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

特許第4738908号
(P4738908)

(45)発行日　平成23年8月3日(2011.8.3)　(24)登録日　平成23年5月13日(2011.5.13)

(51)Int.Cl.		F I		
G06F 13/00	(2006.01)	G06F	13/00	351A
G06F 12/00	(2006.01)	G06F	12/00	545Z
G06F 15/00	(2006.01)	G06F	15/00	310U

請求項の数 24　(全 109 頁)

(21)出願番号　特願2005-175754 (P2005-175754)
(22)出願日　平成17年6月15日 (2005.6.15)
(65)公開番号　特開2006-18821 (P2006-18821A)
(43)公開日　平成18年1月19日 (2006.1.19)
審査請求日　平成20年6月13日 (2008.6.13)
(31)優先権主張番号　10/883,621
(32)優先日　平成16年6月30日 (2004.6.30)
(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　ビー．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】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可能な情報の単位のピアツーピア同期化のための競合処理を提供するためのシステムおよび方法

(57)【特許請求の範囲】

【請求項１】

ピアツーピア同期化システムにおける競合を処理するための方法であって、

前記ピアツーピア同期化システムが、データの変更に関する前記競合を検出するステップであって、前記競合は、同期化オペレーション中にデータストア内に格納可能なデータの単位であるアイテムに関するデータの変更を適用することができない場合に生じる、ステップと、

前記ピアツーピア同期化システムが、第１の競合アイテムを作成するステップであって、前記第１の競合アイテムは、特定のデータの単位に対する変更に関係し、前記変更がなされた前記特定のデータの単位は、特定の変更単位を定める、ステップと、

前記ピアツーピア同期化システムが、前記第１の競合アイテムを永続的データストアにログするステップであって、前記第１の競合アイテムは、前記特定のデータの単位に対する変更に関連する情報を備える、ステップとを備え、

前記ログするステップは、

前記永続的データストアを検索し、前記第１の競合アイテムと同じ特定の変更単位についてすでに存在する先在の競合アイテムがあるかどうかを判定するステップと、

前記永続的データストア内に、前記第１の競合アイテムと同じ特定の変更単位についてすでに存在する先在の競合アイテムがある場合、前記先在の競合アイテムのデータの単位に対する変更が前記第１の競合アイテムの前記データの単位に対する変更によって包含される場合に前記先在の競合アイテムを除去するステップと、

前記永続的データストア内に、前記第１の競合アイテムと同じ特定の変更単位についてすでに存在する先在の競合アイテムがある場合、前記第１の競合アイテムのデータの単位に対する変更が前記先在の競合アイテムの前記データの単位に対する変更によって包含される場合に前記第１の競合アイテムを除去するステップと、

前記永続的データストア内に、前記第１の競合アイテムと同じ特定の変更単位についてすでに存在する先在の競合アイテムがある場合、および、前記第１の競合アイテムの前記データの単位に対する変更が前記先在の競合アイテムの前記データの単位に対する変更によって包含されない場合、および、前記先在の競合アイテムの前記データの単位に対する変更が前記第１の競合アイテムの前記データの単位に対する変更によって包含されない場合、前記先在の競合アイテムを前記永続的データストア内で保持し、前記第１の競合アイテムを前記永続的データストアに追加するステップと

を備えることを特徴とする方法。

【請求項２】

前記ピアツーピア同期化システムが、少なくとも１つの競合ハンドラが規定された競合解決ポリシーに従って前記競合を自動的に解決するステップであって、前記競合ハンドラは、前記競合を評価して前記競合を所定の競合ハンドラに渡すかどうかを判定するフィルタ、前記競合の解決を試みるリゾルバ、及び前記第１の競合アイテムを前記永続的データストア内にログするロガーからなるグループのうちの１つである、ステップをさらに備えることを特徴とする請求項１に記載の方法。

【請求項３】

前記競合ハンドラはアイテムであり、拡張可能であり、前記競合ハンドラのグループは作成可能および拡張可能であることを特徴とする請求項２に記載の方法。

【請求項４】

前記競合解決ポリシーは、競合処理パイプラインを備える少なくとも２つの競合ハンドラが規定されていることを特徴とする請求項３に記載の方法。

【請求項５】

前記ピアツーピア同期化システムが、前記データストア内の古くなった競合アイテムの除去を含む、古くなった競合の自動検出および除去するステップをさらに備えることを特徴とする請求項２に記載の方法。

【請求項６】

ターゲットアイテムのコピーは、前記競合を起こす前記変更を反映することを特徴とする請求項１に記載の方法。

【請求項７】

ピアツーピア同期化システムにおける競合を処理するためのシステムであって、

データの変更に関する前記競合を検出するためのサブシステムであって、前記競合は、同期化オペレーション中にデータストア内に格納可能なデータの単位であるアイテムに関するデータの変更を適用することができない場合に生じる、サブシステムと、

第１の競合アイテムを作成するためのサブシステムであって、前記第１の競合アイテムは、特定のデータの単位に対する変更に関係し、前記変更がなされた前記特定のデータの単位は、特定の変更単位を定める、サブシステムと、

前記第１の競合アイテムを永続的データストアにログするためのサブシステムであって、前記第１の競合アイテムは、前記特定のデータの単位に対する変更に関連する情報を備える、サブシステムと、

前記競合を解決するためのサブシステムとを備え、

前記ログするためのサブシステムは、

前記永続的データストアを検索し、前記第１の競合アイテムと同じ特定の変更単位についてすでに存在する先在の競合アイテムがあるかどうかを判定するためのサブシステムと、

前記永続的データストア内に、前記第１の競合アイテムと同じ特定の変更単位についてすでに存在する先在の競合アイテムがある場合、前記先在の競合アイテムのデータの単位

に対する変更が前記第１の競合アイテムの前記データの単位に対する変更によって包含される場合に前記先在の競合アイテムを除去するためのサブシステムと、
前記永続的データストア内に、前記第１の競合アイテムと同じ特定の変更単位についてすでに存在する先在の競合アイテムがある場合、前記第１の競合アイテムのデータの単位に対する変更が前記先在の競合アイテムの前記データの単位に対する変更によって包含される場合に前記第１の競合アイテムを除去するためのサブシステムと、
前記永続的データストア内に、前記第１の競合アイテムと同じ特定の変更単位についてすでに存在する先在の競合アイテムがある場合、および、前記第１の競合アイテムの前記データの単位に対する変更が前記先在の競合アイテムの前記データの単位に対する変更によって包含されない場合、および、前記先在の競合アイテムの前記データの単位に対する変更が前記第１の競合アイテムの前記データの単位に対する変更によって包含されない場合、前記先在の競合アイテムを前記永続的データストア内で保持し、前記第１の競合アイテムを前記永続的データストアに追加するためのサブシステムと
を備えることを特徴とするシステム。

【請求項８】
少なくとも１つの競合ハンドラが規定された競合解決ポリシーに従って前記競合を自動的に解決するためのサブシステムであって、前記競合ハンドラは、前記競合を評価して前記競合を所定の競合ハンドラに渡すかどうかを判定するフィルタ、前記競合の解決を試みるリゾルバ、及び前記第１の競合アイテムを前記永続的データストア内にログするロガーからなるグループのうちの１つである、サブシステムと
をさらに備え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】
拡張可能な競合ハンドラと、
作成可能および拡張可能な競合ハンドラのグループと
をさらに備えることを特徴とする請求項８に記載のシステム。

【請求項１０】
競合処理パイプラインをさらに備え、前記競合処理パイプラインは少なくとも２つの競合ハンドラを備えることを特徴とする請求項９に記載のシステム。

【請求項１１】
前記データストア内の古くなった競合アイテムの除去を含む、古くなった競合の自動検出および除去のためのサブシステムをさらに備えることを特徴とする請求項８に記載のシステム。

【請求項１２】
ターゲットアイテムのコピーを、前記競合を起こす前記変更を反映するように修正するためのサブシステムをさらに備えることを特徴とする請求項８に記載のシステム。

【請求項１３】
ピアツーピア同期化システムにおける競合を処理するための方法を実行するためのコンピュータ可読命令を格納したコンピュータ可読記録媒体であって、
前記ピアツーピア同期化システムが、データの変更に関する前記競合を検出するステップであって、前記競合は、同期化オペレーション中にデータストア内に格納可能なデータの単位であるアイテムに関するデータの変更を適用することができない場合に生じる、ステップと、
前記ピアツーピア同期化システムが、第１の競合アイテムを作成するステップであって、前記第１の競合アイテムは、特定のデータの単位に対する変更に関係し、前記変更がなされた前記特定のデータの単位は、特定の変更単位を定める、ステップと、
前記ピアツーピア同期化システムが、前記第１の競合アイテムを永続的データストアにログするステップであって、前記第１の競合アイテムは、前記特定のデータの単位に対する変更に関連する情報を備える、ステップとを備え、
前記ログするステップは、
前記永続的データストアを検索し、前記第１の競合アイテムと同じ特定の変更単位につ

いてすでに存在する先在の競合アイテムがあるかどうかを判定するステップと、
前記永続的データストア内に、前記第１の競合アイテムと同じ特定の変更単位についてすでに存在する先在の競合アイテムがある場合、前記先在の競合アイテムのデータの単位に対する変更が前記第１の競合アイテムの前記データの単位に対する変更によって包含される場合に前記先在の競合アイテムを除去するステップと、
前記永続的データストア内に、前記第１の競合アイテムと同じ特定の変更単位についてすでに存在する先在の競合アイテムがある場合、前記第１の競合アイテムのデータの単位に対する変更が前記先在の競合アイテムの前記データの単位に対する変更によって包含される場合に前記第１の競合アイテムを除去するステップと、
前記永続的データストア内に、前記第１の競合アイテムと同じ特定の変更単位についてすでに存在する先在の競合アイテムがある場合、および、前記第１の競合アイテムの前記データの単位に対する変更が前記先在の競合アイテムの前記データの単位に対する変更によって包含されない場合、および、前記先在の競合アイテムの前記データの単位に対する変更が前記第１の競合アイテムの前記データの単位に対する変更によって包含されない場合、前記先在の競合アイテムを前記永続的データストア内で保持し、前記第１の競合アイテムを前記永続的データストアに追加するステップと
を備える方法を実行するためのコンピュータ可読命令を格納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】
前記方法は、前記ピアツーピア同期化システムが、少なくとも１つの競合ハンドラが規定された競合解決ポリシーに従って前記競合を自動的に解決するステップであって、前記競合ハンドラは、前記競合を評価して前記競合を所定の競合ハンドラに渡すかどうかを判定するフィルタ、前記競合の解決を試みるリゾルバ、及び前記第１の競合アイテムを前記永続的データストア内にログするロガーからなるグループのうちの１つである、ステップをさらに備え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】
前記競合ハンドラはアイテムであり、拡張可能であり、
前記競合ハンドラのグループは作成可能および拡張可能であることを特徴とする請求項１４に記載のコンピュータ可読記録媒体。
【請求項１６】
前記競合解決ポリシーは、競合処理パイプラインを備える少なくとも２つの競合ハンドラが規定されていることを特徴とする請求項１５に記載のコンピュータ可読記録媒体。
【請求項１７】
前記方法は、前記ピアツーピア同期化システムが、前記データストア内の古くなった競合アイテムの除去を含む、古くなった競合の自動検出および除去するステップをさらに備えることを特徴とする請求項１４に記載のコンピュータ可読記録媒体。
【請求項１８】
ターゲットアイテムのコピーは、前記競合を起こす前記変更を反映す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１９】
ピアツーピア同期化システムにおける競合を処理するためのハードウェアコントロールデバイスであって、
データの変更に関する前記競合を検出する手段であって、前記競合は、同期化オペレーション中にデータストア内に格納可能なデータの単位であるアイテムに関するデータの変更を適用することができない場合に生じる、手段と、
第１の競合アイテムを作成する手段であって、前記第１の競合アイテムは、特定のデータの単位に対する変更に関係し、前記変更がなされた前記特定のデータの単位は、特定の変更単位を定める、手段と、
前記第１の競合アイテムを永続的データストアにログする手段であって、前記第１の競合アイテムは、前記特定のデータの単位に対する変更に関連する情報を備える、手段とを

