出力システム（ＢＩＯＳ）２６、およびとりわけキーボード４０、マウス４２、モニタ４７、および／またはプリンタ（図に示されていない）などの様々な入出力（Ｉ／Ｏ）デバイスを備えることができる。ハードウェアコンポーネント２０２は、コンピュータシステム２００用の基本的な物理的インフラストラクチャを備える。

【００３２】

アプリケーションプログラムコンポーネント２０６は、限定はしないが、コンパイラ、データベースシステム、ワードプロセッサ、ビジネスプログラム、ビデオゲームなどを含む、様々なソフトウェアプログラムを備える。アプリケーションプログラムは、様々なユーザ（マシン、他のコンピュータシステム、および／またはエンドユーザ）向けに問題を解決し、解決策を提供し、データを処理するためコンピュータ資源を利用するための手段を備える。

【００３３】

ハードウェア／ソフトウェアインターフェースシステムコンポーネント２０４は、それ自体、ほとんどの場合、シェルおよびカーネルを含むオペレーティングシステムを備える（および、いくつかの実施形態では、そのようなオペレーティングシステムのみで構成することができる）。「オペレーティングシステム」（ＯＳ）は、アプリケーションプログラムとコンピュータハードウェアとの間の媒介として動作する特別なプログラムである。ハードウェア／ソフトウェアインターフェースシステムコンポーネント２０４は、さらに、仮想マシンマネージャ（ＶＭＭ）、共通言語ランタイム（ＣＬＲ）またはその機能的等価物、Ｊａｖａ（登録商標）仮想マシン（ＪＶＭ）またはその機能的等価物、またはコンピュータシステム内のオペレーティングシステムの代わりとなる、またはそれへの追加となる他のそのようなソフトウェアコンポーネントを含むこともできる。ハードウェア／ソフトウェアインターフェースシステムの目的は、ユーザがアプリケーションプログラムを実行できるような環境を用意することである。ハードウェア／ソフトウェアインターフェースシステムの目標は、コンピュータシステムを使いやすくするだけでなく、コンピュータハードウェアを効率的な方法で利用できるようにすることである。

【００３４】

ハードウェア／ソフトウェアインターフェースシステムは、一般に、起動時にコンピュータシステムにロードされ、それ以降、コンピュータシステム内のすべてのアプリケーションプログラムを管理する。アプリケーションプログラムは、アプリケーションプログラムインターフェース（ＡＰＩ）を介してサービスを要求することによりハードウェア／ソフトウェアインターフェースシステムとやり取りをする。一部のアプリケーションプログラムでは、エンドユーザはコマンド言語またはグラフィカルユーザインターフェース（ＧＵＩ）などのユーザインターフェースを介してハードウェア／ソフトウェアインターフェースシステムとやり取りすることができる。

【００３５】

ハードウェア／ソフトウェアインターフェースシステムは、従来、アプリケーションに対し様々なサービスを実行するものである。複数のプログラムが同時に実行できるマルチタスクのハードウェア／ソフトウェアインターフェースシステムにおいて、ハードウェア／ソフトウェアインターフェースシステムは、どのアプリケーションをどのような順序で実行し、それぞれのアプリケーションについて次のアプリケーションに切り換えるまでに

どれだけの時間を許すべきかを決定する。ハードウェア／ソフトウェアインターフェースシステムは、さらに、複数のアプリケーション間で内部メモリの共有を管理し、さらに、ハードディスク、プリンタ、およびダイアルアップポートなどの接続されているハードウェアデバイスへの入力およびそこからの出力を処理する。ハードウェア／ソフトウェアインターフェースシステムは、さらに、オペレーションのステータスおよび発生した可能性のあるエラーに関するメッセージをそれぞれのアプリケーション（および、場合によってはエンドユーザ）に送信する。ハードウェア／ソフトウェアインターフェースシステムは、さらに、バッチジョブ（例えば、印刷）の管理の負担を取り除き、開始アプリケーションがこのような作業から解放され、他の処理および／またはオペレーションを再開できるようにすることもできる。並列処理機能を備えることができるコンピュータでは、ハードウェア／ソフトウェアインターフェースシステムは、さらに、プログラムを分割して一度に複数のプロセッサ上で実行させることも管理する、

ハードウェア／ソフトウェアインターフェースシステムのシェル（本明細書では単に「シェル」と呼ぶ）は、ハードウェア／ソフトウェアインターフェースシステムとの対話型エンドユーザインターフェースである。（シェルは、「コマンドインタプリタ」と呼ばれたり、オペレーティングシステムでは、「オペレーティングシステムシェル」と呼ばれたりもする。）シェルは、アプリケーションプログラムおよび／またはエンドユーザにより直接アクセス可能なハードウェア／ソフトウェアインターフェースシステムの外側の層である。シェルとは対照的に、カーネルはハードウェアコンポーネントと直接やり取りするハードウェア／ソフトウェアインターフェースシステムの一番内側の層である。

【００３６】

本発明の多数の実施形態は、コンピュータ化システムに特によく適していると考えられるが、本明細書では、本発明をそのような実施形態に限定することを一切意図していない。それどころか、本明細書で使用されているように、「コンピュータシステム」という用語は、情報を格納し処理することができ、および／または格納されている情報を使用してデバイス自体のビヘイビアまたは実行を、そのようなデバイスがその性質上電子デバイスであろうと、機械デバイスであろうと、論理デバイスであろうと、仮想デバイスであろうと関係なく、制御できることができるありとあらゆるデバイスを包含することが意図されている。

【００３７】

Ｂ．従来のファイルベースのストレージ

ほとんどの今日のコンピュータシステムでは、「ファイル」は、ハードウェア／ソフトウェアインターフェースシステムだけでなくアプリケーションプログラム、データセットなどをも含むことができる格納可能な情報のユニットである。現代的なすべてのハードウェア／ソフトウェアインターフェースシステム（Ｗｉｎｄｏｗｓ（登録商標）、Ｕｎｉｘ（登録商標）、Ｌｉｎｕｘ、Ｍａｃ　ＯＳ、仮想マシンシステムなど）では、ファイルは、ハードウェア／ソフトウェアインターフェースシステムにより操作可能な情報（例えば、データ、プログラムなど）の基本的に離散的（格納可能および取り出し可能な）ユニットである。ファイルのグループは、「フォルダ」単位で一般的には編成される。Ｍｉｃｒｏｓｏｆｔ　Ｗｉｎｄｏｗｓ（登録商標）、Ｍａｃｉｎｔｏｓｈ　ＯＳ、および他のハードウェア／ソフトウェアインターフェースシステムは、フォルダとは、取り出し可能、移動可能、そして単一の情報ユニットとして他の何らかの手段により操作できるファイルのコレクションである。これらのフォルダは、次に、「ディレクトリ」（以下で詳述する）と呼ばれるツリーベースの階層型配列で編成される。ＤＯＳ、ｚ／ＯＳ、およびほとんどのＵｎｉｘ（登録商標）ベースのオペレーティングシステムなど他のいくつかのハードウェア／インターフェースシステムでは、「ディレクトリ」および／または「フォルダ」という用語は交換して使用することができ、また以前のＡｐｐｌｅコンピュータシステム（例えば、Ａｐｐｌｅ　ＩＩｅ）ではディレクトリの代わりに「カタログ」という用語を使用していたが、本明細書で使用されているように、これらの用語はすべて、同義語であり交換可能であるとみなし、階層型情報記憶構造およびそのフォルダおよびファイルコンポ

ーネントに対する他のすべての相当する用語および参照を含むことを意図している。

【００３８】

従来、ディレクトリ（別名、フォルダのディレクタ）は、ツリーベースの階層構造であり、ファイルは複数のフォルダにグループ化され、フォルダは、さらに、ディレクトリツリーを含む相対的ノード位置に応じて配列される。例えば、図２Ａに例示されているように、ＤＯＳベースのファイルシステムのベースフォルダ（または「ルートディレクトリ」）２１２は、複数のフォルダ２１４を含むことができ、それぞれのフォルダは、さらに、追加フォルダ（その特定のフォルダの「サブフォルダ」として）２１６を含むことができ、それぞれ、さらに、追加フォルダ２１８を含むことができ、というように無限に続けることができる。これらのフォルダはそれぞれ、１つまたは複数のファイル２２０を保持できるが、ハードウェア／ソフトウェアインターフェースシステムレベルでは、フォルダ内の個々のファイルは、ツリー階層内のその位置以外共通するものは何も持たない。ファイルをフォルダ階層に編成するこのアプローチは、それらのファイルを格納するために使用される標準的な記憶媒体（例えば、ハードディスク、フロッピー（登録商標）ディスク、ＣＤ－ＲＯＭなど）の物理的構成を間接的に反映することは驚くことではない。

【００３９】

前記に加えて、それぞれのフォルダは、そのサブフォルダおよびそのファイル用のコンテナである－つまり、それぞれのフォルダはそのサブフォルダおよびファイルを所有するということである。例えば、フォルダがハードウェア／ソフトウェアインターフェースシステムにより削除されると、そのフォルダのサブフォルダおよびファイルも削除される（それぞれのサブフォルダの場合、さらにその所有するサブフォルダおよびファイルを再帰的に含む）。同様に、それぞれのファイルは、一般的に、１つのフォルダのみにより所有され、ファイルはコピーされ、そのコピーは異なるフォルダに配置されることができるが、ファイルのコピーはそれ自体、オリジナルとの直接の関連を持たない異なる、別のユニットである（例えば、オリジナルのファイルに変更を加えでも、ハードウェア／ソフトウェアインターフェースシステムレベルではそのコピーファイルに反映されない）。この点に関して、したがって、ファイルおよびフォルダは、本質的に特徴として「物理的」であるが、それは、フォルダは物理的コンテナのように取り扱われ、ファイルはそれらのコンテナ内の離散的な別々の物理的要素として取り扱われるからである。

【００４０】

ＩＩ．データの編成、検索、および共有のためのＷＩＮＦＳストレージプラットフォーム

本発明は、本明細書の前の方で説明したように参照により組み込まれている関連する発明と組み合わせて、データの編成、検索、および共有のためのストレージプラットフォームを対象とするものである。本発明のストレージプラットフォームは、上述の種類の既存のファイルシステムおよびデータベースシステムを超えてデータプラットフォームの概念を拡張し、広げるものであり、Ｉｔｅｍｓと呼ばれるデータの新しい形態を含む、すべての型のデータ用のストアとなるように設計されている。

【００４１】

Ａ．用語

本明細書で使用されているように、また請求項で使用されているように、以下の用語は以下の意味を持つ。

・　「Ｉｔｅｍ」は、単純ファイルとは異なり、ハードウェア／ソフトウェアインターフェースシステムシェルによりエンドユーザに対し公開されるすべてのオブジェクトにわたって一般的にサポートされるプロパティの基本集合を持つオブジェクトであるハードウェア／ソフトウェアインターフェースシステムにアクセスできる格納可能情報のユニットである。Ｉｔｅｍｓは、さらに、新しいプロパティおよび関係を導入することができる機能を含むすべてのＩｔｅｍ型にわたって一般的にサポートされるプロパティおよび関係も有する（本明細書の後の方で詳述する）。

・　「オペレーティングシステム」（ＯＳ）は、アプリケーションプログラムとコンピ

ュータハードウェアとの間の媒介として動作する特別なプログラムである。オペレーティングシステムは、ほとんどの場合、シェルおよびカーネルを備える。

・ 「ハードウェア／ソフトウェアインターフェースシステム」は、コンピュータシステム上で実行されるコンピュータシステムおよびアプリケーションの基礎となるハードウェアコンポーネント間のインターフェースとして使用される、ソフトウェア、またはハードウェアとソフトウェアとの組合せである。ハードウェア／ソフトウェアインターフェースシステムは、通常、オペレーティングシステムを備える（いくつかの実施形態では、オペレーティングシステムのみからなる）。ハードウェア／ソフトウェアインターフェースシステムは、さらに、仮想マシンマネージャ（ＶＭＭ）、共通言語ランタイム（ＣＬＲ）またはその機能的等価物、Ｊａｖａ（登録商標）仮想マシン（ＪＶＭ）またはその機能的等価物、またはコンピュータシステム内のオペレーティングシステムの代わりとなる、またはそれへの追加となる他のそのようなソフトウェアコンポーネントを含むこともできる。ハードウェア／ソフトウェアインターフェースシステムの目的は、ユーザがアプリケーションプログラムを実行できるような環境を用意することである。ハードウェア／ソフトウェアインターフェースシステムの目標は、コンピュータシステムを使いやすくするだけでなく、コンピュータハードウェアを効率的な方法で利用できるようにすることである。

【００４２】

Ｂ．ストレージプラットフォームの概要

図３を参照すると、ストレージプラットフォーム３００は、データベースエンジン３１４上に実装されたデータストア３０２を備える。一実施形態では、データベースエンジンは、オブジェクトリレーショナルの拡張機能を備えるリレーショナルデータベースエンジンを含む。一実施形態では、リレーショナルデータベースエンジン３１４は、Ｍｉｃｒｏｓｏｆｔ　ＳＱＬ　Ｓｅｒｖｅｒリレーショナルデータベースエンジンを含む。データストア３０２は、データの編成、検索、共有、同期、およびセキュリティをサポートするデータモデル３０４を実装する。データの特定の型は、スキーマ３４０などのスキーマで記述され、ストレージプラットフォーム３００は、以下で詳しく説明するように、それらのスキーマを展開するとともに、それらのスキーマを拡張するためのツール３４６を備える。

【００４３】

データストア３０２内に実装された変更追跡メカニズム３０６は、データストアへの変更を追跡する機能を備える。データストア３０２は、さらに、セキュリティ機能３０８および格上げ／格下げ機能３１０も備え、両方とも以下で詳述する。データストア３０２は、さらに、一組のアプリケーションプログラミングインターフェース３１２を備え、データストア３０２の機能を他のストレージプラットフォームコンポーネントおよびそのストレージプラットフォームを利用するアプリケーションプログラム（例えば、アプリケーションプログラム３５０ａ、３５０ｂ、および３５０ｃ）に公開する。本発明のストレージプラットフォームは、さらに、アプリケーションプログラム３５０ａ、３５０ｂ、および３５０ｃなどのアプリケーションプログラムでストレージプラットフォームの前記の機能すべてにアクセスし、スキーマで記述されたデータにアクセスできるようにす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）３２２を備える。ストレージプラットフォームＡＰＩ　３２２は、アプリケーションプログラムにより、ＯＬＥ　ＤＢ　ＡＰＩ　３２４およびＭｉｃｒｏｓｏｆｔ　Ｗｉｎｄｏｗｓ（登録商標）Ｗｉｎ３２　ＡＰＩ　３２６などの他のＡＰＩと併用することができる。

【００４４】

本発明のストレージプラットフォーム３００は、ユーザまたはシステム間でのデータの共有を円滑にする同期サービス３３０を含む、様々なサービス３２８をアプリケーションプログラムに提供することができる。例えば、同期サービス３３０では、データストア３０２と同じ形式を持つ他のデータストア３４０との相互運用性とともに、他の形式を持つデータストア３４２へのアクセスをも可能にすることができる。ストレージプラットフォーム３００は、さらに、データストア３０２とＷｉｎｄｏｗｓ（登録商標）ＮＴＦＳファ

イルシステム３１８などの既存のファイルシステムとデータストア３０２の相互運用を可能にするファイルシステムの機能をも備える。少なくとも一部の実施形態では、ストレージプラットフォーム３２０は、さらに、データへの操作を可能にし、また他のシステムとのやり取りを可能にするための追加機能をアプリケーションプログラミングに付加することもできる。これらの機能は、Ｉｎｆｏ Ａｇｅｎｔサービス３３４および通知サービス３３２などの追加サービス３２８の形態だけでなく、他のユーティリティ３３６の形態で具現化することができる。

【００４５】

少なくとも一部の実施形態では、ストレージプラットフォームは、コンピュータシステムのハードウェア／ソフトウェアインターフェースシステム内に具現化されるか、またはそのなくてはならない一部を形成する。例えば、限定はしないが、本発明のストレージプラットフォームは、オペレーティングシステム、仮想マシンマネージャ（ＶＭＭ）、共通言語ランタイム（ＣＬＲ）またはその機能的等価物、またはＪａｖａ（登録商標）仮想マシン（ＪＶＭ）またはその機能的等価物で具現化されるか、またはそのなくてはならない一部を形成することができる。本発明のストレージプラットフォームでは、共通のストレージ基盤および図式化されたデータを通じて、より効率的なアプリケーション開発を消費者、知識労働者、および企業向けに行うことができる。これは、データモデルに固有の機能を利用可能にするだけでなく、既存のファイルシステムおよびデータベースアクセス方法も包含し、拡張する機能豊富な拡張可能プログラミングサーフェスエリアを提供する。

【００４６】

以下の説明では、また様々な図において、本発明のストレージプラットフォーム３００は、「ＷｉｎＦＳ」と呼ぶことができる。しかし、この名称を使用してストレージプラットフォームを指すのは、説明の便宜上のことにすぎず、いかなる点でも限定することを意図されていない。

【００４７】

Ｃ．データモデル

本発明のストレージプラットフォーム３００のデータストア３０２は、ストア内に置かれているデータの編成、検索、共有、同期、およびセキュリティをサポートするデータモデルを実装する。本発明のデータモデルでは、「Ｉｔｅｍ」は、ストレージ情報の基本ニットである。このデータモデルは、以下で詳述するが、ＩｔｅｍｓおよびＩｔｅｍ拡張を宣言し、Ｉｔｅｍｓ間の関係を確立し、Ｉｔｅｍ ＦｏｌｄｅｒｓとＣａｔｅｇｏｒｉｅｓ内にＩｔｅｍｓを編成するためのメカニズムを備える。

【００４８】

データモデルは、ＴｙｐｅｓとＲｅｌａｔｉｏｎｓｈｉｐｓという2つのプリミティブなメカニズムに依存している。Ｔｙｐｅｓは、Ｔｙｐｅのインスタンスの形態を制御する形式を定める構造である。その形式は、Ｐｒｏｐｅｒｔｉｅｓの順序集合として表される。Ｐｒｏｐｅｒｔｙは、与えられたＴｙｐｅの値または値の集合に対する名前である。例えば、ＵＳＰｏｓｔａｌＡｄｄｒｅｓｓ型は、プロパティＳｔｒｅｅｔ、Ｃｉｔｙ、Ｚｉｐ、Ｓｔａｔｅを持つことができ、その中で、Ｓｔｒｅｅｔ、Ｃｉｔｙ、Ｓｔａｔｅは型Ｓｔｒｉｎｇであり、ＺｉｐはＴｙｐｅ Ｉｎｔ３２型である。Ｓｔｒｅｅｔプロパティに対し住所は複数の値をとりうるので、Ｓｔｒｅｅｔは、多値（つまり、値の集合）とすることができる。システムでは、他の型の構成で使用できるいくつかのプリミティブ型を定義している－これらは、Ｓｔｒｉｎｇ、Ｂｉｎａｒｙ、Ｂｏｏｌｅａｎ、Ｉｎｔ１６、Ｉｎｔ３２、Ｉｎｔ６４、Ｓｉｎｇｌｅ、Ｄｏｕｂｌｅ、Ｂｙｔｅ、ＤａｔｅＴｉｍｅ、Ｄｅｃｉｍａｌ、およびＧＵＩＤを含む。ＴｙｐｅのＰｒｏｐｅｒｔｉｅｓは、プリミティブ型のどれかまたは（下記のいくつかの制約があるが）構成された型のどれかを使用して定義することができる。例えば、Ｐｒｏｐｅｒｔｉｅｓ ＣｏｏｒｄｉｎａｔｅとＡｄｄｒｅｓｓを持つＬｏｃａｔｉｏｎ Ｔｙｐｅが定義されうるが、Ａｄｄｒｅｓｓ Ｐｒｏｐｅｒｔｙは上述のようにＴｙｐｅ ＵＳＰｏｓｔａｌＡｄｄｒｅｓｓである。Ｐｒｏｐｅｒｔｉｅｓは、さらに、必須またはオプションとすることもできる。

【００４９】
Ｒｅｌａｔｉｏｎｓｈｉｐｓは、２つの型のインスタンスの集合同士の間のマッピングとして宣言され、それを表すことができる。例えば、Ｐｅｒｓｏｎ Ｔｙｐｅとどの人々がどの場所に住んでいるかを定義するＬｉｖｅｓＡｔと呼ばれるＬｏｃａｔｉｏｎ Ｔｙｐｅとの間のＲｅｌａｔｉｏｎｓｈｉｐを宣言することができる。Ｒｅｌａｔｉｏｎｓｈｉｐは、１つの名前、２つの終点、つまり、ソース終点とターゲット終点を持つ。Ｒｅｌａｔｉｏｎｓｈｉｐｓは、さらに、プロパティの順序集合を持つこともできる。ＳｏｕｒｃｅとＴａｒｇｅｔの終点は両方ともＮａｍｅとＴｙｐｅを持つ。例えば、ＬｉｖｅｓＡｔ ＲｅｌａｔｉｏｎｓｈｉｐはＴｙｐｅ ＰｅｒｓｏｎのＯｃｃｕｐａｎｔと呼ばれるＳｏｕｒｃｅおよびＴｙｐｅ ＬｏｃａｔｉｏｎのＤｗｅｌｌｉｎｇと呼ばれるＴａｒｇｅｔを持ち、さらに、居住者がその住居に住んでいた期間を示すプロパティＳｔａｒｔＤａｔｅおよびＥｎｄＤａｔｅを持つ。Ｐｅｒｓｏｎは、長い期間にわたっては、複数の住居に住んでいる場合があり、また住居には複数の住人がいる場合もあり、したがってＳｔａｒｔＤａｔｅおよびＥｎｄＤａｔｅ情報を入れる場所として最も可能性の高いのは関係自体であることに注意されたい。
【００５０】
Ｒｅｌａｔｉｏｎｓｈｉｐｓは、終点の型として与えられた型により制約されるインスタンス間のマッピングを定義する。例えば、ＬｉｖｅｓＡｔ関係は、ＡｕｔｏｍｏｂｉｌｅがＯｃｃｕｐａｎｔである関係とはなりえない、というのもＡｕｔｏｍｏｂｉｌｅはＰｅｒｓｏｎではないからである。
【００５１】
データモデルでは、型の間の子型－親型関係の定義を許容する。ＢａｓｅＴｙｐｅ関係とも呼ばれる子型－親型関係は、Ｔｙｐｅ ＡがＴｙｐｅ Ｂに対しＢａｓｅＴｙｐｅならば、Ｂのすべてのインスタンスはインスタンスでもあるという場合でなければならないように定義される。これを表現するもう１つの方法は、Ｂに適合するすべてのインスタンスがさらにＡにも適合しなければならないとするものである。例えば、ＡがＴｙｐｅ Ｓｔｒｉｎｇのプロパティ Ｎａｍｅを持つが、ＢはＴｙｐｅ Ｉｎｔ１６のプロパティＡｇｅを持つ場合、Ｂのインスタンスはどれも、ＮａｍｅとＡｇｅの両方を持たなければならないという結果になる。型階層は、ルートに単一の親型のあるツリーと考えることができる。ルートからの分岐は、第１レベルの子型を定め、このレベルの分岐は、第２レベルの子型を定め、というように、それ自体子型を持たない一番左の子型へ進む。ツリーは、均一な深さとなるように制約されないが、循環を含むことはできない。与えられたＴｙｐｅは、０個またはそれ以上の子型と０個または１個の親型を持つことができる。与えられたインスタンスは、その型の親型とともに高々１つの型に適合することができる。別の言い方をすると、ツリー内の任意のレベルで与えられたインスタンスについて、そのインスタンスはそのレベルで高々１つの子型に適合することができる。型は、型のインスタンスがさらに型の子型のインスタンスでもなければならない場合にＡｂｓｔｒａｃｔであるという。
【００５２】
１．Ｉｔｅｍｓ
Ｉｔｅｍは、単純ファイルとは異なり、ストレージプラットフォームによりエンドユーザまたはアプリケーションプログラムに対し公開されるすべてのオブジェクトにわたって一般的にサポートされるプロパティの基本集合を持つオブジェクトである格納可能情報のユニットである。Ｉｔｅｍｓは、さらに、以下で説明するように、新しいプロパティおよび関係を導入することができる機能を含むすべてのＩｔｅｍ型にわたって一般的にサポートされるプロパティおよび関係も有する。
【００５３】
Ｉｔｅｍｓは、コピー、削除、移動、開く、印刷、バックアップ、リストア、複製などの共通オペレーション用のオブジェクトである。Ｉｔｅｍｓは、格納し、取り出すことのできるユニットであり、ストレージプラットフォームにより操作される格納可能情報のす

べての形態が、Ｉｔｅｍｓ、Ｉｔｅｍｓのプロパティ、またはＩｔｅｍｓ間のＲｅｌａｔｉｏｎｓｈｉｐｓとして存在し、それぞれについては以下で説明される。

【００５４】

Ｉｔｅｍｓは、Ｃｏｎｔａｃｔｓ、Ｐｅｏｐｌｅ、Ｓｅｒｖｉｃｅｓ、Ｌｏｃａｔｉｏｎｓ、Ｄｏｃｕｍｅｎｔｓ（あらゆる種類の）などの現実世界の容易に理解できるデータのユニットを表すことを意図されている。図５Ａは、Ｉｔｅｍの構造を例示するブロック図である。Ｉｔｅｍの未修飾の名前は「Ｌｏｃａｔｉｏｎ」である。Ｉｔｅｍの修飾名は「Ｃｏｒｅ．Ｌｏｃａｔｉｏｎ」であり、これは、このＩｔｅｍ構造がＣｏｒｅ　Ｓｃｈｅｍａ内のＩｔｅｍの特定の型として定義されることを示している。（Ｃｏｒｅ　Ｓｃｈｅｍａは、本明細書の後の方で詳しく説明される。）

【００５５】

Ｌｏｃａｔｉｏｎ　Ｉｔｅｍは、ＥＡｄｄｒｅｓｓｅｓ、ＭｅｔｒｏｐｏｌｉｔａｎＲｅｇｉｏｎ、Ｎｅｉｇｈｂｏｒｈｏｏｄ、およびＰｏｓｔａｌＡｄｄｒｅｓｓｅｓを含む複数のプロパティを持つ。それぞれに対するプロパティの特定の型は、プロパティ名の直後に示され、コロン（「：」）でプロパティ名と区切られる。型名の右には、そのプロパティ型について許された値の個数があり、これは角括弧（「［　］」）の間に示され、コロン（「：」）の右にあるアスタリスク（「＊」）は、未指定および／または無制限の数（「ｍａｎｙ」）を示す。コロンの右にある「１」は、高々１つの値がありえることを示す。コロンの左にあるゼロ（「０」）はプロパティがオプションであることを示す（値がまったくあり得ない）。コロンの左にある「１」は、少なくとも１つの値がなければならないことを示す（プロパティは必須である）。ＮｅｉｇｈｂｏｒｈｏｏｄおよびＭｅｔｒｏｐｏｌｉｔａｎＲｅｇｉｏｎは、両方とも、定義済みのデータ型または「単純型」（また、本明細書では大文字使用でないことにより表される）である、型「ｎｖａｒｃｈａｒ」（または同等の型）である。しかし、ＥＡｄｄｒｅｓｓｅｓおよびＰｏｓｔａｌＡｄｄｒｅｓｓｅｓは、それぞれ型ＥＡｄｄｒｅｓｓおよびＰｏｓｔａｌＡｄｄｒｅｓｓの定義済み型または「複合型」（本明細書では大文字使用で表される）のプロパティである。複合型は、１つまたは複数の単純データ型および／または他の複合型から派生した型である。Ｉｔｅｍのプロパティに対する複合型は、さらに、複合型の詳細はそのプロパティを定義するためにイミディエイトＩｔｅｍ内にネストされるため「ネストされた要素」も構成し、それらの複合型に関係する情報は、（本明細書の後の方で説明するように、Ｉｔｅｍの境界内の）それらのプロパティを持つＩｔｅｍとともに保持される。型付けのこれらの概念は、よく知られており、当業者であれば容易に理解することである。

【００５６】

図５Ｂは、複合プロパティ型ＰｏｓｔａｌＡｄｄｒｅｓｓおよびＥＡｄｄｒｅｓｓを例示するブロック図である。ＰｏｓｔａｌＡｄｄｒｅｓｓプロパティ型は、プロパティ型ＰｏｓｔａｌＡｄｄｒｅｓｓのＩｔｅｍが０または１つのＣｉｔｙ値、０または１つのＣｏｕｎｔｒｙＣｏｄｅ値、０または１つのＭａｉｌＳｔｏｐ値、および任意の数（０から多数）のＰｏｓｔａｌＡｄｄｒｅｓｓＴｙｐｅｓなどを持つことを予想できるものとして定義する。このようにして、Ｉｔｅｍ内の特定のプロパティに対するデータの形状がこれにより定義される。ＥＡｄｄｒｅｓｓプロパティ型も、同様に、示されているとおりに定義される。このＡｐｐｌｉｃａｔｉｏｎでオプションにより使用されるが、Ｌｏｃａｔｉｏｎ　Ｉｔｅｍ内の複合型を表すもう１つの方法は、Ｉｔｅｍをそこにリストされているそれぞれの複合型の個々のプロパティで描画することである。図５Ｃは、複合型がさらに記述されるＬｏｃａｔｉｏｎ　Ｉｔｅｍを例示するブロック図である。しかし、この図５Ｃ内のＬｏｃａｔｉｏｎ　Ｉｔｅｍのこの代替え表現は図５Ａに例示されているのとまったく同じＩｔｅｍに対するものであると理解すべきである。本発明のストレージプラットフォームでは、さらに、サブタイプ化も可能であり、それによって、一方のプロパティ型を他方の子型とすることができる（ただし、一方のプロパティ型は他方の親のプロパティ型のプロパティを継承する）。

【００５７】

プロパティおよびそのプロパティ型と似てはいるが異なる、Ｉｔｅｍｓは、本質的に、サブタイプ化の対象ともなりうる自Ｉｔｅｍ Ｔｙｐｅｓを表す。つまり、本発明のいくつかの実施形態におけるストレージプラットフォームでは、Ｉｔｅｍを他のＩｔｅｍの子型とすることができるということである（それによって、一方のＩｔｅｍは他方の親Ｉｔｅｍのプロパティを継承する）。さらに、本発明の様々な実施形態について、すべてのＩｔｅｍは Ｂａｓｅ Ｓｃｈｅｍａに見られる第１の、基礎的なＩｔｅｍ型である「Ｉｔｅｍ」Ｉｔｅｍ型の子型である。（Ｂａｓｅ Ｓｃｈｅｍａは、さらに、本明細書の後の方で詳しく説明される。）図６Ａは、Ｉｔｅｍ、つまりこのＩｎｓｔａｎｃｅのＬｏｃａｔｉｏｎ ＩｔｅｍをＢａｓｅ ＳｃｈｅｍａにあるＩｔｅｍ Ｉｔｅｍ型の子型であるとして例示している。この図面では、矢印は、Ｌｏｃａｔｉｏｎ Ｉｔｅｍ（他のすべてのＩｔｅｍｓのように）はＩｔｅｍ Ｉｔｅｍ型の子型であることを示している。Ｉｔｅｍ Ｉｔｅｍ型は、他のすべてのＩｔｅｍｓの派生元である基礎のＩｔｅｍとして、ＩｔｅｍＩｄおよび様々なタイムスタンプなどの重要なプロパティを多数持ち、それによって、オペレーティングシステムにおいて、すべてのＩｔｅｍｓの標準プロパティを定義する。この図では、Ｉｔｅｍ Ｉｔｅｍ型のこれらのプロパティは、Ｌｏｃａｔｉｏｎにより継承され、それによって、Ｌｏｃａｔｉｏｎのプロパティになる。

【００５８】

Ｉｔｅｍ Ｉｔｅｍ型から継承されたＬｏｃａｔｉｏｎ Ｉｔｅｍでプロパティを表す他の方法は、そこにリストされている親Ｉｔｅｍから各プロパティ型の個々のプロパティでＬｏｃａｔｉｏｎを描画することである。図６Ｂは、イミディエイトプロパティに加えて継承型が記述されるＬｏｃａｔｉｏｎ Ｉｔｅｍを例示するブロック図である。このＩｔｅｍは図５Ａに例示されているのと同じＩｔｅｍであるが、この図では、Ｌｏｃａｔｉｏｎはそのプロパティのすべてとともに例示されており、両方ともイミディエイトであり－この図と図５Ａの両方に示されている－継承される－この図に示されているが、図５Ａには示されていない－ことに注意し、理解されたい（しかし、図５Ａでは、Ｌｏｃａｔｉｏｎ ＩｔｅｍがＩｔｅｍ Ｉｔｅｍ型の子型であることを矢印で示すことによりそれらのプロパティが参照されている）。

【００５９】

Ｉｔｅｍｓは、スタンドアロンのオブジェクトであり、そのため、Ｉｔｅｍを削除すると、それらのＩｔｅｍｓのイミディエイトおよび継承プロパティもすべて削除される。同様に、Ｉｔｅｍを取り出した場合に受け取る内容は、Ｉｔｅｍおよびそのイミディエイトおよび継承プロパティのすべて（複合プロパティ型に関係する情報を含む）である。本発明のいくつかの実施形態では、特定のＩｔｅｍを取り出す際にプロパティの部分集合を要求することが可能であるが、そのような実施形態の多くの既定として、取り出したときにイミディエイトおよび継承プロパティのすべてをＩｔｅｍに供給する。さらに、Ｉｔｅｍｓのプロパティは、新しいプロパティをそのＩｔｅｍの型の既存のプロパティに追加することにより拡張することもできる。これらの「拡張」は、これ以降、Ｉｔｅｍの本物のプロパティであり、そのＩｔｅｍ型の子型は、自動的に、拡張プロパティを含むことができる。

【００６０】

Ｉｔｅｍの「境界」は、そのプロパティ（複合プロパティ型、拡張などを含む）により表される。Ｉｔｅｍの境界は、さらに、コピー、削除、移動、作成などのＩｔｅｍに対し実行されるオペレーションの限界も表す。例えば、本発明のいくつかの実施形態では、Ｉｔｅｍがコピーされる場合、そのＩｔｅｍの境界内にあるものもすべてコピーされる。Ｉｔｅｍ毎に、境界は以下を包含する。

・ ＩｔｅｍのＩｔｅｍ ＴｙｐｅおよびＩｔｅｍが他のＩｔｅｍの子型の場合（すべてのＩｔｅｍｓがＢａｓｅ Ｓｃｈｅｍａ内の単一ＩｔｅｍおよびＩｔｅｍ Ｔｙｐｅから派生する本発明のいくつかの実施形態の場合のように）は、任意の適用可能な子型情報（つまり、親Ｉｔｅｍ Ｔｙｐｅに関係する情報）。コピー元のオリジナルのＩｔｅｍが他のＩｔｅｍの子型の場合、そのコピーも、その同じＩｔｅｍの子型ともなりうる。

・　もしあればＩｔｅｍの複合型プロパティおよび拡張。オリジナルのＩｔｅｍが複合型のプロパティ（ネイティブまたは拡張）を持つ場合、そのコピーも同じ複合型を持ちうる。

・　「所有関係」に関するＩｔｅｍのレコード、つまり、現在のＩｔｅｍ（「Ｏｗｎｉｎｇ　Ｉｔｅｍ」）によって所有される他のＩｔｅｍｓ（「Ｔａｒｇｅｔ　Ｉｔｅｍｓ」）のＩｔｅｍの自リスト。これは、以下で詳しく説明する、Ｉｔｅｍ　Ｆｏｌｄｅｒｓに関して特に関連性があり、規則では、すべてのＩｔｅｍｓは少なくとも１つのＩｔｅｍ　Ｆｏｌｄｅｒに属していなければならないことを以下で規定している。さらに、埋め込まれているアイテム－以下でさらに詳しく説明する－に関しては、埋め込みアイテムは、コピー、削除などのオペレーションに対して埋め込まれているＩｔｅｍの一部と考えられる。

【００６１】

２．Ｉｔｅｍの識別

Ｉｔｅｍｓは、ＩｔｅｍＩＤを持つ大域的アイテム空間（ｇｌｏｂａｌ　ｉｔｅｍｓ　ｓｐａｃｅ）内で一意に識別される。Ｂａｓｅ．Ｉｔｅｍ型は、ＩｔｅｍのＩＤを格納する型ＧＵＩＤのＩｔｅｍ　ＩＤフィールドを定義する。Ｉｔｅｍは、データストア３０２内にちょうど１つのＩＤがなければならない。

【００６２】

アイテム参照は、Ｉｔｅｍを特定し、識別するための情報を格納するデータ構造体である。データモデルでは、抽象型は、すべてのアイテム参照型の派生元のＩｔｅｍＲｅｆｅｒｅｎｃｅという名前を持つものとして定義される。ＩｔｅｍＲｅｆｅｒｅｎｃｅ型は、Ｒｅｓｏｌｖｅという名前の仮想メソッドを定義する。Ｒｅｓｏｌｖｅメソッドは、ＩｔｅｍＲｅｆｅｒｅｎｃｅを解決して、Ｉｔｅｍを返す。このメソッドは、参照が与えられているＩｔｅｍを取り出す関数を実装するＩｔｅｍＲｅｆｅｒｅｎｃｅの具体的な子型によりオーバーライドされる。Ｒｅｓｏｌｖｅメソッドは、ストレージプラットフォームＡＰＩ　３２２の一部として呼び出される。

【００６３】

ＩｔｅｍＩＤＲｅｆｅｒｅｎｃｅはＩｔｅｍＲｅｆｅｒｅｎｃｅの子型である。これは、ＬｏｃａｔｏｒおよびＩｔｅｍＩＤフィールドを定義する。Ｌｏｃａｔｏｒフィールドではアイテム領域に名前を付ける（つまり、識別する）。これは、Ｌｏｃａｔｏｒの値をアイテム領域に解決できるロケータ解決メソッドにより処理される。ＩｔｅｍＩＤフィールドは型ＩｔｅｍＩＤである。

【００６４】

ＩｔｅｍＰａｔｈＲｅｆｅｒｅｎｃｅは、ＬｏｃａｔｏｒおよびＰａｔｈフィールドを定義するＩｔｅｍＲｅｆｅｒｅｎｃｅの特殊化である。Ｌｏｃａｔｏｒフィールドは、アイテム領域を識別する。これは、Ｌｏｃａｔｏｒの値をアイテム領域に解決できるロケータ解決メソッドにより処理される。Ｐａｔｈフィールドは、Ｌｏｃａｔｏｒにより与えられるアイテム領域をルートとするストレージプラットフォーム名前空間内の（相対）パスを含む。

【００６５】

この型の参照は、ｓｅｔオペレーションで使用することはできない。参照は、一般に、パス解決プロセスを通じて解決されなければならない。ストレージプラットフォームＡＰＩ　３２２のＲｅｓｏｌｖｅメソッドは、この機能を備える。

【００６６】

上述の参照形態は、図１１に例示されている参照型階層を通じて表される。これらの型から継承する追加参照型は、スキーマで定義することができる。それらは、ターゲットフィールドの型として関係宣言の中で使用することができる。

【００６７】

３．Ｉｔｅｍ　ＦｏｌｄｅｒｓおよびＣａｔｅｇｏｒｉｅｓ

以下で詳しく説明するが、Ｉｔｅｍｓのグループは、Ｉｔｅｍ　Ｆｏｌｄｅｒｓ（ファ

イルフォルダと混同すべきでない）と呼ばれる特別なＩｔｅｍｓに編成されることができる。しかし、ほとんどのファイルシステムとは異なり、Ｉｔｅｍは、複数のＩｔｅｍ Ｆｏｌｄｅｒに属すことができ、そのため、Ｉｔｅｍが一方のＩｔｅｍ Ｆｏｌｄｅｒでアクセスされ改訂された場合、この改訂されたＩｔｅｍは、他方のＩｔｅｍフォルダから直接アクセスすることができる。本質的に、Ｉｔｅｍへのアクセスは異なるＩｔｅｍ Ｆｏｌｄｅｒｓから発生しうるが、実際にアクセスされる内容は、事実、まったく同じＩｔｅｍである。しかし、Ｉｔｅｍ Ｆｏｌｄｅｒは、必ずしも、そのメンバＩｔｅｍｓのすべてを所有するわけではなく、または単に、他のフォルダとともにＩｔｅｍｓを共同所有することができ、Ｉｔｅｍ Ｆｏｌｄｅｒを削除しても、その結果Ｉｔｅｍの削除も必ずしも行われるわけではない。しかしながら、本発明のいくつかの実施形態では、Ｉｔｅｍは、少なくとも１つのＩｔｅｍ Ｆｏｌｄｅｒに属していなければならず、したがって、特定のＩｔｅｍに対する単独のＩｔｅｍ Ｆｏｌｄｅｒが削除されると、ある実施形態では、Ｉｔｅｍは自動的に削除されるか、または他の実施形態では、Ｉｔｅｍは自動的に、既定のＩｔｅｍ Ｆｏｌｄｅｒ（例えば、様々なファイル－フォルダベースのシステムで使用される類似名フォルダと概念上類似の「Ｔｒａｓｈ Ｃａｎ」Ｉｔｅｍ Ｆｏｌｄｅｒ）のメンバとなる。

【００６８】

また以下で詳述するが、Ｉｔｅｍｓは、さらに、（ａ）Ｉｔｅｍ Ｔｙｐｅ（またはＴｙｐｅｓ）、（ｂ）特定のイミディエイトまたは継承プロパティ（または複数のプロパティ）、または（ｃ）Ｉｔｅｍプロパティに対応する特定の値（または複数の値）などの共通の記述されている特性に基づいてＣａｔｅｇｏｒｉｅｓに属している場合がある。例えば、個人連絡情報の特定のプロパティを含むＩｔｅｍは、自動的にＣｏｎｔａｃｔ Ｃａｔｅｇｏｒｙに属すことがあり、そうすれば連絡先情報プロパティを含むＩｔｅｍも同様に、自動的にこのＣａｔｅｇｏｒｙに属するのである。同様に、値「Ｎｅｗ Ｙｏｒｋ Ｃｉｔｙ」を持つ場所プロパティが設定されているＩｔｅｍであれば、自動的にＮｅｗＹｏｒｋＣｉｔｙ Ｃａｔｅｇｏｒｙに属すことが可能である。

【００６９】

Ｃａｔｅｇｏｒｉｅｓは、Ｉｔｅｍ Ｆｏｌｄｅｒｓは相互関連性のない（つまり、共通の記述された特性なしの）Ｉｔｅｍｓを含むことができるが、Ｃａｔｅｇｏｒｙのそれぞれのは、そのＣａｔｅｇｏｒｙについて記述された共通の型、プロパティ、または値（「共通性」）を持ち、Ｃａｔｅｇｏｒｙ内の他のＩｔｅｍｓ同士の間の関係の基礎をなすのがこの共通性であるという点で、Ｉｔｅｍ Ｆｏｌｄｅｒｓと概念上異なる。さらに、Ｉｔｅｍの特定のＦｏｌｄｅｒへの帰属関係はそのＩｔｅｍの特定の態様に基づいて強制的ではないが、いくつかの実施形態では、共通性がカテゴリについてＣａｔｅｇｏｒｙに関係しているすべてのＩｔｅｍｓは、ハードウェア／ソフトウェアインターフェースシステムレベルでＣａｔｅｇｏｒｙのメンバに自動的になることが可能である。概念上、Ｃａｔｅｇｏｒｉｅｓは、帰属関係が特定のクエリの結果に基づく（データベースを背景とした場合のように）仮想Ｉｔｅｍ Ｆｏｌｄｅｒｓと考えることもでき、また（Ｃａｔｅｇｏｒｙの共通性により定義された）このクエリの条件を満たすＩｔｅｍｓはＣａｔｅｇｏｒｙの帰属関係を含むであろう。

【００７０】

図４は、Ｉｔｅｍｓ、Ｉｔｅｍ Ｆｏｌｄｅｒｓ、およびＣａｔｅｇｏｒｉｅｓ間の構造的関係を例示する。複数のＩｔｅｍｓ ４０２、４０４、４０６、４０８、４１０、４１２、４１４、４１６、４１８、および４２０は、様々なＩｔｅｍ Ｆｏｌｄｅｒｓ ４２２、４２４、４２６、４２８、および４３０のメンバである。複数のＩｔｅｍ Ｆｏｌｄｅｒに属すＩｔｅｍｓもあり、例えば、Ｉｔｅｍ ４０２はＩｔｅｍ Ｆｏｌｄｅｒｓ ４２２および４２４に属している。いくつかのＩｔｅｍｓ、例えば、Ｉｔｅｍ ４０２、４０４、４０６、４０８、４１０、および４１２は、１つまたは複数のＣａｔｅｇｏｒｉｅｓ ４３２、４３４、および４３６のメンバでもあるが、別のときには、例えば、Ｉｔｅｍｓ ４１４、４１６、４１８、および４２０はどのＣａｔｅｇｏｒｉｅｓにも属し

えない（ただし、これはプロパティの所有が自動的にＣａｔｅｇｏｒｙへの帰属を意味するいくつかの実施形態ではほとんどありえず、Ｉｔｅｍは、そのような実施形態におけるカテゴリのメンバとならないためには完全に無特徴でなければならないであろう）。フォルダの階層構造と対照的に、ＣａｔｅｇｏｒｉｅｓとＩｔｅｍ Ｆｏｌｄｅｒｓは両方とも、図に示されているように有向グラフにより類似している。いずれにせよ、Ｉｔｅｍｓ、Ｉｔｅｍ Ｆｏｌｄｅｒｓ、およびＣａｔｅｇｏｒｉｅｓは、（異なるＩｔｅｍ Ｔｙｐｅｓであるにもかかわらず）すべてＩｔｅｍｓである。

【００７１】

ファイル、フォルダ、およびディレクトリとは対照的に、本発明のＩｔｅｍｓ、Ｉｔｅｍ Ｆｏｌｄｅｒｓ、およびＣａｔｅｇｏｒｉｅｓについては、物理的コンテナに相当する概念がなく、したがって、Ｉｔｅｍｓはそのような複数の場所に存在する可能性があるため、本質的に「物理的」という特徴を持たない。Ｉｔｅｍｓが複数のＩｔｅｍ Ｆｏｌｄｅｒロケーションに存在できることとともに、Ｃａｔｅｇｏｒｉｅｓに編成されることにより、当業で現在利用できるレベルを超える、ハードウェア／ソフトウェアインターフェースレベルで、データ操作およびストレージ構造の機能が高められ、豊富になっている。

【００７２】

４．Ｓｃｈｅｍａｓ

ａ）Ｂａｓｅ Ｓｃｈｅｍａ

Ｉｔｅｍｓの作成および使用に普遍的基盤を用意するため、本発明のストレージプラットフォームの様々な実施形態は、ＩｔｅｍｓおよびプロパティのＢａｓｅ Ｓｃｈｅｍａを備える。Ｂａｓｅ Ｓｃｈｅｍａは、Ｉｔｅｍｓおよびプロパティのいくつかの特殊な型、および子型をさらに派生することができる派生元のこれらの特別な基礎型の特徴を定義する。このＢａｓｅ Ｓｃｈｅｍａを使用することにより、プログラマは、Ｉｔｅｍｓ（およびそれぞれの型）をプロパティ（およびそれぞれの型）から概念的に区別することができる。さらに、Ｂａｓｅ Ｓｃｈｅｍａでは、すべてのＩｔｅｍｓ（およびその対応するＩｔｅｍ Ｔｙｐｅｓ）がＢａｓｅ Ｓｃｈｅｍａ内のこの基礎的Ｉｔｅｍ（およびその対応するＩｔｅｍ Ｔｙｐｅ）から派生するときにすべてのＩｔｅｍｓが所有することができるプロパティの基礎的集合を規定する。

【００７３】

図７に例示されているように、また本発明にいくつかの実施形態に関して、Ｂａｓｅ Ｓｃｈｅｍａは、Ｉｔｅｍ、Ｅｘｔｅｎｓｉｏｎ、およびＰｒｏｐｅｒｔｙＢａｓｅという３つの最上位タイプを定義する。図に示されているように、Ｉｔｅｍ型は、この基礎的な「Ｉｔｅｍ」Ｉｔｅｍ型のプロパティにより定義される。対照的に、最上位レベルのプロパティ型「ＰｒｏｐｅｒｔｙＢａｓｅ」は、定義済みプロパティを持たず、他のすべてのプロパティ型の派生元となる、すべての派生プロパティ型が相互に関係付けられる際の単なるアンカーにすぎない（単一のプロパティ型からふつうは派生する）。Ｅｘｔｅｎｓｉｏｎ型プロパティは、Ｉｔｅｍが複数の拡張を持つ可能性があるときに、拡張でどのＩｔｅｍが拡張されるかとともに、一方の拡張を他方の拡張から区別するための識別を定義する。

【００７４】

ＩｔｅｍＦｏｌｄｅｒは、Ｉｔｅｍから継承されたプロパティに加えて、メンバ（もしあれば）へのリンクを確立するＲｅｌａｔｉｏｎｓｈｉｐを特徴とする、Ｉｔｅｍ Ｉｔｅｍ型の子型であるが、ＩｄｅｎｔｉｔｙＫｅｙとＰｒｏｐｅｒｔｙは両方ともＰｒｏｐｅｒｔｙＢａｓｅの子型である。すると、ＣａｔｅｇｏｒｙＲｅｆは、ＩｄｅｎｔｉｔｙＫｅｙの子型である。

【００７５】

ｂ）Ｃｏｒｅ Ｓｃｈｅｍａ

本発明のストレージプラットフォームの様々な実施形態は、さらに、最上位レベルのＩ

ｔｅｍｓ型構造の概念フレームワークを提供するＣｏｒｅ　Ｓｃｈｅｍａを含む。図８Ａは、Ｃｏｒｅ　Ｓｃｈｅｍａ内のＩｔｅｍｓを例示するブロック図であり、図８Ｂは、Ｃｏｒｅ　Ｓｃｈｅｍａ内のプロパティ型を例示するブロック図である。異なる拡張子（＊．ｃｏｍ、＊．ｅｘｅ、＊．ｂａｔ、＊．ｓｙｓなど）を持つファイルとファイルーフォルダベースのシステムのそのような他の基準を持つファイルとの区別は、Ｃｏｒｅ　Ｓｃｈｅｍａの関数に類似している。Ｃｏｒｅ　Ｓｃｈｅｍａが、Ｉｔｅｍベースのハードウェア／ソフトウェアインターフェースシステムでは、直接的に（Ｉｔｅｍ型により）または間接的に（Ｉｔｅｍ子型により）、すべてのＩｔｅｍｓを、Ｉｔｅｍベースのハードウェア／ソフトウェアインターフェースシステムが理解し、所定の予測可能な方法で処理できる１つまたは複数のＣｏｒｅ　Ｓｃｈｅｍａ　Ｉｔｅｍ型に特徴付ける、コアＩｔｅｍ型の集合を定義する。定義済みＩｔｅｍ型は、Ｉｔｅｍベースのハードウェア／ソフトウェアインターフェースシステム内の最も一般的なＩｔｅｍｓを反映し、したがって、一定レベルの効率が、Ｃｏｒｅ　Ｓｃｈｅｍａを含む定義済みＩｔｅｍ型を認識するＩｔｅｍベースのハードウェア／ソフトウェアインターフェースシステムにより得られる。

【００７６】

いくつかの実施形態では、Ｃｏｒｅ　Ｓｃｈｅｍａは拡張可能でない－つまり、Ｃｏｒｅ　Ｓｃｈｅｍａの一部である特定の定義済み派生Ｉｔｅｍ型を除き、追加Ｉｔｅｍ型を直接、Ｂａｓｅ　ＳｃｈｅｍａのＩｔｅｍ型からサブタイプ化することはできない。後のすべてのＩｔｅｍ型が必ずＣｏｒｅ　Ｓｃｈｅｍａ　Ｉｔｅｍ型の子型であるため、Ｃｏｒｅ　Ｓｃｈｅｍａへの拡張を禁止することにより（つまり、新しいＩｔｅｍｓをＣｏｒｅ　Ｓｃｈｅｍａに追加することを禁止することにより）、ストレージプラットフォームはＣｏｒｅ　Ｓｃｈｅｍａ　Ｉｔｅｍ型の使用を指示する。この構造により、コアＩｔｅｍ型の定義済み集合を持つ利点も維持しながら、追加Ｉｔｅｍ型を定義する自由度も十分に高められる。

【００７７】

本発明の様々な実施形態について、図８Ａを参照すると、Ｃｏｒｅ　Ｓｃｈｅｍａによってサポートされている特定のＩｔｅｍ型は以下のうちの１つまたは複数を含むことができる。

・　Ｃａｔｅｇｏｒｉｅｓ：このＩｔｅｍ　Ｔｙｐｅ（およびそれから派生した子型）のＩｔｅｍｓは、Ｉｔｅｍベースのハードウェア／ソフトウェアインターフェースシステムの有効なＣａｔｅｇｏｒｉｅｓを表す。

・　Ｃｏｍｍｏｄｉｔｉｅｓ：価値ある識別可能なものであるＩｔｅｍｓ。

・　Ｄｅｖｉｃｅｓ：情報処理機能をサポートする論理構造を備えるＩｔｅｍｓ。

・　Ｄｏｃｕｍｅｎｔｓ：Ｉｔｅｍベースのハードウェア／ソフトウェアインターフェースシステムにより解釈されないが、代わりにドキュメント型に対応するアプリケーションプログラムにより解釈されるコンテンツを持つＩｔｅｍｓ。

・　Ｅｖｅｎｔｓ：環境内で生じた何らかの出来事を記録するＩｔｅｍｓ。

・　Ｌｏｃａｔｉｏｎｓ：物理的場所を表すＩｔｅｍｓ（例えば、地理的位置）。

・　Ｍｅｓｓａｇｅｓ：２つ以上のプリンシパルの間の通信のＩｔｅｍｓ（以下に定義する）。

・　Ｐｒｉｎｃｉｐａｌｓ：ＩｔｅｍＩｄだけでなく少なくとも１つの決定的に証明可能なＩＤを持つＩｔｅｍｓ（例えば、人、組織、グループ、世帯、機関、サービスなどの識別）。

・　Ｓｔａｔｅｍｅｎｔｓ：限定はしないが、ポリシー、サブスクリプション、信用などを含む環境に関する特別な情報を持つＩｔｅｍｓ。

【００７８】

同様に、図８Ｂを参照すると、Ｃｏｒｅ　Ｓｃｈｅｍａによってサポートされている特定のプロパティ型は以下のうちの１つまたは複数を含むことができる。

・　Ｃｅｒｔｉｆｉｃａｔｅｓ（Ｂａｓｅ　Ｓｃｈｅｍａ内の基礎のＰｒｏｐｅｒｔｙＢａｓｅ型から派生）

・ Ｐｒｉｎｃｉｐａｌ Ｉｄｅｎｔｉｔｙ Ｋｅｙｓ（Ｂａｓｅ Ｓｃｈｅｍａ内のＩｄｅｎｔｉｔｙＫｅｙ型から派生）
・ Ｐｏｓｔａｌ Ａｄｄｒｅｓｓ（Ｂａｓｅ Ｓｃｈｅｍａ内のＰｒｏｐｅｒｔｙ型から派生）
・ Ｒｉｃｈ Ｔｅｘｔ（Ｂａｓｅ Ｓｃｈｅｍａ内のＰｒｏｐｅｒｔｙ型から派生）
・ ＥＡｄｄｒｅｓｓ（Ｂａｓｅ Ｓｃｈｅｍａ内のＰｒｏｐｅｒｔｙ型から派生）
・ ＩｄｅｎｔｉｔｙＳｅｃｕｒｉｔｙＰａｃｋａｇｅ（Ｂａｓｅ Ｓｃｈｅｍａ内のＲｅｌａｔｉｏｎｓｈｉｐ型から派生）
・ ＲｏｌｅＯｃｃｕｐａｎｃｙ（Ｂａｓｅ Ｓｃｈｅｍａ内のＲｅｌａｔｉｏｎｓｈｉｐ型から派生）
・ ＢａｓｉｃＰｒｅｓｅｎｃｅ（Ｂａｓｅ Ｓｃｈｅｍａ内のＲｅｌａｔｉｏｎｓｈｉｐ型から派生）

【００７９】
これらのＩｔｅｍｓおよびＰｒｏｐｅｒｔｉｅｓは、さらに、図８Ａおよび図８Ｂで述べられているそれぞれのプロパティにより記述される。

【００８０】
５．関係
Ｒｅｌａｔｉｏｎｓｈｉｐｓは、一方のＩｔｅｍがソースとして指定され、他方のＩｔｅｍがターゲットとして指定されている２項関係である。ソースＩｔｅｍおよびターゲットＩｔｅｍは、この関係によって関連付けられる。ソースＩｔｅｍは、一般に、関係の存続期間を制御する。つまり、ソースＩｔｅｍが削除される、それらのＩｔｅｍ間の関係も削除されるということである。

【００８１】
Ｒｅｌａｔｉｏｎｓｈｉｐｓは、包含関係Ｃｏｎｔａｉｎｍｅｎｔと参照関係Ｒｅｆｅｒｅｎｃｅに分類される。包含関係は、ターゲットＩｔｅｍｓの存続期間を制御するが、参照関係は、存続期間管理の意味を持たない。図１２は、関係が分類される仕方を例示する。

【００８２】
Ｃｏｎｔａｉｎｍｅｎｔ関係型は、さらに、保持関係Ｈｏｌｄｉｎｇと埋め込み関係Ｅｍｂｅｄｄｉｎｇに分類される。Ｉｔｅｍに対するすべての保持関係が削除されると、そのＩｔｅｍも削除される。保持関係は、参照カウントメカニズムを使ってターゲットの存続期間を制御する。埋め込み関係を使用すると、複合Ｉｔｅｍｓのモデル化が可能になり、これは排他的保持関係と考えることができる。Ｉｔｅｍは、１つまたは複数の保持関係のターゲットとすることができるが、Ｉｔｅｍは、ちょうど１つの埋め込み関係のターゲットとすることができる。埋め込み関係のターゲットであるＩｔｅｍは、他の保持または埋め込み関係のターゲットとすることはできない。

【００８３】
参照関係は、ターゲットＩｔｅｍの存続期間を制御しない。これらは、懸垂になっていることがある－ターゲットＩｔｅｍが存在しない場合がある。参照関係は、大域的Ｉｔｅｍ名前空間（つまり、リモートデータストアを含む）内のどこかにあるＩｔｅｍｓへの参照をモデル化するために使用できる。

【００８４】
Ｉｔｅｍをフェッチしても、その関係を自動的にフェッチしない。アプリケーション側で、Ｉｔｅｍの関係を明示的に要求しなければならない。さらに、関係を修正しても、ソースまたはターゲットＩｔｅｍは修正されず、同様に、関係を追加しても、ソース／ターゲットＩｔｅｍに影響は及ばない。

【００８５】
ａ）関係の宣言
明示的関係型は、以下の要素により定義される。
・ 関係名は、Ｎａｍｅ属性で指定される。

・　関係型として、Ｈｏｌｄｉｎｇ、Ｅｍｂｅｄｄｉｎｇ、Ｒｅｆｅｒｅｎｃｅのうちの１つ。これは、Ｔｙｐｅ属性で指定される。
・　ソースおよびターゲット終点。それぞれの終点で、参照されているＩｔｅｍの名前および型を指定する。
・　ソース終点フィールドは、一般的に、型ＩｔｅｍＩＤ（宣言されない）であり、関係インスタンスと同じデータストア内のＩｔｅｍを参照しなければならない。
・　ＨｏｌｄｉｎｇおよびＥｍｂｅｄｄｉｎｇ関係については、ターゲット終点フィールドは、型ＩｔｅｍＩＤＲｅｆｅｒｅｎｃｅでなければならず、関係インスタンスと同じストア内のＩｔｅｍを参照しなければならない。Ｒｅｆｅｒｅｎｃｅ関係については、ターゲット終点は任意のＩｔｅｍＲｅｆｅｒｅｎｃｅ型でよく、他のストレージプラットフォームのデータストア内のＩｔｅｍｓを参照することができる。
・　オプションにより、スカラーまたはＰｒｏｐｅｒｔｙＢａｓｅ型の１つまたは複数のフィールドを宣言することができる。これらのフィールドは、関係に関連付けられたデータを格納することができる。
・　関係インスタンスは、大域的関係テーブルに格納される。
・　すべての関係インスタンスは、組合せ（ソースＩｔｅｍＩＤ、関係ＩＤ）により一意に識別される。関係ＩＤは、その型に関係なく与えられたＩｔｅｍをソースとするすべての関係について、与えられたソースＩｔｅｍＩＤ内で一意である。

【００８６】

ソースＩｔｅｍは、関係の所有者である。所有者として指定されているＩｔｅｍは関係の存続期間を制御するが、関係自体はそれが関係するＩｔｅｍｓとは別である。ストレージプラットフォームＡＰＩ　３２２は、Ｉｔｅｍと関連する関係を公開するメカニズムを提供する。

【００８７】

関係宣言の一実施例を以下に示す。

【００８８】

【表１】

```
<Relationship Name="Employment" BaseType="Reference" >
        <Source Name="Employee" ItemType="Contact.Person"/>
        <Target Name="Employer" ItemType="Contact.Organization"
          ReferenceType="ItemIDReference" />
        <Property Name="StartDate" Type="the storage
        platformTypes.DateTime" />
        <Property Name="EndDate" Type="the storage
        platformTypes.DateTime" />
        <Property Name="Office" Type="the storage
        platformTypes.DateTime" />
</Relationship>
```

【００８９】

これは、Ｒｅｆｅｒｅｎｃｅ関係の一実施例である。この関係は、ソース参照によって参照された人のＩｔｅｍが存在しない場合には作成されることはできない。さらに、人のＩｔｅｍが削除された場合、人と組織との間の関係インスタンスが削除される。しかし、Ｏｒｇａｎｉｚａｔｉｏｎ　Ｉｔｅｍが削除された場合、関係は削除されず、懸垂になる。

【００９０】

ｂ）保持関係

保持関係は、ターゲットＩｔｅｍの参照カウントベースの存続期間管理をモデル化するために使用される。

【００９１】

Ｉｔｅｍは、Ｉｔｅｍｓとの０またはそれ以上の関係に対するソース終点とすることが

できる。埋め込まれたＩｔｅｍではないＩｔｅｍは、１つまたは複数の保持関係におけるターゲットとすることができる。

【００９２】

ターゲット終点参照型は、ＩｔｅｍＩＤＲｅｆｅｒｅｎｃｅでなければならず、また関係インスタンスと同じストア内のＩｔｅｍを参照しなければならない。

【００９３】

保持関係は、ターゲット終点の存続期間管理を強制する。保持関係インスタンスおよびそれがターゲットとしているＩｔｅｍの作成は、原子動作である。同じＩｔｅｍをターゲットとしている追加保持関係インスタンスの作成が可能である。与えられたＩｔｅｍをターゲット終点として持つ最後の保持関係インスタンスが削除されると、ターゲットＩｔｅｍも削除される。

【００９４】

関係宣言で指定されている終点Ｉｔｅｍの型は、一般的に、関係のインスタンスが作成されるときに強制される。関係が確立された後では、終点Ｉｔｅｍｓの型は変更できない。

【００９５】

保持関係は、Ｉｔｅｍ名前空間を形成する際に重要な役割を果たす。これらは、ソースＩｔｅｍに相対的にターゲットＩｔｅｍの名前を定義する「Ｎａｍｅ」プロパティを含む。この相対名は、与えられたＩｔｅｍから発したすべての保持関係について一意である。ルートＩｔｅｍから始まり与えられたＩｔｅｍに至るこの相対名の順序付けリストは、Ｉｔｅｍに対する完全な名前を形成する。

【００９６】

保持関係は、非循環有向グラフ（ＤＡＧ）を形成する。保持関係が作成されると、システムは、サイクルが作成されないことを保証し、したがって、Ｉｔｅｍ名前空間がＤＡＧを必ず形成する。

【００９７】

保持関係はターゲットＩｔｅｍの存続期間を制御するが、ターゲット終点Ｉｔｅｍの動作整合性を制御しない。ターゲットＩｔｅｍは、保持関係を通じてそれを所有するＩｔｅｍから動作に関して独立している。保持関係のソースであるＩｔｅｍに対するＣｏｐｙ、Ｍｏｖｅ、Ｂａｃｋｕｐ、およびその他のオペレーションは、同じ関係のターゲットであるＩｔｅｍに影響を及ぼさない－例えば、つまり、Ｆｏｌｄｅｒ　Ｉｔｅｍをバックアップしても、すべてのＩｔｅｍｓをフォルダ（ＦｏｌｄｅｒＭｅｍｂｅｒ関係のターゲット）内に自動的にバックアップするわけではない。

【００９８】

以下は、保持関係の一実施例である。

【００９９】

【表２】

```
<Relationship Name="FolderMembers" BaseType="Holding" >
        <Source Name="Folder" ItemType="Base.Folder"/>
        <Target Name="Item" ItemType="Base.Item"
          ReferenceType="ItemIDReference" />
</Relationship>
```

【０１００】

ＦｏｌｄｅｒＭｅｍｂｅｒｓ関係は、Ｆｏｌｄｅｒの概念をＩｔｅｍｓのジェネリックコレクションとして使用可能にする。

【０１０１】

ｃ）埋め込み関係

埋め込み関係は、ターゲットＩｔｅｍの存続期間の排他的制御という概念をモデル化する。これらは、複合Ｉｔｅｍｓの概念を有効にする。

【０１０２】
埋め込み関係インスタンスおよびそれがターゲットとしているＩｔｅｍの作成は、原子動作である。Ｉｔｅｍは、０またはそれ以上の埋め込み関係のソースとすることができる。しかし、Ｉｔｅｍは、唯一の埋め込み関係のターゲットとすることができる。埋め込み関係のターゲットであるＩｔｅｍは、保持関係のターゲットとすることはできない。
【０１０３】
ターゲット終点参照型は、ＩｔｅｍＩＤＲｅｆｅｒｅｎｃｅでなければならず、また関係インスタンスと同じデータストア内のＩｔｅｍを参照しなければならない。
【０１０４】
関係宣言で指定されている終点Ｉｔｅｍの型は、一般的に、関係のインスタンスが作成されるときに強制される。関係が確立された後では、終点Ｉｔｅｍｓの型は変更できない。
【０１０５】
埋め込み関係は、ターゲット終点の動作整合性を制御する。例えば、Ｉｔｅｍをシリアライズするオペレーションは、そのＩｔｅｍから発するすべての埋め込み関係だけでなくそのターゲットのすべてのシリアライズを含むことができ、Ｉｔｅｍをコピーする、そのすべての埋め込まれているＩｔｅｍｓもコピーする。
【０１０６】
以下は、宣言例である。
【０１０７】
【表３】

```
<Relationship Name="ArchiveMembers" BaseType="Embedding" >
        <Source Name="Archive" ItemType="Zip.Archive" />
        <Target Name="Member" ItemType="Base.Item "
          ReferenceType="ItemIDReference" />
        <Property Name="ZipSize" Type="the storage
        platformTypes.bigint" />
        <Property Name="SizeReduction" Type="the storage
        platformTypes.float" />
</Relationship>
```

【０１０８】
ｄ）参照関係
参照関係は、参照するＩｔｅｍの存続期間を制御しない。さらには、参照関係は、ターゲットの存在を保証せず、また関係宣言で指定された通りにターゲットの型も保証しない。これは、参照関係が懸垂になる可能性のあることを意味している。また、参照関係は、他のデータストア内のＩｔｅｍｓを参照することができる。参照関係は、Ｗｅｂページ内のリンクに類似の概念として考えることができる。
【０１０９】
参照関係宣言の一例を以下に示す。
【０１１０】
【表４】

```
<Relationship Name="DocumentAuthor" BaseType="Reference" >
        <Sourc  ItemType="Document"
        ItemType="Base.Document"/>
        <Target ItemType="Author" ItemType="Base.Author"
          ReferenceType="ItemIDReference" />
        <Property Type="Role" Type="Core.CategoryRef" />
        <Property Type="DisplayName" Type="the storage
        platformTypes.nvarchar(256)" />
</Relationship>
```

【０１１１】
参照型は、ターゲット終点において使用できる。参照関係に関わるＩｔｅｍｓは、任意のＩｔｅｍ型とすることができる。
【０１１２】
参照関係は、Ｉｔｅｍｓ間のほとんどの非存続期間管理関係をモデル化するために使用される。ターゲットの存在は強制されないなめ、参照関係は、疎結合関係をモデル化するために都合がよい。参照関係は、他のコンピュータ上のストアを含む他のデータストア内のＩｔｅｍｓをターゲットとするために使用することができる。
【０１１３】
ｅ）規則と制約条件
以下の追加規則および制約条件を関係について適用する。
・　Ｉｔｅｍは、（ちょうど１つの埋め込み関係）または（１つまたは複数の保持関係）のターゲットでなければならない。例外の１つは、ルートＩｔｅｍである。Ｉｔｅｍは、０またはそれ以上の参照関係のターゲットとすることができる。
・　埋め込み関係のターゲットであるＩｔｅｍは、保持関係のソースとすることはできない。これは、参照関係のソースとすることができる。
・　Ｉｔｅｍは、ファイルから格上げされた場合保持関係のソースとすることはできない。これは、埋め込み関係および参照関係のソースとすることができる。
・　ファイルから格上げされたＩｔｅｍは、埋め込み関係のターゲットとすることはできない。
【０１１４】
ｆ）関係の順序付け
少なくとも一実施形態では、本発明のストレージプラットフォームは、関係の順序付けをサポートする。順序付けは、基本リレーショナル定義の中で「Ｏｒｄｅｒ」という名前のプロパティを通じて実行される。Ｏｒｄｅｒフィールド上には一意性制約条件はない。同じ「順序」プロパティ値を持つ関係の順序は保証されないが、低い「順序」の値を持つ関係の後、および高い「順序」フィールド値を持つ関係の前に、順序付けできることが保証される。
【０１１５】
アプリケーションでは、組合せ（ＳｏｕｒｃｅＩｔｅｍＩＤ、ＲｅｌａｔｉｏｎｓｈｉｐＩＤ、Ｏｒｄｅｒ）の順序付けにより既定の順序で関係を取得することができる。与えられたＩｔｅｍから発するすべての関係インスタンスは、単一コレクションとして、そのコレクション内の関係の型と関係なく、順序付けされる。しかし、これは、与えられた型（例えば、ＦｏｌｄｅｒＭｅｍｂｅｒｓ）のすべての関係が、与えられたＩｔｅｍに対する関係コレクションの順序付き部分集合であることを保証する。
【０１１６】
関係を操作するデータストアＡＰＩ　３１２は、関係の順序付けをサポートするオペレーションの集合を実装している。以下の用語は、オペレーションを説明する補助として導入される。
・　ＲｅｌＦｉｒｓｔは、順序値ＯｒｄＦｉｒｓｔを持つ順序付きコレクション内の最初の関係である。
・　ＲｅｌＬａｓｔは、順序値ＯｒｄＬａｓｔを持つ順序付きコレクション内の最後の関係である。
・　ＲｅｌＸは、順序値ＯｒｄＸを持つコレクション内の与えられた関係である。
・　ＲｅｌＰｒｅｖは、順序値ＯｒｄＰｒｅｖがＯｒｄＸよりも小さい、コレクション内のＲｅｌＸとの最も近い関係である。
・　ＲｅｌＮｅｘｔは、順序値ＯｒｄＮｅｘｔがＯｒｄＸよりも大きい、コレクション内のＲｅｌＸとの最も近い関係である。
【０１１７】
これらのオペレーションは、限定はしないが、以下のものを含む。

・ ＩｎｓｅｒｔＢｅｆｏｒｅＦｉｒｓｔ（ＳｏｕｒｃｅＩｔｅｍＩＤ，Ｒｅｌａｔｉｏｎｓｈｉｐ）は、関係を第１の関係としてコレクション内に挿入する。新しい関係の「Ｏｒｄｅｒ」プロパティの値はＯｒｄＦｉｒｓｔよりも小さい場合がある。

・ ＩｎｓｅｒｔＡｆｔｅｒＬａｓｔ（ＳｏｕｒｃｅＩｔｅｍＩＤ，Ｒｅｌａｔｉｏｎｓｈｉｐ）は、関係を最後の関係としてコレクション内に挿入する。新しい関係の「Ｏｒｄｅｒ」プロパティの値はＯｒｄＬａｓｔよりも大きい場合がある。

・ ＩｎｓｅｒｔＡｔ（ＳｏｕｒｃｅＩｔｅｍＩＤ，ｏｒｄ，Ｒｅｌａｔｉｏｎｓｈｉｐ）は、「Ｏｒｄｅｒ」プロパティに対する指定された値を持つ関係を挿入する。

・ ＩｎｓｅｒｔＢｅｆｏｒｅ（ＳｏｕｒｃｅＩｔｅｍＩＤ，ｏｒｄ，Ｒｅｌａｔｉｏｎｓｈｉｐ）は、与えられた順序値を持つ関係の前に関係を挿入する。新しい関係には、ＯｒｄＰｒｅｖとｏｒｄの間の、しかもそれを含まない、「Ｏｒｄｅｒ」値を割り当てることができる。