備え、
前記ログする手段は、
前記永続的データストアを検索し、前記第１の競合アイテムと同じ特定の変更単位についてすでに存在する先在の競合アイテムがあるかどうかを判定する手段と、
前記永続的データストア内に、前記第１の競合アイテムと同じ特定の変更単位についてすでに存在する先在の競合アイテムがある場合、前記先在の競合アイテムのデータの単位に対する変更が前記第１の競合アイテムの前記データの単位に対する変更によって包含される場合に前記先在の競合アイテムを除去する手段と、
前記永続的データストア内に、前記第１の競合アイテムと同じ特定の変更単位についてすでに存在する先在の競合アイテムがある場合、前記第１の競合アイテムのデータの単位に対する変更が前記先在の競合アイテムの前記データの単位に対する変更によって包含される場合に前記第１の競合アイテムを除去する手段と、
前記永続的データストア内に、前記第１の競合アイテムと同じ特定の変更単位についてすでに存在する先在の競合アイテムがある場合、および、前記第１の競合アイテムの前記データの単位に対する変更が前記先在の競合アイテムの前記データの単位に対する変更によって包含されない場合、および、前記先在の競合アイテムの前記データの単位に対する変更が前記第１の競合アイテムの前記データの単位に対する変更によって包含されない場合、前記先在の競合アイテムを前記永続的データストア内で保持し、前記第１の競合アイテムを前記永続的データストアに追加する手段と
を備えることを特徴とするハードウェアコントロールデバイス。

【請求項２０】
少なくとも１つの競合ハンドラが規定された競合解決ポリシーに従って前記競合を自動的に解決する手段であって、前記競合ハンドラは、前記競合を評価して前記競合を所定の競合ハンドラに渡すかどうかを判定するフィルタ、前記競合の解決を試みるリゾルバ、及び前記第１の競合アイテムを前記永続的データストア内にログするロガーからなるグループのうちの１つである、手段と
をさらに備えることを特徴とする請求項１９に記載のハードウェアコントロールデバイス。

【請求項２１】
前記競合ハンドラを拡張可能アイテムとする手段と、
前記競合ハンドラのグループを作成可能および拡張可能とする手段と
をさらに備えることを特徴とする請求項２０に記載のハードウェアコントロールデバイス。

【請求項２２】
前記競合解決ポリシーは、競合処理パイプラインを備える少なくとも２つの競合ハンドラが規定されていることを特徴とする請求項２０に記載のハードウェアコントロールデバイス。

【請求項２３】
前記データストア内の古くなった競合アイテムの除去を含む、古くなった競合の自動検出および除去の手段をさらに備えることを特徴とする請求項１９に記載のハードウェアコントロールデバイス。

【請求項２４】
ターゲットアイテムのコピーは、前記競合を起こす前記変更を反映する、手段をさらに備えることを特徴とする請求項１９に記載のハードウェアコントロールデバイス。

【発明の詳細な説明】
【技術分野】
【０００１】
本出願は一般に情報のストレージおよび検索の分野、ならびに、異なるタイプのデータをコンピュータ化されたシステムにおいて編成、検索および共有するためのアクティブストレージプラットフォームに関する。具体的には、本発明は、データプラットフォームの

複数のインスタンスの間のデータのピアツーピア同期化のための競合処理に関する。

【背景技術】

【０００２】

個々のディスク容量は、過去１０年間で１年あたりほぼ７０％増大してきた。ムーアの法則は、長年にわたって起こっている、中央処理装置（ＣＰＵ）能力における途方もなく大きい進歩を正確に予測した。有線および無線技術は、途方もなく大きい接続性および帯域幅を提供している。現在の傾向が継続すると仮定すると、数年以内に、平均的なラップトップコンピュータは約１テラバイト（ＴＢ）のストレージを保有するようになり、何百万のファイルを含むようになり、５００ギガバイト（ＧＢ）のドライブが一般的となるであろう。

【０００３】

【特許文献１】米国特許出願第１０／６４６６４６号

【特許文献２】米国特許出願第１０／６４６６４６号

【０００４】

消費者は自分のコンピュータを、従来の個人情報マネージャ（ＰＩＭ）スタイルのデータであるか、デジタル音楽または写真などのメディアであるかにかかわらず、主に通信および個人情報の編成のために使用する。デジタルコンテンツの量、および生のバイトを格納する能力は、大いに増大しているが、消費者がこのデータを編成および単一化するために使用可能な方法は、この増大についてきていない。知識労働者は情報の管理および共有に莫大な量の時間を費やしており、ある調査では、知識労働者は１５～２５％の時間を非生産的情報関連のアクティビティに費やすと推定する。他の調査では、通常の知識労働者は１日につき約２．５時間を情報の検索に費やすと推定する。

【０００５】

開発者および情報技術（ＩＴ）部門は、著しい量の時間および費用を、人々、場所、時間およびイベントのような物を表現するための共通ストレージ抽象化のための、それら独自のデータストアの構築に投資する。これは重複した作業をもたらす結果となるだけでなく、そのデータを共通して検索または共有するためのメカニズムのない、共通データのアイランドをも作り出す。まさに、どれだけの数のアドレス帳が今日、Ｍｉｃｒｏｓｏｆｔ Ｗｉｎｄｏｗｓ（登録商標）オペレーティングシステムを実行するコンピュータ上に存在する可能性があるかを考慮されたい。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、多数のアプリケーションは個々のアドレス帳を保持しており、アプリケーション間で、このような各プログラムが個々に維持するアドレス帳データの共有はほとんどない。したがって、ファイナンスプログラム（Ｍｉｃｒｏｓｏｆｔ Ｍｏｎｅｙなど）は、受取人のためのアドレスを、電子メール連絡先フォルダ（Ｍｉｃｒｏｓｏｆｔ Ｏｕｔｌｏｏｋ内のフォルダなど）内で維持されたアドレスと共有しない。実際に、多数のユーザは複数のデバイスを有し、論理的にはそれらの個人データをそれらのデバイスの間で、および、携帯電話からＭＳＮおよびＡＯＬなどの商業サービスまでを含む幅広い種類の追加のソースにわたって同期化するべきであり、それにもかかわらず、共有ドキュメントのコラボレーションは主として、ドキュメントを電子メールメッセージに添付することによって、すなわち、手動および非効率的に達成される。

【０００６】

このコラボレーションの欠乏の１つの理由は、コンピュータシステム内の情報の編成に対する従来の手法が、ファイルフォルダおよびディレクトリベースのシステム（「ファイルシステム」）の使用を中心として、複数のファイルをフォルダのディレクトリ階層に、これらのファイルを格納するために使用されたストレージメディアの物理的編成の抽象化に基づいて編成してきたことである。Ｍｕｌｔｉｃｓオペレーティングシステムは１９６０年代に開発され、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するためのファイル、フォルダおよびディレクトリの使用を開拓したことで功績があると考えることができる。具体的には、Ｍｕｌｔｉｃｓは記号アドレスをファイルの階層内で使用し（それにより、ファイルパスのアイデアを導入し）、ファイルの物理アドレスはユーザ

（アプリケーションおよびエンドユーザ）に透過的ではなかった。このファイルシステムはまったく、いかなる個々のファイルのファイルフォーマットにも無関心であり、複数のファイルの間のリレーションシップは、オペレーティングシステムレベル（すなわち、階層内のファイルのロケーション以外）で無関係と見なされた。Ｍｕｌｔｉｃｓの出現以来、格納可能データはファイル、フォルダおよびディレクトリへ、オペレーティングシステムレベルで編成されてきた。これらのファイルは一般にファイル階層自体（「ディレクトリ」）を含み、ディレクトリはファイルシステムによって維持された特殊ファイル内で実施される。このディレクトリは、ディレクトリ内の他のファイルのすべてに対応するエントリのリスト、および、階層内のこのようなファイルのノードのロケーションを維持する（本明細書でフォルダと称する）。このような物は、約４０年間にわたって現況技術であった。

【０００７】

しかし、コンピュータの物理的ストレージシステム内に存在する情報の妥当な表現を提供するが、ファイルシステムはそれにもかかわらず、その物理ストレージシステムの抽象化であり、したがって、ファイルの利用には、ユーザが操作する物（コンテキスト、特徴および他の単位とのリレーションシップを有する単位）と、オペレーティングシステムが提供する物（ファイル、フォルダおよびディレクトリ）の間であるレベルの間接化（解釈）が必要である。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、そうすることが非効率的であり、矛盾しており、そうでない場合は望ましくない場合でも、情報の単位をファイルシステム構造に強いる以外の選択肢はない。また、既存のファイルシステムは、個々のファイル内に格納されたデータの構造についてほとんど知らず、このため、大部分の情報はファイル内でロックされたままであり、それらの情報を書き込んだアプリケーションにとってのみアクセスする（および理解可能である）ことが可能である。したがって、この情報の概略記述、および、情報を管理するためのメカニズムの欠如は、個々のサイロ間のデータ共有がほとんどない、データのサイロの作成につながる。例えば、多数のパーソナルコンピュータ（ＰＣ）ユーザは、あるレベル、例えば、Ｏｕｔｌｏｏｋ　Ｃｏｎｔａｃｔｓ、オンラインアカウントアドレス、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）バディリストで、対話する相手の人々についての情報を含む、５つ以上の異なるストアを有し、これは、ファイルの編成がこれらのＰＣユーザに対して著しい課題を提示するからである。大部分の既存のファイルシステムは、ネストされたフォルダのメタファを、ファイルおよびフォルダの編成のために利用するので、ファイルの数が増すにつれて、柔軟性があり効率的である編成スキームを維持するために必要な取り組みは極めて困難になる。このような状況で、単一のファイルの複数の分類を有することは大変有用となるが、既存のファイルシステムにおけるハードまたはソフトリンクの使用は、維持することが煩わしく困難である。

【０００８】

ファイルシステムの欠点に対処するためのいくつかの不成功な試みが過去に行われている。これらの以前の試みのいくつかは、コンテンツアドレス可能メモリを使用して、それにより物理アドレスではなくコンテンツによってデータにアクセスすることができるメカニズムを提供することに関係した。しかし、これらの取り組みは不成功であることが判明しており、これは、コンテンツアドレス可能メモリが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の使用に有用であることは判明しているが、物理ストレージメディアなどのデバイス向けの大規模の使用は、様々な理由のためにまだ可能となっておらず、したがって、このような解決法は単に存在しないからである。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が行われているが、これらの試みは、強力なデータベース特性および十分な非ファイル表現を特徴とするが、ファイル表現の処理において有効ではなく、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階層構造に基づいたファイルおよびフォルダの速度、効率および単純さを複製することはできなかった。ＳｍａｌｌＴａｌｋ（および他の派生物）を使用するために試みら

れた物など、他の取り組みは、ファイルおよび非ファイル表現の処理で大変有効であるが、様々なデータファイルの間に存在するリレーションシップを効率的に編成および利用するために必要なデータベース機能が不足していることが判明しており、したがってこのようなシステムの全体の効率は受け入れ不可能であった。ＢｅＯＳ（および他のこのようなオペレーティングシステム研究）を使用するためのさらに他の試みは、いくつかの必要なデータベース機能を提供しながらファイルを適切に表現することができるにもかかわらず、非ファイル表現の処理、すなわち、従来のファイルシステムの同じ中心となる欠点に対して不適切であることが判明した。