・ ＩｎｓｅｒｔＡｆｔｅｒ（ＳｏｕｒｃｅＩｔｅｍＩＤ，ｏｒｄ，Ｒｅｌａｔｉｏｎｓｈｉｐ）は、与えられた順序値を持つ関係の後に関係を挿入する。新しい関係には、ｏｒｄとＯｒｄＮｅｘｔの間の、しかもそれを含まない、「Ｏｒｄｅｒ」値を割り当てることができる。

・ ＭｏｖｅＢｅｆｏｒｅ（ＳｏｕｒｃｅＩｔｅｍＩＤ，ｏｒｄ，ＲｅｌａｔｉｏｎｓｈｉｐＩＤ）は、与えられた関係ＩＤを持つ関係を指定された「Ｏｒｄｅｒ」値を持つ関係の前に移動する。この関係には、ＯｒｄＰｒｅｖとｏｒｄの間の、しかもそれを含まない、新しい「Ｏｒｄｅｒ」値を割り当てることができる。

・ ＭｏｖｅＡｆｔｅｒ（ＳｏｕｒｃｅＩｔｅｍＩＤ，ｏｒｄ，ＲｅｌａｔｉｏｎｓｈｉｐＩＤ）は、与えられた関係ＩＤを持つ関係を指定された「Ｏｒｄｅｒ」値を持つ関係の後に移動する。この関係には、ｏｒｄとＯｒｄＮｅｘｔの間の、しかもそれを含まない、新しい順序値を割り当てることができる。

【０１１８】

すでに述べたように、すべてのＩｔｅｍはＩｔｅｍ Ｆｏｌｄｅｒのメンバでなければならない。Ｒｅｌａｔｉｏｎｓｈｉｐｓに関して、すべてのＩｔｅｍはＩｔｅｍ Ｆｏｌｄｅｒと関係を持っていなければならない。本発明のいくつかの実施形態では、Ｉｔｅｍｓ間に存在するＲｅｌａｔｉｏｎｓｈｉｐｓにより特定の関係が表される。

【０１１９】

本発明の様々な実施形態で実装されているように、Ｒｅｌａｔｉｏｎｓｈｉｐは一方のＩｔｅｍ（ソース）により他方のＩｔｅｍ（ターゲット）に「拡張」される有向２項関係を実現する。Ｒｅｌａｔｉｏｎｓｈｉｐは、ソースＩｔｅｍ（それを拡張したＩｔｅｍ）により所有され、したがってＲｅｌａｔｉｏｎｓｈｉｐは、ソースが削除されると削除される（例えば、Ｒｅｌａｔｉｏｎｓｈｉｐは、ソースＩｔｅｍが削除されると削除される）。さらに、いくつかのインスタンスでは、Ｒｅｌａｔｉｏｎｓｈｉｐは、ターゲットＩｔｅｍの所有権を共有（共同所有）することができ、そのような所有権は、ＲｅｌａｔｉｏｎｓｈｉｐのＩｓＯｗｎｅｄプロパティ（またはその等価物）内に反映されることが可能である（図７に、Ｒｅｌａｔｉｏｎｓｈｉｐプロパティ型について示されているように）。これらの実施形態では、新しいＩｓＯｗｎｅｄ Ｒｅｌａｔｉｏｎｓｈｉｐを作成すると、自動的にターゲットＩｔｅｍの参照カウントがインクリメントされ、そのようなＲｅｌａｔｉｏｎｓｈｉｐを削除すると、ターゲットＩｔｅｍの参照カウントがデクリメントされる。これらの特定の実施形態では、Ｉｔｅｍｓは、参照カウントが０よりも大きければ存在し続け、カウントが０になった場合に自動的に削除される。またも、Ｉｔｅｍ Ｆｏｌｄｅｒは、他のＩｔｅｍｓとのＲｅｌａｔｉｏｎｓｈｉｐｓの集合を持つ（または持つことができる）Ｉｔｅｍであり、それらの他のＩｔｅｍｓは、Ｉｔｅｍ Ｆｏｌｄｅｒの帰属関係を含む。Ｒｅｌａｔｉｏｎｓｈｉｐｓの他の実際の実装も可能であり、本発明により本明細書で説明されている機能を実現することが予想される。

【０１２０】

実際の実装に関係なく、Ｒｅｌａｔｉｏｎｓｈｉｐは、一方のオブジェクトから他方の

オブジェクトへの選択可能な接続である。Ｉｔｅｍが複数のＩｔｅｍ　Ｆｏｌｄｅｒに属すことができることとともに、１つまたは複数のＣａｔｅｇｏｒｉｅｓに属すことができること、さらに、これらのＩｔｅｍｓ、Ｆｏｌｄｅｒｓ、およびＣａｔｅｇｏｒｉｅｓがパブリックであるがプライベートであるかは、Ｉｔｅｍベースの構造内で存在すること（または存在しないこと）に対し与えられた意味により決定される。これらの論理的Ｒｅｌａｔｉｏｎｓｈｉｐｓは、本明細書で説明されている機能を実現するために特に採用されている、物理的実装に関係ない、Ｒｅｌａｔｉｏｎｓｈｉｐｓの集合に割り当てられた意味である。論理的Ｒｅｌａｔｉｏｎｓｈｉｐｓは、ＩｔｅｍとそのＩｔｅｍ　Ｆｏｌｄｅｒ（ｓ）またはＣａｔｅｇｏｒｉｅｓとの間で確立されている（およびその逆に）が、それは、本質的にＩｔｅｍ　ＦｏｌｄｅｒｓおよびＣａｔｅｇｏｒｉｅｓはそれぞれＩｔｅｍの特殊な型であるからである。したがって、Ｉｔｅｍ　ＦｏｌｄｅｒｓおよびＣａｔｅｇｏｒｉｅｓは、他のＩｔｅｍと同じようにして作用－限定はしないが、コピー、電子メールメッセージへの追加、ドキュメントへの埋め込み、など－を受けることができ、Ｉｔｅｍ　ＦｏｌｄｅｒｓおよびＣａｔｅｇｏｒｉｅｓに対して、他のＩｔｅｍｓの場合と同じメカニズムを使用してシリアライズおよびデシリアライズ（インポートおよびエクスポート）を行うことができる。（例えば、ＸＭＬでは、すべてのＩｔｅｍｓは、シリアライズ形式を持つことができ、この形式はＩｔｅｍ　Ｆｏｌｄｅｒｓ、Ｃａｔｅｇｏｒｉｅｓ、およびＩｔｅｍｓにも等しく適用される。）

前述のＲｅｌａｔｉｏｎｓｈｉｐｓは、ＩｔｅｍとそれのＩｔｅｍ　Ｆｏｌｄｅｒ（ｓ）と間の関係を表すが、ＩｔｅｍからＩｔｅｍ　Ｆｏｌｄｅｒに、Ｉｔｅｍ　ＦｏｌｄｅｒからＩｔｅｍに、またはその両方に論理的に拡張することができる。論理的にＩｔｅｍからＩｔｅｍ　Ｆｏｌｄｅｒに拡張するＲｅｌａｔｉｏｎｓｈｉｐは、Ｉｔｅｍ　ＦｏｌｄｅｒがそのＩｔｅｍに対しパブリックであることを表し、そのＩｔｅｍとの帰属関係情報を共有するが、逆に、ＩｔｅｍからＩｔｅｍ　Ｆｏｌｄｅｒへの論理的Ｒｅｌａｔｉｏｎｓｈｉｐが存在しないことは、Ｉｔｅｍ　ＦｏｌｄｅｒがそのＩｔｅｍに対しプライベートであり、そのＩｔｅｍとの帰属関係情報を共有しない。同様に、Ｉｔｅｍ　ＦｏｌｄｅｒからＩｔｅｍに論理的に拡張するＲｅｌａｔｉｏｎｓｈｉｐは、ＩｔｅｍがそのＩｔｅｍ　Ｆｏｌｄｅｒに対しパブリックであり共有可能であることを表すが、Ｉｔｅｍ　ＦｏｌｄｅｒからＩｔｅｍへの論理的Ｒｅｌａｔｉｏｎｓｈｉｐが存在しないことは、Ｉｔｅｍがプライベートであり、非共有可能であることを表す。そのため、Ｉｔｅｍ　Ｆｏｌｄｅｒが他のシステムにエクスポートされる場合、それは新しい文脈で共有される「パブリック」Ｉｔｅｍｓであり、ＩｔｅｍがそのＩｔｅｍｓ　Ｆｏｌｄｅｒｓ内で他の共有可能なＩｔｅｍｓを検索する場合、それは、Ｉｔｅｍにそれに属する共有可能Ｉｔｅｍｓに関する情報を供給する「パブリック」Ｉｔｅｍ　Ｆｏｌｄｅｒｓである。

【０１２１】

図９は、Ｉｔｅｍ　Ｆｏｌｄｅｒ（これもまた、Ｉｔｅｍ自体である）、そのメンバＩｔｅｍｓ、およびＩｔｅｍ　ＦｏｌｄｅｒとそのメンバＩｔｅｍｓとの間の相互接続Ｒｅｌａｔｉｏｎｓｈｉｐｓを例示するブロック図である。Ｉｔｅｍ　Ｆｏｌｄｅｒ　９００は、複数のＩｔｅｍｓ　９０２、９０４、および９０６をメンバとして持つ。Ｉｔｅｍ　Ｆｏｌｄｅｒ　９００は、Ｉｔｅｍ　９０２がパブリックでありＩｔｅｍ　Ｆｏｌｄｅｒ　９００、そのメンバ９０４および９０６、およびＩｔｅｍ　Ｆｏｌｄｅｒ　９００にアクセスする可能性のある他のＩｔｅｍ　Ｆｏｌｄｅｒｓ、Ｃａｔｅｇｏｒｉｅｓ、またはＩｔｅｍｓ（図に示されていない）に対し共有可能であることを表す、それ自体からＩｔｅｍ　９０２へのＲｅｌａｔｉｏｎｓｈｉｐ　９１２を持つ。しかし、Ｉｔｅｍ　Ｆｏｌｄｅｒ　９００がＩｔｅｍ　９０２に対しプライベートであり、Ｉｔｅｍ　９０２と帰属関係情報を共有しないことを表す、Ｉｔｅｍ　９０２からＩｔｅｍ　Ｆｏｌｄｅｒ　９００へのＲｅｌａｔｉｏｎｓｈｉｐはない。他方、Ｉｔｅｍ　９０４は、Ｉｔｅｍ　Ｆｏｌｄｅｒ　９００がパブリックであり、Ｉｔｅｍ　９０４と帰属関係情報を共有することを表す、それ自体からＩｔｅｍ　Ｆｏｌｄｅｒ　９００へのＲｅｌａｔｉｏｎｓｈｉｐ　９２４を持つ。しかし、Ｉｔｅｍ　９０４がプライベートであり、Ｉｔｅｍ　Ｆｏｌｄｅｒ

９００、それの他のメンバ９０２および９０６、およびＩｔｅｍ Ｆｏｌｄｅｒ ９００にアクセスする可能性のある他のＩｔｅｍ Ｆｏｌｄｅｒｓ、Ｃａｔｅｇｏｒｉｅｓ、またはＩｔｅｍｓ（図に示されていない）に対し共有可能でないことを表す、Ｉｔｅｍ Ｆｏｌｄｅｒ ９００からＩｔｅｍ ９０４へのＲｅｌａｔｉｏｎｓｈｉｐはない。Ｉｔｅｍｓ ９０２および９０４に対するそのＲｅｌａｔｉｏｎｓｈｉｐｓ（またはそれが存在しないこと）とは対照的に、Ｉｔｅｍ Ｆｏｌｄｅｒ ９００はそれ自体からＩｔｅｍ ９０６へのＲｅｌａｔｉｏｎｓｈｉｐ ９１６を持ち、Ｉｔｅｍ ９０６はＩｔｅｍ Ｆｏｌｄｅｒ ９００に戻るＲｅｌａｔｉｏｎｓｈｉｐ ９２６を持ち、これらは併せて、Ｉｔｅｍ ９０６がパブリックであり、Ｉｔｅｍ Ｆｏｌｄｅｒ ９００、それのメンバ９０２および９０４、およびＩｔｅｍ Ｆｏｌｄｅｒ ９００にアクセスする可能性のある他のＩｔｅｍ Ｆｏｌｄｅｒｓ、Ｃａｔｅｇｏｒｉｅｓ、またはＩｔｅｍｓ（図に示されていない）に対し共有可能であること、およびＩｔｅｍ Ｆｏｌｄｅｒ ９００がパブリックであり、Ｉｔｅｍ ９０６と帰属関係情報を共有することを表す。

【０１２２】

すでに説明したように、Ｉｔｅｍ Ｆｏｌｄｅｒ内のＩｔｅｍｓは、Ｉｔｅｍ Ｆｏｌｄｅｒｓが「記述」されていないため共通性を共有する必要はない。他方、Ｃａｔｅｇｏｒｉｅｓは、そのメンバＩｔｅｍｓのすべてに共通である共通性により記述される。したがって、Ｃａｔｅｇｏｒｙの帰属関係は、本質的に、記述されている共通性を持つＩｔｅｍｓに制限され、いくつかの実施形態では、Ｃａｔｅｇｏｒｙの記述を満たすすべてのＩｔｅｍｓは自動的にＣａｔｅｇｏｒｙのメンバにされる。したがって、Ｉｔｅｍ Ｆｏｌｄｅｒｓでは自明な型構造をその帰属関係により表すことができるが、Ｃａｔｅｇｏｒｉｅｓでは、帰属関係は定義済み共通性に基づくようにできる。

【０１２３】

もちろん、Ｃａｔｅｇｏｒｙ記述は、その性質上論理的であり、したがって、Ｃａｔｅｇｏｒｙは、型、プロパティ、および／または値の論理表現により記述することができる。例えば、Ｃａｔｅｇｏｒｙの論理表現は、Ｉｔｅｍｓを含む帰属関係が２つのプロパティのうちの一方または両方を持つものとすることができる。Ｃａｔｅｇｏｒｙに対するこれらの記述されたプロパティが「Ａ」および「Ｂ」の場合、Ｃａｔｅｇｏｒｉｅｓ帰属関係は、プロパティＡを持つがＢを持たないＩｔｅｍｓ、プロパティＢを持つがＡを持たないＩｔｅｍｓ、およびプロパティＡおよびＢの両方を持つＩｔｅｍｓを含むことができる。プロパティのこの論理表現は、論理演算子「ＯＲ」により記述され、Ｃａｔｅｇｏｒｙにより記述されるメンバの集合はプロパティＡ ＯＲ Ｂを持つＩｔｅｍｓとなる。類似の論理オペランド（限定はしないが、「ＡＮＤ」、「ＸＯＲ」、および「ＮＯＴ」を単独で、または組み合わせて含む）も、カテゴリを記述するために使用することができることは、当業者であれば理解するであろう。

【０１２４】

Ｉｔｅｍ Ｆｏｌｄｅｒｓ（記述されていない）とＣａｔｅｇｏｒｉｅｓ（記述されている）との区別にもかかわらず、Ｃａｔｅｇｏｒｉｅｓ Ｒｅｌａｔｉｏｎｓｈｉｐ対ＩｔｅｍｓとＩｔｅｍｓ Ｒｅｌａｔｉｏｎｓｈｉｐ対Ｃａｔｅｇｏｒｉｅｓは、本発明の多くの実施形態におけるＩｔｅｍ ＦｏｌｄｅｒｓおよびＩｔｅｍｓについて本明細書でこれまでに開示されたのと本質的に同じものである。

【０１２５】

図１０は、Ｃａｔｅｇｏｒｙ（またも、Ｉｔｅｍ自体である）、そのメンバＩｔｅｍｓ、およびＣａｔｅｇｏｒｙとそのメンバＩｔｅｍｓとの間の相互接続のＲｅｌａｔｉｏｎｓｈｉｐｓを例示するブロック図である。Ｃａｔｅｇｏｒｙ １０００は、複数のＩｔｅｍｓ １００２、１００４、および１００６をメンバとして持ち、それらはすべて、Ｃａｔｅｇｏｒｙ １０００により記述されているように（共通性記述１００８’）共通のプロパティ、値、または型１００８の何らかの組合せを共有する。Ｃａｔｅｇｏｒｙ １０００は、Ｉｔｅｍ １００２がパブリックであり、Ｃａｔｅｇｏｒｙ １０００、そのメンバ１００４および１００６、およびＣａｔｅｇｏｒｙ １０００にアクセスする可能性

のある他のＣａｔｅｇｏｒｉｅｓ、Ｉｔｅｍ Ｆｏｌｄｅｒｓ、またはＩｔｅｍｓ（図に示されていない）に対し共有可能であることを表す、それ自体からＩｔｅｍ １００２へのＲｅｌａｔｉｏｎｓｈｉｐ １０１２を持つ。しかし、Ｃａｔｅｇｏｒｙ １０００がＩｔｅｍ １００２に対しプライベートであり、Ｉｔｅｍ １００２と帰属関係情報を共有しないことを表す、Ｉｔｅｍ １００２からＣａｔｅｇｏｒｙ １０００へのＲｅｌａｔｉｏｎｓｈｉｐはない。他方、Ｉｔｅｍ １００４は、Ｃａｔｅｇｏｒｙ １０００がパブリックであり、Ｉｔｅｍ １００４と帰属関係情報を共有することを表す、それ自体からＣａｔｅｇｏｒｙ １０００へのＲｅｌａｔｉｏｎｓｈｉｐ １０２４を持つ。しかし、Ｉｔｅｍ １００４がプライベートであり、Ｃａｔｅｇｏｒｙ １０００、それの他のメンバ１００２および１００６、およびＣａｔｅｇｏｒｙ １０００にアクセスする可能性のある他のＣａｔｅｇｏｒｉｅｓ、Ｉｔｅｍ Ｆｏｌｄｅｒｓ、またはＩｔｅｍｓ（図に示されていない）に対し共有可能でないことを表す、Ｃａｔｅｇｏｒｙ １０００からＩｔｅｍ １００４へのＲｅｌａｔｉｏｎｓｈｉｐはない。Ｉｔｅｍｓ １００２および１００４に対するそのＲｅｌａｔｉｏｎｓｈｉｐｓ（またはそれが存在しないこと）とは対照的に、Ｃａｔｅｇｏｒｙ １０００はそれ自体からＩｔｅｍ １００６へのＲｅｌａｔｉｏｎｓｈｉｐ １０１６を持ち、Ｉｔｅｍ １００６はＣａｔｅｇｏｒｙ １０００に戻るＲｅｌａｔｉｏｎｓｈｉｐ １０２６を持ち、これらは併せて、Ｉｔｅｍ １００６がパブリックであり、Ｃａｔｅｇｏｒｙ １０００、それのＩｔｅｍメンバ１００２および１００４、およびＣａｔｅｇｏｒｙ １０００にアクセスする可能性のある他のＣａｔｅｇｏｒｉｅｓ、Ｉｔｅｍ Ｆｏｌｄｅｒｓ、またはＩｔｅｍｓ（図に示されていない）に対し共有可能であること、およびＣａｔｅｇｏｒｙ １０００がパブリックであり、Ｉｔｅｍ １００６と帰属関係情報を共有することを表す。

【０１２６】

最後に、他のいくつかの実施形態では、ＣａｔｅｇｏｒｉｅｓおよびＩｔｅｍ Ｆｏｌｄｅｒｓは、それ自体Ｉｔｅｍｓであり、Ｉｔｅｍｓは互いとのＲｅｌａｔｉｏｎｓｈｉｐを、ＣａｔｅｇｏｒｉｅｓはＩｔｅｍ ＦｏｌｄｅｒｓへのＲｅｌａｔｉｏｎｓｈｉｐおよびその逆を持ち、Ｃａｔｅｇｏｒｉｅｓ、Ｉｔｅｍ Ｆｏｌｄｅｒｓ、およびＩｔｅｍｓはそれぞれ他のＣａｔｅｇｏｒｉｅｓ、Ｉｔｅｍ Ｆｏｌｄｅｒｓ、およびＩｔｅｍに対するＲｅｌａｔｉｏｎｓｈｉｐを持つことができる。しかし、様々な実施形態では、Ｉｔｅｍ Ｆｏｌｄｅｒ構造および／またはＣａｔｅｇｏｒｙ構造は、ハードウェア／ソフトウェアインターフェースシステムレベルでサイクルを含むことが禁止されている。Ｉｔｅｍ ＦｏｌｄｅｒおよびＣａｔｅｇｏｒｙ構造は、有効グラフに似ており、サイクルを禁止する実施形態は、グラフ理論の分野の数学的定義により、どのような経路も同じ頂点から始まり、終わるということのない有向グラフである、非循環有向グラフ（ＤＡＧｓ）に似ている。

【０１２７】

６．拡張性

ストレージプラットフォームに対して、上述のように、スキーマの初期集合３４０を与えることが意図されている。しかし、それに加えて、少なくともいくつかの実施形態では、ストレージプラットフォームにより、独立系ソフトウェアベンダ（ＩＳＶ）を含む顧客は、新しいスキーマ３４４（つまり、新しいＩｔｅｍおよびＮｅｓｔｅｄ Ｅｌｅｍｅｎｔ型）を作成することができる。このセクションでは、スキーマの初期集合３４０内で定義されているＩｔｅｍ型およびＮｅｓｔｅｄ Ｅｌｅｍｅｎｔ型（または単純に、「Ｅｌｅｍｅｎｔ」型）を拡張することによりそのようなスキーマを作成するためのメカニズムを取りあげる。

【０１２８】

ＩｔｅｍおよびＮｅｓｔｅｄ Ｅｌｅｍｅｎｔ型の初期集合の拡張は、以下のように制約されるのが好ましい。

・ ＩＳＶは新しいＩｔｅｍ型、つまり、子型Ｂａｓｅ．Ｉｔｅｍを導入することが許される。

・　ＩＳＶは新しいＮｅｓｔｅｄ　Ｅｌｅｍｅｎｔ型、つまり、子型Ｂａｓｅ．ＮｅｓｔｅｄＥｌｅｍｅｎｔを導入することが許される。
・　ＩＳＶは新しい拡張、つまり、子型Ｂａｓｅ．ＮｅｓｔｅｄＥｌｅｍｅｎｔを導入することが許される。
・　ＩＳＶは、ストレージプラットフォームのスキーマの初期集合３４０により定義されている型（Ｉｔｅｍ、Ｎｅｓｔｅｄ　Ｅｌｅｍｅｎｔ、またはＥｘｔｅｎｓｉｏｎ型）をサブタイプ化することはできない。

【０１２９】

ストレージプラットフォームスキーマの初期集合により定義されているＩｔｅｍ型またはＮｅｓｔｅｄ　Ｅｌｅｍｅｎｔ型はＩＳＶアプリケーションの要求条件に正確に一致しない場合があるので、ＩＳＶ側で型をカスタマイズできるようにする必要がある。これは、Ｅｘｔｅｎｓｉｏｎｓという概念で可能になる。Ｅｘｔｅｎｓｉｏｎｓは、強く型付けされたインスタンスであるが、（ａ）それらは独立して存在することはできず、（ｂ）ＩｔｅｍまたはＮｅｓｔｅｄ　Ｅｌｅｍｅｎｔに付随しなければならない。

【０１３０】

スキーマの拡張性の必要性に応える他に、Ｅｘｔｅｎｓｉｏｎｓは、「多重型付け」問題を解消することも意図されている。いくつかの実施形態では、ストレージプラットフォームは、多重継承または重複子型をサポートしていない場合があるため、アプリケーション側でＥｘｔｅｎｓｉｏｎｓを重複子型のインスタンスをモデル化する一手段として使用することができる（例えば、Ｄｏｃｕｍｅｎｔは、法律文書であるとともにセキュリティに関する文書でもある）。

【０１３１】

ａ）アイテム拡張

Ｉｔｅｍ拡張を行うために、データモデルで、さらに、Ｂａｓｅ．Ｅｘｔｅｎｓｉｏｎという名前の抽象型を定義する。これは、拡張型の階層に対するルート型である。アプリケーションでは、Ｂａｓｅ．Ｅｘｔｅｎｓｉｏｎをサブタイプ化して、特定の拡張型を作成することができる。

【０１３２】

Ｂａｓｅ．Ｅｘｔｅｎｓｉｏｎ型は、以下のようにＢａｓｅ　ｓｃｈｅｍａで定義される。

【０１３３】

【表５】

```
<Type Name="Base.Extension" IsAbstract="True">
      <Propety Name="ItemID"
               Type="the storage platformTypes.uniqueidentified"
               Nullable="false"
               MultiValued="false"/>
      <Property Name="ExtensionID"
               Type="the storage platformTypes.uniqueidentified"
               Nullable="false"
               MultiValued="false"/>
</Type>
```

【０１３４】

ＩｔｅｍＩＤフィールドは、拡張子が関連付けられているアイテムのＩｔｅｍＩＤを含む。このＩｔｅｍＩＤを持つＩｔｅｍは存在していなければならない。拡張は、与えられたＩｔｅｍＩＤを持つアイテムが存在しない場合には、作成することができない。Ｉｔｅｍが削除されると、同じＩｔｅｍＩＤを持つ拡張はすべて削除される。タプル（ＩｔｅｍＩＤ，ＥｘｔｅｎｓｉｏｎＩＤ）は拡張インスタンスを一意に識別する。

【０１３５】

拡張型の構造は、アイテム型のものと類似している。

・　拡張型はフィールドを持つ。

・　フィールドは、プリミティブまたはネスト要素型とすることができる。
・　拡張型は、サブタイプ化することができる。

【０１３６】

以下の制約が拡張型に対し適用される。
・　拡張は、関係のソースおよびターゲットとすることはできない。
・　拡張型インスタンスは、アイテムから独立しては存在し得ない。
・　拡張型は、ストレージプラットフォームの型定義でのフィールド型として使用することはできない。

【０１３７】

与えられたＩｔｅｍ型に関連付けられる拡張の型には制約はない。どのような拡張型も、アイテム型の拡張に使用することができる。複数の拡張インスタンスがアイテムに付随する場合、それらは、構造とビヘイビアの両面で互いに独立している。

【０１３８】

拡張インスタンスは、格納され、アイテムから別にアクセスされる。すべての拡張型インスタンスは大域的拡張ビューからアクセス可能である。関連するアイテムの型がどうであれ、与えられた拡張の型のすべてのインスタンスを返す効率的なクエリを作成することができる。ストレージプラットフォームＡＰＩは、アイテム上の拡張の格納、取り出し、および修正を行うことができるプログラミングモデルを提供する。

【０１３９】

拡張型は、ストレージプラットフォームの単一継承モデルを使用してサブタイプ化された型とすることができる。拡張型からの派生は、新しい拡張型を生み出す。拡張の構造またはビヘイビアは、アイテム型階層の構造またはビヘイビアをオーバーライドまたは置き換えることができない。Ｉｔｅｍ型と同様に、Ｅｘｔｅｎｓｉｏｎ型インスタンスは、拡張型に関連付けられているビューを通じて直接アクセスされることが可能である。拡張のＩｔｅｍＩＤは、どのアイテムに属しているかを示し、これを使用して、大域的Ｉｔｅｍビューから対応するＩｔｅｍオブジェクトを取り出すことができる。拡張は、動作整合性の目的のためにアイテムの一部と考えられる。ストレージプラットフォームが定義するＣｏｐｙ／Ｍｏｖｅ、Ｂａｃｋｕｐ／Ｒｅｓｔｏｒｅ、およびその他の共通オペレーションは、そのアイテムの一部として拡張上で動作することができる。

【０１４０】

以下の例を考察する。Ｃｏｎｔａｃｔ型は、Ｗｉｎｄｏｗｓ（登録商標）Ｔｙｐｅ設定で定義される。

【０１４１】

【表６】

```
<Type Name="Contact" BaseType="Base.Item" >
        <Property Name="Name"
                  Type="String"
                  Nullable="false"
                  MultiValued="false"/>
        <Property Name="Address"
                  Type="Address"
                  Nullable="true"
                  MultiValued="false"/>

</Type>
```

【０１４２】

ＣＲＭアプリケーション開発者は、ＣＲＭアプリケーション拡張をストレージプラットフォームに格納されている連絡先に付随させたい。アプリケーション開発者は、アプリケーション側で操作することができる追加データ構造を含むＣＲＭ拡張を定義する。

【０１４３】

【表７】

```
<Type Name="CRMExtension" BaseType="Base.Extension" >
        <Property Name="CustomerID"
                  Type="String"
                  Nullable="false"
                  MultiValued="false"/>
        ...
</Type>
```

【０１４４】

ＨＲアプリケーション開発者は、Ｃｏｎｔａｃｔに追加データを付随させたいとも考えている。このデータは、ＣＲＭアプリケーションデータと無関係である。ここでもまた、このアプリケーション開発者は拡張を作成することができる。

【０１４５】

【表８】

```
<Type Name="HRExtension" EBaseType="Base.Extension" >
        <Property Name="EmployeeID"
                  Type="String"
                  Nullable="false"
                  MultiValued="false"/>
        ...
</Type>
```

【０１４６】

ＣＲＭＥｘｔｅｎｓｉｏｎおよびＨＲＥｘｔｅｎｓｉｏｎは、Ｃｏｎｔａｃｔアイテムに付随させることができる２つの独立の拡張である。これらは、互いに独立に作成されアクセスされる。

【０１４７】

上記の例では、ＣＲＭＥｘｔｅｎｓｉｏｎ型のフィールドおよびメソッドは、Ｃｏｎｔａｃｔ階層のフィールドまたはメソッドをオーバーライドすることができない。ＣＲＭＥｘｔｅｎｓｉｏｎ型のインスタンスは、Ｃｏｎｔａｃｔ以外のＩｔｅｍ型に不随させることができる。

【０１４８】

Ｃｏｎｔａｃｔアイテムが取り出される場合、そのアイテム拡張は自動的に取り出されはしない。Ｃｏｎｔａｃｔアイテムが与えられた場合、関連するアイテム拡張は、同じＩｔｅｍＩｄで拡張の大域的拡張ビューにクエリを実行することによりアクセスすることが可能である。

【０１４９】

システム内のすべてのＣＲＭＥｘｔｅｎｓｉｏｎ拡張は、どのアイテムに属しているか関係なく、ＣＲＭＥｘｔｅｎｓｉｏｎ型ビューを通じてアクセスすることができる。アイテムのすべてのアイテム拡張は、同じアイテムｉｄを共有する。上記の例では、Ｃｏｎｔａｃｔアイテムインスタンスおよび付随するＣＲＭＥｘｔｅｎｓｉｏｎおよびＨＲＥｘｔｅｎｓｉｏｎは同じＩｔｅｍＩＤをインスタンス化する。

【０１５０】

以下のテーブルは、Ｉｔｅｍ、Ｅｘｔｅｎｓｉｏｎ、およびＮｅｓｔｅｄＥｌｅｍｅｎｔ型同士の間の類似点と相違点をまとめたものである。

【０１５１】

【表９】

Ｉｔｅｍ、Ｉｔｅｍ拡張、ＮｅｓｔｅｄＥｌｅｍｅｎｔの対比

	Item	Item拡張	NestedElement
アイテムID	自アイテムidを持つ	アイテムのアイテムidを共有する	自アイテムidを持たない。ネストされている要素はアイテムの一部である。
ストレージ	アイテム階層は自テーブルに格納される。	アイテム拡張階層は自テーブルに格納される。	アイテムとともに格納される。
クエリ/検索	アイテムテーブルに対しクエリを実行できる。	アイテム拡張テーブルに対しクエリを実行できる。	一般的に、包含アイテムコンテキスト内でのみクエリを実行できる。
クエリ/検索範囲	アイテム型のすべてのインスタンスにわたって検索できる。	アイテム拡張型のすべてのインスタンスにわたって検索できる。	一般的に、単一の（包含）アイテムのネストされた要素型インスタンス内でのみ検索できる。
関係意味	アイテムとのRelationshipsを持つことができる。	アイテム拡張とのRelationshipsがない。	ネストされた要素とのRelationshipsがない。
アイテムへの関連付け	保持、埋め込み、およびソフトRelationshipsを介して他のアイテムに関連付けられる。	一般的に拡張を介してのみ関連付けられる。拡張意味は、埋め込まれアイテム意味と類似している。	フィールドを介してアイテムに関係する。ネストされている要素はアイテムの一部である。

【０１５２】

ｂ）ＮｅｓｔｅｄＥｌｅｍｅｎｔ型の拡張

ＮｅｓｔｅｄＥｌｅｍｅｎｔ型は、Ｉｔｅｍ型と同じメカニズムで拡張されない。ネストされた要素の拡張は、ネストされた要素型のフィールドと同じメカニズムにより格納され、アクセスされる。

【０１５３】

データモデルは、Ｅｌｅｍｅｎｔという名前のネストされた要素型のルートを定義する。

【０１５４】

【表１０】

```
<Type Name="Element"
    IsAbstract="True">
        <Property Name="ElementID"
                Type="the storage platformTypes.uniqueidentifier"
                Nullable="false"
                MultiValued="false"/>
</Type>
```

【０１５５】

ＮｅｓｔｅｄＥｌｅｍｅｎｔ型はこの型から継承する。ＮｅｓｔｅｄＥｌｅｍｅｎｔ要素型は、さらに、Ｅｌｅｍｅｎｔｓの多重集合であるフィールドを定義する。

【０１５６】

【表１１】

```
<Type Name="NestedElement" BaseType="Base.Element"
      IsAbstract="True">
      <Property Name="Extensions"
                Type="Base.Element"
                Nullable="false"
                MultiValued="true"/>
</Type>
```

【０１５７】

ＮｅｓｔｅｄＥｌｅｍｅｎｔ拡張は、以下の点でアイテム拡張と異なる。

・　ネストされた要素拡張は、拡張型ではない。それらは、Ｂａｓｅ．Ｅｘｔｅｎｓｉｏｎ型をルートとする拡張型階層に属さない。

・　ネストされた要素拡張は、アイテムの他のフィールドとともに格納され、大域的にはアクセス可能でない－与えられた拡張型のすべてのインスタンスを取り出すクエリを作成できない。

・　これらの拡張は、他のネストされた要素（アイテムの）が格納されるとの同じ方法で格納される。他のネストされた集合のように、ＮｅｓｔｅｄＥｌｅｍｅｎｔ拡張はＵＤＴに格納される。これらは、ネストされた要素型のＥｘｔｅｎｓｉｏｎｓフィールドを通じてアクセス可能である。

・　多値プロパティにアクセスするために使用されるコレクションインターフェースも、型拡張の集合についてアクセスおよび反復するために使用される。

【０１５８】

以下のテーブルは、Ｉｔｅｍ　ＥｘｔｅｎｓｉｏｎｓおよびＮｅｓｔｅｄＥｌｅｍｅｎｔ拡張のまとめと比較である。

【０１５９】

【表１２】

ＩｔｅｍとＮｅｓｔｅｄＥｌｅｍｅｎｔの対比

	Item拡張	NestedElement拡張
ストレージ	アイテム拡張階層は自テーブルに格納される。	ネストされた要素のように格納される。
クエリ/検索	アイテム拡張テーブルに対しクエリを実行できる。	一般的に、包含アイテムコンテキスト内でクエリを実行できるだけである。
クエリ/検索範囲	アイテム拡張型のすべてのインスタンスにわたって検索できる。	一般的に、単一の（包含）アイテムのネストされた要素型インスタンス内でのみ検索できる。
プログラム可能性	特別な拡張APIおよび拡張テーブルに対する特別なクエリを必要とする。	NestedElement拡張は、ネストされた要素の他の多値フィールドに似ている。通常のネストされた要素型APIが使用される。
ビヘイビア	ビヘイビアを関連付けることができる。	ビヘイビアは許容されない（?）
関係意味	アイテム拡張とのRelationshipsがない。	NestedElement拡張とのRelationshipsがない。
アイテムID	アイテムのアイテムidを共有する	自アイテムidを持たない。NestedElement拡張はアイテムの一部である。

【０１６０】

Ｄ．データベースエンジン

上述のように、データストアは、データベースエンジン上に実装される。本発明では、データベースエンジンは、オブジェクトリレーショナル拡張とともに、Ｍｉｃｒｏｓｏｆｔ　ＳＱＬ　Ｓｅｒｖｅｒエンジンなどの、ＳＱＬクエリ言語を実装するリレーショナルデータベースエンジンを含む。このセクションでは、データストアが実装するデータモデルのリレーショナルストアへのマッピングについて説明し、また本発明によるストレージプラットフォームクライアントによって消費される論理ＡＰＩに関する情報も掲載する。しかし、異なるデータベースエンジンが使用される場合に、異なるマッピングを採用できることは理解される。実際、ストレージプラットフォームの概念データモデルをリレーショナルデータベースエンジン上に実装することに加えて、さらに、他のタイプのデータベース、例えば、オブジェクト指向およびＸＭＬデータベース上に実装されることも可能である。

【０１６１】

オブジェクト指向（ＯＯ）データベースシステムは、プログラミング言語オブジェクト（例えば、Ｃ＋＋、Ｊａｖａ（登録商標））に対する永続性およびトランザクションを提供する。ストレージプラットフォームにおける「アイテム」の概念は、オブジェクト指向システムにおける「Ｏｂｊｅｃｔ」にうまくマッピングされるが、ただし、埋め込まれたコレクションはＯｂｊｅｃｔｓに追加されなければならないであろう。継承およびネストされた要素型のような他のストレージプラットフォームの型概念も、オブジェクト指向型システムをマッピングする。オブジェクト指向システムでは、通常、オブジェクト識別をすでにサポートしており、したがって、アイテム識別は、オブジェクト識別にマッピングすることができる。アイテムビヘイビア（オペレーション）は、オブジェクトメソッドにうまくマッピングされる。しかし、オブジェクト指向システムは、通常、組織機能を欠いており、検索能力が劣る。また、オブジェクト指向システムは、非構造化および半構造化データのサポートも行わない。本明細書で説明されている完全なストレージプラットフォ

ームデータモデルをサポートするため、関係、フォルダ、および拡張などの概念は、オブジェクトデータモデルに追加される必要があるであろう。さらに、格上げ、同期、通知、およびセキュリティなどのメカニズムも、実装される必要があるであろう。

【０１６２】

オブジェクト指向システムと同様に、ＸＭＬデータベースは、ＸＳＤ（ＸＭＬスキーマ定義）に基づいており、単一継承ベースの型システムをサポートする。本発明のアイテム型システムは、ＸＳＤ型モデルにマッピングされることも可能である。ＸＳＤは、さらに、ビヘイビアのサポートも行わない。アイテムに対するＸＳＤは、アイテムビヘイビアにより増強されなければならないであろう。ＸＭＬデータベースでは、単一のＸＳＤドキュメントを取り扱い、編成および広範な検索機能を欠いている。オブジェクト指向データベースの場合と同様、本明細書で説明されているデータモデルをサポートするため、関係、およびフォルダなどの他の概念は、そのようなＸＭＬデータベースに組み込まれる必要があり、また、同期、通知、およびセキュリティなどのメカニズムも実装される必要がある。

【０１６３】

以下のサブセクションに関して、開示されている一般的情報を利用しやすくするためいくつかの図が用意されており、図１３は、通知メカニズムを例示する図である。図１４は、２つのトランザクションが両方とも新しいレコードを同じＢ－Ｔｒｅｅ内に挿入する実施例を示す図である。図１５は、データ変更検出プロセスを例示する図である。図１６は、ディレクトリツリー例を示す図である。図１７は、ディレクトリベースのファイルシステムの既存のフォルダがストレージプラットフォームのデータストアに移動される例を示す図である。

【０１６４】

１．ＵＤＴを使用したデータストアの実装

本発明の実施形態では、一実施形態ではＭｉｃｒｏｓｏｆｔ　ＳＱＬ　Ｓｅｒｖｅｒエンジンを含むリレーショナルデータベースエンジン３１４は、組み込みスカラー型をサポートする。組み込みスカラー型は「ネイティブ」と「単純」である。それらは、ユーザが独自の型を定義することはできないという点でネイティブであり、複合構造をカプセル化できないという点で単純である。ユーザ定義型（これ以降、ＵＤＴ）は、複合構造型を定義することによりユーザが型システムを拡張できるようにすることによりネイティブのスカラー型システムの上の、またそれを超える、型拡張性のためのメカニズムを用意する。ＵＤＴは、ユーザにより定義された後、型システムにおいて組み込みスカラー型が使用できる場所であればどこででも使用することができる。

【０１６５】

本発明の一態様によれば、ストレージプラットフォームスキーマは、データベースエンジンストア内のＵＤＴクラスにマッピングされる。データストアＩｔｅｍｓは、Ｂａｓｅ．Ｉｔｅｍ型から派生するＵＤＴクラスにマッピングされる。Ｉｔｅｍｓのように、ＥｘｔｅｎｓｉｏｎｓもＵＤＴクラスにマッピングされ、継承を使用する。ルートＥｘｔｅｎｓｉｏｎ型は、Ｂａｓｅ．Ｅｘｔｅｎｓｉｏｎであり、そこからすべてのＥｘｔｅｎｓｉｏｎ型が派生する。

【０１６６】

ＵＤＴはＣＬＲクラスである－これは状態（つまり、データフィールド）とビヘイビア（つまり、ルーチン）を持つ。ＵＤＴは、マネージド言語－Ｃ＃、ＶＢ．ＮＥＴなどを使用して定義される。ＵＤＴメソッドおよび演算子は、その型のインスタンスに対してＴ－ＳＱＬで呼び出すことができる。ＵＤＴは、１行の中の１列の型、Ｔ－ＳＱＬのルーチンのパラメータの型、またはＴ－ＳＱＬの変数の型とすることができる。

【０１６７】

ストレージプラットフォームスキーマをＵＤＴクラスにマッピングすることは、高水準でかなり容易である。一般に、ストレージプラットフォームＳｃｈｅｍａは、ＣＬＲ名前空間にマッピングされる。ストレージプラットフォームＴｙｐｅは、ＣＬＲクラスにマッ

ピングされる。ＣＬＲクラス継承は、ストレージプラットフォームＴｙｐｅ継承を反映し、ストレージプラットフォームＰｒｏｐｅｒｔｙは、ＣＬＲクラスプロパティにマッピングされる。

【０１６８】

２．アイテムのマッピング

Ｉｔｅｍｓが大域的に検索可能であることが望ましく、継承および型代替可能性に対する本発明のリレーショナルデータベースでのサポートがあれば、データベースストア内のＩｔｅｍストレージの１つの可能な実装は、すべてのＩｔｅｍｓを型Ｂａｓｅ．Ｉｔｅｍの列を含む単一テーブルに格納することである。型代替可能性を使用すると、すべての型のＩｔｅｍｓを格納することが可能になり、またＹｕｋｏｎの「ｉｓ　ｏｆ（Ｔｙｐｅ）」演算子を使用してＩｔｅｍ型と子型により検索をフィルタ処理することが可能になる。

【０１６９】

しかし、本発明の実施形態においては、そのようなアプローチに関連するオーバーヘッドに関する心配から、Ｉｔｅｍｓは、最上位の型により分割され、それぞれの型の「族」のＩｔｅｍｓが別々のテーブルに格納される。このパーティション分割スキームに従って、Ｂａｓｅ．Ｉｔｅｍから直接継承するＩｔｅｍ型毎に１つのテーブルが作成される。これらの下の型継承は、上述のように、型代替可能性を使用して適切な型族テーブルに格納される。Ｂａｓｅ．Ｉｔｅｍからの継承の第１のレベルのみが特別に処理される。

【０１７０】

すべてのＩｔｅｍｓに対する大域的に検索可能なプロパティのコピーを格納するために「シャドウ」テーブルが使用される。このテーブルは、すべてのデータ変更が行われる際に使用される、ストレージプラットフォームＡＰＩのＵｐｄａｔｅ（）メソッドにより保持することができる。型族テーブルと異なり、大域的Ｉｔｅｍテーブルは、完全なＵＤＴ　Ｉｔｅｍオブジェクトではなく、Ｉｔｅｍの最上位スカラープロパティのみを含む。大域的Ｉｔｅｍテーブルでは、ＩｔｅｍＩＤおよびＴｙｐｅＩＤを公開することにより型族テーブルに格納されているＩｔｅｍオブジェクトにナビゲートすることができる。ＩｔｅｍＩＤは、一般に、データストア内のＩｔｅｍを一意に識別する。ＴｙｐｅＩＤは、ここでは説明しないメタデータを使用して、型名およびＩｔｅｍを含むビューにマッピングすることができる。ＩｔｅｍをそのＩｔｅｍＩＤにより見つけることはごくふつうのオペレーションといえるので、大域的Ｉｔｅｍテーブルおよび他の何らかの手段の両方の文脈において、ＩｔｅｍのＩｔｅｍＩＤを指定するとＩｔｅｍオブジェクトを取り出すＧｅｔＩｔｅｍ（）関数が用意される。

【０１７１】

アクセスを簡単に、実装の詳細をできる限り隠すために、Ｉｔｅｍｓのすべてのクエリは、上述のＩｔｅｍテーブル上に構築されたビューに対して行われるようにできる。特に、ビューは、該当する型族テーブルと突き合わせてＩｔｅｍ型毎に作成されうる。これらの型ビューでは、子型を含む、関連付けられた型のすべてのＩｔｅｍｓを選択することができる。便宜のため、ＵＤＴオブジェクトに加えて、それらのビューは、継承されたフィールドを含む、その型の最上位レベルのフィールドのすべてに対する列を公開することができる。

【０１７２】

３．拡張マッピング

Ｅｘｔｅｎｓｉｏｎｓは、Ｉｔｅｍｓに非常によく似ており、同じ要求条件のうちいくつかを持つ。継承をサポートする他のルート型として、Ｅｘｔｅｎｓｉｏｎｓはストレージに対する同じ考慮事項とトレードオフの関係の多くの影響を受ける。このため、類似の型族マッピングは、単一テーブルアプローチではなく、むしろ、Ｅｘｔｅｎｓｉｏｎｓに適用される。もちろん、他の実施形態では、単一テーブルアプローチを使用することが可能である。本発明の実施形態では、ＥｘｔｅｎｓｉｏｎはＩｔｅｍＩＤによりちょうど１つのＩｔｅｍに関連付けられ、Ｉｔｅｍの文脈において一意であるＥｘｔｅｎｓｉｏｎＩＤを含む。Ｉｔｅｍｓの場合と同様に、ＩｔｅｍＩＤとＥｘｔｅｎｓｉｏｎＩＤのペアか

らなる、識別を与えられたＥｘｔｅｎｓｉｏｎを取り出す関数を用意することが可能である。Ｖｉｅｗは、Ｅｘｔｅｎｓｉｏｎ型毎に作成され、Ｉｔｅｍ型ビューに類似している。

【０１７３】

４．ネストされている要素のマッピング

Ｎｅｓｔｅｄ　Ｅｌｅｍｅｎｔｓは、深くネストされた構造を形成するためにＩｔｅｍｓ、Ｅｘｔｅｎｓｉｏｎｓ、Ｒｅｌａｔｉｏｎｓｈｉｐｓ、または他のＮｅｓｔｅｄ　Ｅｌｅｍｅｎｔｓ内に埋め込むことができる型である。ＩｔｅｍｓおよびＥｘｔｅｎｓｉｏｎｓのように、Ｎｅｓｔｅｄ　Ｅｌｅｍｅｎｔｓは、ＵＤＴとして実装されるが、ＩｔｅｍｓとＥｘｔｅｎｓｉｏｎｓ内に格納される。したがって、Ｎｅｓｔｅｄ　Ｅｌｅｍｅｎｔｓは、そのＩｔｅｍおよびＥｘｔｅｎｓｉｏｎコンテナものを超えるストレージマッピングを持たない。つまり、ＮｅｓｔｅｄＥｌｅｍｅｎｔ型のインスタンスを直接格納するテーブルはシステムにはなく、Ｎｅｓｔｅｄ　Ｅｌｅｍｅｎｔｓ専用のビューもない。

【０１７４】

５．オブジェクト識別

データモデル内の各エンティティ、つまり、各Ｉｔｅｍ、Ｅｘｔｅｎｓｉｏｎ、およびＲｅｌａｔｉｏｎｓｈｉｐは一意のキー値を持つ。Ｉｔｅｍは、ＩｔｅｍＩｄにより一意に識別される。Ｅｘｔｅｎｓｉｏｎは、（ＩｔｅｍＩｄ，ＥｘｔｅｎｓｉｏｎＩｄ）の複合キーにより一意に識別される。Ｒｅｌａｔｉｏｎｓｈｉｐは、複合キー（ＩｔｅｍＩｄ，ＲｅｌａｔｉｏｎｓｈｉｐＩｄ）により識別される。ＩｔｅｍＩｄ、ＥｘｔｅｎｓｉｏｎＩｄ、およびＲｅｌａｔｉｏｎｓｈｉｐＩｄはＧＵＩＤ値である。

【０１７５】

６．ＳＱＬオブジェクトの命名規則

データストア内に作成されたすべてのオブジェクトは、ストレージプラットフォームスキーマ名から派生されたＳＱＬスキーマ名で格納することができる。例えば、ストレージプラットフォームＢａｓｅスキーマ（「Ｂａｓｅ」と呼ばれることが多い）は、「［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．Ｉｔｅｍ」などの「［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］」ＳＱＬスキーマで型を生成することができる。生成された名前は、命名の競合をなくすため、修飾子が先頭に付けられる。適切であれば、感嘆符（！）が、名前の各論理部分の仕切として使用される。以下のテーブルは、データストア内のオブジェクトに使用される命名規則の概要である。それぞれのスキーマ要素（Ｉｔｅｍ、Ｅｘｔｅｎｓｉｏｎ、Ｒｅｌａｔｉｏｎｓｈｉｐ、およびＶｉｅｗ）は、データストア内のインスタンスにアクセスするために使用される装飾された命名規則とともに一覧にされている。

【０１７６】

【表１３】

オブジェクト	名前装飾	説明	例
マスター アイテム 検索ビュー	Master!Item	現在のアイテム領域内のアイテムの概要を示す。	[System.Storage]. [Master!Item]
型付き アイテム 検索ビュー	ItemType	itemおよび親型からのすべてのプロパティデータを与える。	[AcmeCorp.Doc]. [OfficeDoc]
マスター 拡張検索 ビュー	Master!Extension	現在のアイテム領域内のすべての拡張の概要を示す。	[System.Storage]. [Master!Extension]
型付き 拡張検索 ビュー	Extension!extension Type	extensionに対するすべてのプロパティデータを与える。	[AcmeCorp.Doc]. [Extension!Sticky Note]
マスター 関係ビュー	Master!Relationship	現在のアイテム領域内のすべての関係の概要を示す。	[System.Storage]. [Master!Relationship]
関係ビュー	Relationship!relationshipName	与えられたrelationshipに関連付けられているすべてのデータを与える。	[AcmeCorp.Doc]. [Relationship!AuthorsFromDocument]
ビュー	View!viewName	スキーマview定義に基づく列/型を与える。	[AcmeCorp.Doc]. [View!DocumentTitles]

【０１７７】

７．列の命名規則

オブジェクトモデルをストアにマッピングする場合、アプリケーションオブジェクトとともに追加情報が格納されているため、命名競合が発生する可能性がある。命名競合を回避するため、すべての非型固有列（型宣言で名前付きＰｒｏｐｅｒｔｙに直接マッピングされない列）の先頭に、下線（＿）文字を付ける。本発明の実施形態では、下線（＿）文字は、識別子プロパティの先頭文字としては禁止されている。さらに、ＣＬＲとデータストアとの間の命名規則の統一のため、ストレージプラットフォームの型またはスキーマ要素（関係など）のすべてのプロパティは、先頭文字を大文字にしていなければならない。

【０１７８】

８．検索ビュー

ビューは、格納されているコンテンツを検索するためストレージプラットフォームにより用意されている。ＳＱＬビューは、各ＩｔｅｍおよびＥｘｔｅｎｓｉｏｎ型に用意されている。さらに、ビューは、ＲｅｌａｔｉｏｎｓｈｉｐｓとＶｉｅｗｓをサポートするためにも用意されている（Ｄａｔａ　Ｍｏｄｅｌでの定義に従って）。ストレージプラットフォーム内のすべてのＳＱＬビューおよび基礎のテーブルは読み取り専用である。データは、以下に詳述するように、ストレージプラットフォームＡＰＩのＵｐｄａｔｅ（）メソッドを使用して格納または変更することができる。

【０１７９】

ストレージプラットフォームスキーマで明示的に定義されているそれぞれのビューは（スキーマデザイナにより定義され、ストレージプラットフォームにより自動的には生成されない）は、名前付きＳＱＬビュー［＜ｓｃｈｅｍａ－ｎａｍｅ＞］．［Ｖｉｅｗ！＜ｖｉｅｗ－ｎａｍｅ＞］によりアクセス可能である。例えば、スキーマ「ＡｃｍｅＰｕｂｌｉｓｈｅｒ．Ｂｏｏｋｓ」内の「ＢｏｏｋＳａｌｅｓ」という名前のビューは、名前「［

ＡｃｍｅＰｕｂｌｉｓｈｅｒ．Ｂｏｏｋｓ］．［Ｖｉｅｗ！ＢｏｏｋＳａｌｅｓ］」を使用するとアクセス可能である。ビューの出力形式はビュー毎に変更できるので（ビューを定義する当事者により与えられる任意のクエリにより定義される）、列は、スキーマビュー定義に基づいて直接マッピングされる。

【０１８０】

ストレージプラットフォームのデータストア内のすべてのＳＱＬ検索ビューは、列に対し以下の順序付け規則を使用する。

・　ＩｔｅｍＩｄ、ＥｌｅｍｅｎｔＩｄ、ＲｅｌａｔｉｏｎｓｈｉｐＩｄ、．．．などのビュー結果の（複数の）論理「ｋｅｙ」列。

・　ＴｙｐｅＩｄなどの結果の型に関するメタデータ情報。

・　Ｃｒｅａｔｅ　Ｖｅｒｓｉｏｎ、Ｕｐｄａｔｅ　Ｖｅｒｓｉｏｎ、．．．などの変更追跡列。

・　（複数の）型特有の列（宣言された型のＰｒｏｐｅｒｔｉｅｓ）

・　型特有のビュー（族ビュー）も、オブジェクトを返すオブジェクト列を含む。

【０１８１】

各型族のメンバは、一連のＩｔｅｍビューを使用して検索可能であり、データストア内にＩｔｅｍ型毎に１つのビューがある。図２８は、Ｉｔｅｍの検索ビューの概念を例示する図である。

【０１８２】

ａ）アイテム

各Ｉｔｅｍ検索ビューは、特定の型またはその子型のＩｔｅｍの各インスタンスに対する１つの行を含む。例えば、Ｄｏｃｕｍｅｎｔに対するビューは、Ｄｏｃｕｍｅｎｔ、ＬｅｇａＩＤｏｃｕｍｅｎｔ、およびＲｅｖｉｅｗＤｏｃｕｍｅｎｔのインスタンスを返すことができる。この例では、Ｉｔｅｍビューは、図２９に示されているように概念化することができる。

【０１８３】

（１）マスターアイテム検索ビュー

ストレージプラットフォームのデータストアのそれぞれのインスタンスは、マスターアイテムビューという特別なＩｔｅｍビューを定義する。このビューは、データストア内のそれぞれのＩｔｅｍに関する概要情報を示す。ビューは、Ｉｔｅｍ型プロパティ毎に１つの列を示し、列は、変更追跡および同期情報を供給するために使用されるＩｔｅｍおよび複数の列の型を記述している。マスターアイテムビューは、名前「［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．［Ｍａｓｔｅｒ！Ｉｔｅｍ］」を使用してデータストア内で識別される。

【０１８４】

【表１４】

列	型	説明
ItemId	ItemId	Itemのストレージプラットフォームの識別。
_TypeId	TypeId	ItemのTypeId-Itemの正確な型を識別し、またMetadataカタログを使用して型に関する情報を取り出すため使用することができる。
_RootItemId	ItemId	このアイテムの存続期間を制御する第1の埋め込みでない祖先のItemId。
<global change tracking>	…	大域的変更追跡情報
<Item props>	n/a	Item型プロパティ毎に1つの列。

【０１８５】

（２）型付きアイテム検索ビュー

各Ｉｔｅｍ型は、さらに、検索ビューを持つ。ルートＩｔｅｍビューに似ているが、このビューは、さらに、「＿Ｉｔｅｍ」列を介してＩｔｅｍオブジェクトにアクセスできる。各型付きアイテム検索ビューは、名前［ｓｃｈｅｍａＮａｍｅ］．［ｉｔｅｍＴｙｐｅＮａｍｅ］を使用してデータストア内で識別される。例えば、［ＡｃｍｅＣｏｒｐ．Ｄｏｃ］．［ＯｆｆｉｃｅＤｏｃ］。

【０１８６】

【表１５】

列	型	説明
ItemId	ItemId	Itemのストレージプラットフォームの識別。
<type change tracking>	…	型変更追跡情報
<parent props>	<property specific>	親プロパティ毎に1つの列。
<item props>	<property specific>	この型の排他的プロパティ毎に1つの列。
_Item	ItemのCLR型	CLRオブジェクト-宣言されたItemの型

【０１８７】

ｂ）アイテム拡張

ＷｉｎＦＳ　Ｓｔｏｒｅ内のすべてのＩｔｅｍ　Ｅｘｔｅｎｓｉｏｎｓは、検索ビューを使用してアクセスすることもできる。

【０１８８】

（１）マスター拡張検索ビュー

データストアのそれぞれのインスタンスは、マスター拡張ビューという特別なＥｘｔｅｎｓｉｏｎビューを定義する。このビューは、データストア内のそれぞれのＥｘｔｅｎｓｉｏｎに関する概要情報を示す。ビューは、Ｅｘｔｅｎｓｉｏｎプロパティ毎に１つの列を持ち、列は、変更追跡および同期情報を供給するために使用されるＥｘｔｅｎｓｉｏｎおよび複数の列の型を記述している。マスター拡張ビューは、名前「［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．［Ｍａｓｔｅｒ！Ｅｘｔｅｎｓｉｏｎ］」を使用してデータストア内で識別される。

【０１８９】

【表１６】

列	型	説明
ItemId	ItemId	この拡張が関連付けられているItemのストレージプラットフォーム識別。
ExtensionId	ExtensionId (GUID)	この拡張インスタンスのId
_TypeId	TypeId	ExtensionのTypeId-拡張の正確な型を識別し、またMetadataカタログを使用して拡張に関する情報を取り出すため使用することができる。
<global change tracking>	…	大域的変更追跡情報
<ext properties>	<property specific>	Extension型プロパティ毎に1つの列。

【０１９０】

（２）型付き拡張検索ビュー

各Ｅｘｔｅｎｓｉｏｎ型は、さらに、検索ビューを持つ。マスター拡張ビューに似てい

るが、このビューは、さらに、「_Ｅｘｔｅｎｓｉｏｎ」列を介してＩｔｅｍオブジェクトにアクセスできる。各型付き拡張検索ビューは、名前［ｓｃｈｅｍａＮａｍｅ］．［Ｅｘｔｅｎｓｉｏｎ！ｅｘｔｅｎｓｉｏｎＴｙｐｅＮａｍｅ］を使用してデータストア内で識別される。例えば、［ＡｃｍｅＣｏｒｐ．Ｄｏｃ］［Ｅｘｔｅｎｓｉｏｎ！ＯｆｆｉｃｅＤｏｃＥｘｔ］。

【０１９１】

【表１７】

列	型	説明
ItemId	ItemId	この拡張が関連付けられているItemのストレージプラットフォーム識別。
ExtensionId	ExtensionId (GUID)	この拡張インスタンスのId
<type change tracking>	…	型変更追跡情報
<parent props>	<property specific>	親プロパティ毎に1つの列。
<ext props>	<property specific>	この型の排他的プロパティ毎に1つの列。
_Extension	Extension インスタンスのCLR型	CLRオブジェクト-宣言されたExtensionの型

【０１９２】

ｃ）ネストされている要素

すべてのネストされている要素は、Ｉｔｅｍｓ、Ｅｘｔｅｎｓｉｏｎｓ、またはＲｅｌａｔｉｏｎｓｈｉｐｓインスタンス内に格納される。したがって、該当するＩｔｅｍ、Ｅｘｔｅｎｓｉｏｎ、またはＲｅｌａｔｉｏｎｓｈｉｐ検察ビューにクエリを行うことで、これらはアクセスされる。

【０１９３】

ｄ）関係

上述のように、Ｒｅｌａｔｉｏｎｓｈｉｐｓは、ストレージプラットフォームのデータストア内にＩｔｅｍｓ間のリンキングの基本ユニットを形成する。

【０１９４】

（１）マスター関係検索ビュー

それぞれのデータストアは、Ｍａｓｔｅｒ Ｒｅｌａｔｉｏｎｓｈｉｐ Ｖｉｅｗを与える。このビューは、データストア内のすべての関係インスタンスに関する情報を示す。マスター関係ビューは、名前「［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．［Ｍａｓｔｅｒ！Ｒｅｌａｔｉｏｎｓｈｉｐ］」を使用してデータストア内で識別される。

【０１９５】

【表１８】

列	型	説明
ItemId	ItemId	ソース終点の識別（ItemId）
RelationshipId	RelationshipId (GUID)	関係インスタンスのid
_RelTypeId	Relationship TypeId	RelationshipのRelTypeId-Metadataカタログを使用して関係インスタンスの型を識別する。
<global change tracking>	…	大域的変更追跡情報
TargetItem Reference	ItemReference	ターゲット終点の識別
_Relationship	Relationship	このインスタンスに対するRelationshipオブジェクトのインスタンス。

【０１９６】

（２）関係インスタンス検索ビュー

それぞれの宣言されたＲｅｌａｔｉｏｎｓｈｉｐは、さらに、特定の関係のすべてのインスタンスを返す検索ビューを持つ。マスター関係ビューに似ているが、このビューは、さらに、関係データのプロパティ毎に名前付き列を示す。各関係インスタンス検索ビューは、名前［ｓｃｈｅｍａＮａｍｅ］．［Ｒｅｌａｔｉｏｎｓｈｉｐ！ｒｅｌａｔｉｏｎｓｈｉｐＮａｍｅ］を使用してデータストア内で識別される。例えば、［ＡｃｍｅＣｏｒｐ．Ｄｏｃ］．［Ｒｅｌａｔｉｏｎｓｈｉｐ！ＤｏｃｕｍｅｎｔＡｕｔｈｏｒ］。

【０１９７】

【表１９】

列	型	説明
ItemId	ItemId	ソース終点の識別（ItemId）
RelationshipId	RelationshipId (GUID)	関係インスタンスのid
<type change tracking>	…	型変更追跡情報
TargetItem Reference	ItemReference	ターゲット終点の識別
<source name>	ItemId	ソース終点識別の名前付きプロパティ（ItemIdの別名）。
<target name>	ItemReference または 派生クラス	ターゲット終点識別の名前付きプロパティ（TargetItemReferenceの別名およびキャスト）。
<rel property>	<property specific>	関係定義のプロパティ毎に1つの列。
_Relationship	Relationship インスタンスの CLR型	CLRオブジェクト-宣言されたRelationshipの型

【０１９８】

９．更新

ストレージプラットフォームのデータストア内のすべてのビューは読み取り専用である。データモデル要素（アイテム、拡張、または関係）の新しいインスタンスを作成する、

または既存のインスタンスを更新するには、ストレージプラットフォームＡＰＩのＰｒｏｃｅｓｓＯｐｅｒａｔｉｏｎまたはＰｒｏｃｅｓｓＵｐｄａｔｅｇｒａｍメソッドが使用されなければならない。ＰｒｏｃｅｓｓＯｐｅｒａｔｉｏｎメソッドは、実行されるアクションの詳細を決める「オペレーション」を消費するデータストアにより定義された単一のストアドプロシージャである。ＰｒｏｃｅｓｓＵｐｄａｔｅｇｒａｍメソッドは、実行されるアクションの集合の詳細を包括的に決める、「アップデートグラム」と呼ばれる、オペレーションの順序付き集合を受け取るストアドプロシージャである。

【０１９９】

オペレーション形式は、拡張可能であり、スキーマ要素に対する様々なオペレーションを用意している。共通オペレーションとしては以下のものがある。

１．Ｉｔｅｍオペレーション：

ａ．ＣｒｅａｔｅＩｔｅｍ（埋め込みまたは保持関係の文脈において新しいアイテムを作成する）

ｂ．ＵｐｄａｔｅＩｔｅｍ（既存のＩｔｅｍを更新する）

２．Ｒｅｌａｔｉｏｎｓｈｉｐオペレーション：

ａ．ＣｒｅａｔｅＲｅｌａｔｉｏｎｓｈｉｐ（参照または保持関係のインスタンスを作成する）

ｂ．ＵｐｄａｔｅＲｅｌａｔｉｏｎｓｈｉｐ（関係インスタンスを更新する）

ｃ．ＤｅｌｅｔｅＲｅｌａｔｉｏｎｓｈｉｐ（関係インスタンスを削除する）

３．Ｅｘｔｅｎｓｉｏｎオペレーション：

ａ．ＣｒｅａｔｅＥｘｔｅｎｓｉｏｎ（既存のＩｔｅｍに拡張を追加する）

ｂ．ＵｐｄａｔｅＥｘｔｅｎｓｉｏｎ（既存の拡張を更新する）

ｃ．ＤｅｌｅｔｅＥｘｔｅｎｓｉｏｎ（拡張を削除する）

１０．変更追跡＆ツームストーン

変更追跡およびツームストーンサービスは、以下で詳述するように、データストアにより提供される。このセクションでは、データストア内に公開された変更追跡情報の概要を述べる。

【０２００】

ａ）変更追跡

データストアによって与えられるそれぞれの検索ビューは、変更追跡情報を供給するために使用される列を含み、それらの列は、すべてのＩｔｅｍ、Ｅｘｔｅｎｓｉｏｎ、およびＲｅｌａｔｉｏｎｓｈｉｐビュー間で共通である。ストレージプラットフォームのＳｃｈｅｍａ　Ｖｉｅｗｓは、スキーマデザイナにより明示的に定義され、変更追跡情報を自動的に供給することはしない－そのような情報は、ビュー自体が構築される検索ビューを通じて間接的に供給される。

【０２０１】

データストア内の要素毎に、変更追跡情報は２つの場所－「マスター」要素ビューと「型付き」要素ビューから利用できる。例えば、ＡｃｍｅＣｏｒｐ．Ｄｏｃｕｍｅｎｔ．Ｄｏｃｕｍｅｎｔ　Ｉｔｅｍ型に関する変更追跡情報は、マスターアイテムビュー「［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．［Ｍａｓｔｅｒ！Ｉｔｅｍ］」および型付きアイテム検索ビュー［ＡｃｍｅＣｏｒｐ．Ｄｏｃｕｍｅｎｔ］．［Ｄｏｃｕｍｅｎｔ］から利用できる。

【０２０２】

（１）「マスター」検索ビューでの変更追跡

マスター検索ビュー内の変更追跡情報は、要素の作成および更新バージョンに関する情報、要素を作成した同期パートナに関する情報、要素を最後に更新した同期パートナに関する情報、および作成および更新に関する各パートナからのバージョン番号を示す。同期関係にあるパートナ（後述）は、パートナキーにより識別される。型［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ．Ｓｔｏｒｅ］．ＣｈａｎｇｅＴｒａｃｋｉｎｇＩｎｆｏの＿ＣｈａｎｇｅＴｒａｃｋｉｎｇＩｎｆｏという名前の単一のＵＤＴオブジェクトがこの情報のすべてを

格納する。この型は、Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅスキーマで定義される。＿ＣｈａｎｇｅＴｒａｃｋｉｎｇＩｎｆｏは、Ｉｔｅｍ、Ｅｘｔｅｎｓｉｏｎ、およびＲｅｌａｔｉｏｎｓｈｉｐについてすべての大域的検索ビューで利用可能である。ＣｈａｎｇｅＴｒａｃｋｉｎｇＩｎｆｏの型定義は以下のとおりである。

【０２０３】

【表２０】

```
<Type Name="ChangeTrackingInfo" BaseType="Base.NestedElement">
  <FieldProperty Name="CreationLocalTS"       Type="SqlTypes.SqlInt64"
  Nullable="False" />
  <FieldProperty Name="CreatingPartnerKey"
  Type="SqlTypes.SqlInt32"        Nullable="False" />
  <FieldProperty Name="CreatingPartnerTS"
  Type="SqlTypes.SqlInt64"        Nullable="False" />
  <FieldProperty Name="LastUpdateLocalTS"
  Type="SqlTypes.SqlInt64"        Nullable="False" />
  <FieldProperty Name="LastUpdatingPartnerKey"
  Type="SqlTypes.SqlInt32"        Nullable="False" />
  <FieldProperty Name="LastUpdatingPartnerTS" Type="SqlTypes.SqlInt64"
  Nullable="False" />
</Type>
```

【０２０４】

これらのプロパティは、以下の情報を含む。

【０２０５】

【表２１】

列	説明
_CreationLocalTS	ローカルマシンによる作成時タイムスタンプ。
_CreatingPartnerKey	このエンティティを作成したパートナのPartnerKey。このエンティティがローカルで作成された場合、これはローカルマシンのPartnerKeyである。
_CreatingPartnerTS	このエンティティが_CreatingPartnerKeyに対応するパートナで作成された時間のタイムスタンプ。
_LastUpdateLocalTS	ローカルマシンの更新時間に対応するローカルタイムスタンプ。
_LastUpdatingPartnerKey	このエンティティを最後に更新したパートナのPartnerKey。このエンティティに対する最後の更新がローカルで実行された場合、これはローカルマシンのPartnerKeyである。
_LastUpdatingPartnerTS	このエンティティが_LastUpdatingPartnerKeyに対応するパートナで更新された時間のタイムスタンプ。

【０２０６】

（２）「型付き」検索ビューでの変更追跡

大域的検索ビューと同じ情報を供給することに加えて、それぞれの型付き検索ビューは、同期トポロジ内の各要素の同期状態を記録した追加情報を供給する。

【０２０７】

【表２２】

列	型	説明
<global change tracking>	…	大域的変更追跡からの情報。
_ChangeUnit Versions	MultiSet<ChangeUnit Version>	特定の要素内の変更ユニットのバージョン番号の記述。
_ElernentSync Metadata	ElementSyncMetadata	Synchronizationランタイムにのみ関係のあるこのアイテムに関するバージョン独立の追加メタデータ。
_VersionSync Metadata	VersionSyncMetadata	Synchronizationランタイムにのみ関係のあるこのバージョンに関するバージョン特有の追加メタデータ。

【０２０８】

ｂ）ツームストーン

データストアは、Ｉｔｅｍｓ、Ｅｘｔｅｎｓｉｏｎｓ、およびＲｅｌａｔｉｏｎｓｈｉｐｓのツームストーン情報を与える。ツームストーンビューは、ある場所にあるライブ状態のエンティティとツームストーン状態のエンティティ（ｌｉｖｅ ａｎｄ ｔｏｍｂｓｔｏｎｅｄ ｅｎｔｉｔｉｅｓ）（アイテム、拡張、および関係）の両方に関する情報を示す。アイテムおよび拡張ツームストーンビューは、対応するオブジェクトへのアクセスを行えないが、関係ツームストーンビューは、関係オブジェクトへのアクセスを行える（関係オブジェクトは、ツームストーン状態の関係の場合にはＮＵＬＬである）。

【０２０９】

（１）アイテムツームストーン

アイテムツームストーンは、ビュー［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．［Ｔｏｍｂｓｔｏｎｅ！Ｉｔｅｍ］を介してシステムから取り出される。

【０２１０】

【表２３】

列	型	説明
ItemId	ItemId	Itemの識別。
_TypeID	TypeId	Itemの型。
<Item properties>	…	すべてのアイテムについて定義されているプロパティ。
_RootItemId	ItemId	このアイテムを含む第1の非埋め込みアイテムのItemId。
_ChangeTrackingInfo	型ChangeTrackingInfoのCLRインスタンス	このアイテムの変更追跡情報。
_IsDeleted	BIT	これは、ライブ状態アイテムについては0、ツームストーン状態アイテムについては1であるフラグである。
_DeletionWallclock	UTCDATETIME	アイテムを削除したパートナによる時計のUTC日時。Itemがライブ状態であればnullである。

【０２１１】

（２）拡張ツームストーン

拡張ツームストーンは、ビュー［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．［Ｔｏｍｂｓｔｏｎｅ！Ｅｘｔｅｎｓｉｏｎ］を使用してシステムから取り出される。拡張変更追跡情報は、ＥｘｔｅｎｓｉｏｎＩｄプロパティの追加を含むＩｔｅｍｓについて与えられる情報と類似している。