【０００９】

データベース技術は、類似の課題が存在する当技術分野のもう１つのエリアである。例えば、リレーショナルデータベースモデルは大きい商業的成功となっているが、実際には、独立系ソフトウェアベンダー（ＩＳＶ）は一般に、リレーショナルデータベースソフトウェア製品（Ｍｉｃｒｏｓｏｆｔ　ＳＱＬ　Ｓｅｒｖｅｒなど）で使用可能な機能性のごく一部を用いる。その代わりに、アプリケーションによるこのような製品との対話の大部分は、単純な「ｇｅｔｓ」および「ｐｕｔｓ」の形態である。プラットフォームまたはデータベースに寛容であるなど、このためのいくつかの容易に明らかな理由はあるが、しばしば気付かれずに終わる１つの重要な理由は、データベースが必ずしも、主要なビジネスアプリケーションベンダーが本当に必要とする正確な抽象化を提供するとは限らないことである。例えば、現実の世界は「顧客」または「注文」など、「アイテム」の概念を（それら自体においてそれら自体のアイテムとして、注文の埋め込まれた「ラインアイテム」と共に）有するが、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のみ物を言う。したがって、アプリケーションは（２、３例を挙げると）アイテムレベルで整合性、ロック、セキュリティおよび／またはトリガの態様を有することを望む場合があるが、一般にデータベースはこれらの機能をテーブル／行レベルでのみ提供する。これは、各アイテムがデータベース内のあるテーブル内の単一の行にマップされる場合、うまく機能する可能性があるが、複数のラインアイテムを有する注文の場合、１つのアイテムが実際には複数のテーブルにマップされる理由がある可能性があり、その場合、単純なリレーショナルデータベースシステムはあまり正しい抽象化を提供しない。したがって、アプリケーションはロジックをデータベースの最上部で構築して、これらの基本的な抽象化を提供しなければならない。すなわち、基本リレーショナルモデルは、その上でより高いレベルのアプリケーションを容易に開発することができる、データのストレージのための十分なプラットフォームを提供せず、これは、基本リレーショナルモデルがアプリケーションとストレージシステムの間にあるレベルの間接化を必要とし、データの意味構造が場合によってはアプリケーション内でのみ可視である可能性があるからである。いくつかのデータベースベンダーは、オブジェクトリレーショナル機能、新しい組織モデルなど、より高いレベルの機能性を自社製品に組み込んでいるが、いずれもまだ、必要とされた包括的な解決法の種類を提供しておらず、真に包括的な解決法は、両方の有用なデータモデル抽象化（「Ｉｔｅｍｓ」、「Ｅｘｔｅｎｓｉｏｎｓ」、「Ｒｅｌａｔｉｏｎｓｈｉｐｓ」など）を有用なドメイン抽象化（「Ｐｅｒｓｏｎｓ」、「Ｌｏｃａｔｉｏｎｓ」、「Ｅｖｅｎｔｓ」など）のために提供する解決法である。

【発明の開示】

【発明が解決しようとする課題】

【００１０】

既存のデータストレージおよびデータベース技術における前述の欠陥に鑑みて、コンピュータシステム内ですべてのタイプのデータを編成、検索および共有するための、改善された能力を提供する新しいストレージプラットフォーム、すなわち、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および拡大し、すべてのタイプのデータのためのストアとなるように設計されるストレージプラットフォームの必要性がある。

【００１１】

ある開示された発明はこの必要性を満たす（例えば、２００３年８月２１日出願の「STORAGE PLATFORM FOR ORGANIZING, SEARCHING, AND SHARING DATA」という名称の特許文献１参照。）。このストレージプラットフォームのための同期化サービスはさらに開示された発明において提供される（例えば、２００３年１０月２４日出願の「SYSTEMS AND METHODS FOR PROVIDING RELATIONAL AND HIERARCHICAL SYNCHRONIZATION SERVICES FOR UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM」という名称の特許文献２参照。）。本発明の目的は、開示されたストレージプラットフォームにおける開示された同期化サービスのため、ならびに、主張されたシステムおよび方法を適用することができる他の同期化サービスおよび／または他のストレージプラットフォームための、競合（ｃｏｎｆｌｉｃｔ）処理のためのシステムおよび方法を提供するにある。

【課題を解決するための手段】

【００１２】

以下の本発明の概要は、以下の詳細な説明および図への導入としての機能を果たすように意図される。

【００１３】

関連発明は集合的に、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ストレージの概念を拡張および拡大する、データを編成、検索および共有するためのストレージプラットフォームを対象とする。このストレージプラットフォームは、構造化、非構造化、または半構造化データを含む、すべてのタイプのデータのためのストアとなるように設計される。

【００１４】

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で実装されたデータストアを備える。データベースエンジンは、オブジェクトリレーショナルエクステンションを有するリレーショナルデータベースエンジンを備える。データストアは、データの編成、検索、共有、同期化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。特定のタイプのデータがスキーマにおいて記述され、プラットフォームは、新しいタイプのデータ（本質的に、スキーマによって提供された基本タイプのサブタイプ）を定義するようにスキーマのセットを拡張するためのメカニズムを提供する。同期化機能は、ユーザまたはシステムの間のデータの共有を実施する。ファイルシステム同様の機能が提供され、この機能により、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が可能となるが、このような従来のファイルシステムの制限はない。変更追跡メカニズムは、データストアへの変更を追跡する能力を提供する。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プログラムインターフェースのセットを備え、これによりアプリケーションは、ストレージプラットフォームの前述の機能のすべてにアクセスすること、および、スキーマ内に記述されたデータにアクセスすることが可能となる。

【００１５】

データストアによって実装されたデータモデルは、データストレージの単位を、アイテム、要素およびリレーションシップに関して定義する。アイテムは、データストア内に格納可能なデータの単位であり、１つまたは複数の要素およびリレーションシップを備えることができる。要素は、１つまたは複数のフィールドを備えるタイプのインスタンスである（本明細書でプロパティとも称する）。リレーションシップは、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る場合、これらおよび他の特定の用語を、近接して使用された他の用語からそれらを並置するために、大文字で書く場合があるが、大文字で書かれた用語、例えば「Ｉｔｅｍ」と、同じ用語が大文字で書かれない場合、例えば「ｉｔｅｍ」の間で区別するいかなる意図もなく、このような区別は仮定または暗示されるべきではない。）

【００１６】

コンピュータシステムはさらに複数のアイテムを備え、各アイテムは、ハードウェア／ソフトウェアインターフェースシステムによって操作することができる情報の離散的格納可能単位を構成し、さらに、前記アイテムのための組織構造を構成する複数のアイテムフ

ォルダ、および、複数のアイテム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、各アイテムは少なくとも１つのアイテムフォルタに属し、複数のアイテムフォルダに属することができる。

【００１７】

アイテム、または、アイテムのプロパティ値のいくつかを、永続的ストアから導出される物とは対照的に、動的に計算することができる。すなわち、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、アイテムが格納されることを必要とせず、ストレージプラットフォームの、アイテムの現在のセットをエニュメレートする能力、または、その識別子が与えられたアイテムを検索する能力（アプリケーションプログラミングインターフェースまたはＡＰＩを説明するセクションでより十分に後述する）など、あるオペレーションがサポートされ、例えば、あるアイテムは携帯電話の現在のロケーションであってもよく、温度センサー上の温度読み取りであってもよい。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アイテムを操作することができ、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って管理された複数のリレーションシップによって相互接続されたアイテムを備えることができる。

【００１８】

コンピュータシステムの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、コアアイテムのセットを定義するためのコアスキーマを備える。複数のアイテムを操作するため、コンピュータシステムは前記アイテムを複数のリレーションシップと相互接続し、前記リレーションシップをハードウェア／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セットにおいて定義された、各アイテム、アイテムエクステンションおよびリレーションシップのためのデータクラスを提供する。加えて、アプリケーションプログラミングインターフェースはフレームワーククラスのセットを提供し、フレームワーククラスのセットは、データクラスのためのビヘイビアの共通セットを定義し、データクラスと共に、ストレージプラットフォームＡＰＩのための基本プログラミングモデルを提供する。ストレージプラットフォームＡＰＩは簡易クエリモデルを提供し、簡易クエリモデルは、アプリケーションプログラマを基礎となる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方法で、アプリケーションプログラマがクエリを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形成すること可能にする。ストレージプラットフォームＡＰＩはまた、アプリケーションプログラムによって行われたアイテムへの変更を収集し、次いでこれらの変更を、その上でデータストアが実装されるデータベースエンジン（またはいずれかの種類のストレージエンジン）によって必要とされた正しいアップデートに編成する。これにより、アプリケーションプログラマは、データストアアップデートの複雑さをＡＰＩに任せながら、メモリ内のアイテムに変更を行うことができる。

【００２０】

その共通ストレージ基盤およびスキーマ化されたデータを通じて、本発明のストレージプラットフォームは、消費者、知識労働者および企業のためのより効率的なアプリケーション開発を可能にする。本発明のストレージプラットフォームは、そのデータモデル内に固有の機能を使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を包含および拡張する、豊富で拡張可能なアプリケーションプログラミングインターフェースを提供する。

【００２１】

この相互に関係する発明の包括的な構造（詳細な説明のセクションＩＩで詳細に説明）の一部として、関連発明のいくつかは特に同期ＡＰＩ（詳細な説明のセクションＩＩＩで詳細に説明）を対象とし、これはストレージプラットフォームの幅広い同期化機能を記述した。本発明のいくつかの実施形態にこれらの同期化機能が組み込まれて、ピアツーピア

同期化中に競合が生じる場合に競合を処理するようになることを予想されたい。

【００２２】

競合を正確および効率的に処理する能力は、よい有用性を保持しながらデータ損失を最小限にし、同期化中のユーザの介入の必要性を低減する。詳細な説明のセクションＩＶは、それだけに限定されないが、関連発明の同期化システムまたはストレージプラットフォームを含む、ピアツーピア同期化システムにおける競合を処理するためのシステムおよび方法を対象とする、本発明の様々な実施形態の詳細な説明である。様々な実施形態は、（ａ）競合処理のための共通拡張可能スキーマを確立すること、および（ｂ）競合が検出されてから解決され最終的に除去されるまで進行する場合に競合を表現する状態を追跡することによって、ピアツーピア同期化システム内で発生する競合を表現するための解決法を提供する。いくつかの実施形態の重要な特徴には、競合がこれらの前述の状態を通じて進行する場合に競合を追跡する能力、ならびに、解決された競合の自動クリーンアップを実行する能力が含まれる。ある実施形態には、古くなった競合（古くなっている変化を表現する競合）を検出する能力、および、このような競合を排除するための対応する能力が含まれる。他の実施形態は、競合を同期的に自動的に解決するための競合解決ポリシーの使用を対象とする。最後だが重要でないということではなく、ある実施形態にはまた、ユーザによって後に解決するためにログされている競合をプログラム的に解決する能力（ユーザインターフェースによって、ユーザ対象の競合解決を実行するために使用することができる）もが含まれる可能性がある。

【００２３】

本発明の具体的な特徴および利点は単独で、および関連発明と共に、以下の本発明の詳細な説明および添付の図面から明らかになるであろう。

【００２４】

前述の概要、ならびに以下の本発明の詳細な説明は、添付の図面と共に読まれる場合によりよく理解される。本発明を例示するため、図面において本発明の様々な態様の例示的実施形態を示すが、本発明は開示された特定の方法および手段に限定されない。図面においては以下の通りである。

【発明を実施するための最良の形態】

【００２５】

Ｉ．序論

本発明の主題を、法的要件を満たすための特定性を有して説明する。しかし、説明自体は本特許の範囲を限定するように意図されない。むしろ、発明者は、主張された主題もまた他の方法で、他の現在および将来の技術と共に、異なるステップ、または本書で説明するステップに類似のステップの組合せを含むように実施される可能性もあることを企図している。また、「ステップ」という用語は本明細書で、使用された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る場合を除いて、本明細書で開示された様々なステップの間のいかなる特定の順序をも暗示するように解釈されるべきではない。

【００２６】

Ａ．例示的コンピューティング環境

本発明の多数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および以下の考察は、本発明を実施することができる適切なコンピューティング環境の簡単な全体的説明を提供するように意図される。必要ではないが、本発明の様々な態様を一般に、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って実行される、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に関連して説明することができる。一般に、プログラムモジュールには、ルーチン、プログラム、オブジェクト、コンポーネント、データ構造などが含まれ、これらは特定のタスクを実行するか、あるいは特定の抽象データタイプを実装する。また、本発明を他のコンピュータシステム構成により実施することができ、これらの構成に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またはプログラマブルなコンシューマエレクトロニクス、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が含まれる。本発明をまた分散コンピューティング環境において実施することもでき、この環境ではタスクが、通信ネットワークを通じてリンクされるリモート処理デバイスによって実行される。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が、ローカルおよびリモートのメモリストレージデバイス内に位置することができる。

【００２７】

図１のように、例示的汎用コンピューティングシステムには従来のパーソナルコンピュータ２０などが含まれ、パーソナルコンピュータ２０には、処理装置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装置２１に結合するシステムバス２３が含まれる。システムバス２３を、いくつかのタイプのバス構造のいずれにすることもでき、これらのバス構造には、様々なバスアーキテクチャのいずれかを使用する、メモリバスまたはメモリコントローラ、周辺バス、およびローカルバスが含まれる。システムメモリには、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が含まれる。基本入出力システム２６（ＢＩＯＳ）は、起動中など、パーソナルコンピュータ２０内の複数の要素の間で情報を転送する助けとなる基本ルーチンを含み、ＲＯＭ２４に格納される。パーソナルコンピュータ２０にはさらに、図示しないハードディスクに対する読み書きを行うためのハードディスクドライブ２７、リムーバブル磁気ディスク２９に対する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または他の光メディアなど、リムーバブル光ディスク３１に対する読み書きを行うための光ディスクドライブ３０が含まれる可能性がある。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はシステムバス２３に、それぞれ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によって接続される。これらのドライブおよびそれらの関連付けられたコンピュータ可読メディアは、パーソナルコンピュータ２０のためのコンピュータ可読命令、データ構造、プログラムモジュールおよび他のデータの不揮発性ストレージを提供する。本明細書で説明した例示的環境は、ハードディスク、リムーバブル磁気ディスク２９およびリムーバブル光ディスク３１を使用するが、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、コンピュータによってアクセス可能であるデータを格納することができる他のタイプのコンピュータ可読メディアもまた例示的オペレーティング環境内で使用することができることは、当業者には理解されよう。同様に、例示的環境にはまた、熱感知器およびセキュリティまたは火災警報システムなど、多数のタイプのモニタリングデバイスおよび他の情報のソースが含まれる場合もある。

【００２８】

いくつかのプログラムモジュールをハードディスク、磁気ディスク２９、光ディスク３１、ＲＯＭ２４またはＲＡＭ２５上で格納することができ、これらのプログラムモジュールには、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、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が含まれる。ユーザはコマンドおよび情報をパーソナルコンピュータ２０へ、キーボード４０およびポインティングデバイス４２など、入力デバイスを通じて入力することができる。他の入力デバイス（図示せず）には、マイクロフォン、ジョイスティック、ゲームパッド、衛星放送受信アンテナ、スキャナなどが含まれる可能性がある。これらおよび他の入力デバイスはしばしば処理装置２１へ、システムバスに結合されるシリアルポートインターフェース４６を通じて接続されるが、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、他のインターフェースによって接続することができる。モニタ４７または他のタイプの表示デバイスもまたシステムバス２３へ、ビデオアダプタ４８などのインターフェースを介して接続される。モニタ４７に加えて、パーソナルコンピュータには通常、スピーカおよびプリンタなど、他の周辺出力デバイス（図示せず）が含まれる。図１の例示的システムにはまた、ホストアダプタ５５、Ｓｍａｌｌ　Ｃｏｍｐｕｔｅｒ　Ｓｙｓ

ｔｅｍ　Ｉｎｔｅｒｆａｃｅ（ＳＣＳＩ）バス５６、および、ＳＣＳＩバス５６に接続された外部ストレージデバイス６２もが含まれる。

【００２９】

パーソナルコンピュータ２０はネットワーク環境において、リモートコンピュータ４９など、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動作することができる。リモートコンピュータ４９は、別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他の共通ネットワークノードにすることができ、通常は、パーソナルコンピュータ２０に関連して上述した要素の多数またはすべてを含むが、メモリストレージデバイス５０のみが図１に例示される。図１に示す論理接続には、ローカルエリアネットワーク（ＬＡＮ）５１およびワイドエリアネットワーク（ＷＡＮ）５２が含まれる。このようなネットワーキング環境は、オフィス、企業全体の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において一般的である。

【００３０】

ＬＡＮネットワーキング環境において使用する場合、パーソナルコンピュータ２０はＬＡＮ５１へ、ネットワークインターフェースまたはアダプタ５３を通じて接続される。ＷＡＮネットワーキング環境において使用する場合、パーソナルコンピュータ２０には通常、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を介して通信を確立するための他の手段が含まれる。モデム５４は内部であっても外部であってもよく、システムバス２３へ、シリアルポートインターフェース４６を介して接続される。ネットワーク環境では、パーソナルコンピュータ２０に関連して示したプログラムモジュールまたはその一部を、リモートメモリストレージデバイスに格納することができる。図示のネットワーク接続は例示的であり、複数のコンピュータの間で通信リンクを確立する他の手段を使用することができることは理解されよう。

【００３１】

図２のブロック図に例示するように、コンピュータシステム２００を大まかに３つのコンポーネントグループ、すなわち、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（本明細書のある文脈では「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」とも称する）に分割することができる。

【００３２】

コンピュータシステム２００の様々な実施形態では、図１に戻って参照すると、ハードウェアコンポーネント２０２はとりわけ、中央処理装置（ＣＰＵ）２１、メモリ（ＲＯＭ２４およびＲＡＭ２５の両方）、基本入出力システム（ＢＩＯＳ）２６、ならびに、キーボード４０、マウス４２、モニタ４７および／またはプリンタ（図示せず）など、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のための基本的な物理的インフラストラクチャを備える。

【００３３】

アプリケーションプログラムコンポーネント２０６は様々なソフトウェアプログラムを備え、ソフトウェアプログラムには、それだけに限定されないが、コンパイラ、データベースシステム、ワープロ、ビジネスプログラム、ビデオゲームなどが含まれる。アプリケーションプログラムは、それによってコンピュータリソースが様々なユーザ（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）のために、問題を解決するため、解決を提供するため、およびデータを処理するために利用される手段を提供する。

【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コンポーネント２０４は、それ自体が大抵の場合にシェルおよびカーネルを備える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、オペレーティングシステムのみからなる場合がある）。「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の間の仲介の機能を果たす、特殊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はまた、仮想マシンマネージ

ャ（ＶＭＭ）、共通言語ランタイム（ＣＬＲ）もしくはその機能的同等物、Ｊａｖａ（登録商標）仮想マシン（ＪＶＭ）もしくはその機能的同等物、または他のこのようなソフトウェアコンポーネントを、コンピュータシステム内でオペレーティングシステムの代わりに、あるいはそれに加えて備えることもできる。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することができる環境を提供することである。いかなるハードウェア／ソフトウェアインターフェースシステムの目標も、コンピュータシステムを使いやすくすること、ならびに、コンピュータハードウェアを効率的な方法で利用することである。

【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一般にコンピュータシステムへ、起動の際にロードされ、その後、コンピュータシステム内のアプリケーションプログラムのすべてを管理する。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って、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対話する。いくつかのアプリケーションプログラムは、エンドユーザがハードウェア／ソフトウェアインターフェースシステムと、共通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、ユーザインターフェースを介して対話することを可能にする。

【００３６】

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、アプリケーションのための様々なサービスを実行する。マルチタスキングハードウェア／ソフトウェアインターフェースシステムでは、複数のプログラムが同時に実行中である場合があり、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、どのアプリケーションがどの順序で実行するべきであるか、および、各アプリケーションについて、順番で別のアプリケーションに切り替える前にどれだけの時間が許可されるべきであるかを判定する。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また、複数のアプリケーションの間の内部メモリの共有も管理し、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、付属のハードウェアデバイスとの入出力も処理する。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また、オペレーションの状況および発生している可能性のあるいかなるエラーに関しても、メッセージを各アプリケーションへ（および、ある場合にはエンドユーザへ）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またバッチジョブ（例えば、印刷）の管理をオフロードして、開始側アプリケーションがこの作業から解放されて他の処理および／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。並列処理を提供す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またプログラムの分割を管理して、プログラムが一度に複数のプロセッサ上で実行するようにする。

【００３７】

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（本明細書で単に「シェル」と称する）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へのインタラクティブなエンドユーザインターフェースである。（シェルはまた、「コマンドインタプリタ」、またはある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」とも呼ばれる場合がある）。シェルは、ハードウェア／ソフトウェアインターフェースシステムの外部層であり、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って直接アクセス可能である。シェルとは対照的に、カーネルはハードウェア／ソフトウェアインターフェースシステムの最も内側の層であり、ハードウェアコンポーネントと直接対話する。

【００３８】

本発明の多数の実施形態は特にコンピュータ化されたシステムに適切であることが想定されるが、本書の何物も、本発明をこのような実施形態に限定するように意図されない。それどころか、本明細書で使用される場合、「コンピュータシステム」という用語は、そのようなデバイスが性質において電子的であるか、機械的であるか、論理的であるか仮想

的であるかにかかわらず、情報を格納および処理することができる、および／または、格納された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コントロールすることができる、いかなるおよびすべてのデバイスをも包含するように意図される。

【００３９】

Ｂ．従来のファイルベースのストレージ

今日の大部分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、ならびにアプリケーションプログラム、データセットなどを含むことができる、格納可能な情報の単位である。すべての現代のハードウェア／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、ファイルは情報（例えば、データ、プログラムなど）の基本の離散的（格納可能および検索可能な）単位であり、ハードウェア／ソフトウェアインターフェースシステムによって操作することができる。ファイルのグループは一般に「フォルダ」内に編成される。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では、フォルダは、単一の情報の単位として検索、移動およびそうでない場合は操作することが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、「ディレクトリ」と呼ばれるツリーベースの階層構成において編成される（本明細書で以下でより詳細に論じる）。ＤＯＳ、ｚ／ＤＯＳおよび大部分のＵｎｉｘ（登録商標）ベースのオペレーティングシステムなど、ある他のハードウェア／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」という用語は交換可能であり、初期のＡｐｐｌｅコンピュータシステム（例えば、ＡｐｐｌｅＩＩｅ）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たが、本明細書で使用される場合、これらのすべての用語は同義語および交換可能であると見なされ、階層的情報ストレージ構造ならびにそれらのフォルダおよびファイルコンポーネントについてのすべての他の等価語、およびそれらへの参照をさらに含むように意図される。