【０２１２】

【表２４】

列	型	説明
ItemId	ItemId	Extensionを所有するItemの識別。
ExtensionId	ExtensionId	ExtensionのExtension Id
_TypeID	TypeId	拡張の型。
_ChangeTrackingInfo	型ChangeTrackingInfoのCLRインスタンス	この拡張の変更追跡情報。
_IsDeleted	BIT	これは、ライブ状態アイテムについては0、ツームストーン状態拡張については1であるフラグである。
_DeletionWallclock	UTCDATETIME	拡張を削除したパートナによる時計のUTC日時。拡張がライブ状態であればNULLである。

【０２１３】

（３）関係ツームストーン

関係ツームストーンは、ビュー［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．［Ｔｏｍｂｓｔｏｎｅ！Ｒｅｌａｔｉｏｎｓｈｉｐ］を介してシステムから取り出される。関係ツームストーン情報は、Ｅｘｔｅｎｓｉｏｎｓについて与えられる情報と類似している。しかし、追加情報は、関係インスタンスのターゲットＩｔｅｍＲｅｆ上で与えられる。さらに、関係オブジェクトも選択される。

【０２１４】

【表２５】

列	型	説明
ItemId	ItemId	関係を所有していたItemの識別（関係ソース終点の識別）。
RelationshipId	RelationshipId	関係のRelationshipId。
_TypeID	TypeId	関係の型。
_ChangeTrackingInfo	型ChangeTrackingInfoのCLRインスタンス	この関係の変更追跡情報。
_IsDeleted	BIT	これは、ライブ状態アイテムについては0、ツームストーン状態拡張については1であるフラグである。
_DeletionWallclock	UTCDATETIME	関係を削除したパートナによる時計のUTC日時。関係がライブ状態であればNULLである。
_Relationship	RelationshipのCLRインスタンス	これは、ライブ状態の関係に対する関係オブジェクトである。ツームストーン状態の関係についてはNULLである。
TargetItemReference	ItemReference	ターゲット終点の識別

【０２１５】

（４）ツームストーンのクリーンアップ

ツームストーン情報の際限のない増殖を防ぐために、データストアは、ツームストーンのクリーンアップタスクを用意している。このタスクは、ツームストーン情報をいつ破棄できるかを決定する。このタスクは、ローカルの作成／更新バージョンに対する限界を計算し、その後、旧いすべてのツームストーンバージョンを破棄することによりツームストーン情報を切り詰める。

【０２１６】

１１．ヘルパＡＰＩおよび関数

Ｂａｓｅマッピングは、さらに、多数のヘルパ関数も備える。これらの関数は、データモデルに対する共通のオペレーションを補助するために用意されている。

【０２１７】

ａ）関数［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．ＧｅｔＩｔｅｍ

【０２１８】

【表２６】

```
Returns an Item object given an ItemId
//
Item GetItem (ItemId ItemId)
```

【０２１９】

ｂ）関数［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．ＧｅｔＥｘｔｅｎｓｉｏｎ

【０２２０】

【表２７】

```
// Returns an extension object given an ItemId and ExtensionId
//
Extension GetExtension (ItemId ItemId, ExtensionId ExtensionId)
```

【０２２１】

ｃ）関数［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．ＧｅｔＲｅｌａｔｉｏｎｓｈｉｐ

【０２２２】

【表２８】

```
// Returns an relationship object given an ItemId and RelationshipId
//
Relationship GetRelationship (ItemId ItemId, RelationshipId RelationshipId)
```

【０２２３】

１２．メタデータ

ストアで表されるメタデータは、インスタンスメタデータ（Ｉｔｅｍの型など）および型メタデータの２種類がある。

【０２２４】

ａ）スキーマメタデータ

スキーマメタデータは、ＭｅｔａスキーマからのＩｔｅｍ型のインスタンスとしてデータストア内に格納される。

【０２２５】

ｂ）インスタンスメタデータ

インスタンスメタデータは、Ｉｔｅｍの型についてクエリを実行するためにアプリケーションによって使用され、これによりＩｔｅｍに関連付けられている拡張を見つける。Ｉｔｅｍに対するＩｔｅｍＩｄが与えられれば、アプリケーションは、大域的アイテムビューにクエリを実行して、Ｉｔｅｍの型を返し、この値を使用して、Ｍｅｔａ．Ｔｙｐｅビューにクエリを実行し、Ｉｔｅｍの宣言された型に関する情報を返す。例えば、以下のようになる。

【０２２６】

【表２９】

```
// Return metadata Item object for given Item instance
//
SELECT m._Item AS metadataInfoObj
FROM [System.Storage].[Item] i INNER JOIN [Meta].[Type] m ON i._TypeId = m.ItemId
WHERE i.ItemId = @ItemId
```

【０２２７】

Ｅ．セキュリティ

一般に、すべてのセキュリティ設定可能なオブジェクトは、図２６に示されているアクセスマスク形式を使用してそのアクセス権をアレンジする。この形式では、下位１６ビットはオブジェクト特有のアクセス権に、次の７ビットは標準アクセス権に使用され、これは、オブジェクトのほとんどの型に適用され、上位４ビットは各オブジェクト型で標準およびオブジェクト特有の権利の集合にマッピングできる汎用アクセス権を指定するために使用される。ＡＣＣＥＳＳ＿ＳＹＳＴＥＭ＿ＳＥＣＵＲＩＴＹビットは、オブジェクトのＳＡＣＬにアクセスする権利に対応する。

【０２２８】

図２６のアクセスマスク構造では、アイテム特有の権利は「Ｏｂｊｅｃｔ　Ｓｐｅｃｉｆｉｃ　Ｒｉｇｈｔｓ」セクション（下位１６ビット）に置かれる。本発明の実施形態では、ストレージプラットフォームは、セキュリティを管理する２組のＡＰＩ－Ｗｉｎ３２およびストレージプラットフォームＡＰＩ－を公開しているため、ファイルシステムオブジェクト特有の権利は、ストレージプラットフォームオブジェクト特有の権利の設計の動機付けをするために考慮されなければならない。

【０２２９】

本発明のストレージプラットフォームのセキュリティモデルは、本明細書の前の方で参照により組み込まれている関連出願において詳しく説明されている。この点に関して、図２７（パートａ、ｂ、およびｃ）は、セキュリティモデルの一実施形態による、既存のセキュリティ領域から切り出される新しいまったく同じように保護されているセキュリティ領域を示す図である。

【０２３０】
Ｆ．通知および変更追跡
本発明の他の態様によれば、ストレージプラットフォームは、データ変更をアプリケーションで追跡できるようにする通知機能を備える。この機能は、主に、揮発性状態を維持するか、またはデータ変更イベントに関するビジネスロジックを実行するアプリケーション向けである。アプリケーションは、アイテム、アイテム拡張、およびアイテム関係に関する通知を登録する。通知は、データ変更がコミットされた後に非同期に送り出される。アプリケーション側では、アイテム、拡張、および関係型さらにオペレーションの種類により、通知をフィルタ処理することができる。
【０２３１】
一実施形態により、ストレージプラットフォームＡＰＩ　３２２は、２種類のインターフェースを通知用に用意している。第１に、アプリケーションはアイテム、アイテム拡張、およびアイテム関係への変更によりトリガされた単純データ変更イベントを登録する。第２に、アプリケーションは、アイテム、アイテム拡張、およびアイテム間の関係の集まりを監視する「ウォッチャ」オブジェクトを作成する。ウォッチャオブジェクトの状態は、システム障害の後、またはシステムが長時間オフライン状態になってしまった後に、保存され、再作成されるようにできる。単一の通知で、複数の更新を反映する場合がある。
【０２３２】
この機能に関する追加詳細は、本明細書の前の方で参照により組み込まれている関連出願で説明されている。
【０２３３】
Ｇ．従来のファイルシステムの相互運用性
上述のように、本発明のストレージプラットフォームは、少なくとも一部の実施形態では、コンピュータシステムのハードウェア／ソフトウェアインターフェースシステムのなくてはならない一部として具現化されることを意図されている。例えば、本発明のストレージプラットフォームは、Ｍｉｃｒｏｓｏｆｔ　Ｗｉｎｄｏｗｓ（登録商標）ファミリのオペレーティングシステムなどのオペレーティングシステムの重要な一部として具現化されることができる。そのようなキャパシティにおいて、ストレージプラットフォームＡＰＩは、アプリケーションプログラムがオペレーティングシステムとやり取りするために使用するオペレーティングシステムＡＰＩの一部となる。したがって、ストレージプラットフォームは、アプリケーションプログラムがオペレーティングシステムに関する情報を格納するために使用する手段となり、そのため、ストレージプラットフォームのＩｔｅｍベースのデータモデルは、そのようなオペレーティングシステムの従来のファイルシステムの代替えとなる。例えば、Ｍｉｃｒｏｓｏｆｔ　Ｗｉｎｄｏｗｓ（登録商標）ファミリのオペレーティングシステムで具現化されているように、ストレージプラットフォームでは、そのオペレーティングシステムで実装されているＮＴＦＳファイルシステムを置き換えることも可能である。現在、アプリケーションプログラムは、Ｗｉｎｄｏｗｓ（登録商標）ファミリのオペレーティングシステムにより公開されているＷｉｎ３２　ＡＰＩを通じてＮＴＦＳファイルシステムのサービスにアクセスしている。
【０２３４】
しかし、ＮＴＦＳファイルシステムを本発明のストレージプラットフォームで完全に置き換えるには、既存のＷｉｎ３２ベースのアプリケーションプログラムをコーディングし直す必要があること、またそのようなコーディングし直しは望ましくないことを理解すると、本発明のストレージプラットフォームでＮＴＦＳなどの既存のファイルシステムとの何らかの相互運用性を実現することが有益であろう。したがって、本発明の一実施形態では、ストレージプラットフォームにおいて、Ｗｉｎ３２プログラミングモデルに依存するアプリケーションプログラムはストレージプラットフォームのデータストアと従来のＮＴＦＳファイルシステムの両方の内容にアクセスすることができる。この目的のために、ストレージプラットフォームでは、簡単な相互運用性を高めるＷｉｎ３２の命名規則の上位集合となる命名規則を使用する。さらに、ストレージプラットフォームでは、Ｗｉｎ３２

ＡＰＩを通じてストレージプラットフォームボリューム内に格納されているファイルおよびディレクトリのアクセスをサポートする。

【０２３５】

この機能に関する追加詳細は、本明細書の前の方で参照により組み込まれている関連出願で説明されている。

【０２３６】

Ｈ．ストレージプラットフォームＡＰＩ

ストレージプラットフォームは、アプリケーションプログラム側で上述のストレージプラットフォームの機能および能力にアクセスし、データストアに格納されているアイテムにアクセスするために使用できるＡＰＩを備える。このセクションでは、本発明のストレージプラットフォームのストレージプラットフォームＡＰＩの一実施形態について説明する。この機能に関する詳細は、本明細書の前の方で参照により組み込まれている関連出願で説明されているが、便宜のため以下にこの情報の一部をまとめた。

【０２３７】

図１８を参照すると、Ｃｏｎｔａｉｎｍｅｎｔ　Ｆｏｌｄｅｒは他のＩｔｅｍｓとの保持Ｒｅｌａｔｉｏｎｓｈｉｐｓを含むアイテムであり、ファイルシステムのフォルダの共通概念の等価物である。それぞれのＩｔｅｍは少なくとも１つの包含フォルダ内に「含まれる」。

【０２３８】

図１９は、本発明の一実施形態による、ストレージプラットフォームＡＰＩの基本アーキテクチャを例示する。ストレージプラットフォームＡＰＩでは、ＳＱＬＣＩｉｅｎｔ１９００を使用してローカルデータストア３０２とやり取りし、またＳＱＬＣｌｉｅｎｔ１９００を使用してリモートデータストア（例えば、データストア３４０）とやり取りすることができる。ローカルストア３０２は、さらに、ＤＱＰ（分散クエリプロセッサ）を使用するか、または後述のストレージプラットフォーム同期サービス（「Ｓｙｎｃ」）を通じて、リモートデータストア３４０ともやり取りできる。ストレージプラットフォームＡＰＩ　３２２は、さらに、データストア通知用のブリッジＡＰＩとして機能し、上述のように、アプリケーションのサブスクリプションを通知エンジン３３２に受け渡し、通知をアプリケーション（例えば、アプリケーション３５０ａ、３５０ｂ、または３５０ｃ）にルーティングする。一実施形態では、ストレージプラットフォームＡＰＩ　３２２は、さらに、Ｍｉｃｒｏｓｏｆｔ　ＥｘｃｈａｎｇｅおよびＡＤでデータにアクセスできるように制限された「プロバイダ」アーキテクチャを定義することもできる。

【０２３９】

図２０は、ストレージプラットフォームＡＰＩの様々なコンポーネントを表す概略図である。ストレージプラットフォームＡＰＩは、（１）ストレージプラットフォーム要素およびアイテム型を表す、データクラス２００２、（２）オブジェクト永続性を管理し、サポートクラス２００６を提供する、ランタイムフレームワーク２００４、および（３）ストレージプラットフォームスキーマからＣＬＲクラスを生成するために使用される、ツール２００８の各コンポーネントからなる。

【０２４０】

与えられたスキーマから生じるクラスの階層は、直接、そのスキーマ内の型の階層を反映する。例えば、図２１Ａおよび図２１Ｂに示されているようなＣｏｎｔａｃｔｓスキーマで定義されているＩｔｅｍ型を考察する。

【０２４１】

図２２は、動作中のランタイムフレームワークを例示している。ランタイムフレームワークは以下のように動作する。

１．アプリケーション３５０ａ、３５０ｂ、または３５０ｃは、ストレージプラットフォーム内のアイテムにバインドする。

２．フレームワーク２００４は、バインドされたアイテムに対応するＩｔｅｍＣｏｎｔｅｘｔオブジェクト２２０２を作成し、アプリケーションに返す。

３．アプリケーションは、このＩｔｅｍＣｏｎｔｅｘｔ上でＦｉｎｄをサブミットして、Ｉｔｅｍｓのコレクションを取得し、返されるコレクションは、概念上オブジェクトグラフ２２０４となる（関係により）。

４．アプリケーションは、データを変更、削除、および挿入する。

５．アプリケーションは、Ｕｐｄａｔｅ（）メソッドを呼び出して変更を保存する。

【０２４２】

図２３は、「ＦｉｎｄＡｌｌ」オペレーションの実行を例示する。

【０２４３】

図２４は、ストレージプラットフォームＡＰＩクラスがストレージプラットフォームＳｃｈｅｍａから生成されるプロセスを例示する。

【０２４４】

図２５は、Ｆｉｌｅ　ＡＰＩが基づくスキーマを例示している。ストレージプラットフォームＡＰＩは、ファイルオブジェクトを取り扱うための名前空間を含む。この名前空間は、Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ．Ｆｉｌｅｓと呼ばれる。Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ．Ｆｉｌｅｓのクラスのデータメンバは、ストレージプラットフォームストアに格納されている情報を直接反映し、この情報は、ファイルシステムオブジェクトから「格上げ」されるか、またはＷｉｎ３２　ＡＰＩを使用してネイティブな形で作成することができる。Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ．Ｆｉｌｅｓ名前空間は、ＦｉｌｅＩｔｅｍおよびＤｉｒｅｃｔｏｒｙＩｔｅｍの２つのクラスを持つ。これらのクラスおよびそのメソッドのメンバは、図２５のスキーマ図を見ると容易に推測できる。ＦｉｌｅＩｔｅｍおよびＤｉｒｅｃｔｏｒｙＩｔｅｍは、ストレージプラットフォームＡＰＩからは読み取り専用である。それらを修正するためには、Ｗｉｎ３２　ＡＰＩを使用するか、またはＳｙｓｔｅｍ．ＩＯのクラスを使用する必要がある。

【０２４５】

ＡＰＩに関して、プログラミングインターフェース（またはより単純に、インターフェース）は、コードの１つまたは複数のセグメントがコードの１つまたは複数の他のセグメントにより提供される機能と通信またはアクセスできるようにするメカニズム、プロセス、プロトコルとしてみなすことができる。代替えとして、プログラミングインターフェースは、他の（複数の）コンポーネントの１つまたは複数のメカニズム、メソッド、関数呼び出し、モジュールなどに通信するように結合することができるシステムのコンポーネントの１つまたは複数のメカニズム、メソッド、関数呼び出し、モジュール、オブジェクトなどとみなすことができる。前文の「ｓｅｇｍｅｎｔ　ｏｆ　ｃｏｄｅ」という用語は、適用される用語、またはコードセグメントが別々にコンパイルされるかどうか、コードセグメントがソースコード、中間コード、またはオブジェクトコードとして供給されるかどうか、コードセグメントがランタイムシステムまたはプロセスで利用されるかどうか、それらが同じまたは異なるマシン上に配置されるか、または複数のマシンにまたがって分散されるかどうか、コードのセグメントにより表される機能がソフトウェアで全部実装されるのか、ハードウェアで全部実装されるのか、それともハードウェアとソフトウェアの組合せで実装されるのかに関係なく、１つまたは複数の命令またはコード行を含むことを意図しており、例えば、コードモジュール、オブジェクト、サブルーチン、関数などを含む。

【０２４６】

概念上、プログラミングインターフェースは、図３０Ａまたは図３０Ｂに示されているように、総称的に示すことができる。図３０Ａは、第１および第２のコードセグメントが通信に使用する情報伝達ルートとしてインターフェースＩｎｔｅｒｆａｃｅ１を例示している。図３０Ｂは、システムの第１および第２のコードセグメントが媒体Ｍを介して通信できるようにするインターフェースオブジェクトＩ１およびＩ２（第１および第２のコードセグメントの一部である場合もない場合もある）を含むものとしてインターフェースを例示している。図３０Ｂを見ると、インターフェースオブジェクトＩ１およびＩ２は同じシステムの別のインターフェースとして考えることができ、またオブジェクトＩ１および

Ｉ２さらに媒体Ｍはそのインターフェースを含むものとして考えることもできる。図３０Ａおよび３０Ｂは、双方向の流れおよびその流れのいずれかの側のインターフェースを示しているが、いくつかの実装では、一方向のみ情報の流れがある（または後述のようにまったく情報の流れがない）か、または片側にのみインターフェースオブジェクトがある場合がある。例えば、限定はしないが、アプリケーションプログラミングインターフェース（ＡＰＩ）、エントリポイント、メソッド、関数、サブルーチン、リモートプロシージャ呼び出し、およびコンポーネントオブジェクトモデル（ＣＯＭ）インターフェースなどの用語は、プログラミングインターフェースの定義内に包含される。

【０２４７】

このようなプログラミングインターフェースの複数の態様は、第１のコードセグメントが情報（「情報」は、最も広い意味で使用されており、データ、コマンド、要求などを含む）を第２のコードセグメントに送信するためのメソッド、第２のコードセグメントがその情報を受け取るためのメソッド、および情報の構造、系列、構文、編成、スキーマ、タイミング、および内容を含むことができる。この点に関して、基礎となる搬送媒体自体は、媒体が有線であろうと無線であろうと、両方の組合せであろうと、情報がインターフェースにより定義された方法で搬送される限り、そのインターフェースのオペレーションにとっては重要でないと考えられる。いくつかの状況では、情報は、従来の意味で一方向または双方向で受け渡されない場合があるが、それは、１つのコードセグメントが単に第２のコードセグメントにより実行される機能にアクセスする場合のように、情報転送が他のメカニズム（例えば、情報がコードセグメント間の情報の流れとは別のバッファ、ファイルなどに置かれる）を介するか、または存在しない場合があるからである。これらの態様のどれかまたは全部は、例えばコードセグメントが疎結合または密結合の構成のシステムの一部かどうかに応じて、与えられた状況では重要である場合もあり、そのため、このリストは説明を目的としているのであって、制限することを意図していないと考えるべきである。

【０２４８】

プログラミングインターフェースのこの概念は、当業者には知られており、本発明の前述の詳細な説明から明らかである。しかし、プログラミングインターフェースを実装する方法は他にもあり、特に断らない限り、これらも、本明細書に付属する請求項により取り込まれることが意図されている。このような他の方法は、図３０Ａおよび３０Ｂの簡略化した図よりも詳しい、または複雑なものとして現れる場合があるが、そうであっても、総体的に同じ結果が得られる類似の関数を実行する。そこで、プログラミングインターフェースのいくつかの説明的な他の実装について簡単に述べることにする。

【０２４９】

因数分解：一方のコードセグメントから他方のコードセグメントへの通信は、通信を複数の離散的通信に分割することにより間接的に実行されるようにできる。これは、図３１Ａおよび３１Ｂに概略が示されている。図に示されているように、いくつかのインターフェースは、機能の分割可能な集合に関して説明することができる。そこで、図３０Ａおよび３０Ｂのインターフェース機能は、ちょうど２４、つまり２×２×３×２と数学的に示すことができるのと同じ結果が得られるように因数分解することができる。したがって、図３１Ａに例示されているように、インターフェースＩｎｔｅｒｆａｃｅ１により提供される関数を細分し、インターフェースの通信を複数のインターフェースＩｎｔｅｒｆａｃｅ１Ａ、Ｉｎｔｅｒｆａｃｅ１Ｂ、Ｉｎｔｅｒｆａｃｅ１Ｃに変換し、しかも同じ結果が得られるようにすることができる。図３１Ｂに例示されているように、インターフェースＩ１により提供される関数を複数のインターフェースＩ１ａ、Ｉ１ｂ、Ｉ１ｃに細分し、しかも同じ結果が得られるようにすることができる。同様に、第１のコードセグメントから情報を受け取る第２のコードセグメントのインターフェースＩ２は、複数のインターフェースＩ２ａ、Ｉ２ｂ、Ｉ２ｃなどに因数分解することができる。因数分解の際に、第１のコードセグメントとともに含まれるインターフェースの個数は、第２のコードセグメントとともに含まれるインターフェースの個数と一致している必要はない。図３１Ａおよび

３１Ｂの場合のいずれも、インターフェースＩｎｔｅｒｆａｃｅ１およびＩ１の機能の本質は、それぞれ、図３０Ａおよび３０Ｂの場合と同じままである。インターフェースの因数分解は、さらに、結合的特性、可換特性、およびその他の数学的特性にも従い、因数分解は理解しにくい場合がある。例えば、オペレーションの順序は重要でない場合があり、そのため、インターフェースにより実行される関数は、コードまたはインターフェースの他の断片により、そのインターフェースに到達するいくらか前に実行されるか、またはシステムの別のコンポーネントにより実行されることができる。さらに、プログラミング技術の通常の技能を有する者であれば、同じ結果を出す異なる関数呼び出しを実行する方法として様々な方法があることを理解できるであろう。

【０２５０】

再定義：場合によっては、意図した結果をそのまま達成しながら、プログラミングインターフェースのいくつかの態様（例えば、パラメータ）を無視、追加、または再定義することが可能な場合がある。これは、図３２Ａおよび３２Ｂに例示されている。例えば、図３０Ａのインターフェースｉｎｔｅｒｆａｃｅ１が関数呼び出しＳｑｕａｒｅ（ｉｎｐｕｔ，ｐｒｅｃｉｓｉｏｎ，ｏｕｔｐｕｔ）を含み、呼び出しは、３つのパラメータｉｎｐｕｔ、ｐｒｅｃｉｓｉｏｎ、およびｏｕｔｐｕｔを含み、第１のＣｏｄｅ　Ｓｅｇｍｅｎｔから第２のＣｏｄｅ　Ｓｅｇｍｅｎｔに発行されると仮定する。真ん中のパラメータｐｒｅｃｉｓｉｏｎが与えられたシナリオにおいて無関係であれば、図３２Ａに示されているように、ただ無視するか、さらには意味のない（この状況では）パラメータと置き換えることが可能である。また、関係のない追加パラメータを加えることも可能である。いずれにせよ、平方の機能は、第２のコードセグメントにより入力が平方された後、出力が返される限り、達成されうる。ｐｒｅｃｉｓｉｏｎは、コンピューティングシステムの何らかの下流または他の部分にとって意味のあるパラメータである可能性も十分ありえるが、平方を計算するという狭い目的についてはｐｒｅｃｉｓｉｏｎは不要であることがわかれば、置き換えるか、または無視することができる。例えば、有効な精度値を受け渡す代わりに、誕生日の日付など意味のない値を渡しても、結果に悪影響を及ぼさないであろう。同様に、図３２Ｂに示されているように、インターフェースＩ１をインターフェースＩ１’で置き換えて、再定義して、そのインターフェースへのパラメータを無視するか、または追加する。インターフェースＩ２は、同様に、インターフェースＩ２’と再定義することができ、不要なパラメータ、または他のところで処理することができるパラメータを無視するように再定義することができる。ここでの要点は、場合によっては、プログラミングインターフェースは、ある目的には必要のない、パラメータなどの態様を含むことがあり、したがって、それらを無視または再定義するか、または他の目的のために他の場所で処理することができる。

【０２５１】

インラインコーディング：また、間に入る「インターフェース」が形態を変更するように２つの別々のコードモジュールの機能の一部または全部をマージすることが可能な場合がある。例えば、図３０Ａおよび３０Ｂの機能は、それぞれ、図３３Ａおよび３３Ｂの機能に変換することができる。図３３Ａでは、図３０Ａの前の第１および第２のコードセグメントは、それらの両方を含む１つのモジュールにマージされる。この場合、コードセグメントは、そのまま、他のインターフェースと通信していることが可能であるが、インターフェースを、単一モジュールにより好適な形態に適合させることができる。そこで、例えば、形式的ＣａｌｌおよびＲｅｔｕｒｎステートメントは、もはや、必要なくなるが、インターフェースＩｎｔｅｒｆａｃｅ１による類似の処理または応答はいぜんとして有効である。同様に、図３３Ｂに示されているように、図３０Ｂからのインターフェース Ｉ２の一部（または全部）を、インラインでインターフェースＩ１に書き込んで、インターフェースＩ１”を形成することができる。例示されているように、インターフェースＩ２はＩ２ａとＩ２ｂに分割され、インターフェース部Ｉ２ａは、インターフェースＩ１とインラインでコーディングされており、インターフェースＩ１”を形成する。具体的な例として、図３０Ｂのインターフェース Ｉ１は、インターフェースＩ２によって受け取られ、第

２のコードセグメントにより（平方するために）入力とともに渡された値を処理した後、平方された結果を出力とともに戻す、関数呼び出しｓｑｕａｒｅ（ｉｎｐｕｔ，ｏｕｔｐｕｔ）を実行すると考える。このよう場合、第２のコードセグメントによりされる処理（入力の平方）は、インターフェースを呼び出さずに、第１のコードセグメントにより実行することができる。

【０２５２】

分離：一方のコードセグメントから他方のコードセグメントへの通信は、通信を複数の離散的通信に分割することにより間接的に実行されるようにできる。これは、図３４Ａおよび３４Ｂに概略が示されている。図３４Ａに示されているように、ミドルウェアの１つまたは複数の断片（（複数の）Ｄｉｖｏｒｃｅ　Ｉｎｔｅｒｆａｃｅ。機能および／またはインターフェース関数をオリジナルのインターフェースから分離するので）が用意されており、これにより、別のインターフェース、この場合インターフェースＩｎｔｅｒｆａｃｅ２Ａ、Ｉｎｔｅｒｆａｃｅ２Ｂ、およびＩｎｔｅｒｆａｃｅ２Ｃに適合するように第１のインターフェースＩｎｔｅｒｆａｃｅ１上の通信を変換する。これは、例えば、Ｉｎｔｅｒｆａｃｅ１プロトコルに従ってオペレーティングシステムと通信するように設計されているアプリケーションのインストールベースがあるが、そのときに、オペレーティングシステムは異なるインターフェース、この場合、インターフェースＩｎｔｅｒｆａｃｅ２Ａ、Ｉｎｔｅｒｆａｃｅ２Ｂ、およびＩｎｔｅｒｆａｃｅ２Ｃを使用するように変更される場合に行うことが可能である。この要点は、第２のコードセグメントにより使用されるオリジナルのインターフェースが変更され、第１のコードセグメントにより使用されるインターフェースと互換性がとれなくなり、そのため、媒介を使用して旧インターフェースと新インターフェースとの互換性をとるという点である。同様に、図３４Ｂに示されているように、第３のコードセグメントを、インターフェースＩ１からの通信を受け取るために分離インターフェースＤＩ１とともに導入し、インターフェース機能を例えばＤＩ２と連携するが、同じ機能的結果を供給するように再設計されたインターフェースＩ２ａおよびＩ２ｂに送るために分離インターフェースＤＩ２とともに導入することができる。同様に、ＤＩ１およびＤＩ２は、同じまたは類似の機能的結果が得られるようにしながら、連携して動作し、図３０Ｂのインターフェース１１およびＩ２の機能を新しいオペレーティングシステムに翻訳することができる。

【０２５３】

書き換え：さらに他の可能な変更形態として、コードを動的に書き換えることで、インターフェース機能を他の何かの機能で置き換えるが、ただし全体として同じ結果が得られるようにする方法がある。例えば、中間言語（例えば、Ｍｉｃｒｏｓｏｆｔ　ＩＬ、Ｊａｖａ（登録商標）ＢｙｔｅＣｏｄｅなど）で書かれたコードセグメントを実行環境（．Ｎｅｔフレームワーク、Ｊａｖａ（登録商標）ランタイム環境、または他の類似のランタイム型環境によって実現されるような）内のジャストインタイム（ＪＩＴ）コンパイラまたはインタプリタに送るシステムもある。ＪＩＴコンパイラは、第１のコードセグメントから第２のコードセグメントに通信を動的に変換する、つまり、第２のコードセグメント（オリジナルまたは異なる第２のコードセグメントのいずれか）が必要とするとおりにそれらを異なるインターフェースに適合させるように書くことができる。これは、図３５Ａおよび３５Ｂに示されている。図３５Ａからわかるように、このアプローチは上述の分離シナリオに似ている。これは、例えば、アプリケーションのインストールベースがＩｎｔｅｒｆａｃｅ１プロトコルに従ってオペレーティングシステムと通信するように設計されているが、そのときに、オペレーティングシステムは異なるインターフェースを使用するように変更される場合に行うことが可能である。ＪＩＴコンパイラは、インストールベースのアプリケーションからオペレーティングシステムの新しいインターフェースへ通信をオンザフライで適合させる場合に使用することが可能である。図３５Ｂに示されているように、（複数の）インターフェースを動的に書き換えるこのアプローチを適用することで、動的に因数分解するか、または他の何らかの手段により、（複数の）インターフェースも変更することができる。

【０２５４】
他の実施形態を介してインターフェースと同じまたは類似の結果を得る上述のシナリオは、さらに、様々な方法で、直列に、および／または並列に、または他の介入コードを使用して、組み合わせることもできることに注意すべきである。こうして、上記の他の実施形態は、相互排他的ではなく、図３０Ａおよび３０Ｂに示されている汎用シナリオと同じまたは同等のシナリオを生成するために混合し、照合し、組み合わせることができる。また、ほとんどのプログラミング構文の場合と同様、本明細書で説明できないが、それでも、本発明の精神および範囲により表される、インターフェースの同じまたは類似の機能を実現する他の類似の方法があることにも注意されたい、つまり、少なくとも一部は、インターフェースの値の基礎となるインターフェースによって表される機能および有効にされる有利な結果であることに注意されたい。
【０２５５】
ＩＩＩ．同期ＡＰＩ
Ｉｔｅｍベースのハードウェア／ソフトウェアインターフェースシステムにおいて同期に対するいくつかのアプローチが考えられる。セクションＡでは、本発明のいくつかの実施形態を開示し、セクションＢでは、同期に関してＡＰＩの様々な実施形態に注目する。
【０２５６】
Ａ．同期の概要
本発明のいくつかの実施形態において、図３に関し、ストレージプラットフォームは、（ｉ）ストレージプラットフォーム（それぞれ自データストア３０２を備える）の複数のインスタンスが柔軟な規則の集まりに従ってその内容の部分の同期をとれるようにし、（ｉｉ）本発明のストレージプラットフォームのデータストアと専用プロトコルを実装する他のデータソースとの同期をとるサードパーティ用のインフラストラクチャを備える同期サービス３３０を提供する。
【０２５７】
ストレージプラットフォームとストレージプラットフォームとの間の同期は、参加「レプリカ」のグループのうちで実行される。例えば、図３を参照すると、おそらく異なるコンピュータシステム上で稼働しているであろう、ストレージプラットフォームの他のインスタンスの制御の下で、ストレージプラットフォーム３００のデータストア３０２と他のリモートデータストア３０８との同期を与えることが望ましい場合がある。このグループの全体的な帰属関係は、必ずしも、与えられた時刻で与えられたレプリカに知られているわけではない。
【０２５８】
異なるレプリカは、変更を独立して行うことができる（つまり、同時に）。同期処理のプロセスは、他のレプリカによって行われる変更をすべてのレプリカに気づかせることとして定義される。この同期機能は、本質的に、マルチマスターである。
【０２５９】
本発明の同期機能では、レプリカは以下のようにできる。
・　他のレプリカがどの変更を認識するかを決定する。
・　このレプリカが認識しない変更に関する情報を要求する。
・　他のレプリカが認識しない変更に関する情報を伝達する。
・　２つの変更がいつ互いに競合するかを決定する。
・　変更をローカルで適用する。
・　競合する解決を他のレプリカに伝達し、収束を保証する。
・　競合する解決に対する指定されたポリシーに基づき競合状態を解決する。
【０２６０】
１．ストレージプラットフォームとストレージプラットフォームとの間の同期処理
本発明のストレージプラットフォームの同期サービス３３０の主要な適用は、ストレージプラットフォームの複数のインスタンスの同期をとることである（それぞれ、自データストアを有する）。同期サービスは、ストレージプラットフォームのスキーマレベルで動

作する（データベースエンジン３１４の基礎のテーブルではなく）。したがって、例えば、「Ｓｃｏｐｅｓ」は、後述のように同期を定義するために使用される。

【０２６１】

同期サービスは、「純変化」の原理に基づいて動作する。同期サービスでは、個別のオペレーションを（トランザクション複製などにより）記録し、送信するのではなく、それらのオペレーションの最終結果を送り、そのため、複数のオペレーションの結果を得られた単一の変更に統合することが多い。

【０２６２】

同期サービスは、一般的にはトランザクション境界を尊重しない。つまり、単一のトランザクションでストレージプラットフォームのデータストアに２つの変更が加えられた場合、それらの変更が他のすべてのレプリカに最小単位として適用されることは保証されない－一方が他方なしで現れることがある。この原理の例外は、同じトランザクションで２つの変更が同じＩｔｅｍに加えられた場合に、それらの変更は送信され、他のレプリカに最小単位として適用されることが保証されるという点である。したがって、Ｉｔｅｍｓは、同期サービスの整合性ユニットである。

【０２６３】

ａ）同期（Ｓｙｎｃ）制御アプリケーション

どのアプリケーションも、同期サービスに接続し、同期処理オペレーションを開始することができる。そのようなアプリケーションは、同期処理を実行するために必要なすべてのパラメータを用意する（以下のｓｙｎｃプロファイルを参照）。このようなアプリケーションは、本明細書では、同期制御アプリケーション（ＳＣＡ）と呼ばれる。

【０２６４】

２つのストレージプラットフォームインスタンスの同期をとる場合、ｓｙｎｃはＳＣＡにより片方の側で開始される。そのＳＣＡは、リモートパートナと同期することをローカル同期サービスに通知する。他方の側では、同期サービスは、発信側マシンから同期サービスにより送られるメッセージによって目覚める。これは、送信先マシン上に存在する永続的構成情報（以下のマッピングを参照）に基づいて応答する。同期サービスは、スケジュールに従って実行されるか、またはイベントに対する応答として実行されることが可能である。これらの場合に、スケジュールを実行する同期サービスがＳＣＡとなる。

【０２６５】

同期処理を有効にするには、２つのステップが実行される必要がある。第１に、スキーマデザイナが適切なｓｙｎｃの意味でストレージプラットフォームスキーマに注釈を入れなければならない（後述のようにＣｈａｎｇｅ　Ｕｎｉｔｓを指定する）。第２に、同期処理は、同期処理に関わるストレージプラットフォームのインスタンスを持つすべてのマシン上で適切に構成されなければならない（後述のように）。

【０２６６】

ｂ）スキーマ注釈

同期サービスの基本概念は、Ｃｈａｎｇｅ　Ｕｎｉｔの概念である。Ｃｈａｎｇｅ　Ｕｎｉｔは、ストレージプラットフォームにより個別に追跡されるスキーマ最小断片である。すべてのＣｈａｎｇｅ　Ｕｎｉｔについて、同期サービスは、最後のｓｙｎｃ以降に変化したか、変化しなかったかを判別することができる。

【０２６７】

スキーマでＣｈａｎｇｅ　Ｕｎｉｔｓを指定することは、複数の目的に使用される。第１に、これは、同期サービスが回線上でどれだけ混雑しているかを判別する。Ｃｈａｎｇｅ　Ｕｎｉｔ内で変更が加えられた場合、同期サービスはＣｈａｎｇｅ　Ｕｎｉｔのどの部分が変更されたかを認識していないので、Ｃｈａｎｇｅ　Ｕｎｉｔ全体が他のレプリカに送信される。第２に、これは、競合検出の精度を決定する。２つの同時変更（これらの用語は、後のセクションで詳しく定義される）が同じＣｈａｎｇｅ　Ｕｎｉｔに対し加えられた場合、同期サービスは競合の通知を発し、その一方で、同時変更が異なるＣｈａｎｇｅ　Ｕｎｉｔｓに加えられた場合、競合の通知は発せず、変更は自動的にマージされる

。第３に、これは、システムによって保持されているメタデータの量を強く左右する。同期サービスのメタデータの多くは、Ｃｈａｎｇｅ　Ｕｎｉｔ毎に保持され、そのため、Ｃｈａｎｇｅ　Ｕｎｉｔｓを小さくすれば、ｓｙｎｃのオーバーヘッドが大きくなる。

【０２６８】

Ｃｈａｎｇｅ　Ｕｎｉｔｓを定義するには、正しいトレードオフを見つける必要がある。そういうわけで、同期サービスを使用すると、スキーマデザイナはそのプロセスに関与することができる。

【０２６９】

一実施形態では、同期サービスは、要素よりも大きなＣｈａｎｇｅ　Ｕｎｉｔｓをサポートしていない。しかし、スキーマデザイナが要素よりも小さいＣｈａｎｇｅ　Ｕｎｉｔｓを指定する能力－つまり、要素の複数の属性を１つの独立したＣｈａｎｇｅ　Ｕｎｉｔｓにグループ化すること－はサポートしている。その実施形態では、これは、以下の構文を使用して行われる。

【０２７０】

【表３０】

```
<Type Name="Appointment" MajorVersion="1" MinorVersion="0"
      ExtendsType="Base.Item"                    ExtendsVersion="1">

   <Field Name="MeetingStatus“  Type="the storage platformTypes.uniqueidentifier
      Nullable="False"/>
   <Field Name="OrganizerName“ Type="the storage platformTypes.nvarchar(512)"
      Nullable="False"/>
   <Field Name="OrganizerEmail“  Type="the storage platformTypes.nvarchar(512)"
      TypeMajorVersion="1“                    MultiValued="True"/>
    …

   <ChangeUnit Name=”CU_Status”>
       <Field Name=”MeetingStatus”/>
   </ChangeUnit>

   <ChangeUnit Name=”CU_Organizer”/>
       <Field Name=”OrganizerName” />
       <Field Name=”OrganizerEmail” />
   </ChangeUnit>
   …

</Type>
```

【０２７１】

ｃ）同期構成

データの特定の部分の同期を維持したいストレージプラットフォームパートナのグループは、ｓｙｎｃコミュニティと呼ばれる。コミュニティのメンバが同期を維持したい場合、必ずしも、まったく同じ方法でデータを表すわけではない、つまり、ｓｙｎｃパートナは、同期処理しているデータを変換することができる。

【０２７２】

ピアツーピアのシナリオでは、ピアがそのパートナのすべてについて変換マッピングを維持することは実際的ではない。そうする代わりに、同期サービスは、「コミュニティフ

ォルダ」を定義するアプローチをとる。コミュニティフォルダは、すべてのコミュニティメンバが同期をとる仮想「共有フォルダ」を表す抽象化である。

【０２７３】

この概念は、例を使うと最もよくわかる。Ｊｏｅが複数のコンピュータのＭｙ　Ｄｏｃｕｍｅｎｔｓフォルダの同期をとりたい場合、Ｊｏｅは、例えば、ＪｏｅｓＤｏｃｕｍｅｎｔｓというコミュニティフォルダを定義する。次に、すべてのコンピュータ上で、Ｊｏｅは仮想ＪｏｅｓＤｏｃｕｍｅｎｔｓフォルダとローカルのＭｙ　Ｄｏｃｕｍｅｎｔｓフォルダとの間のマッピングを構成する。この時点以降、Ｊｏｅのコンピュータが互いに同期した場合、そのローカルアイテムではなく、ＪｏｅｓＤｏｃｕｍｅｎｔｓ内のドキュメントに関してやり取りすることになる。このようにして、すべてのＪｏｅのコンピュータは相手が誰であるかを知らなくても互いに理解する－そのコミュニティフォルダが同期コミュニティの共通語となるのである。

【０２７４】

同期サービスの構成は、（１）ローカルフォルダとコミュニティフォルダとの間のマッピングを定義するステップ、（２）何が同期されるか（例えば、同期する相手、送信すべき部分集合、および受信すべきもの）を決定するｓｙｎｃプロファイルを定義するステップ、および（３）異なるｓｙｎｃプロファイルが実行されるスケジュールを定義する、または手動で実行するステップの３つのステップからなる。

【０２７５】

（１）コミュニティフォルダ－マッピング

コミュニティフォルダマッピングは、個々のマシン上にＸＭＬ構成ファイルとして格納される。それぞれのマッピングは、以下のスキーマを持つ。

/mappings/communityFolder

この要素は、このマッピングの対象となるコミュニティフォルダの名前を指定する。名前はフォルダの構文規則に従う。

【０２７６】

/mappings/localFolder

この要素は、このマッピングの変換先のローカルフォルダの名前を指定する。名前はフォルダの構文規則に従う。フォルダは、マッピングが有効であるためにすでに存在していなければならない。このフォルダ内のアイテムは、このマッピングに従って同期対象とみなされる。

【０２７７】

/mappings/transformations

この要素は、アイテムをコミュニティフォルダからローカルフォルダに、またその逆方向に変換する方法を定義する。存在しない、または空の場合、変換は実行されない。特に、これは、ＩＤがマッピングされないことを意味する。この構成は、主に、Ｆｏｌｄｅｒのキャッシュを作成する際に使用される。

【０２７８】

/mappings/transformations/mapIDs

この要素は、コミュニティＩＤを再利用するのではなく、新しく生成されたローカルＩＤがコミュニティフォルダからマッピングされたアイテムのすべてに割り当てられることを要求する。Ｓｙｎｃ　Ｒｕｎｔｉｍｅは、アイテムの往復変換を行うＩＤマッピングを保持する。

【０２７９】

/mappings/transformations/localRoot

この要素は、コミュニティフォルダ内のすべてのルートアイテムが指定されたルートの子にされることを要求する。

【０２８０】

/mappings/runAs

この要素は、このマッピングに対する要求が処理される際の権限を制御する。存在しな

い場合、送信者が想定される。

【０２８１】

/mappings/runAs/sender

この要素が存在すると、このマッピングへのメッセージの送信者は、偽装され、要求はその信用証明書に従って処理されなければならない。

【０２８２】

（２）プロファイル

同期プロファイルは、同期をキックオフするために必要なパラメータ一式である。これは、ＳＣＡにより、同期処理を開始するためＳｙｎｃ　Ｒｕｎｔｉｍｅによって供給される。ストレージプラットフォームとストレージプラットフォームとの間の同期をとるための同期プロファイルは、以下の情報を含む。

・　Ｌｏｃａｌ　Ｆｏｌｄｅｒ、変更の送り元および送り先として使用される。

・　同期をとるＲｅｍｏｔｅ　Ｆｏｌｄｅｒ名－このＦｏｌｄｅｒは、上で定義されたとおりにマッピングを使用してリモートパートナからパブリッシュされなければならない。

・　Ｄｉｒｅｃｔｉｏｎ－同期サービスは、送信のみ、受信のみ、および送受信同期をサポートする。

・　Ｌｏｃａｌ　Ｆｉｌｔｅｒ－リモートパートナに送信するローカル情報を選択する。ローカルフォルダに対するストレージプラットフォームクエリとして表される。

・　Ｒｅｍｏｔｅ　Ｆｉｌｔｅｒ－リモートパートナから取り出すリモート情報を選択する－コミュニティフォルダに対するストレージプラットフォームクエリとして表される。

・　Ｔｒａｎｓｆｏｒｍａｔｉｏｎｓ－ローカル形式のアイテムの変換方法を定義する。

・　ローカルセキュリティ－リモート終点から取り出された変更がリモート終点（偽装）の許可を得て適用するか、またはローカルで同期を開始したユーザの許可を得て適用するかを指定する。

・　競合解決ポリシー－競合が拒絶されるか、ログに記録されるか、または自動的に解決されるかを指定する－後者の場合、使用する競合リゾルバとともにそれに対する構成パラメータを指定する。

【０２８３】

同期サービスは、同期プロファイルの単純構築を可能にするランタイムＣＬＲクラスを提供する。プロファイルは、さらに、格納を簡単に行えるようにＸＭＬファイルとの間のシリアライズも行える（スケジュールと一緒であるのが多い）。しかし、すべてのプロファイルが格納されるストレージプラットフォーム内の標準的な場所はなく、ＳＣＡは、それを永続せずにその場でプロファイルを構築できるため願ってもないものである。同期を開始するためにローカルマッピングを用意する必要がないことに注意されたい。すべての同期情報は、プロファイル内に指定することができる。しかし、マッピングは、リモート側で開始された同期要求に応答するために必要である。

【０２８４】

（３）スケジュール

一実施形態では、同期サービスは、それ専用のスケジューリングインフラストラクチャを用意しない。その代わりに、このタスクを実行するために他のコンポーネントに頼る－Ｍｉｃｒｏｓｏｆｔ　Ｗｉｎｄｏｗｓ（登録商標）オペレーティングシステムに付属するＷｉｎｄｏｗｓ（登録商標）Ｓｃｈｅｄｕｌｅｒを使用する。同期サービスは、ＳＣＡとして動作し、ＸＭＬファイルに保存されている同期プロファイルに基づいて同期をトリガするコマンドラインユーティリティを備える。このユーティリティを使用すると、スケジュールに従って、またはユーザがログオンまたはログオフするなどのイベントへの応答として同期処理を実行するようにＷｉｎｄｏｗｓ（登録商標）Ｓｃｈｅｄｕｌｅｒを非常に簡単に構成できる。

【０２８５】
ｄ）競合回避処理
同期サービスにおける競合処理は、（１）変更適用時に実行される、競合検出－このステップでは、変更が安全に適用可能か判別する、（２）自動競合解決およびログ記録－このステップでは（競合が検出された直後に実行される）、競合を解決できるか自動競合リゾルバに問い合わせがゆき、できなければオプションで競合をログに記録できる、（３）競合検査および解決－このステップは、何らかの競合がログに記録されている場合に実行され、同期セッションの状況の外部で実行されるが、このときに、ログに記録された競合は解決され、ログから削除されるようにできる－の３つの段階に分けられる。
【０２８６】
（１）競合検出
本発明の実施形態では、同期サービスは、知識ベースと制約ベースの２種類の競合を検出する。
【０２８７】
（ａ）知識ベースの競合
知識ベースの競合は、２つのレプリカが同じＣｈａｎｇｅ　Ｕｎｉｔに対し独立の変更を加えたときに発生する。２つの変更は、それらがお互いについての知識なしで行われる場合に独立して呼び出される－つまり、第１のバージョンは、第２のバージョンについての知識の対象とならず、またその逆もいえる。同期サービスは、上述のように、レプリカの知識に基づいてそのようなすべての競合を自動的に検出する。
【０２８８】
競合をＣｈａｎｇｅ　Ｕｎｉｔのバージョン履歴におけるフォークと考えると役立つことがある。Ｃｈａｎｇｅ　Ｕｎｉｔの存続期間中に競合が発生しない場合、そのバージョン履歴は単純連鎖－それぞれの変更は前の変更の後に発生する－である。知識ベースの競合の場合、２つの変更は並行して発生し、そのため、連鎖は分割され、バージョンツリーとなる。
【０２８９】
（ｂ）制約ベースの競合
いっしょに適用された場合に独立の変更が完全性制約に違反する場合がある。例えば、２つのレプリカが同じディレクトリ内に同じ名前を持つファイルを作成しようとすると、そのような競合が発生するおそれがある。
【０２９０】
制約ベースの競合は、２つの独立した変更を伴うが（知識ベースの競合とまったく同様に）、同じＣｈａｎｇｅ　Ｕｎｉｔには影響を及ぼさない。むしろ、異なるＣｈａｎｇｅ　Ｕｎｉｔｓに影響を及ぼすが、制約がそれらの間に存在する。
【０２９１】
同期サービスは、変更適用時に制約違反を検出し、制約ベースの競合の通知を自動的に発する。制約ベースの競合を解決するには、通常、制約に違反しないような方法で変更を修正するカスタムコードを必要とするが、同期サービスは、そのようなことを行うための汎用メカニズムを備えていない。
【０２９２】
（２）競合処理
競合が検出されると、同期サービスは、（１）変更を拒絶し、それを送信者に送り返すアクション、（２）競合を競合ログに記録するアクション、または（３）競合を自動的に解決するアクションのうちの１つ（同期プロファイル内の同期イニシエータにより選択される）を実行することができる。
【０２９３】
変更が拒絶された場合、同期サービスは、変更がレプリカに到着しなかったかのように動作する。否定応答が発信者側に送り返される。この解決ポリシーは、主に、競合のログ記録が可能でないヘッドレスレプリカ（ファイルサーバなど）で使用することができる。

その代わりに、そのようなレプリカは拒絶することにより競合を処理するように他のレプリカを強制する。

【０２９４】

同期イニシエータは、同期プロファイルで競合解決を構成する。同期サービスは、以下のようにして単一プロファイル内の複数の競合リゾルバの組合せをサポートする－第１に、どれか１つが成功するまで次々に試みられる競合リゾルバのリストを指定し、第２に、競合リゾルバを競合のタイプに関連付ける、例えば、更新と更新との間の知識ベースの競合を１つのリゾルバに振り向け、他の競合をすべてログに振り向ける。

【０２９５】

（ａ）自動競合解決

同期サービスは、多数の既定の競合リゾルバを備える。このリストは以下のものを含む。

・　ｌｏｃａｌ－ｗｉｎｓ：ローカルに格納されているデータとの競合がある場合に受け取った変更を破棄する。

・　ｒｅｍｏｔｅ－ｗｉｎｓ：受け取った変更との競合がある場合にローカルデータを破棄する。

・　ｌａｓｔ－ｗｒｉｔｅｒ－ｗｉｎｓ：変更のタイムスタンプに基づきＣｈａｎｇｅ Ｕｎｉｔに従ってｌｏｃａｌ－ｗｉｎｓまたはｒｅｍｏｔｅ－ｗｉｎｓのいずれかを選択する（同期サービスは、一般に、クロック値に依存しないことに注意されたい。この競合リゾルバはその規則に対する唯一の例外である）。

・　Ｄｅｔｅｒｍｉｎｉｓｔｉｃ：すべてのレプリカ上で同じであることを保証される方法で勝利者を選択するが、他の方法では意味がない－同期サービスの一実施形態では、この機能を実装するためにパートナＩＤの辞書式比較を使用する。

【０２９６】

さらに、ＩＳＶは、独自の競合リゾルバを実装し、インストールすることができる。カスタム競合リゾルバは、構成パラメータを受け付けることができ、そのようなパラメータは、同期プロファイルのＣｏｎｆｌｉｃｔ ＲｅｓｏｌｕｔｉｏｎセクションにあるＳＣＡにより指定されなければならない。

【０２９７】

競合リゾルバが競合を処理すると、これは、（競合変更の代わりに）実行される必要のあるオペレーションのリストをランタイムに返す。その後、同期サービスは、競合ハンドラ側で考慮していた内容を含むようにリモート知識を適切に調整してから、それらのオペレーションを適用する。

【０２９８】

解決を適用している間に他の競合が検出される可能性がある。このような場合、新しい競合は、元の処理が再開する前に解決されなければならない。

【０２９９】

競合をアイテムのバージョン履歴内の分岐として考えた場合、競合解決は、結合－２つの分岐を組み合わせて１つのポイントを形成すること－であるとみなすことができる。したがって、競合解決は、バージョン履歴をＤＡＧに変える。

【０３００】

（ｂ）競合ログ作成

本当に特別な種類の競合リゾルバとして、Ｃｏｎｆｌｉｃｔ Ｌｏｇｇｅｒがある。同期サービスは、競合を型ＣｏｎｆｌｉｃｔＲｅｃｏｒｄのＩｔｅｍｓとしてログに記録する。これらの記録は、競合が生じているアイテムに再び関係する（アイテム自体が削除されていない限り）。それぞれの競合記録は、競合を引き起こした受け取った変更、競合のタイプ、更新－更新、更新－削除、削除－更新、挿入－挿入、または制約、および受け取った変更のバージョンとそれを送信するレプリカの知識を含む。ログに記録された競合は、後述のように、検査および解決のために使用できる。

【０３０１】

（ｃ）競合の検査および解決

同期サービスは、アプリケーション側で競合ログを調べ、その中の競合の解決を提案するためのＡＰＩを備えている。このＡＰＩにより、アプリケーションは、すべての競合、または与えられたＩｔｅｍに関係する競合を列挙することができる。これにより、さらに、そのようなアプリケーションは、ログに記録された競合を解決するために、（１）ｒｅｍｏｔｅ　ｗｉｎｓ－ログに記録された変更を受け入れ、競合しているローカルの変更を上書きする、（２）ｌｏｃａｌ　ｗｉｎｓ－ログに記録された変更の競合部分を無視する、および（３）ｓｕｇｇｅｓｔ　ｎｅｗ　ｃｈａｎｇｅ－アプリケーションが、オプションにより、競合を解決するマージを提案する場合、の３つの方法のうちの１つを使用することができる。アプリケーションにより競合が解決された後、同期サービスはそれらをログから削除する。

【０３０２】

（ｄ）レプリカの収束および競合解決の伝播

複雑な同期シナリオでは、同じ競合が複数のレプリカで検出される場合がある。このような状況が生じた場合、（１）競合が一方のレプリカで解決され、その解決が他方のレプリカに送られるようにすることができるか、または（２）競合が両方のレプリカ上で自動的に解決されるか、または（３）競合が両方のレプリカで手動で解決される（競合検査ＡＰＩを通じて）。

【０３０３】

収束を保証するため、同期サービスは、競合解決を他のレプリカに転送する。競合を解決する変更がレプリカに届くと、同期サービスは、この更新により解決されるログに含まれる競合記録を自動的に見つけ出し、それらを削除する。この意味で、一方のレプリカでの競合解決は他方のすべてのレプリカ上でのバインディングになる。

【０３０４】

同じ競合について異なるレプリカにより異なる勝利者が選択された場合、同期サービスは競合解決をバインドする原理を適用し、２つの解決のうちの１つを選び、自動的に他方を獲得する。勝利者は、いつでも同じ結果が得られるように保証される決定論的方法で選ばれる（一実施形態ではレプリカＩＤ辞書式比較を使用する）。

【０３０５】

同じ競合について異なるレプリカにより異なる「新しい変更」が提案された場合、同期サービスは、この新しい競合を特別な競合として取り扱い、Ｃｏｎｆｌｉｃｔ　Ｌｏｇｇｅｒを使用して、それが他のレプリカに伝搬しないようにする。そのような状況は、一般に、手動による競合解決において生じる。

【０３０６】

２．非ストレージプラットフォームデータストアの同期処理

本発明のストレージプラットフォームの他の態様によれば、ストレージプラットフォームは、ＩＳＶが、ストレージプラットフォームとＭｉｃｒｏｓｏｆｔ　Ｅｘｃｈａｎｇｅ、ＡＤ、Ｈｏｔｍａｉｌなどの従来システムとを同期させるＳｙｎｃ　Ａｄａｐｔｅｒｓを実装するためのアーキテクチャを備える。Ｓｙｎｃ　Ａｄａｐｔｅｒｓは、後述のように、同期サービスにより提供される多くのＳｙｎｃ　Ｓｅｒｖｉｃｅを利用する。

【０３０７】

その名にもかかわらず、Ｓｙｎｃ　Ａｄａｐｔｅｒｓは、何らかのストレージプラットフォームアーキテクチャにプラグインとして実装する必要はない。そうしたければ、「ｓｙｎｃ　ａｄａｐｔｅｒ」は、単に、変更列挙およびアプリケーションなどのサービスを取得するため同期サービスランタイムインターフェースを利用するアプリケーションとすることもできる。

【０３０８】

与えられたバックエンドとの同期を他から構成し、実行することを簡単にするため、Ｓｙｎｃ　Ａｄａｐｔｅｒ作成者は、上述のように同期プロファイルが与えられた場合に同期処理を実行する標準のＳｙｎｃ　Ａｄａｐｔｅｒインターフェースを公開するよう奨励

される。このプロファイルで、構成情報をアダプタに伝えるが、アダプタは、その一部をランタイムサービス（例えば、同期をとるＦｏｌｄｅｒ）を制御するＳｙｎｃ　Ｒｕｎｔｉｍｅに受け渡す。

【０３０９】

ａ）同期サービス

同期サービスは、多数の同期サービスをアダプタ作成者に提供する。このセッションの残り部分では、ストレージプラットフォームが同期処理を実行しているマシンを「クライアント」と呼び、アダプタの通信先の非ストレージプラットフォームバックエンドを「サーバ」と呼ぶと都合がよい。

【０３１０】

（１）Ｃｈａｎｇｅ　Ｅｎｕｍｅｒａｔｉｏｎ

Ｃｈａｎｇｅ　Ｅｎｕｍｅｒａｔｉｏｎを使用すると、同期サービスにより保持されている変更追跡データに基づき、同期処理アダプタは、データストアＦｏｌｄｅｒに対しこのパートナとの同期が最後に試みられた以降に発生した変更を簡単に列挙することができる。

【０３１１】

変更は、「アンカー」－最後の同期に関する情報を表す隠蔽型構造－という概念に基づいて列挙される。アンカーは、前のセクションで説明されているように、ストレージプラットフォームの知識という形をとる。変更列挙サービスを使用する同期処理アダプタは、「ストアドアンカー」（ｓｔｏｒｅｄ　ａｎｃｈｏｒｓ）を使用するものと「サプライドアンカー」（ｓｕｐｐｌｉｅｄ　ａｎｃｈｏｒｓ）を使用するものという２つの広いカテゴリに分類される。

【０３１２】

この区別は、最後の同期に関する情報が格納されている場所－クライアントか、それともサーバか－に基づく。アダプタはこの情報をクライアント上に格納するのが簡単である場合が多い－バックエンドは、この情報を簡単に格納できない場合が多い。その一方で、複数のクライアントが同じバックエンドと同期をとる場合、この情報をクライアント上に格納するのは不効率であり、場合によっては正しくない－場合によって一方のクライアントが他方のクライアントがすでにサーバにプッシュアップしている変更に気づかない。アダプタ側でサーバストアドアンカーを使用したい場合、そのアダプタは、変更列挙時にストレージプラットフォームへ送り返す必要がある。

【０３１３】

ストレージプラットフォームがアンカー（ローカルストレージまたはリモートストレージのいずれかの）を保持するために、ストレージプラットフォームに対し、サーバ側で正常に適用された変更を認識させる必要がある。これらの変更およびこれらの変更のみをアンカーに含めることができる。変更列挙の際にＳｙｎｃ　Ａｄａｐｔｅｒｓは、Ａｃｋｎｏｗｌｅｄｇｅｍｅｎｔインターフェースを使用して、正常に適用された変更を報告する。同期の終了時に、サプライドアンカーを使用するアダプタは、新しいアンカー（正常に適用された変更すべてを組み込んだ）を読み込み、それをバックエンドに送る必要がある。

【０３１４】

多くの場合、Ａｄａｐｔｅｒｓは、ストレージプラットフォームデータストア内に挿入するアイテムとともにアダプタ特有のデータを格納する必要がある。このようなデータの一般的な例として、リモートＩＤおよびリモートバージョン（タイムスタンプ）がある。同期サービスは、このデータを格納するメカニズムを備えており、Ｃｈａｎｇｅ　Ｅｎｕｍｅｒａｔｉｏｎは、返される変更とともにこの付加データを受け取るためのメカニズムを備える。これにより、ほとんどの場合に、アダプタ側がデータベースに対し再度クエリを実行する必要がなくなる。

【０３１５】

（２）Ｃｈａｎｇｅ　Ａｐｐｌｉｃａｔｉｏｎ

Ｃｈａｎｇｅ Ａｐｐｌｉｃａｔｉｏｎを使用すると、Ｓｙｎｃ Ａｄａｐｔｅｒｓ側でバックエンドから受け取った変更をローカルストレージプラットフォームに適用することができる。アダプタは、変更をストレージプラットフォームスキーマに変換することが期待される。図２４は、ストレージプラットフォームＡＰＩクラスがストレージプラットフォームＳｃｈｅｍａから生成されるプロセスを例示している。

【０３１６】

変更適用の主な機能は、競合を自動的に検出することである。ストレージプラットフォームとストレージプラットフォームとの同期の場合のように、競合は、お互いについての知識なしで２つの重なり合う変更が行われることと定義される。アダプタは、Ｃｈａｎｇｅ Ａｐｐｌｉｃａｔｉｏｎを使用する場合、どの競合検出が実行されるかについてアンカーを指定しなければならない。Ｃｈａｎｇｅ Ａｐｐｌｉｃａｔｉｏｎは、アダプタの知識の対象とならない重なり合うローカル変更が検出された場合に競合の通知を発する。Ｃｈａｎｇｅ Ｅｎｕｍｅｒａｔｉｏｎと同様に、アダプタは、ストアドアンカーまたはサプライドアンカーのいずれかを使用することができる。Ｃｈａｎｇｅ Ａｐｐｌｉｃａｔｉｏｎでは、アダプタ特有のメタデータの効率的格納をサポートする。このようなデータは、アダプタにより、適用される変更に付随させることができ、また同期サービスにより格納することも可能である。データは、次の変更列挙のときに返すことができるであろう。

【０３１７】

（３）Ｃｏｎｆｌｉｃｔ Ｒｅｓｏｌｕｔｉｏｎ

上述のＣｏｎｆｌｉｃｔ Ｒｅｓｏｌｕｔｉｏｎメカニズム（ログ記録および自動解決オプション）も、同期処理アダプタで利用できる。同期処理アダプタは、変更を適用する場合に競合解決のポリシーを指定することができる。指定した場合、競合は、指定された競合ハンドラに受け渡され、解決されるようにできる（可能ならば）。競合をログに記録することもできる。アダプタは、ローカル変更をバックエンドに適用しようと試みる際に競合を検出する可能性がある。このような場合、アダプタは、それでも、Ｓｙｎｃ Ｒｕｎｔｉｍｅにその競合を受け渡し、ポリシーに従って解決することができる。さらに、同期処理アダプタは、同期サービスにより検出された競合が処理のため送り返されるように要求することもできる。これは、バックエンドが競合を格納または解決することができる場合に便利である。

【０３１８】

ｂ）アダプタの実装

いくつかの「アダプタ」が単に、ランタイムインターフェースを使用するアプリケーションである場合には、アダプタ側で標準アダプターインターフェースを実装することが奨励される。それらのインターフェースを使用することにより、Ｓｙｎｃ Ｃｏｎｔｒｏｌｌｉｎｇ Ａｐｐｌｉｃａｔｉｏｎｓは、与えられた同期プロファイルに従ってアダプタが同期処理を実行するよう要求し、進行中の同期処理をキャンセルし、進行中の同期に関する進行中報告（完了率）を受け取ることができる。

【０３１９】

３．セキュリティ

同期サービスは、ストレージプラットフォームにより実装されたセキュリティモデルに導入する物をできるだけ少なくしようとしている。同期処理のための新しい権利を定義するのではなく、既存の権利が使用される。特に、

・　データストアＩｔｅｍを読み込める人は誰でも、そのアイテムへの変更を列挙することができ、

・　データストアＩｔｅｍに書き込める人は誰でも、そのアイテムに変更を適用することができ、

・　データストアＩｔｅｍを拡張できる人は誰でも、そのアイテムに同期メタデータを関連付けることができるということである。

【０３２０】

同期サービスでは、セキュリティで保護された著作権情報を保持しない。ユーザＵによりレプリカＡで変更が加えられ、レプリカＢに転送された場合、変更が元々Ａで（またはＵにより）行われたという事実は失われる。Ｂがこの変更をレプリカＣに転送した場合、これは、Ａの権限ではなくＢの権限の下で実行される。このため、レプリカがアイテムに対し独自の変更を加えることについて信頼されていない場合、他者により加えられた変更を転送することはできないという制限が生じる。

【０３２１】

同期サービスは、開始されると、Ｓｙｎｃ　Ｃｏｎｔｒｏｌｌｉｎｇ　Ａｐｐｌｉｃａｔｉｏｎによって実行される。同期サービスは、ＳＣＡの識別を偽装し、その識別の下ですべてのオペレーション（ローカルとをリモートの両方）を実行する。説明のため、ユーザＵは、ローカル同期サービスに、ユーザＵが読み取りアクセス権を持たないアイテムについてリモートストレージプラットフォームから変更を取り出させることはできないことを観察する。

【０３２２】

４．管理可能性

レプリカの分散コミュニティの監視は複雑な問題である。同期サービスは、「スイープ」アルゴリズムを使用して、レプリカのステータスに関する情報を収集し分配することができる。スイープアルゴリズムの特性は、構成されたすべてのレプリカに関する情報は、最終的に回収されること、および失敗（非応答）レプリカが検出されることを保証する。

【０３２３】

このコミュニティ規模の監視情報は、すべてのレプリカで利用可能になっている。監視ツールを、任意に選択されたレプリカで実行し、この監視情報を調べて、管理上の決定を下すことができる。構成変更は、影響を受けるレプリカにおいて直接行われなければならない。

【０３２４】

Ｂ．同期処理ＡＰＩの概要

分散化傾向を強めるデジタル世界において、個人とワークグループは、情報とデータを様々なデバイスおよび場所に格納することが多くなっている。これが、これらの独立した多くの場合本質的に異なるデータストアを常に、ユーザの介入を最小限に抑えて同期させることができるデータ同期サービスの開発の推進要因であった。

【０３２５】

本発明の同期処理プラットフォームは、本明細書のセクションＩＩで説明されているリッチストレージプラットフォーム（別名「ＷｉｎＦＳ」）の一部であり、次の３つの主要な目標に取り組むものである。

・　アプリケーションおよびサービスが異なる「ＷｉｎＦＳ」ストア間のデータを効率よく同期させることができる。

・　開発者が「ＷｉｎＦＳ」ストアと非「ＷｉｎＦＳ」ストアとの間のデータの同期をとるリッチソリューションを構築することができる。

・　開発者に、同期処理ユーザエクスペリエンス（ｓｙｎｃｈｒｏｎｉｚａｔｉｏｎ　ｕｓｅｒ　ｅｘｐｅｒｉｅｎｃｅ）をカスタマイズするのに適したインターフェースを提供する。

【０３２６】

１．一般的用語

本明細書では、以下に、このセクションのＩＩＩ．Ｂの後の方の説明に関連するいくつかの他の精緻化された定義および重要概念を示す。

・　同期レプリカ（Ｓｙｎｃ　Ｒｅｐｌｉｃａ）：ほとんどのアプリケーションは、ＷｉｎＦＳストア内のアイテムの与えられた部分集合に対する変更の追跡、列挙、および同期処理にしか注目していない。同期処理オペレーションに関わるアイテムの集合は、同期レプリカと呼ばれる。レプリカは、与えられたＷｉｎＦＳ包含階層（通常は、Ｆｏｌｄｅｒアイテムをルートとする）内に含まれるアイテムに関して定義される。すべての同期サ

ービスは、与えられたレプリカのコンテキスト内で実行される。ＷｉｎＦＳ　Ｓｙｎｃが、レプリカを定義し、管理し、クリーンアップするためのメカニズムを提供する。レプリカはすべて、与えられたＷｉｎＦＳストア内で一意にそれを識別するＧＵＩＤ識別子を持つ。

・　同期パートナ（Ｓｙｎｃ　Ｐａｒｔｎｅｒ）：同期パートナは、ＷｉｎＦＳアイテム、拡張、および関係に対する変更に影響を及ぼすことが可能なエンティティとして定義される。したがって、すべてのＷｉｎＦＳストアは、同期パートナと呼ぶことができる。非ＷｉｎＦＳストアと同期をとる場合、外部データソース（ＥＤＳ）も同期パートナと呼ばれる。すべてのパートナは、それを一意に識別するＧＵＩＤ識別子を持つ。

・　同期コミュニティ（Ｓｙｎｃ　Ｃｏｍｍｕｎｉｔｙ）：同期コミュニティは、ピアツーピア同期処理オペレーションを使用して同期が保持されるレプリカのコレクションとして定義される。これらのレプリカは、すべて、同じＷｉｎＦＳストア、異なるＷｉｎＦＳストア内にあるか、さらには、非ＷｉｎＦＳストア上の仮想レプリカとしてさえ現れることができる。ＷｉｎＦＳ　Ｓｙｎｃでは、特にコミュニティ内の同期処理オペレーションのみがＷｉｎＦＳ　Ｓｙｎｃサービス（ＷｉｎＦＳアダプタ）を通じて行われる場合に、コミュニティの特定のトポロジを規定または指令しない。同期アダプタ（以下に定義されている）は、独自のトポロジ制約を導入することができる。

・　変更追跡、変更ユニット、およびバージョン：ＷｉｎＦＳストアはすべて、すべてのローカルのＷｉｎＦＳ　Ｉｔｅｍｓ、Ｅｘｔｅｎｓｉｏｎｓ、およびＲｅｌａｔｉｏｎｓｈｉｐｓへの変更を追跡する。変更は、スキーマ内で定義されている変更ユニット精度のレベルで追跡される。Ｉｔｅｍ、Ｅｘｔｅｎｓｉｏｎ、およびＲｅｌａｔｉｏｎｓｈｉｐ型の最上位レベルのフィールドは、スキーマデザイナにより複数の変更ユニットに細分されることが可能であり、最小精度が１つの最上位レベルのフィールドとなる。変更追跡のために、すべての変更ユニットに、同期パートナｉｄとバージョン番号のペアであるＶｅｒｓｉｏｎが割り当てられる（バージョン番号は、パートナ特有の単調増加番号である）。Ｖｅｒｓｉｏｎｓは、ローカルのストア内で変更が生じたり、他のレプリカから取得されるときに更新される。

・　同期知識：知識は、いつでも与えられた同期レプリカの状態を表す、つまり、ローカルまたは他のレプリカからの与えられたレプリカが認識するすべての変更に関するメタデータをカプセル化する。ＷｉｎＦＳ　Ｓｙｎｃは、同期処理オペレーション間で同期レプリカにする知識を保持し、更新する。注意すべき重要なこととして、知識表現により、コミュニティ全体に関して解釈できるのであって、知識が格納されている特定のレプリカに関してだけではないことである。

・　同期アダプタ：同期アダプタは、Ｓｙｎｃ　Ｒｕｎｔｉｍｅ　ＡＰＩを通じてＷｉｎＦＳ　Ｓｙｎｃサービスにアクセスし、ＷｉｎＦＳデータと非ＷｉｎＦＳデータストアとの同期をとることができるマネージドコードアプリケーションである。シナリオの要求条件に応じて、ＷｉｎＦＳデータのどの部分集合およびどのようなＷｉｎＦＳデータ型の同期をとるかは、アダプタ開発者次第である。アダプタは、ＥＤＳとの通信、ＷｉｎＦＳスキーマとＥＤＳサポートスキーマとの間の変換、独自構成およびメタデータの定義および管理を受け持つ。アダプタでは、ＷｉｎＦＳ　Ｓｙｎｃ　Ａｄａｐｔｅｒ　ＡＰＩを実装して、ＷｉｎＦＳ　Ｓｙｎｃチームにより提供されるアダプタの共通構成および制御インフラストラクチャを利用するよう強く奨励される。詳細については、文献（例えば、非特許文献１および非特許文献２）を参照されたい。

【０３２７】

ＷｉｎＦＳデータと外部の非ＷｉｎＦＳストアとの同期をとるが、ＷｉｎＦＳ形式で知識を出力または保持できないアダプタのために、ＷｉｎＦＳ　Ｓｙｎｃは、その後の変更列挙またはアプリケーションオペレーションに使用できるリモート知識を取得するサービスを提供する。バックエンドストアの能力に応じて、アダプタ側で、このリモート知識をバックエンドまたはローカルのＷｉｎＦＳストア上に格納したい場合がある。