【００４０】

従来、ディレクトリ（別名、フォルダのディレクトリ）はツリーベースの階層構造であり、ファイルはフォルダにグループ化され、フォルダは、ディレクトリツリーを備える相対的なノードのロケーションに従って構成される。例えば、図２Ａに例示するように、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を備えることができ、これらの各フォルダはさらに追加のフォルダ２１６を（その特定のフォルダの「サブフォルダ」として）備えることができ、これらの各々もまた追加のフォルダ２１８を無限に備えることができる。これらの各フォルダは１つまたは複数のファイル２２０を有することができるが、ハードウェア／ソフトウェアインターフェースシステムのレベルで、フォルダ内の個々のファイルは、ツリー階層内のそれらの位置以外、共通の何物をも有していない。当然のことながら、ファイルをフォルダ階層に編成するこの手法は、これらのファイルを格納するために使用された通常のストレージメディア（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）の物理的編成を間接的に反映する。

【００４１】

前述に加えて、各フォルダは、そのサブフォルダおよびそのファイルのためのコンテナであり、すなわち、各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。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って削除される場合、そのフォルダのサブフォルダおよびファイルもまた削除される（各サブフォルダの場合、さらにそれ自体のサブフォルダおよびファイルを再帰的に含む）。同様に、各ファイルは一般にただ１つのフォルダによって所有され、ファイルをコピーすることができ、コピーは異なるフォルダ内に位置することができるが、ファイルのコピー自体は、元のファイルとの直接接続を有していない、異なる分離した単位である（例えば、元のファイルへの変更は、ハードウェア／ソフトウェアインターフェースシステムのレベルでコピーファイルにミラーリングされない）。これについて、ファイルおよびフォルダはしたがって、

特徴的に性質において「物理的」であり、これは、フォルダが物理的コンテナのように処理され、ファイルがこれらのコンテナの内側で、離散的で分離した物理的要素として処理されるからである。

【００４２】

ＩＩ．データを編成、検索および共有するためのＷＩＮＦＳストレージプラットフォーム

本発明は、本明細書で前述したように参照により組み込まれた関連発明と組み合わせて、データを編成、検索および共有する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、上述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の種類を超えてデータプラットフォームを拡張および拡大し、アイテムと呼ばれる新しい形式のデータを含む、すべてのタイプのデータのためのストアとなるように設計される。

【００４３】

Ａ．用語集

本明細書および特許請求の範囲で使用される場合、以下の用語は以下の意味を有する。

【００４４】

・「アイテム」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にとってアクセス可能な格納可能情報の単位であり、単純な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ってエンドユーザにエクスポーズされたすべてのオブジェクトにわたって共通してサポートされる、プロパティの基本セットを有するオブジェクトである。アイテムはまた、新しいプロパティおよびリレーションシップが導入されることを可能にする特徴を含むすべてのアイテムタイプにわたって共通してサポートされる、プロパティおよびリレーションシップをも有する（本明細書で以下で詳細に論じる）。

【００４５】

・「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の間の仲介の機能を果たす、特殊なプログラムである。オペレーティングシステムは、大抵の場合、シェルおよびカーネルを備える。

【００４６】

・「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、ソフトウェア、または、ハードウェアおよびソフトウェアの組合せであり、コンピュータシステムの基礎となるハードウェアコンポーネントと、コンピュータシステム上で実行するアプリケーションの間のインターフェースとしての機能を果たす。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場合がある）。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また、仮想マシンマネージャ（ＶＭＭ）、共通言語ランタイム（ＣＬＲ）もしくはその機能的同等物、Ｊａｖａ（登録商標）仮想マシン（ＪＶＭ）もしくはその機能的同等物、または他のこのようなソフトウェアコンポーネントを、コンピュータシステム内でオペレーティングシステムの代わりに、あるいはそれに加えて備えることもできる。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することができる環境を提供することである。いかなるハードウェア／ソフトウェアインターフェースシステムの目標も、コンピュータシステムを使いやすくすること、ならびに、コンピュータハードウェアを効率的な方法で利用することである。

【００４７】

Ｂ．ストレージプラットフォーム概観

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上で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エクステンションを有するリレーショナルデータベースエンジンを備える。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、Ｍ

ｉｃｒｏｓｏｆｔ　ＳＱＬ　Ｓｅｒｖｅｒリレーショナルデータベースエンジンを備える。データストア３０２は、データの編成、検索、共有、同期化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。特定のタイプのデータは、スキーマ３０４などのスキーマ内に記述され、ストレージプラットフォーム３００は、これらのスキーマを配置するため、ならびに、これらのスキーマを拡張するためのツール３４６を提供し、これを以下でより十分に説明する。

【００４８】

変更追跡メカニズム３０６はデータストア３０２内で実装され、データストアへの変更を追跡する能力を提供する。データストア３０２はまた、セキュリティ機能３０８およびプロモーション／デモーション機能３１０をも提供し、両方の機能を以下でより十分に論じる。データストア３０２はまた、データストア３０２の機能を、ストレージプラットフォームを利用する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）にエクスポーズするための、アプリケーションプログラミングインターフェースのセット３１２をも提供する。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ミングインターフェース（ＡＰＩ）３２２を備え、これにより、アプリケーションプログラム３５０ａ、３５０ｂおよび３５０ｃなど、アプリケーションプログラムがストレージプラットフォームの前述の機能のすべてにアクセスすること、および、スキーマ内で記述されたデータにアクセスすることが可能となる。ストレージプラットフォームＡＰＩ３２２をアプリケーションプログラムによって、ＯＬＥ　ＤＢ　ＡＰＩ３２４およびＭｉｃｒｏｓｏｆｔ　Ｗｉｎｄｏｗｓ（登録商標）Ｗｉｎ３２　ＡＰＩ３２６など、他のＡＰＩと組み合わせて使用することができる。

【００４９】

本発明のストレージプラットフォーム３００は様々なサービス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、これらのサービスには、ユーザまたはシステムの間のデータの共有を実施する同期化サービス３３０が含まれる。例えば、同期化サービス３３０は、データストア３０２と同じフォーマットを有する他のデータストア３４０との相互運用性、ならびに、他のフォーマットを有するデータストア３４２へのアクセスを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はまた、Ｗｉｎｄｏｗｓ（登録商標）ＮＴＦＳファイルシステム３１８など、既存のファイルシステムとの、データストア３０２の相互運用性を可能にする、ファイルシステム機能をも提供する。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３００はまたアプリケーションプログラムに、データが他のシステム上で動作されることを可能にするため、および他のシステムとの対話を可能にするための追加の機能を提供することもできる。これらの機能を、情報エージェントサービス３３４および通知サービス３３２など、追加のサービス３２８の形態において、ならびに他のユーティリティ３３６の形態において実施することができる。

【００５０】

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において実施され、あるいはその一体部分を形成する。例えば、限定なく、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）もしくはその機能的同等物、または、Ｊａｖａ（登録商標）仮想マシン（ＪＶＭ）もしくはその機能的同等物において実施される場合があり、あるいはその一体部分を形成する場合がある。その共通のストレージ基盤、および、スキーマ化されたデータを通じて、本発明のストレージプラットフォームは、消費者、知識労働者および企業のためのより効率的なアプリケーション開発を可能にする。本発明のストレージプラットフォームは、そのデータモデル内に固有の機能を使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を包含および拡張する、豊富で拡張可能なプログラミングのサーフェスエリアを提供する。

【００５１】
以下の説明において、および様々な図面において、本発明のストレージプラットフォーム３００を「ＷｉｎＦＳ」と称する場合がある。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ことは、説明の便宜上のためのみであり、決して限定するように意図されない。
【００５２】
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、ストア内に存在するデータの編成、検索、共有、同期化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、「アイテム」はストレージ情報の基本的単位である。データモデルは、アイテムおよびアイテムエクステンションを宣言するため、および、複数のアイテムの間のリレーションシップを確立するため、および、アイテムをアイテムフォルダ内およびカテゴリ内に編成するためのメカニズムを提供し、これを以下でより十分に説明する。
【００５３】
データモデルは２つの基本的メカニズムであるタイプおよびリレーションシップに依拠する。タイプは、タイプのインスタンスの形態を管理するフォーマットを提供する構造である。フォーマットは、プロパティの順序付きセットとして表現される。プロパティは、所与のタイプの値または値のセットのための名前である。例えば、ＵＳＰｏｓｔａｌＡｄｄｒｅｓｓタイプは、プロパティＳｔｒｅｅｔ、Ｃｉｔｙ、Ｚｉｐ、Ｓｔａｔｅを有する可能性があり、Ｓｔｒｅｅｔ、ＣｉｔｙおよびＳｔａｔｅはＳｔｒｉｎｇのタイプであり、ＺｉｐはタイプＩｎｔ３２である。Ｓｔｒｅｅｔを多値（すなわち、値のセット）にして、アドレスがＳｔｒｅｅｔプロパティのための複数の値を有することを可能にすることができる。システムは、他のタイプの構成において使用することができるある基本的タイプを定義し、これらのタイプには、Ｓｔｒｉｎｇ、Ｂｉｎａｒｙ、Ｂｏｏｌｅａｎ、Ｉｎｔ１６、Ｉｎｔ３２、Ｉｎｔ６４、Ｓｉｎｇｌｅ、Ｄｏｕｂｌｅ、Ｂｙｔｅ、ＤａｔａＴｉｍｅ、ＤｅｃｉｍａｌおよびＧＵＩＤが含まれる。タイプのプロパティを、基本的タイプのいずれか、または、（以下に示すいくつかの制限を有して）構築されたタイプのいずれかを使用して、定義することができる。例えば、プロパティのＣｏｏｒｄｉｎａｔｅおよびＡｄｄｒｅｓｓを有したＬｏｃａｔｉｏｎタイプを定義することができ、Ａｄｄｒｅｓｓプロパティは、上述のタイプＵＳＰｏｓｔａｌＡｄｄｒｅｓｓである。プロパティはまた必須であってもオプショナルであってもよい。
【００５４】
リレーションシップを宣言することができ、リレーションシップは２つのタイプのインスタンスのセットの間のマッピングを表現することができる。例えば、Ｐｅｒｓｏｎタイプと、ＬｉｖｅｓＡｔと呼ばれるＬｏｃａｔｉｏｎタイプの間で宣言されたリレーションシップがある可能性があり、このリレーションシップはどの人がどのロケーションに住むかを定義する。リレーションシップは名前、２つのエンドポイント、すなわち、ソースエンドポイントおよびターゲットエンドポイントを有する。リレーションシップはまた、プロパティの順序付きセットを有することもできる。ソースおよびターゲットエンドポイントは、名前およびタイプを有する。例えば、ＬｉｖｅｓＡｔリレーションシップは、タイプＰｅｒｓｏｎのＯｃｃｕｐａｎｔと呼ばれるソース、および、タイプＬｏｃａｔｉｏｎのＤｗｅｌｌｉｎｇと呼ばれるターゲットを有し、加えて、居住者が住居に住んだ期間を示すプロパティＳｔａｒｔＤａｔｅおよびＥｎｄＤａｔｅを有する。Ｐｅｒｓｏｎは経時的に複数の住居に住む可能性があり、住居は複数の居住者を有する可能性があり、そのため、ＳｔａｒｔＤａｔｅおよびＥｎｄＤａｔｅ情報を置くための最も可能性の高い場所は、リレーションシップ自体の上であることに留意されたい。
【００５５】
リレーションシップは、エンドポイントタイプとして与えられたタイプによって制約される複数のインスタンスの間のマッピングを定義する。例えば、ＬｉｖｅｓＡｔリレーシ