【０３２８】

簡単のため、同期「レプリカ」は、単一の論理的ロケーション内に存在する「ＷｉｎＦＳ」ストア内のデータの集合を表す構造であるが、非「ＷｉｎＦＳ」ストア上のデータは、「データソース」と呼ばれ、一般的にアダプタを使用する必要がある。

・　リモート知識：与えられた同期レプリカで他のレプリカから変更を取得したい場合、他のレプリカが変更を列挙する際に突き合わせるベースラインとして自分の持つ知識を提供する。同様に、与えられたレプリカ側で、他のレプリカに変更を送りたい場合に、競合の検出のためにリモートレプリカにより使用されることができるベースラインとして自分の持つ知識を提供する。同期変更の列挙および適用時に提供される他のレプリカに関する知識が、リモート知識と呼ばれる。

【０３２９】

２．同期処理ＡＰＩの主要事項

いくつかの実施形態では、同期ＡＰＩは、同期構成ＡＰＩと同期コントローラＡＰＩの２つの部分に分けられる。同期構成ＡＰＩを使用すると、アプリケーション側で、同期を構成し、２つのレプリカの間の特定の同期セッションのパラメータを指定することができる。与えられた同期セッションについて、構成パラメータは、同期をとるＩｔｅｍｓの集合、同期のタイプ（一方向または両方向）、リモートデータソースに関する情報、および競合解決ポリシーを含む。同期コントローラＡＰＩは、同期セッションを開始し、同期をキャンセルし、進行中の同期処理に関する進行状況およびエラー情報を受け取る。さらに、所定のスケジュールに従って同期処理を実行する必要のある特定の実施形態では、そのようなシステムは、スケジューリングをカスタマイズするためのスケジューリングメカニズムを備えることができる。

【０３３０】

本発明のいくつかの実施形態では、「ＷｉｎＦＳ」データソースと非「ＷｉｎＦＳ」データソースとの間の情報の同期をとるために同期アダプタを採用する。アダプタの例として、「ＷｉｎＦＳ」連絡先フォルダと非ＷｉｎＦＳメールボックスとの間のアドレス帳情報の同期をとるアダプタがある。これらの場合、アダプタ開発者は、「ＷｉｎＦＳ」スキーマと非「ＷｉｎＦＳ」データソーススキーマとの間のスキーマ変換コードを開発するため、「ＷｉｎＦＳ」同期プラットフォームにより提供される本明細書で説明されている「ＷｉｎＦＳ」同期処理コアサービスＡＰＩを、サービスにアクセスするために使用することも可能である。さらに、アダプタ開発者は、非「ＷｉｎＦＳ」データソースの変更を伝達するためのプロトコルをサポートする。同期アダプタは、同期コントローラＡＰＩを使用することにより呼び出され、制御され、このＡＰＩを使用して進捗状況およびエラーを報告する。

【０３３１】

しかし、本発明のいくつかの実施形態では、「ＷｉｎＦＳ」データストアと他の「ＷｉｎＦＳ」データストアとの同期をとるときに、「ＷｉｎＦＳ」－「ＷｉｎＦＳ」間同期サービスがハードウェア／ソフトウェアインターフェースシステム内に統合されていれば、同期アダプタは不要と考えられる。いずれにせよ、いくつかのそのような実施形態は、「ＷｉｎＦＳ」－「ＷｉｎＦＳ」と同期アダプタソリューションの両方に対する同期サービス群を提供し、これらは以下を含む。

・　「ＷｉｎＦＳ」アイテム、拡張、および関係への変更の追跡。
・　与えられた過去の状態以降の効率的なインクリメント的変更列挙のサポート。
・　「ＷｉｎＦＳ」への外部変更の適用。
・　変更適用時の競合処理。

【０３３２】

図３６を参照すると、これは、共通データストアの３つのインスタンスおよびそれらの同期をとるためのコンポーネントを例示している。第１のシステム３６０２は、利用できるように同期ＡＰＩ　３６５２を公開する（３６４６）、ＷｉｎＦＳ－非ＷｉｎＦＳ同期のためのＷｉｎＦＳ－ＷｉｎＦＳ同期サービス３６２２およびコア同期サービス３６２４を含むＷｉｎＦＳデータストア３６１２を備える。第１のシステム３６０２と同様に、第

２のシステム３６０４は、利用できるように同期ＡＰＩ　３６５２を公開する（３６４６）、ＷｉｎＦＳ－非ＷｉｎＦＳ同期のためのＷｉｎＦＳ－ＷｉｎＦＳ同期サービス３６３２およびコア同期サービス３６３４を含むＷｉｎＦＳデータストア３６１４を備える。第１のシステム３６０２および第２のシステム３６０４は、それぞれのＷｉｎＦＳ－ＷｉｎＦＳ同期サービス３６２２および３６３２を介して同期をとる（３６４２）。第３のシステム３６０６は、ＷｉｎＦＳシステムではないが、ＷｉｎＦＳレプリカとの同期コミュニティでデータソースを保持するためＷｉｎＦＳ同期３６６６を使用するためのアプリケーションを備える。このアプリケーションは、ＷｉｎＦＳ　Ｓｙｎｃ　Ｃｏｎｆｉｇ／Ｃｏｎｔｒｏｌサービス３６６４を利用して、ＷｉｎＦＳ－ＷｉｎＦＳ同期サービス３６２２を介して（それ自身をＷｉｎＦＳデータストアとして仮想化できる場合）、または同期ＡＰＩ　３６５２とインターフェースする（３６４８）同期アダプタ３６６２を介して、直接ＷｉｎＦＳデータストア３６１２とインターフェースする（３６４４）ことができる。

【０３３３】

この図に例示されているように、第１のシステム３６０２は、第２のシステム３６０４と第３のシステム３６０６の両方を認識し、直接同期をとる。しかし、第２のシステム３６０４も第３のシステム３６０６も、互いを認識せず、したがって、互いに直接変更を同期させないが、代わりに、一方のシステムに生じた変更は、第１のシステム３６０２を通じて伝搬しなければならない。

【０３３４】

Ｃ．同期処理ＡＰＩサービス

本発明のいくつかの実施形態は、変更列挙および変更適用という２つの基本サービスを含む同期サービスを対象とする。

【０３３５】

１．変更列挙

本明細書の前の方で説明されているように、Ｃｈａｎｇｅ　Ｅｎｕｍｅｒａｔｉｏｎを使用すると、同期処理アダプタは、同期サービスにより保持されている変更追跡データに基づき、このパートナとの同期が最後に試みられた以降に、データストアＦｏｌｄｅｒに対し発生した変更を簡単に列挙することができる。変更列挙に関して、本発明のいくつかの実施形態は以下を対象とする。

・　指定されたＫｎｏｗｌｅｄｇｅインスタンスに関して、与えられたレプリカ内のＩｔｅｍｓ、Ｅｘｔｅｎｓｉｏｎｓ、およびＲｅｌａｔｉｏｎｓｈｉｐｓの変更の効率的列挙。

・　ＷｉｎＦＳスキーマ内で指定された変更ユニット精度のレベルでの変更の列挙。

・　複合アイテムに関して列挙された変更のグループ化。複合アイテムは、アイテム、そのすべての拡張、そのアイテムとのすべての保持関係、およびその埋め込まれたアイテムに対応するすべての複合アイテムからなる。アイテム間の参照関係の変更は、別々に列挙される。

・　変更列挙に対するバッチ処理。バッチ処理の精度は、複合アイテムまたは関係変更（参照関係の）である。

・　変更列挙時のレプリカ内のアイテムに対するフィルタの指定、例えば、レプリカは与えられたフォルダ内のすべてのアイテムからなるが、この特定の変更列挙については、アプリケーション側では、第１の名前が「Ａ」で始まるすべてのＣｏｎｔａｃｔアイテムに対する変更を列挙することのみを望む（このサポートはＢマイルストーンの後に（ｐｏｓｔ　Ｂ－ｍｉｌｅｓｔｏｎｅ）追加される）。

・　個々の変更ユニット（または、アイテム、拡張、または関係全体）を知識の中で同期失敗として記録し、それらを次回に再列挙させることができる、列挙された変更に対するリモート知識の使用。

・　変更列挙時に変更とともにメタデータを返すことによりＷｉｎＦＳ同期メタデータを理解できる場合のある上級アダプタの使用。

【０３３６】

２．変更適用
本明細書のはじめの方で説明されているように、変更適用では、同期アダプタは、ストレージプラットフォームスキーマに変更を変換することが予期されているため、そのバックエンドから受け取った変更をローカルストレージプラットフォームに適用できる。変更適用に関して、本発明のいくつかの実施形態は以下を対象とする。
・　ＷｉｎＦＳ変更メタデータへの対応する更新のある、他のレプリカ（または非ＷｉｎＦＳストア）からのインクリメント的変更の適用。
・　変更ユニットの精度での変更適用後の競合の検出。
・　変更適用後に個々の変更ユニットレベルで生じる成功、失敗、および競合の報告であって、アプリケーション（アダプタおよび同期制御アプリケーションを含む）がその情報を進捗状況、エラー、およびステータス報告およびもしあればそのバックエンド状態の更新に使用できるようにする報告。
・　次の変更列挙オペレーションでアプリケーションにより供給された変更の「反映」を防ぐために変更適用時にリモート知識を更新すること。
・　ＷｉｎＦＳ同期メタデータを変更とともに理解し提供することができる上級アダプタの使用。

【０３３７】
３．サンプルコード
以下に、ＦＯＯ同期アダプタが同期ランタイムとやり取りする方法を示すコードサンプルを掲載した（すべてのアダプタ特有の関数の先頭にはＦＯＯが付いている）。

【０３３８】
```
ItemContext ctx = new ItemContext ("＼.＼System＼UserData＼dshah＼My Contacts"
, true);

//レプリカアイテムｉｄとリモートパートナｉｄをプロファイルから取得する。
//ほとんどのアダプタは、同期プロファイルからこの情報を取得する。

Guid replicaItemId = FOO_GetReplicaId();
Guid remotePartnerId = FOO_Get_RemotePartnerId();

//
//上記のようにｓｔｏｒｅｄＫｎｏｗｌｅｄｇｅＩｄを使用してストア内に格納されている知識を検索する。
//
ReplicaKnowledge remoteKnowledge = ...;

//
//ＲｅｐｌｉｃａＳｙｎｃｈｒｏｎｉｚｅｒを初期化する。
//
ctx.ReplicaSynchronizer = new ReplicaSynchronizer( replicaItemId, remotePartne
rId);
ctx.ReplicaSynchronizer.RemoteKnowledge = remoteKnowledge; ChangeReader reader
 = ctx.ReplicaSynchronizer.GetChangeReader();

//
//変更を列挙し、それらを処理する。
//
bool bChangesToRead = true;
while (bChangesToRead)
{
```

```
    ChangeCollection<object> changes = null;
    bChangesToRead = reader.ReadChanges( 10, out changes);
    foreach (object change in changes)
    {
      //列挙されているオブジェクトを処理し、アダプタは、それ自体の
      //スキーマ変換およびＩＤマッピングを実行する。この目的のために
      //Ｃｔｘから追加オブジェクトを取り出して、変更がリモートストアに
      //適用された後のアダプタメタデータを修正することすらできる。
      ChangeStatus status = FOOProcessAndApplyToRemoteStore(change);

      //学習された知識をステータスで更新する
      reader.AcknowledgeChange ( changeStatus);
    }
  }

  remoteKnowledge = ctx.ReplicaSynchronizer.GetUpdatedRemoteKnowledge(); reader.
Close();

  //
  //もしあれば更新された知識とアダプタメタデータを保存する。
  //
  ctx.Update();

  //
  //変更適用のサンプル、まず最初に、前のようにｓｔｏｒｅｄＫｎｏｗｌｅｄｇｅＩｄ
  //を使用してリモート知識を初期化する。
  //
  remoteKnowledge = ...;

  ctx.ReplicaSynchronizer.ConflictPolicy = conflictPolicy;
  ctx.ReplicaSynchronizer.RemotePartnerId = remotePartnerId;
  ctx.ReplicaSynchronizer.RemoteKnowledge = remoteKnowledge;
  ctx.ReplicaSynchronizer.ChangeStatusEvent += FOO_OnChangeStatusEvent;

  //
  //リモートストアから変更を取得する。アダプタは、ストアから
  //バックエンド特有のメタデータを取り出す役割を持つ。これは
  //レプリカ上の拡張とすることができる。
  //

  object remoteAnchor = FOO_GetRemoteAnchorFromStore();
  FOO_RemoteChangeCollection remoteChanges = FOO_GetRemoteChanges(remoteAnchor);

  //
  //変更コレクションを充填する。
  //
  foreach( FOO_RemoteChange change in remoteChanges)
  {
    //ＩＤマッピングを実行する役割を持つアダプタ
    Guid localId = FOO_MapRemoteId(change);
```

```
    //例えば、Ｐｅｒｓｏｎオブジェクトの同期をとることにする。
    ItemSearcher searcher = Person.GetSearcher(ctx);
    searcher.Filters.Add("PersonId=@localId");
    searcher.Parameters["PersonId"] = localId;
    Person person = searcher.FindOne();

    //
    //アダプタは、リモート変更をＰｅｒｓｏｎオブジェクト上の修正に変換する
    //このアダプタの一部として、変更をリモートオブジェクトのアイテム
    //レベルのバックエンド特有のメタデータに加えることさえできる。
    //
    FOO_TransformRemoteToLocal ( remoteChange, person);
  }

  ctx.Update();

  //
  //新しいリモートアンカーを保存する（これは、レプリカ上の拡張とすることができる
)
  //
  FOO_SaveRemoteAnchor();

  //
  //これは、リモート知識の同期がとれていないので、通常のＷｉｎＦＳ　ＡＰＩ保存で
ある。
  //
  remoteKnowledge = ctx.ReplicaSynchronizer.GetUpdatedRemoteKnowledge();
  ctx.Update();
  ctx.Close();

  //
  //アダプタは、アプリケーションステータスコールバックの処理をコールバックする
  //
  void FOO_OnEntitySaved( object sender, ChangeStatusEventArgs args)
  {
    remoteAnchor.AcceptChange( args.ChangeStatus);
  }
```

【０３３９】

４．ＡＰＩ同期処理のメソッド

本発明の一実施形態では、ＷｉｎＦＳストアと非ＷｉｎＦＳストアとの間で同期をとることは、ＷｉｎＦＳベースのハードウェア／ソフトウェアインターフェースシステムにより公開されている同期ＡＰＩを介して可能である。

【０３４０】

一実施形態では、すべての同期アダプタは、同期アダプタＡＰＩ、共通言語ランタイム（ＣＬＲ）マネージドＡＰＩを実装し、矛盾なく展開、初期化、および制御できるようにする必要がある。アダプタＡＰＩは以下を備える。

・　ハードウェア／ソフトウェアインターフェースシステム同期フレームワークにアダプタを登録する標準メカニズム

・　アダプタがその機能およびアダプタを初期化するために必要な構成情報の型を宣言

する標準メカニズム
・　初期化情報をアダプタに受け渡す標準メカニズム
・　アダプタが進捗状況を、同期を呼び出すアプリケーションに送り返して報告するメカニズム
・　同期処理時に発生したエラーを報告するメカニズム
・　進行中の同期処理オペレーションのキャンセルを要求するメカニズム
【０３４１】
シナリオの要求条件に応じて、アダプタには２つの潜在的プロセスモデルがある。アダプタは、それを呼び出すアプリケーションと同じプロセス空間内で、または独立のプロセスですべて自動的に実行することができる。アダプタは、別の自プロセスで実行するために、アダプタをインスタンス化するために使用される自ファクトリクラスを定義する。ファクトリは、呼び出し側アプリケーションと同じプロセスでアダプタのインスタンスを返すか、または異なるＭｉｃｒｏｓｏｆｔ共通言語ランタイムアプリケーション領域またはプロセス内でアダプタのリモートインスタンスを返すことができる。同じプロセス内のアダプタをインスタンス化する既定のファクトリ実装が実現される。実際には、多くのアダプタが呼び出し側アプリケーションと同じプロセスで実行される。処理外モデルは、通常、以下の理由の１つまたは両方について必要である。
・　セキュリティ目的。アダプタは、特定のプロセスまたはサービスのプロセス空間内で実行されなければならない。
・　アダプタは、呼び出し側アプリケーションからの要求を処理する他に、他のソースからの要求－例えば、受け取ったネットワーク要求－を処理しなければならない。
【０３４２】
図３７を参照すると、本発明の一実施形態は、状態がどのように計算されるか、またはその関連するメタデータがどのように交換されるかを認識しない単純なアダプタを想定している。この実施形態では、同期処理は、同期をとりたいデータソースに関してレプリカにより行われるが、そのために、最初に、ステップ３７０２で、前記データソースと最後に同期した以降に発生した変更を決定し、その後、レプリカは現在の状態の情報に基づいてこの最後の同期以降に発生したインクリメント的変更を送り、この状態情報およびインクリメント的変更は、アダプタを介してデータソースに送られる。ステップ３７０４で、アダプタは、前のステップでレプリカから変更データを受け取った後、できる限り多くのデータに対する変更を実装し、どの変更が成功し、どの変更が失敗したかを追跡し、成功－失敗情報をＷｉｎＦＳ（レプリカの）に送り返す。レプリカ（ＷｉｎＦＳ）のハードウェア／ソフトウェアインターフェースシステムは、ステップ３７０６で、レプリカから成功－失敗情報を受け取った後、データソースに対する新しい状態情報を計算し、この情報を将来そのレプリカで使用するため格納しておき、この新しい状態情報をデータソース、つまりアダプタに送り返して、格納し、アダプタによって後から使用できるようにする。
【０３４３】
Ｄ．同期スキーマの追加態様
以下は、本発明の様々な態様に対する同期スキーマの追加（またはより具体的な）態様である。
・　それぞれのレプリカは、データストアの全体から取り出した定義済み同期部分集合－複数のインスタンスを持つスライス－である。
・　競合解決ポリシーは、それぞれのレプリカ（およびアダプタ／データソースの組合せ）により処理される－つまり、それぞれのレプリカは、自基準と競合解決スキーマに基づいて競合を解決することができる。さらに、データストアのそれぞれのインスタンスの違いが生じ、その結果、将来さらに競合が生じるが、更新された状態情報に反映されるような競合のインクリメント的および順次的列挙は、その更新された状態情報を受け取る他のレプリカからは見えない。
・　同期スキーマのルートには、一意的なＩＤを持つルートフォルダ（実際には、ルートＩｔｅｍ）を定義するための基本型を持つレプリカ、それがメンバである同期コミュニ

ティに対するＩＤ、および特定のレプリカに対し必要な、または望ましい何らかのフィルタおよびその他の要素がある。

・　それぞれのレプリカの「マッピング」は、レプリカ内に保持され、したがって、任意の特定のレプリカのマッピングは、そのようなレプリカが関知している他のレプリカに制限される。このマッピングは、同期コミュニティ全体の部分集合しか含まないが、前記レプリカに対する変更は、それでも、共通の共有レプリカを介して同期コミュニティ全体に伝搬する（ただし、特定のレプリカは、不明なレプリカと他のどのレプリカをふつうに共有しているかを認識しない）。さらに、それぞれのレプリカは、同じ同期コミュニティにおいて異なる同期パートナとの異なる同期ビヘイビアを許容するように複数のマッピングを備えることができる。

・　レプリカのマッピングは、コミュニティ識別および同期パートナのマッピング識別を含むだけでよく、このため、レプリカは、同期パートナレプリカの物理的位置を必ずしも知らなくてもパートナと同期をとることができる（したがって、同期パートナレプリカのセキュリティが向上する）。

・　同期スキーマは、すべてのレプリカから利用できる複数の定義済み競合ハンドラとともに、ユーザ／開発者定義済みカスタム競合ハンドラの機能を含む。スキーマは、さらに、３つの特別な「競合リゾルバ」として、（ａ）例えば、（ｉ）同じ変更ユニットが２つの場所で変更された場合の取り扱い方法、（ｉｉ）変更ユニットが一方の場所で変更され、他方の場所で削除される場合の取り扱い方法、および（ｉｉｉ）２つの異なる変更ユニットが２つの異なる場所で同じ名前を持つ場合の取り扱い方法に基づく異なる方法で異なる競合を解決する競合「フィルタ」、（ｂ）リストの各要素で競合が正常に解決されるまで順番に試みる一連のアクションを指定する場合の競合「ハンドラリスト」、および（ｃ）競合を追跡するが、ユーザ介入なしでアクションをさらに実行することはしない「何もしない」ログを含むこともできる。

・　レプリカの同期スキーマおよび使用により、真の分散型ピアツーピアマルチマスター同期コミュニティが使用可能になる。さらに、同期コミュニティタイプはないが、同期コミュニティは、単に、レプリカ自体のコミュニティフィールドの中の値としてだけ存在する。

・　すべてのレプリカは、インクリメント的変更列挙を追跡し、同期コミュニティで知られている他のレプリカに対する状態情報を格納するため自メタデータを持つ。

・　変更ユニットは、自メタデータを持ち、これは、パートナキーとパートナ変更番号を含むバージョン番号、各変更ユニットのＩｔｅｍ／Ｅｘｔｅｎｓｉｏｎ／Ｒｅｌａｔｉｏｎｓｈｉｐのバージョニング、同期コミュニティからレプリカが見た／受信した変更に関する知識、ＧＵＩＤおよびローカルＩＤ構成、およびクリーンアップのため参照関係上に格納されているＧＵＩＤを含む。

【０３４４】

ＩＶ．結論

これまでに例示したように、本発明は、データの編成、検索、および共有のためのストレージプラットフォームを対象とする。本発明のストレージプラットフォームは、既存のファイルシステムおよびデータベースシステムを超えるデータストレージの概念を拡張し、広げるものであり、リレーショナル（表形式）データ、ＸＭＬ、およびＩｔｅｍｓと呼ばれる新しいデータ形態などの、構造化、非構造化、または半構造化データを含むすべての型のデータ用のストアとなるように設計されている。本発明のストレージプラットフォームでは、共通のストレージ基盤および図式化されたデータを通じて、より効率的なアプリケーション開発を消費者、知識労働者、および企業向けに行うことができる。これは、データモデルに固有の機能を利用可能にするだけでなく、既存のファイルシステムおよびデータベースアクセス方法も包含し、拡張する機能豊富な拡張可能アプリケーションプログラミングインターフェースを備える。本発明の広範な概念から逸脱することなく、上述の実施形態に対し変更を加えられることは理解される。したがって、本発明は、開示され

ている特定の実施形態に限定されず、付属の請求項の定義に従って本発明の精神および範囲内にあるすべての修正形態を対象とすることを意図されている。

【０３４５】

上述の内容から明らかなように、本発明の様々なシステム、方法、および態様の全部または一部は、プログラムコード（つまり、命令）の形で具現化できる。このプログラムコードは、限定はしないが、フロッピー（登録商標）ディスケット、ＣＤ－ＲＯＭ、ＣＤ－ＲＷ、ＤＶＤ－ＲＯＭ、ＤＶＤ－ＲＡＭ、磁気テープ、フラッシュメモリ、ハードディスクドライブ、またはその他のマシン可読記憶媒体を含む、磁気、電気、または光記憶媒体などのコンピュータ可読媒体上に格納することができ、プログラムコードがコンピュータまたはサーバなどのマシン内にロードされ実行されると、マシンは本発明を実践するための装置となる。本発明は、さらに、電気配線またはケーブル配線、光ファイバの使用、インターネットまたはイントラネットを含むネットワークなどの何らかの伝送媒体で、または他の形態の伝送を介して、伝送されるプログラムコードの形態で具現化されることも可能であり、プログラムコードがコンピュータなどのマシンにより受信され、読み込まれ、実行されると、マシンは本発明を実施するための装置となる。汎用プロセッサ上に実装された場合、プログラムコードはプロセッサとの組合せで、特定のロジック回路と類似の動作をする独自の装置を実現する。

【図面の簡単な説明】

【０３４６】

【図１】本発明の複数の態様が組み込まれうるコンピュータシステムを表すブロック図である。

【図２】３つのコンポーネントグループであるハードウェアコンポーネント、ハードウェア／ソフトウェアインターフェースシステムコンポーネント、およびアプリケーションプログラムコンポーネントに分割されたコンピュータシステムを例示するブロック図である。

【図２Ａ】ファイルベースのオペレーティングシステムにおけるディレクトリ内の複数のフォルダにグループ化されているファイルの従来のツリーベースの階層構造を例示する図である。

【図３】ストレージプラットフォームを例示するブロック図である。

【図４】Ｉｔｅｍｓ、Ｉｔｅｍ　Ｆｏｌｄｅｒｓ、およびＣａｔｅｇｏｒｉｅｓ間の構造的関係を例示する図である。

【図５Ａ】Ｉｔｅｍの構造を例示するブロック図である。

【図５Ｂ】図５ＡのＩｔｅｍの複合プロパティ型を例示するブロック図である。

【図５Ｃ】複合型がさらに記述されている（明示的にリストされている）「Ｌｏｃａｔｉｏｎ」Ｉｔｅｍを例示するブロック図である。

【図６Ａ】Ｂａｓｅ　ＳｃｈｅｍａにあるＩｔｅｍの子型としてＩｔｅｍを例示する図である。

【図６Ｂ】継承型が明示的にリストされている（イミディエイトプロパティの他に）図６Ａの子型Ｉｔｅｍを例示するブロック図である。

【図７】２つの最上位レベルのクラス型を含むＢａｓｅ　Ｓｃｈｅｍａ、ＩｔｅｍおよびＰｒｏｐｅｒｔｙＢａｓｅ、およびそれから派生した追加Ｂａｓｅ　Ｓｃｈｅｍａ型を例示するブロック図である。

【図８Ａ】Ｃｏｒｅ　Ｓｃｈｅｍａ内のＩｔｅｍを例示するブロック図である。

【図８Ｂ】Ｃｏｒｅ　Ｓｃｈｅｍａ内のプロパティ型を例示するブロック図である。

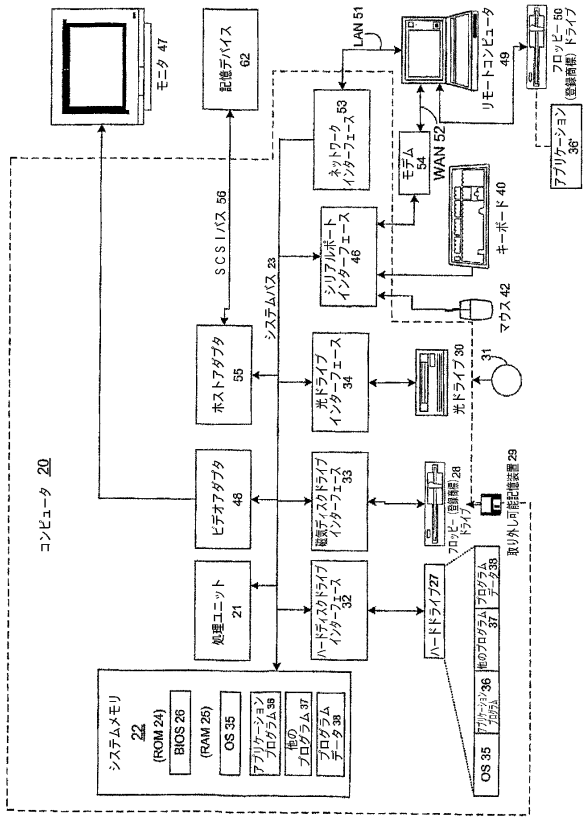
【図９】Ｉｔｅｍ　Ｆｏｌｄｅｒ、そのメンバＩｔｅｍｓ、およびＩｔｅｍ　ＦｏｌｄｅｒとそのメンバＩｔｅｍｓとの間の相互接続のＲｅｌａｔｉｏｎｓｈｉｐｓを例示するブロック図である。

【図１０】Ｃａｔｅｇｏｒｙ（またも、Ｉｔｅｍ自体である）、そのメンバＩｔｅｍｓ、およびＣａｔｅｇｏｒｙとそのメンバＩｔｅｍｓとの間の相互接続のＲｅｌａｔｉｏｎｓｈｉｐｓを例示するブロック図である。

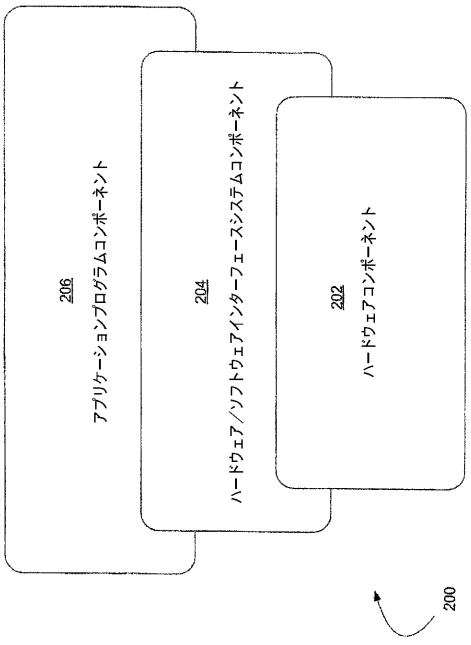
【図１１】ストレージプラットフォームのデータモデルの参照型階層を例示する図である。
【図１２】関係を分類する方法を例示する図である。
【図１３】通知メカニズムを例示する図である。
【図１４】２つのトランザクションが両方とも新しいレコードを同じＢ－Ｔｒｅｅ内に挿入する実施例を示す図である。
【図１５】データ変更検出プロセスを例示する図である。
【図１６】ディレクトリツリー例を例示する図である。
【図１７】ディレクトリベースのファイルシステムの既存のフォルダがストレージプラットフォームのデータストアに移動される例を例示する図である。
【図１８】Ｃｏｎｔａｉｎｍｅｎｔ　Ｆｏｌｄｅｒｓの概念を例示する図である。
【図１９】ストレージプラットフォームＡＰＩの基本アーキテクチャを例示する図である。
【図２０】ストレージプラットフォームＡＰＩスタックの様々なコンポーネントの概略を表す図である。
【図２１Ａ】Ｃｏｎｔａｃｔｓ　Ｉｔｅｍスキーマ例の図である。
【図２１Ｂ】図２１ＡのＣｏｎｔａｃｔｓ　Ｉｔｅｍスキーマ例のＥｌｅｍｅｎｔｓの図である。
【図２２】ストレージプラットフォームＡＰＩのランタイムフレームワークを例示する図である。
【図２３】「ＦｉｎｄＡｌｌ」オペレーションの実行を例示する図である。
【図２４】ストレージプラットフォームＡＰＩクラスがストレージプラットフォームＳｃｈｅｍａから生成されるプロセスを例示する図である。
【図２５】Ｆｉｌｅ　ＡＰＩが基づくスキーマを例示する図である。
【図２６】データセキュリティの目的のために使用されるアクセスマスク形式を例示する図である。
【図２７】（パートａ、ｂ、およびｃ）既存のセキュリティ領域から切り出される新しいまったく同じように保護されているセキュリティ領域を示す図である。
【図２８】Ｉｔｅｍの検索ビューの概念を例示するブロック図である。
【図２９】Ｉｔｅｍ階層例を例示する図である。
【図３０Ａ】第１および第２のコードセグメントが通信に使用する情報伝達ルートとしてインターフェースＩｎｔｅｒｆａｃｅ１を例示する図である。
【図３０Ｂ】システムの第１および第２のコードセグメントが媒体Ｍを介して通信できるようにするインターフェースオブジェクトＩ１およびＩ２を含むものとしてインターフェースを例示する図である。
【図３１Ａ】インターフェースＩｎｔｅｒｆａｃｅ１により提供される機能を細分し、インターフェースの通信を複数のインターフェースＩｎｔｅｒｆａｃｅ１Ａ、Ｉｎｔｅｒｆａｃｅ１Ｂ、Ｉｎｔｅｒｆａｃｅ１Ｃに変換する方法を例示する図である。
【図３１Ｂ】インターフェースＩ１により提供される機能を複数のインターフェースＩ１ａ、Ｉ１ｂ、Ｉ１ｃに細分する方法を例示する図である。
【図３２Ａ】意味のないパラメータ精度を無視できる、または任意のパラメータで置き換えることができるシナリオを例示する図である。
【図３２Ｂ】パラメータを無視するか、またはインターフェースに追加するように定義される代替えインターフェースによりインターフェースが置き換えられるシナリオを例示する図である。
【図３３Ａ】第１および第２のＣｏｄｅ　Ｓｅｇｍｅｎｔｓがその両方を含むモジュールにマージされるシナリオを例示する図である。
【図３３Ｂ】インターフェースの一部または全部が他のインターフェースにインラインで書き込まれマージされたインターフェースを形成するシナリオを例示する図である。
【図３４Ａ】ミドルウェアの１つまたは複数が第１のインターフェース上の通信を変換し

、１つまたは複数の異なるインターフェースに適合するようにする方法を例示する図である。
【図３４Ｂ】一方のインターフェースから通信を受け取るが、機能を第２および第３のインターフェースに送信するようにコードセグメントをインターフェースとともに導入する方法を例示する図である。
【図３５Ａ】ジャストインタイムコンパイラ（ＪＩＴ）が一方のコードセグメントから他方のコードセグメントに通信を変換する方法を例示する図である。
【図３５Ｂ】１つまたは複数のインターフェースを動的に書き換えるＪＩＴ方法が適用され、インターフェースを動的に因数分解するか、または他の何らかの方法で変更することができることを例示する図である。
【図３６】共通データストアの３つのインスタンスおよびそれらの同期をとるためのコンポーネントを例示する図である。
【図３７】状態がどのように計算されるか、またはその関連するメタデータがどのように交換されるかを認識しない単純なアダプタを想定する本発明の一実施形態を例示する図である。

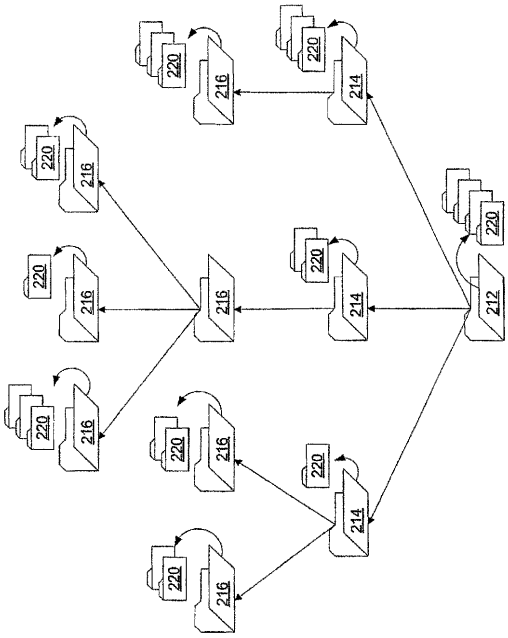
【図１】



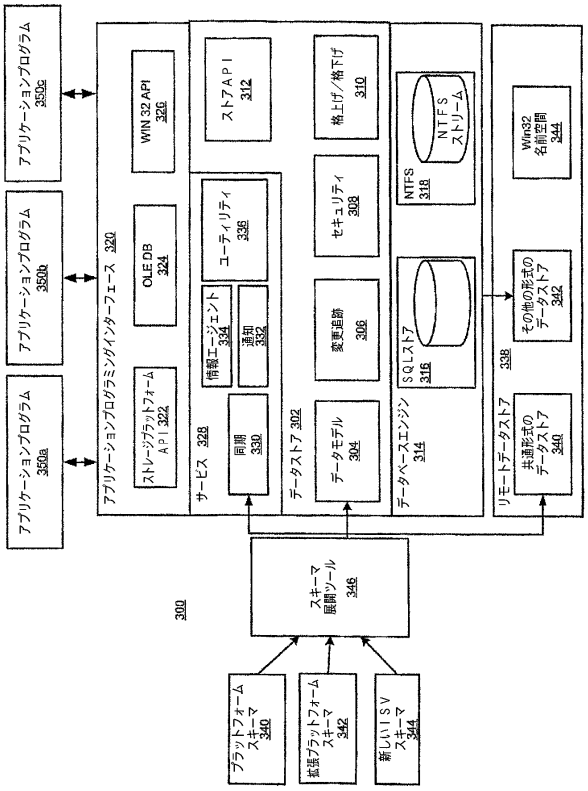
【図２】



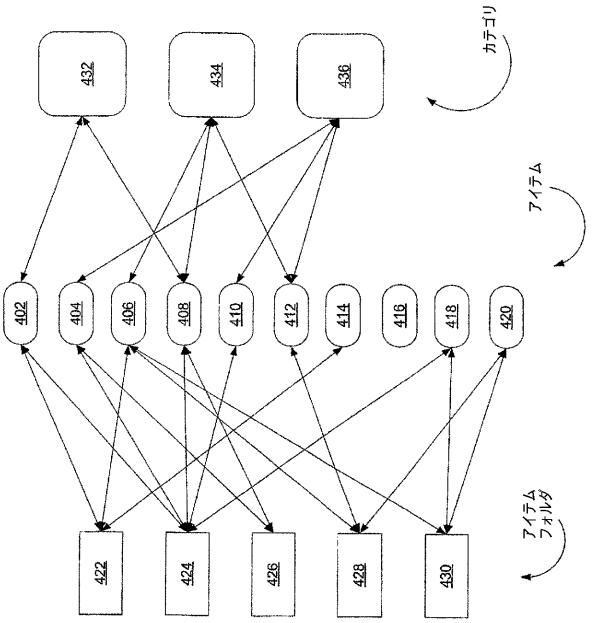
【図２Ａ】



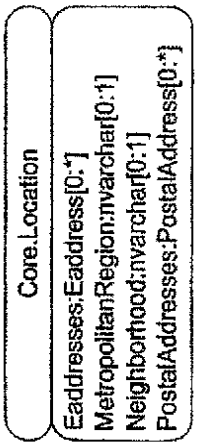
【図３】



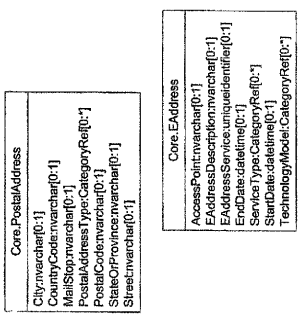
【図４】



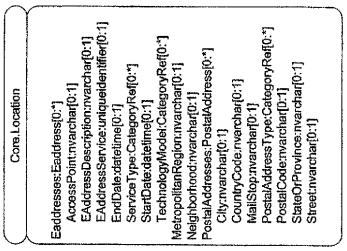
【図５Ａ】



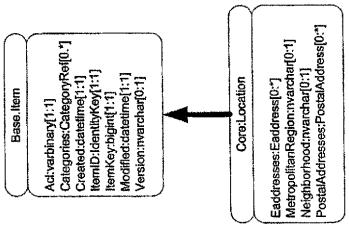
【図５Ｂ】



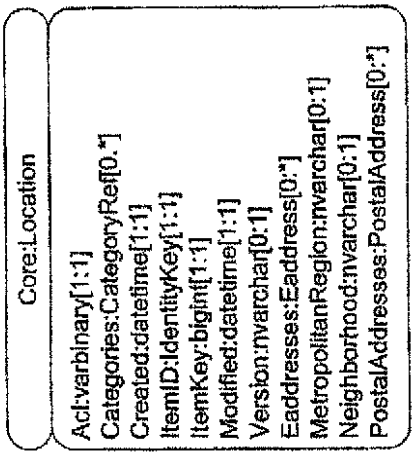
【図５Ｃ】



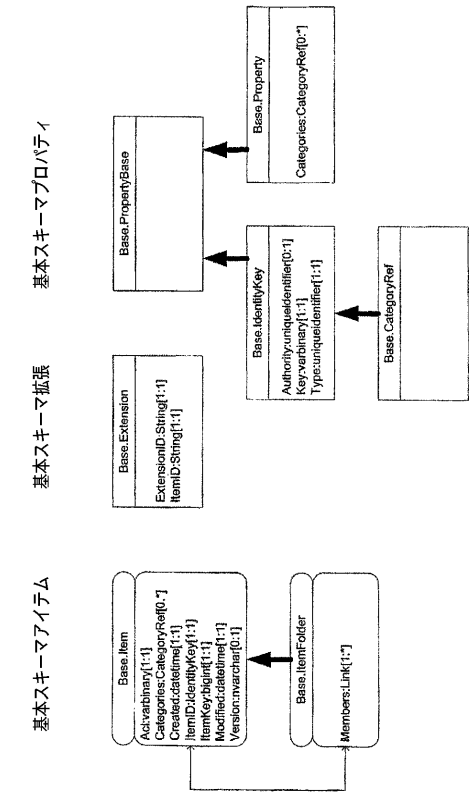
【図６Ａ】



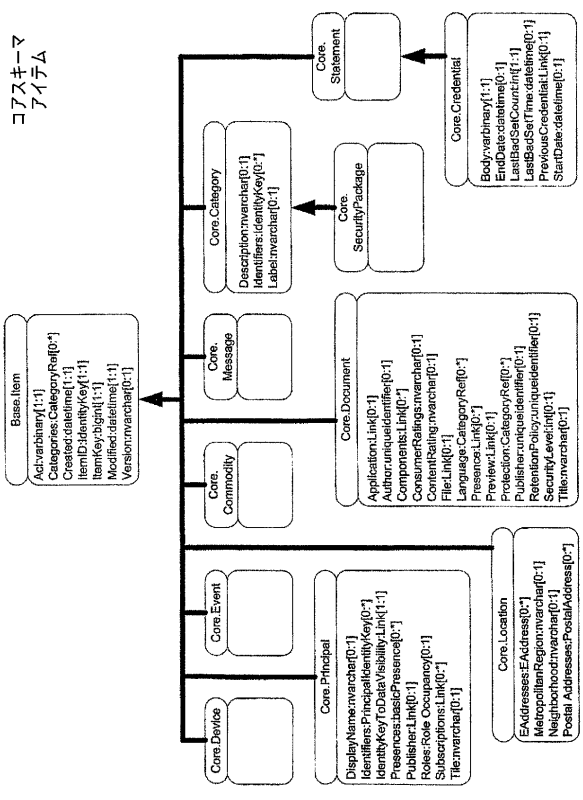
【図６Ｂ】



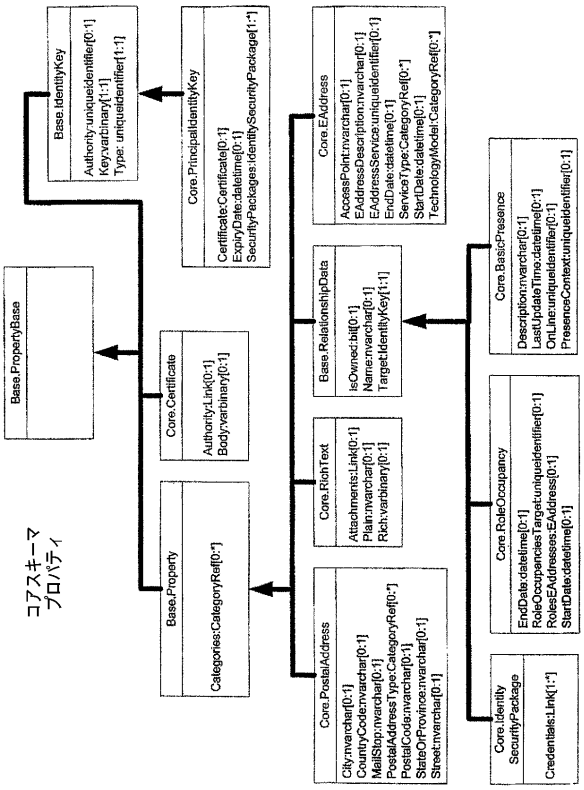
【図７】



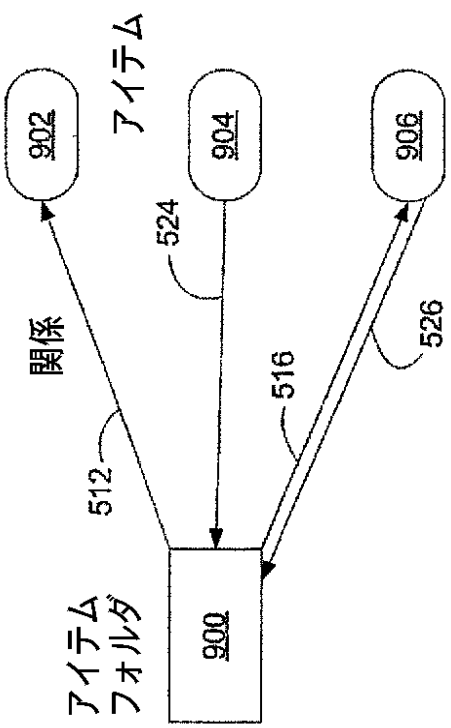
【図８Ａ】



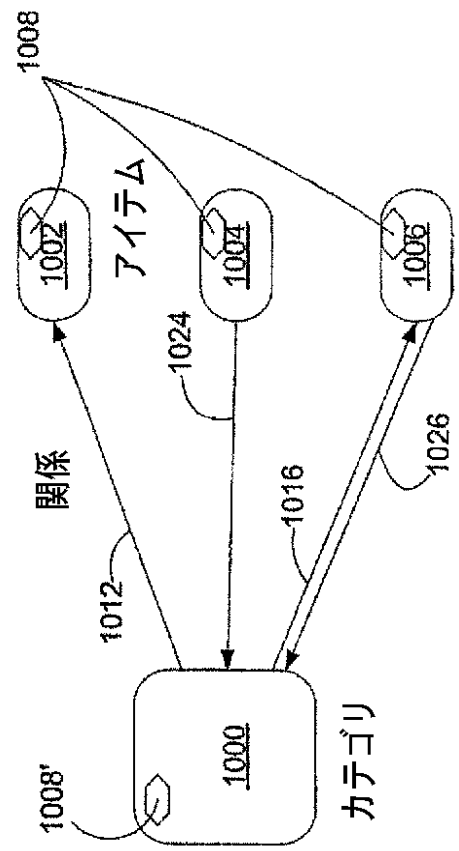
【図８Ｂ】



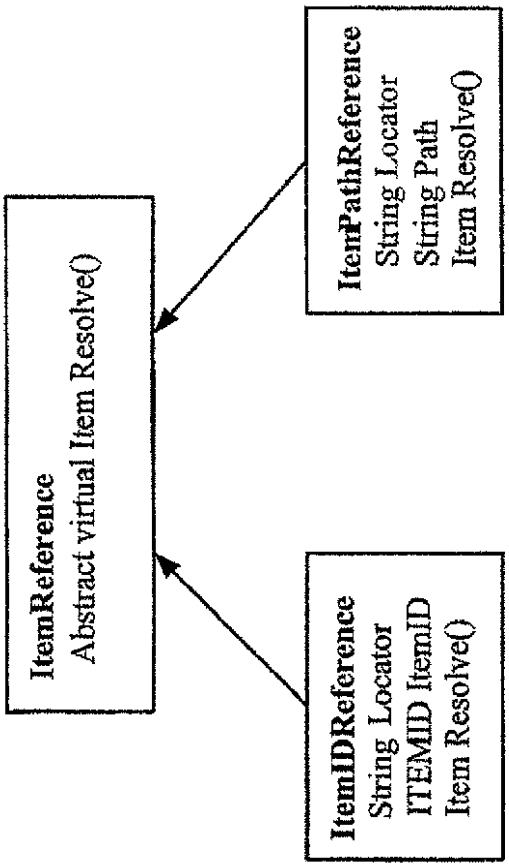
【図９】



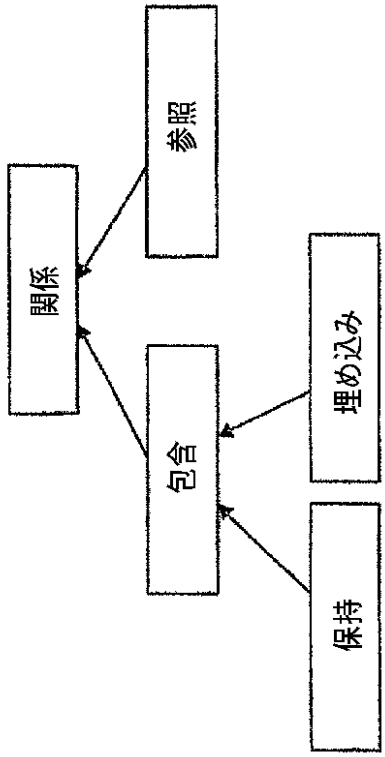
【図１０】



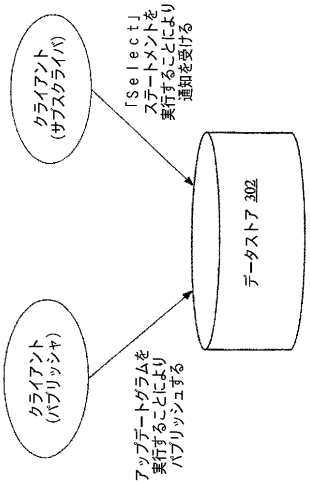
【図１１】



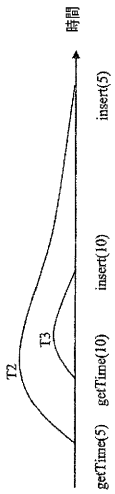
【図１２】



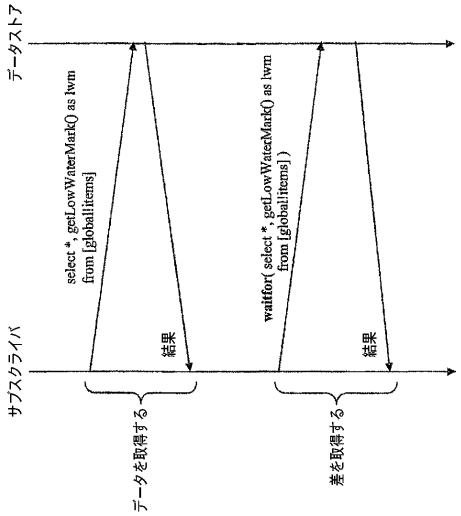
【図１３】



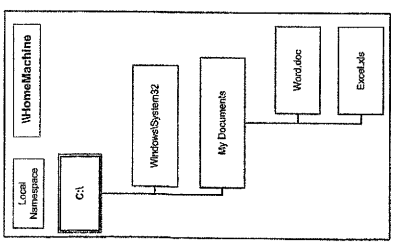
【図１４】



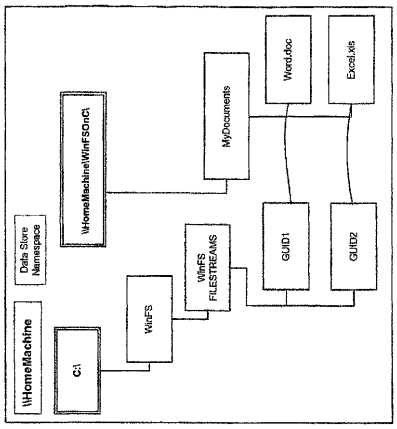
【図１５】



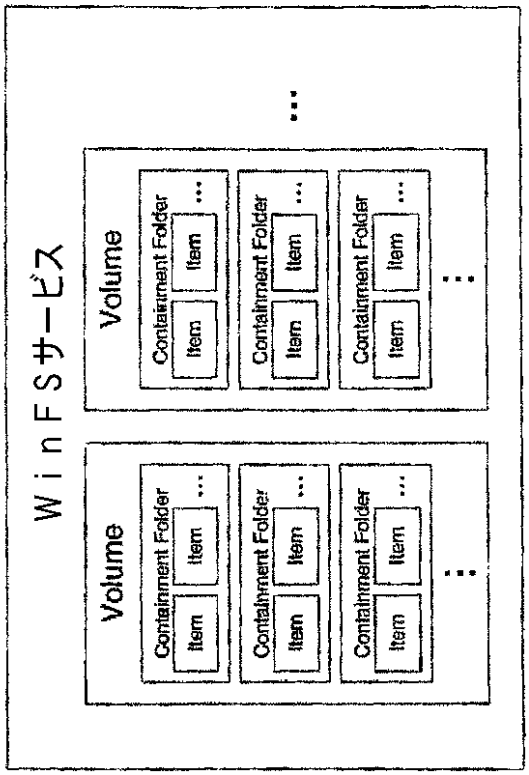
【図１６】



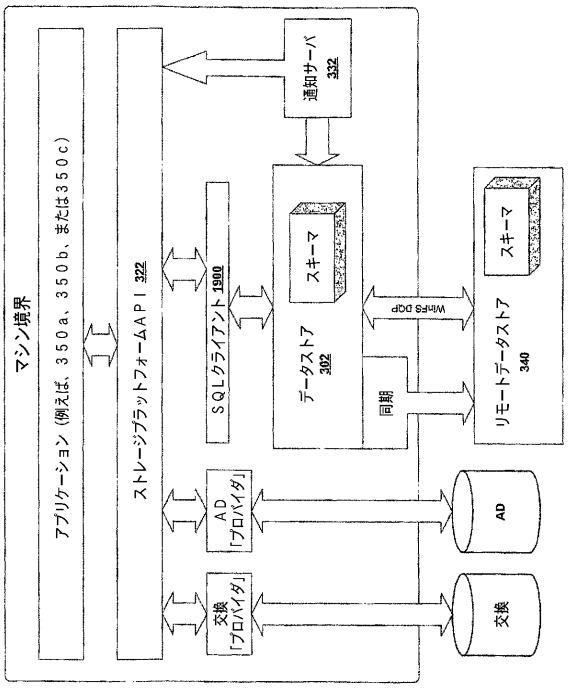
【図１７】



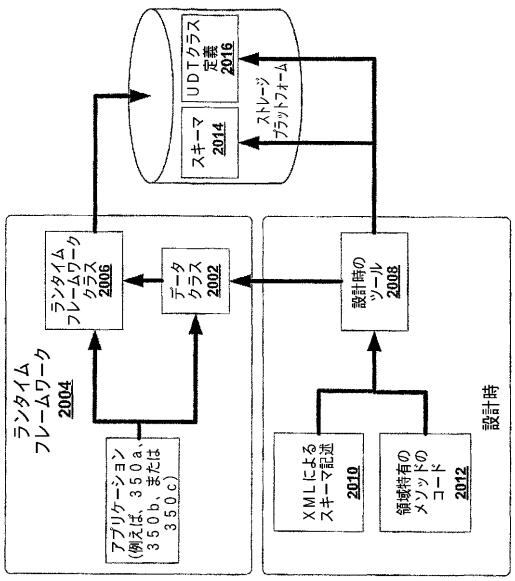
【図１８】



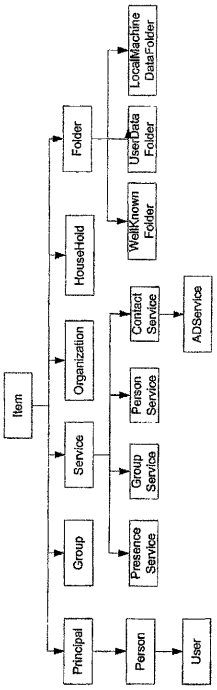
【図１９】



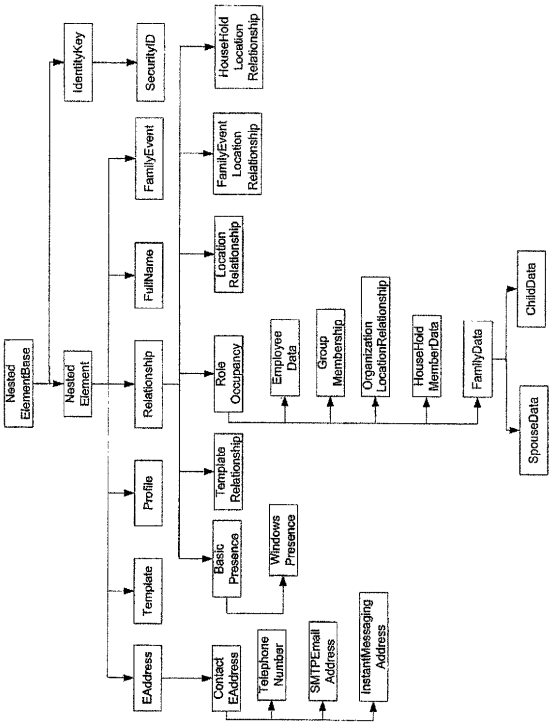
【図２０】



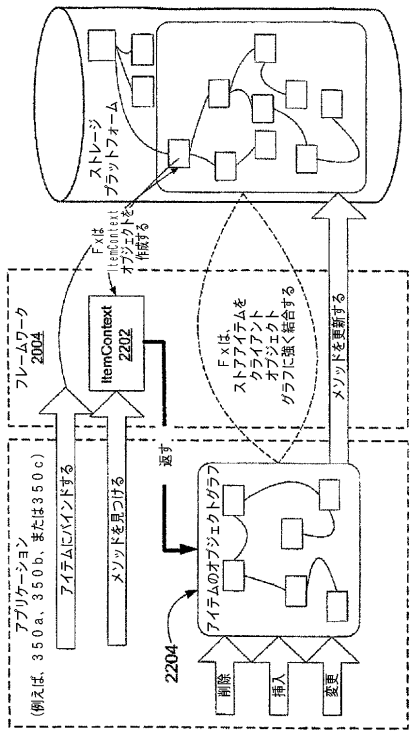
【図２１Ａ】



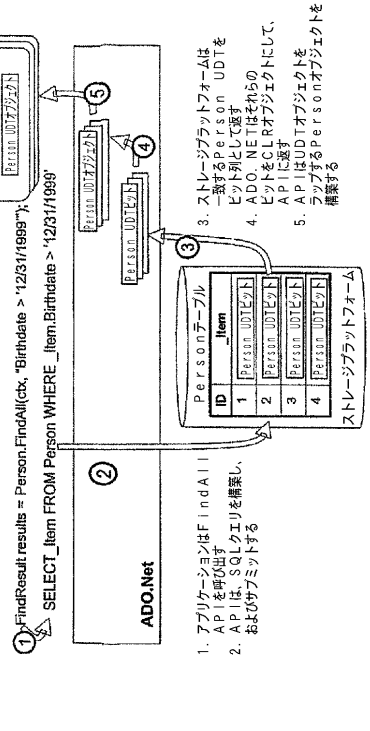
【図２１Ｂ】



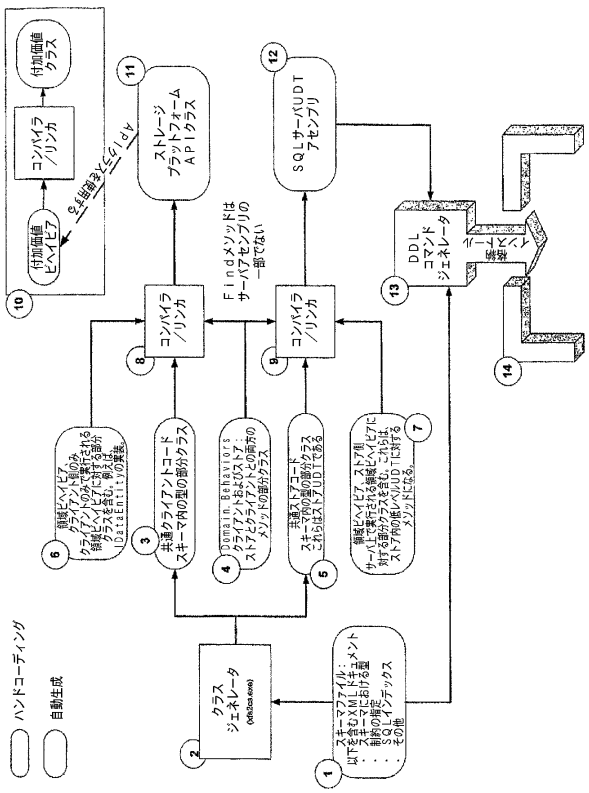
【図２２】



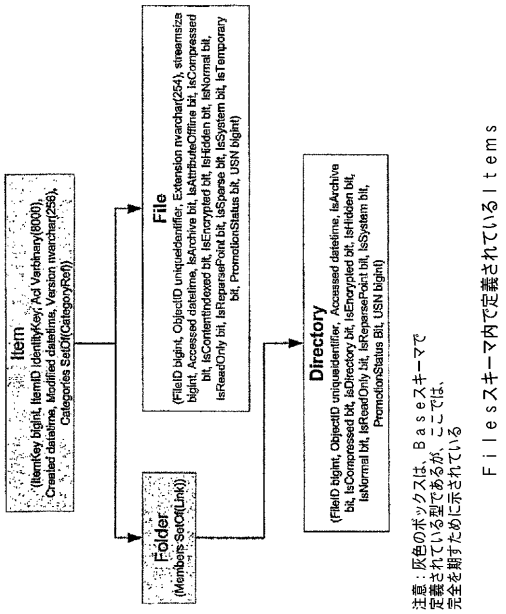
【図２３】



【図２４】

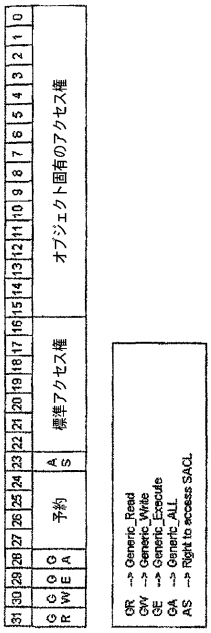


【図２５】

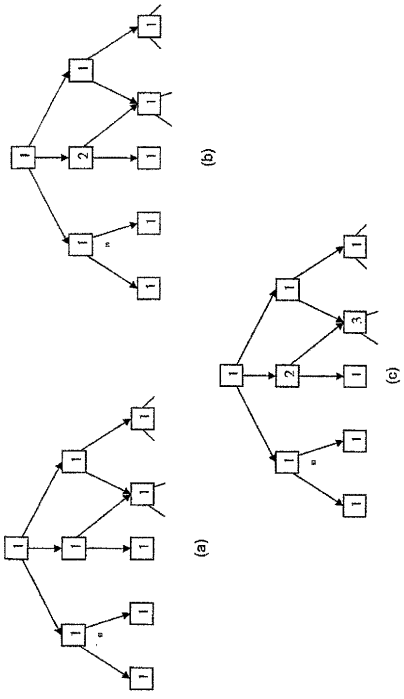


注意：灰色の ボックスは、BaseスキーマでFilesスキーマ内で定義されているItems
定義されている型であるが、ここでは、
完全を期すために示されている

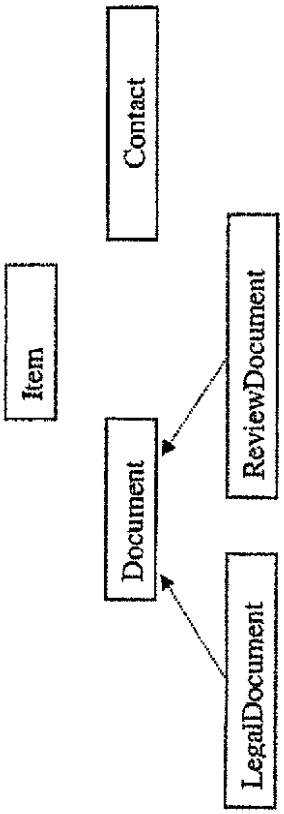
【図２６】



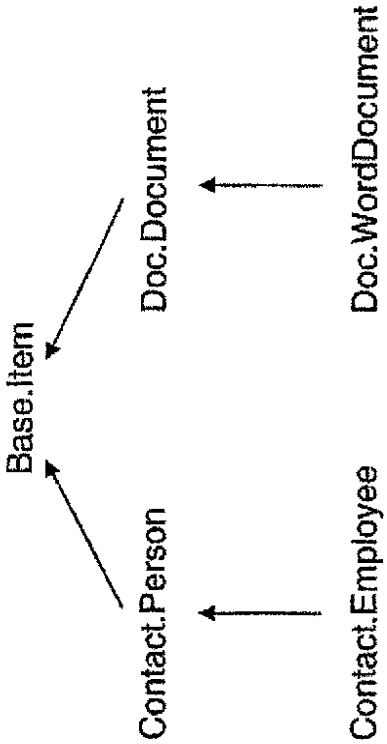
【図２７】



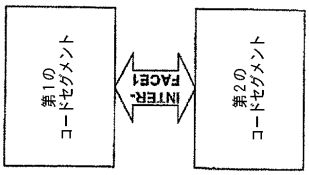
【図２８】



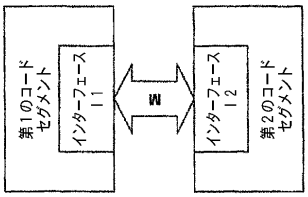
【図２９】



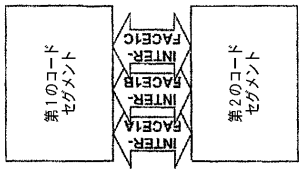
【図３０Ａ】



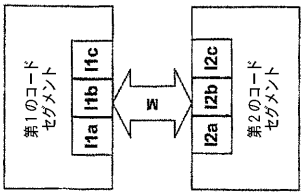
【図３０Ｂ】



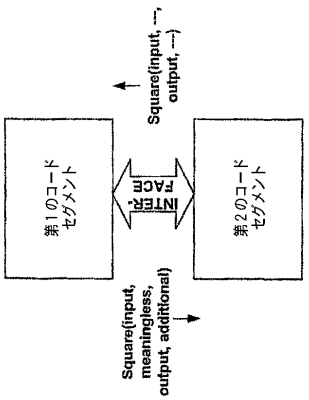
【図３１Ａ】



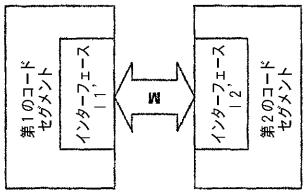
【図３１Ｂ】



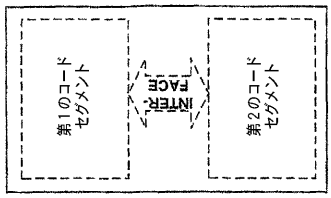
【図３２Ａ】



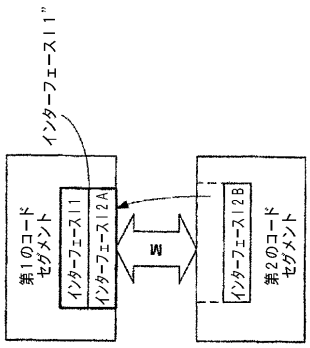
【図３２Ｂ】



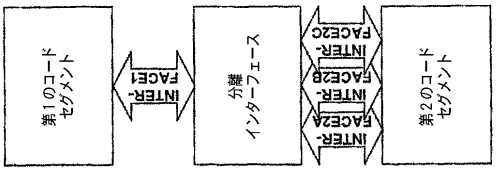
【図３３Ａ】



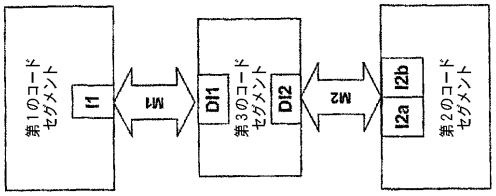
【図３３Ｂ】



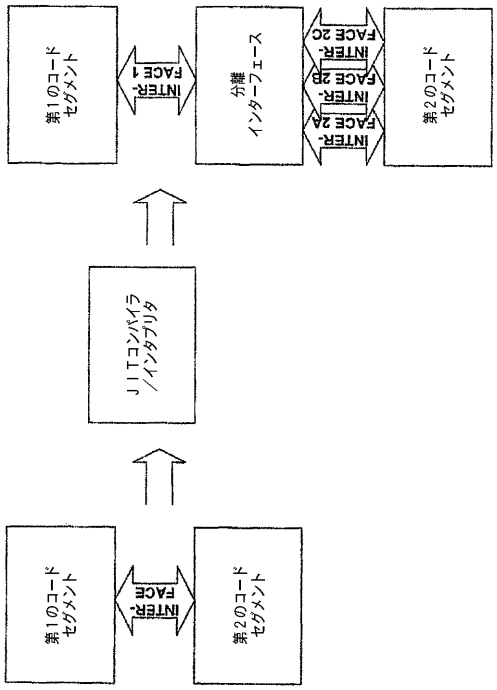
【図３４Ａ】



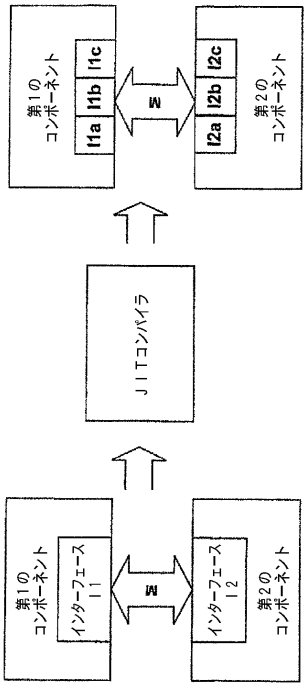
【図３４Ｂ】



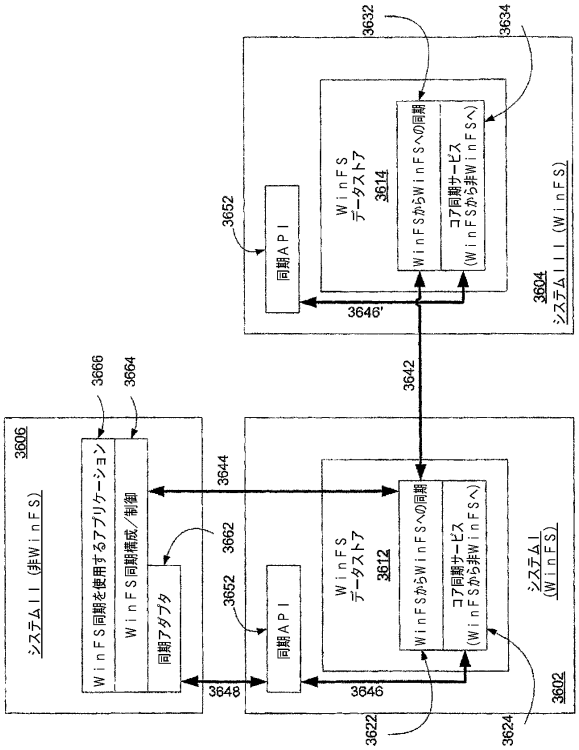
【図３５Ａ】



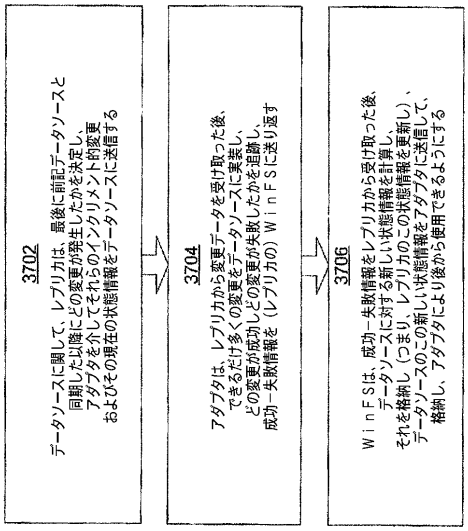
【図３５Ｂ】



【図３６】


システムI (WinFS)
システムII (非WinFS)
システムIII (WinFS)
WinFS同期を使用するアプリケーション
WinFS同期構成／制御
同期アダプタ
同期API
WinFS データストア
WinFSからWinFSへの同期
コア同期サービス (WinFSから非WinFSへ)
3602
3604
3606
3612
3614
3622
3624
3632
3634
3642
3644
3646
3646'
3648
3652
3662
3664
3666

【図３７】


3702
データソースに関して、レプリカは、最後に前記データソースと同期した以降にどの変更が発生したかを決定し、アダプタを介してそれらのインクリメント的変更およびその現在の状態情報をデータソースに送信する
3704
アダプタは、レプリカから変更データを受け取った後、できるだけ多くの変更をデータソースに実装し、どの変更が成功しどの変更が失敗したかを追跡し、成功－失敗情報を（レプリカの）WinFSに送り返す
3706
WinFSは、成功－失敗情報をレプリカから受け取った後、データソースに対する新しい状態情報を計算し、それを格納し（つまり、レプリカの古い状態情報を更新し）、データソースの新しい状態情報をアダプタに送信して、格納し、アダプタにより後から使用できるようにする

フロントページの続き

(31)優先権主張番号 10/693,362
(32)優先日 平成15年10月24日(2003.10.24)
(33)優先権主張国 米国(US)

(72)発明者 レブ ノビク
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション内
(72)発明者 イリナ フーディス
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション内
(72)発明者 トーマス タリアス
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション内
(72)発明者 ジェイ．パトリック トンプソン
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション内

審査官 田川 泰宏

(56)参考文献 特開２００２－１４９４６４（ＪＰ，Ａ）
国際公開第０３／０４４６９８（WO，Ａ１）
特表２００５－５０９９７９（ＪＰ，Ａ）
国際公開第０２／０９７６２３（WO，Ａ１）
特表２００５－５０７１００（ＪＰ，Ａ）
国際公開第０３／０４６７５９（WO，Ａ１）
特表２００５－５１０８０５（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 12/00
G06F 17/30