ョンシップを、ＡｕｔｏｍｏｂｉｌｅがＯｃｃｕｐａｎｔであるリレーションシップにすることはできず、これはＡｕｔｏｍｏｂｉｌｅがＰｅｒｓｏｎではないからである。

【００５６】

データモデルは、複数のタイプの間のサブタイプ－スーパータイプリレーションシップの定義を可能にする。サブタイプ－スーパータイプリレーションシップはまたＢａｓｅＴｙｐｅリレーションシップとしても知られ、タイプＡがタイプＢのためのＢａｓｅＴｙｐｅである場合、ＢのあらゆるインスタンスもまたＡのインスタンスである場合でなければならないような方法で、定義される。これを別の表現にすると、Ｂに適合するあらゆるインスタンスはまたＡにも適合しなければならないということである。例えば、ＡがタイプＳｔｒｉｎｇのプロパティＮａｍｅを有するが、ＢはタイプＩｎｔ１６のプロパティＡｇｅを有する場合、これは、ＢのいかなるインスタンスもＮａｍｅおよびＡｇｅを有していなければならないということになる。タイプ階層を、単一のスーパータイプをルートで有するツリーとして想定することができる。ルートからのブランチは第１のレベルのサブタイプを提供し、このレベルのブランチは第２のレベルのサブタイプを提供するなど、最もリーフのサブタイプまで提供し、最もリーフのサブタイプ自体はいかなるサブタイプも有していない。ツリーは、一様の深度であるように制約されないが、いかなるサイクルをも含むことはできない。所与のタイプは、ゼロまたは多数のサブタイプおよびゼロまたは１つのスーパータイプを有することができる。所与のインスタンスは、多くても１つのタイプに、そのタイプのスーパータイプと共に適合することができる。別の言い方をすれば、ツリー内のいずれかのレベルでの所与のインスタンスについて、インスタンスはそのレベルの多くても１つのサブタイプに適合することができる。タイプのインスタンスがまたタイプのサブタイプのインスタンスでもなければならない場合、タイプはＡｂｓｔｒａｃｔであると呼ばれる。

【００５７】

１．アイテム

アイテムは格納可能な情報の単位であり、単純なファイルとは異なり、ストレージプラットフォームによってエンドユーザまたはアプリケーションプログラムにエクスポーズされたすべてのオブジェクトにわたって共通してサポートされる、プロパティの基本セットを有するオブジェクトである。アイテムはまた、新しいプロパティおよびリレーションシップが導入されることを可能にする特徴を含むすべてのアイテムタイプにわたって共通してサポートされる、プロパティおよびリレーションシップをも有し、これを以下で論じる。

【００５８】

アイテムは、コピー、削除、移動、開く、印刷、バックアップ、リストア、レプリケートなど、共通オペレーションのためのオブジェクトである。アイテムは、格納および検索することができる単位であり、ストレージプラットフォームによって操作された格納可能な情報のすべての形態はアイテム、アイテムのプロパティ、または複数のアイテムの間のリレーションシップとして存在し、その各々を本明細書で以下でより詳細に論じる。

【００５９】

アイテムは、（すべての様々な種類の）Ｃｏｎｔａｃｔｓ、Ｐｅｏｐｌｅ、Ｓｅｒｖｉｃｅｓ、Ｌｏｃａｔｉｏｎｓ、Ｄｏｃｕｍｅｎｔｓなどのようなデータの、現実世界の容易に理解可能な単位を表現するように意図される。図５Ａは、アイテムの構造を例示するブロック図である。アイテムの非修飾名は「Ｌｏｃａｔｉｏｎ」である。アイテムの修飾名は「Ｃｏｒｅ．Ｌｏｃａｔｉｏｎ」であり、このアイテム構造がＣｏｒｅスキーマ内のアイテムの特定のタイプとして定義されることを示す。（Ｃｏｒｅスキーマを本明細書で以下でより詳細に論じる。）

【００６０】

Ｌｏｃａｔｉｏｎアイテムは、ＥＡｄｄｒｅｓｓｅｓ、ＭｅｔｒｏｐｏｌｉｔａｎＲｅｇｉｏｎ、ＮｅｉｇｈｂｏｒｈｏｏｄおよびＰｏｓｔａｌＡｄｄｒｅｓｓｅｓを含む、複数のプロパティを有する。各々のためのプロパティの特定のタイプは、プロパティ名の直

後に示され、プロパティ名からコロン（「：」）によって区切られる。タイプ名の右側に、そのプロパティタイプのために許可された値の数が角括弧（「［］」）の間で示され、コロン（「：」）の右側のアスタリスク（「＊」）は、規定されていない、および／または、無制限の数（「多数」）を示す。コロンの右側の「１」は、多くても１つの値がある可能性があることを示す。コロンの左側のゼロ（「０」）は、プロパティがオプショナルである（値がまったくない場合がある）ことを示す。コロンの左側の「１」は、少なくとも１つの値がなければならない（このプロパティが必要とされる）ことを示す。ＮｅｉｇｈｂｏｒｈｏｏｄおよびＭｅｔｒｏｐｏｌｉｔａｎＲｅｇｉｏｎは共にタイプ「ｎｖａｒｃｈａｒ」（または同等物）であり、事前定義されたデータタイプまたは「単純タイプ」である（本明細書では大文字化の欠如によって示す）。しかし、ＥＡｄｄｒｅｓｓｅｓおよびＰｏｓｔａｌＡｄｄｒｅｓｓｅｓは、それぞれタイプＥＡｄｄｒｅｓｓおよびＰｏｓｔａｌＡｄｄｒｅｓｓの定義済みタイプのプロパティまたは「複合タイプ」（本明細書では大文字化によって示す）である。複合タイプは、１つまたは複数の単純データタイプから、および／または、他の複合タイプから派生されるタイプである。複合タイプの詳細が直接のアイテムにネストされてそのプロパティが定義されるので、アイテムのプロパティのための複合タイプもまた「ネストされた要素」を構成し、これらの複合タイプに関係する情報は、これらのプロパティを有するアイテムと共に（本明細書で以下で論じるように、アイテムの境界内で）維持される。これらのタイプ定義（ｔｙｐｉｎｇ）の概念は周知であり、当業者には容易に理解される。

【００６１】

図５Ｂは、複合プロパティタイプＰｏｓｔａｌＡｄｄｒｅｓｓおよびＥＡｄｄｒｅｓｓを例示するブロック図である。ＰｏｓｔａｌＡｄｄｒｅｓｓプロパティタイプは、プロパティタイプＰｏｓｔａｌＡｄｄｒｅｓｓのアイテムがゼロまたは１つのＣｉｔｙ値、ゼロまたは１つのＣｏｕｎｔｒｙＣｏｄｅ値、ゼロまたは１つのＭａｉｌＳｔｏｐ値、およびいずれかの数（ゼロから多数）のＰｏｓｔａｌＡｄｄｒｅｓｓＴｙｐｅｓその他などを有するように予想することができると定義する。このように、アイテム内の特定のプロパティのためのデータの形状はこれによって定義される。ＥＡｄｄｒｅｓｓプロパティタイプは同様に図示のように定義される。本明細書でこの適用をオプショナルで使用したが、Ｌｏｃａｔｉｏｎアイテム内の複合タイプを表現するもう１つの方法は、その中にリストされた各複合タイプの個々のプロパティと共にアイテムを描くことである。図５Ｃは、その複合タイプがさらに記述されるＬｏｃａｔｉｏｎアイテムを例示するブロック図である。しかし、この図５ＣのＬｏｃａｔｉｏｎアイテムのこの代替表現は、図５Ａに例示したまったく同じアイテムのためであることを理解されたい。本発明のストレージプラットフォームはまたサブタイプ定義（ｓｕｂｔｙｐｉｎｇ）も可能にし、それによりあるプロパティタイプを別のプロパティタイプのサブタイプにすることができる（あるプロパティタイプは別の、親プロパティタイプのプロパティを継承する）。

【００６２】

プロパティおよびそれらのプロパティタイプと類似しているが異なるように、アイテムはそれら自体のアイテムタイプを本質的に表現し、このアイテムタイプもまたサブタイプ定義の対象になる可能性がある。すなわち、本発明のいくつかの実施形態のストレージプラットフォームは、アイテムを別のアイテムのサブタイプにすることを可能にする（それにより、あるアイテムは、他の親アイテムのプロパティを継承する）。また、本発明の様々な実施形態では、あらゆるアイテムは「Ｉｔｅｍ」アイテムタイプのサブタイプであり、「Ｉｔｅｍ」はベーススキーマ内で発見される第１の基本的アイテムタイプである。（ベーススキーマもまた本明細書で以下で詳細に論じる。）図６Ａは、アイテムである、このインスタンスにおけるＬｏｃａｔｉｏｎアイテムを、ベーススキーマ内で発見されるＩｔｅｍアイテムタイプのサブタイプとして例示する。この図では、矢印は、Ｌｏｃａｔｉｏｎアイテムは（他のすべてのアイテムと同様に）Ｉｔｅｍアイテムタイプのサブタイプであることを示す。Ｉｔｅｍアイテムタイプは、他のすべてのアイテムがそれから派生される基本的アイテムとして、ＩｔｅｍＩｄなど、いくつかの重要なプロパティおよび様々

なタイムスタンプを有し、それにより、オペレーティングシステム内のすべてのアイテムの標準プロパティを定義する。この図では、Ｉｔｅｍアイテムタイプのこれらのプロパティは Ｌｏｃａｔｉｏｎによって継承され、それによりＬｏｃａｔｉｏｎのプロパティとなる。

【００６３】

Ｉｔｅｍアイテムタイプから継承されたＬｏｃａｔｉｏｎアイテムのプロパティを表現するもう１つの方法は、その中にリストされた親アイテムからの各プロパティタイプの個々のプロパティと共にＬｏｃａｔｉｏｎを描くことである。図６Ｂは、その継承されたタイプがその直接のプロパティに加えて記述されるＬｏｃａｔｉｏｎアイテムを例示するブロック図である。このアイテムは、図５Ａに例示された同じアイテムであるが、この図ではＬｏｃａｔｉｏｎはそのプロパティのすべて、すなわち、この図および図５Ａに示す直接のプロパティ、およびこの図に示すが図５Ａには示さない継承されたプロパティと共に例示される（しかし、図５Ａではこれらのプロパティは、矢印により、Ｌｏｃａｔｉｏｎアイテムがアイテムタイプのサブタイプであることを示すことによって、参照される）ことに留意され、理解されたい。

【００６４】

アイテムはスタンドアロンオブジェクトであり、したがって、アイテムを削除する場合、アイテムの直接および継承プロパティのすべても削除される。同様に、アイテムを検索する場合、受信される物は、アイテムおよびその直接および継承プロパティ（その複合プロパティタイプに関係する情報を含む）のすべてである。本発明のある実施形態は、特定のアイテムを検索する場合にプロパティのサブセットを要求することを可能にする場合があるが、多数のこのような実施形態の初期設定は、検索される場合にアイテムをその直接および継承プロパティのすべてと共に提供することである。また、アイテムのプロパティをまた、新しいプロパティをそのアイテムのタイプの既存のプロパティに追加することによって、拡張することもできる。これらの「エクステンション」はその後、アイテムの真正なプロパティであり、そのアイテムタイプのサブタイプは自動的にエクステンションプロパティを含むことができる。

【００６５】

アイテムの「境界」はそのプロパティ（複合プロパティタイプ、エクステンションなどを含む）によって表現される。アイテムの境界はまた、コピー、削除、移動、作成など、アイテムにおいて実行されるオペレーションの制限も表現する。例えば、本発明のいくつかの実施形態では、アイテムがコピーされる場合、そのアイテムの境界内のすべての物もコピーされる。各アイテムについて、境界は以下を包含する。

【００６６】

・アイテムのＩｔｅｍタイプ、および、アイテムが別のアイテムのサブタイプである（すべてのアイテムは単一のアイテムおよびベーススキーマのＩｔｅｍタイプから派生される、本発明のいくつかの実施形態の場合のように）場合、いずれかの適用可能なサブタイプ情報（すなわち、親アイテムタイプに関係する情報）。コピーされる元のアイテムが別のアイテムのサブタイプである場合、このコピーもまたその同じアイテムのサブタイプである場合がある。

【００６７】

・ある場合、アイテムの複合タイププロパティおよびエクステンション。元のアイテムが複合タイプ（ネイティブまたは拡張された）のプロパティを有する場合、このコピーもまたその同じ複合タイプを有する場合がある。

【００６８】

・「所有権リレーションシップ」上のアイテムのレコード、すなわち、他の何のアイテム（「ターゲットアイテム」）が現在のアイテム（「所有側アイテム」）によって所有されるかのアイテム自体のリスト。これは特に、以下でより十分に論じるアイテムフォルダ、および、すべてのアイテムは少なくとも１つのアイテムフォルダに属していなければならないという後述のルールについて関連がある。また、以下でより十分に論じる埋め込み

アイテムについては、埋め込みアイテムは、その中に埋め込みアイテムがコピー、削除などのオペレーションのために埋め込まれるアイテムの一部であると見なされる。

【００６９】

２．アイテム識別

アイテムはＩｔｅｍＩＤにより、グローバルアイテムスペース内で一意に識別される。Ｂａｓｅ．Ｉｔｅｍタイプは、そのアイテムのための識別を格納するタイプＧＵＩＤのフィールドＩｔｅｍＩＤを定義する。アイテムは、データストア３０２内で厳密に１つの識別を有していなければならない。

【００７０】

アイテム参照は、アイテムを位置付けおよび識別するための情報を含むデータ構造である。データモデルでは、抽象タイプは、そこからすべてのアイテム参照タイプが派生する、ＩｔｅｍＲｅｆｅｒｅｎｃｅという名前で定義される。ＩｔｅｍＲｅｆｅｒｅｎｃｅタイプは、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、アイテムを戻す。このメソッドはＩｔｅｍＲｅｆｅｒｅｎｃｅの具象サブタイプによってオーバーライドされ、ＩｔｅｍＲｅｆｅｒｅｎｃｅの具象サブタイプは、参照が与えられたアイテムを検索するファンクションを実装する。Ｒｅｓｏｌｖｅメソッドは、ストレージプラットフォームＡＰＩ３２２の一部として呼び出される。

【００７１】

ＩｔｅｍＩＤＲｅｆｅｒｅｎｃｅはＩｔｅｍＲｅｆｅｒｅｎｃｅのサブタイプである。これはＬｏｃａｔｏｒおよびＩｔｅｍＩＤフィールドを定義する。Ｌｏｃａｔｏｒフィールドはアイテムドメインを名前付け（すなわち、識別）する。これは、Ｌｏｃａｔｏｒの値をアイテムドメインへ解決することができるロケータ解決メソッドによって処理される。ＩｔｅｍＩＤフィールドはタイプＩｔｅｍＩＤである。

【００７２】

ＩｔｅｍＰａｔｈＲｅｆｅｒｅｎｃｅはＩｔｅｍＲｅｆｅｒｅｎｃｅの特殊化であり、ＬｏｃａｔｏｒおよびＰａｔｈフィールドを定義する。Ｌｏｃａｔｏｒフィールドはアイテムドメインを識別する。これは、Ｌｏｃａｔｏｒの値をアイテムドメインへ解決することができるロケータ解決メソッドによって処理される。Ｐａｔｈフィールドは、Ｌｏｃａｔｏｒによって提供されたアイテムドメインでルートとされたストレージプラットフォームネームスペース内の（相対）パスを含む。

【００７３】

このタイプの参照をｓｅｔオペレーション内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ＡＰＩ３２２のＲｅｓｏｌｖｅメソッドは、この機能性を提供する。

【００７４】

上述の参照の形態は、図１１に例示した参照タイプ階層を通じて表現される。これらのタイプから継承する追加の参照タイプを、スキーマ内で定義することができる。これらをリレーションシップ宣言においてターゲットフィールドのタイプとして使用することができる。

【００７５】

３．アイテムフォルダおよびカテゴリ

以下でより十分に論じるように、アイテムのグループを、アイテムフォルダ（ファイルフォルダと混同されるべきではない）と呼ばれる特殊アイテムに編成することができる。しかし、大部分のファイルシステムとは異なり、アイテムは複数のアイテムフォルダに属することができ、アイテムがあるアイテムフォルダ内でアクセスされて訂正される場合、この訂正されたアイテムに別のアイテムフォルダから直接アクセスすることができるようになる。本質的に、アイテムへのアクセスは異なるアイテムフォルダから発生する可能性があるが、実際にアクセスされているのは実はまったく同じアイテムである。しかし、アイテムフォルダは必ずしもそのメンバアイテムのすべてを所有するとは限らず、または、

単に他のフォルダと共にアイテムを共同所有することができ、あるアイテムフォルダの削除は必ずしもアイテムの削除の結果になるとは限らないようになる。それにもかかわらず、本発明のいくつかの実施形態では、アイテムは少なくとも１つのアイテムフォルダに属していなければならず、特定のアイテムのためのただ１つのアイテムフォルダが削除される場合、いくつかの実施形態では、そのアイテムは自動的に削除されるか、または代替実施形態では、アイテムは自動的にデフォルトのアイテムフォルダ（例えば、様々なファイルおよびフォルダベースのシステムで使用される類似の名前のフォルダに概念的に類似する「ごみ箱」アイテムフォルダ）のメンバとなるようになる。

【００７６】

同じく以下でより十分に論じるように、アイテムはまた共通の記述された特性に基づいたカテゴリにも属することができ、カテゴリは、（ａ）アイテムタイプ（またはタイプ）、（ｂ）特定の直接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）アイテムプロパティに対応する特定の値（または複数の値）などである。例えば、個人連絡情報のための特定のプロパティを備えるアイテムは自動的にＣｏｎｔａｃｔカテゴリに属する可能性があり、連絡情報プロパティを有するいかなるアイテムも、同様に自動的にこのカテゴリに属するようになる。同様に、「Ｎｅｗ　Ｙｏｒｋ　Ｃｉｔｙ」の値を有するロケーションプロパティを有するいかなるアイテムも、自動的にＮｅｗＹｏｒｋＣｉｔｙカテゴリに属する可能性がある。

【００７７】

カテゴリはアイテムフォルダとは概念的に異なり、アイテムフォルダは、相関されない（すなわち、共通の記述された特性がない）複数のアイテムを備える可能性があるのに対して、カテゴリ内の各アイテムは、そのカテゴリについて記述される共通のタイプ、プロパティまたは値（「共通性」）を有し、そのアイテムとカテゴリ内の他のアイテムとのリレーションシップ、およびカテゴリ内他のアイテムの間のリレーションシップの基礎を形成するのはこの共通性である。また、特定のフォルダ内のアイテムのメンバシップはそのアイテムのいずれかの特定のアスペクトに基づいて強制的ではないのに対して、ある実施形態では、あるカテゴリにカテゴリ的に関係する共通性を有するすべてのアイテムは自動的にハードウェア／ソフトウェアインターフェースシステムレベルでそのカテゴリのメンバとなる可能性がある。概念上、カテゴリをまた、そのメンバシップが特定のクエリ（データベースのコンテキスト内など）の結果に基づく仮想アイテムフォルダと見なすこともでき、このクエリ（カテゴリの共通性によって定義される）の条件を満たすアイテムはしたがってカテゴリのメンバシップを備えるようになる。

【００７８】

図４は、アイテム、アイテムフォルダおよびカテゴリの間の構造リレーションシップを例示する。複数のアイテム４０２、４０４、４０６、４０８、４１０、４１２、４１４、４１６、４１８および４２０は、様々なアイテムフォルダ４２２、４２４、４２６、４２８および４３０のメンバである。いくつかのアイテムは複数のアイテムフォルダに属することができ、例えば、アイテム４０２はアイテムフォルダ４２２および４２４に属する。いくつかのアイテム、例えばアイテム４０２、４０４、４０６、４０８、４１０および４１２はまた、１つまたは複数のカテゴリ４３２、４３４および４３６のメンバでもあるが、他の時間、例えばアイテム４１４、４１６、４１８および４２０は、どのカテゴリにも属さない場合がある（これは、いかなるプロパティの所有も自動的にカテゴリ内のメンバシップを暗黙指定するある実施形態では概して可能性が低く、したがって、アイテムは、このような実施形態ではいかなるカテゴリのメンバでないようにするために完全に特徴がないようでなければならない）。フォルダの階層構造と対照的に、カテゴリおよびアイテムフォルダは、図示の有向グラフにより類似の構造を有する。いずれにしても、アイテム、アイテムフォルダおよびカテゴリは、（異なるアイテムタイプにもかかわらず）すべてアイテムである。

【００７９】

ファイル、フォルダおよびディレクトリとは対照的に、本発明のアイテム、アイテムフ

ォルダおよびカテゴリは物理的コンテナの概念的同等物を有していないので、性質において特徴的に「物理的」ではなく、したがって、アイテムは複数のこのようなロケーション内に存在することができるからである。アイテムが複数のアイテムフォルダのロケーション内に存在し、ならびにカテゴリに編成されるための能力は、当技術分野で現在使用可能な機能を超えた、ハードウェア／ソフトウェアインターフェースレベルでの強化され、豊富にされた程度のデータ操作およびストレージ構造機能を提供する。

【００８０】

４．スキーマ

ａ）ベーススキーマ

アイテムの作成および使用のための汎用の基礎を提供するために、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、アイテムおよびプロパティを作成および編成するための概念的フレームワークを確立するベーススキーマを備える。ベーススキーマはある特殊なタイプのアイテムおよびプロパティ、および、それからサブタイプをさらに導出することができるこれらの特殊な基礎タイプの特徴を定義する。このベーススキーマの使用により、プログラマはアイテム（およびそれらの各タイプ）をプロパティ（およびそれらの各タイプ）から概念的に区別することができる。また、ベーススキーマは、すべてのアイテムが所有することができるプロパティの基礎セットを示し、これは、すべてのアイテム（およびそれらの対応するアイテムタイプ）はベーススキーマ（およびその対応するアイテムタイプ）内のこの基礎アイテムから導出されるからである。

【００８１】

図７に例示するように、また、本発明のいくつかの実施形態に関して、ベーススキーマは３つのトップレベルタイプであるＩｔｅｍ、ＥｘｔｅｎｓｉｏｎおよびＰｒｏｐｅｒｔｙＢａｓｅを定義する。図のように、Ｉｔｅｍタイプは、この基礎の「Ｉｔｅｍ」アイテムタイプのプロパティによって定義される。対照的に、トップレベルプロパティタイプ「ＰｒｏｐｅｒｔｙＢａｓｅ」は事前定義されたプロパティを有しておらず、それからすべての他のプロパティタイプが派生され、それを通じてすべての派生されたプロパティタイプが相関される（単一のプロパティタイプから共通して派生される）、アンカーに過ぎない。Ｅｘｔｅｎｓｉｏｎタイププロパティは、どのアイテムをエクステンションが拡張するか、ならびに、アイテムが複数のエクステンションを有する可能性がある場合にあるエクステンションを別のエクステンションから区別するための識別を定義する。

【００８２】

ＩｔｅｍＦｏｌｄｅｒは、Ｉｔｅｍから継承されたプロパティに加えて、そのメンバ（ある場合）へのリンクを確立するためのリレーションシップを特徴付ける、Ｉｔｅｍアイテムタイプのサブタイプであるのに対して、ＩｄｅｎｔｉｔｙＫｅｙおよびＰｒｏｐｅｒｔｙはＰｒｏｐｅｒｔｙＢａｓｅのサブタイプである。ＣａｔｅｇｏｒｙＲｅｆは、ＩｄｅｎｔｉｔｙＫｅｙのサブタイプである。

【００８３】

ｂ）コアスキーマ

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、トップレベルアイテムタイプ構造のための概念的フレームワークを提供するコアスキーマを備える。図８Ａは、コアスキーマ内のアイテムを例示するブロック図であり、図８Ｂは、コアスキーマ内のプロパティタイプを例示するブロック図である。ファイルおよびフォルダベースのシステムにおける、異なる拡張子（＊．ｃｏｍ、＊．ｅｘｅ、＊．ｂａｔ、＊．ｓｙｓなど）を有するファイルと他のこのような基準の間でなされる区別は、コアスキーマのファンクションに類似している。アイテム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、コアスキーマはコアアイテムタイプのセットを定義し、このセットは直接的（アイテムタイプによる）または間接的（アイテムサブタイプによる）にすべてのアイテムを、アイテム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て所定の予測可能な方法で直接処理することができる１つまたは複数のコアスキーマアイテムタイプに特徴付ける。事前定義されたアイテムタイプは、アイテムベースのハードウ

ェア／ソフトウェアインターフェースシステム内の最も共通のアイテムを反映し、したがって、アイテム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、コアスキーマを備えるこれらの事前定義されたアイテムタイプを理解することによって、あるレベルの効率が得られる。

【００８４】

ある実施形態では、コアスキーマは拡張可能ではなく、すなわち、追加のアイテムタイプをベーススキーマ内のアイテムタイプから直接サブタイプ定義することはできず、ただし、コアスキーマの一部である特定の事前定義された派生アイテムタイプを除く。コアスキーマへのエステンションを防止することによって（すなわち、コアスキーマへの新しいアイテムの追加を防止することによって）、ストレージプラットフォームはコアスキーマアイテムタイプの使用を命令し、これはあらゆる後続のアイテムタイプが必ずコアスキーマアイテムタイプのサブタイプであるからである。この構造は、事前定義されたセットのコアアイテムタイプを有する利点をも保持しながら、追加のアイテムタイプの定義において妥当な程度の柔軟性を可能にする。

【００８５】

本発明の様々な実施形態では、また図８Ａを参照すると、コアスキーマによってサポートされた特定のアイテムタイプには、以下の１つまたは複数が含まれる可能性がある。

【００８６】

・Ｃａｔｅｇｏｒｙ：このアイテムタイプのアイテム（およびそれから派生されたサブタイプ）は、アイテム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有効なカテゴリを表現する。

【００８７】

・Ｃｏｍｍｏｄｉｔｙ：値の識別可能な物であるアイテム。

【００８８】

・Ｄｅｖｉｃｅ：情報処理機能をサポートする論理構造を有するアイテム。

【００８９】

・Ｄｏｃｕｍｅｎｔ：アイテム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って解釈されないが、その代わりにドキュメントタイプに対応するアプリケーションプログラムによって解釈される、コンテンツを有するアイテム。

【００９０】

・Ｅｖｅｎｔ：環境内のあるオカレンス（ｏｃｃｕｒｒｅｎｃｅ）を記録するアイテム。

【００９１】

・Ｌｏｃａｔｉｏｎ：物理ロケーション（例えば、地理的ロケーション）を表現するアイテム。

【００９２】

・Ｍｅｓｓａｇｅ：２つ以上のプリンシパル（以下で定義）の間の通信のアイテム。

【００９３】

・Ｐｒｉｎｃｉｐａｌ：ＩｔｅｍＩｄの他に、少なくとも１つの決定的に証明可能な識別を有するアイテム（例えば、人、組織、グループ、世帯、権限、サービスなどの識別）。

【００９４】

・Ｓｔａｔｅｍｅｎｔ：ポリシー、購読、クレデンシャルなどを限定なしに含む、環境に関する特殊情報を有するアイテム。

【００９５】

同様に、図８Ｂを参照すると、コアスキーマによってサポートされた特定のプロパティタイプには、以下の１つまたは複数が含まれる可能性がある。

【００９６】

・Ｃｅｒｔｉｆｉｃａｔｅ（ベーススキーマの基礎的なＰｒｏｐｅｒｔｙＢａｓｅタイプから派生される）。

【００９７】
・Ｐｒｉｎｃｉｐａｌ　Ｉｄｅｎｔｉｔｙ　Ｋｅｙ（ベーススキーマのＩｄｅｎｔｉｔｙＫｅｙタイプから派生される）。
【００９８】
・Ｐｏｓｔａｌ　Ａｄｄｒｅｓｓ（ベーススキーマのＰｒｏｐｅｒｔｙタイプから派生される）。
【００９９】
・Ｒｉｃｈ　Ｔｅｘｔ（ベーススキーマのＰｒｏｐｅｒｔｙタイプから派生される）。
【０１００】
・ＥＡｄｄｒｅｓｓ（ベーススキーマのＰｒｏｐｅｒｔｙタイプから派生される）。
【０１０１】
・ＩｄｅｎｔｉｔｙＳｅｃｕｒｉｔｙＰａｃｋａｇｅ（ベーススキーマのＲｅｌａｔｉｏｎｓｈｉｐタイプから派生される）。
【０１０２】
・ＲｏｌｅＯｃｃｕｐａｎｃｙ（ベーススキーマのＲｅｌａｔｉｏｎｓｈｉｐタイプから派生される）。
【０１０３】
・ＢａｓｉｃＰｒｅｓｅｎｃｅ（ベーススキーマのＲｅｌａｔｉｏｎｓｈｉｐタイプから派生される）。
【０１０４】
これらのアイテムおよびプロパティはさらに、図８Ａおよび図８Ｂに示すそれらの各プロパティによって記述される。
【０１０５】
５．リレーションシップ
リレーションシップは、一方のアイテムがソースとして指定され、他方のアイテムがターゲットとして指定される、２項リレーションシップである。ソースアイテムおよびターゲットアイテムはリレーションシップによって関係付けられる。ソースアイテムは一般にリレーションシップのライフタイムをコントロールする。すなわち、ソースアイテムが削除される場合、これらのアイテムの間のリレーションシップも削除される。
【０１０６】
リレーションシップは、包含および参照リレーションシップに分類される。包含リレーションシップはターゲットアイテムのライフタイムをコントロールするが、参照リレーションシップはいかなるライフタイム管理セマンティクスも提供しない。図１２は、リレーションシップが分類される方法を例示する。
【０１０７】
包含リレーションシップタイプはさらに保留および埋め込みリレーションシップに分類される。あるアイテムへのすべての保留リレーションシップが除去される場合、このアイテムは削除される。保留リレーションシップはターゲットのライフタイムを、参照カウントメカニズムを通じてコントロールする。埋め込みリレーションシップは、複合アイテムのモデリングを可能にし、排他的な保留リレーションシップと見なすことができる。アイテムを１つまたは複数の保留リレーションシップのターゲットにすることができるが、アイテムを厳密に１つの埋め込みリレーションシップのターゲットにすることができる。埋め込みリレーションシップのターゲットであるアイテムを、いかなる他の保留または埋め込みリレーションシップのターゲットにすることもできない。
【０１０８】
参照リレーションシップは、ターゲットアイテムのライフタイムをコントロールしない。これらのリレーションシップはダングリングである可能性があり、ターゲットアイテムは存在しない可能性がある。参照リレーションシップを使用して、グローバルなアイテムネームスペース（すなわち、リモートデータストアを含む）内のいかなる場所のアイテムへの参照もモデリングすることができる。

【０１０９】
アイテムのフェッチは自動的にそのリレーションシップをフェッチしない。アプリケーションは明示的にアイテムのリレーションシップを要求しなければならない。加えて、リレーションシップの修正は、ソースまたはターゲットアイテムを修正せず、同様に、リレーションシップの追加はソース／ターゲットアイテムに影響を及ぼさない。
【０１１０】
リレーションシップ宣言
明示的リレーションシップタイプは以下の要素により定義される。
【０１１１】
・リレーションシップ名はＮａｍｅ属性内で規定される。
【０１１２】
・保留、埋め込み、参照のうち１つであるリレーションタイプ。これはＴｙｐｅ属性内で規定される。
【０１１３】
・ソースおよびターゲットエンドポイント。各エンドポイントは、参照されたアイテムの名前およびタイプを規定する。
【０１１４】
・ソースエンドポイントフィールドは一般にタイプＩｔｅｍＩＤ（宣言されない）であり、リレーションシップインスタンスと同じデータストア内のアイテムを参照しなければならない。
【０１１５】
・保留および埋め込みリレーションシップでは、ターゲットエンドポイントフィールドはタイプＩｔｅｍＩＤＲｅｆｅｒｅｎｃｅでなければならず、リレーションシップインスタンスと同じデータストア内のアイテムを参照しなければならない。参照リレーションシップでは、ターゲットエンドポイントはいずれかのＩｔｅｍＲｅｒｅｆｅｎｃｅタイプにすることができ、他のストレージプラットフォームデータストア内のアイテムを参照することができる。
【０１１６】
・オプショナルで、スカラまたはＰｒｏｐｅｒｔｙＢａｓｅタイプの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、リレーションシップに関連付けられたデータを含むことができる。
【０１１７】
・リレーションシップインスタンスはグローバルリレーションシップテーブル内に格納される。
【０１１８】
・あらゆるリレーションシップインスタンスは組合せ（ソースＩｔｅｍＩＤ、リレーションシップＩＤ）によって一意に識別される。リレーションシップＩＤは、それらのタイプにかかわらず、所与のアイテム内で供給されたすべてのリレーションシップのための所与のソースＩｔｅｍＩＤ内で一意である。
【０１１９】
ソースアイテムはリレーションシップの所有者である。所有者として指定されたアイテムはリレーションシップのライフタイムをコントロールするが、リレーションシップ自体は、それが関係するアイテムから分離している。ストレージプラットフォームＡＰＩ３２２は、アイテムに関連付けられたリレーションシップをエクスポーズするためのメカニズムを提供する。
【０１２０】
これはリレーションシップ宣言の一実施例である。
【０１２１】

【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>
```

【０１２２】

これは、参照リレーションシップの一実施例である。ソース参照によって参照されるｐｅｒｓｏｎアイテムが存在しない場合、このリレーションシップを作成することはできない。また、ｐｅｒｓｏｎアイテムが削除される場合、ｐｅｒｓｏｎとｏｒｇａｎｉｚａｔｉｏｎの間のリレーションシップインスタンスは削除される。しかし、Ｏｒｇａｎｉｚａｔｉｏｎアイテムが削除される場合、リレーションシップは削除されず、それはダングリングである。

【０１２３】

保留リレーションシップ

保留リレーションシップは、ターゲットアイテムのライフタイム管理に基づいて参照カウントをモデリングするために使用される。

【０１２４】

アイテムを、アイテムへのゼロまたは複数のリレーションシップのためのソースエンドポイントにすることができる。埋め込みアイテムではないアイテムを、１つまたは複数の保留リレーションシップ内のターゲットにすることができる。

【０１２５】

ターゲットエンドポイント参照タイプはＩｔｅｍＩＤＲｅｆｅｒｅｎｃｅでなければならず、リレーションシップインスタンスと同じストア内のアイテムを参照しなければならない。

【０１２６】

保留リレーションシップは、ターゲットエンドポイントのライフタイム管理を実施する。保留リレーションシップインスタンス、および、それがターゲットにしているアイテムの作成は、アトミックオペレーションである。同じアイテムをターゲットにする追加の保留リレーションシップインスタンスを作成することができる。所与のアイテムをターゲットエンドポイントとして有する最後の保留リレーションシップインスタンスが削除される場合、ターゲットアイテムも削除される。

【０１２７】

リレーションシップ宣言で規定されたエンドポイントアイテムのタイプは一般に、リレーションシップのインスタンスが作成される場合に実施されるようになる。エンドポイントアイテムのタイプを、リレーションシップが確立された後に変更することはできない。

【０１２８】

保留リレーションシップは、アイテムネームスペースの形成において重要な役割を果たす。保留リレーションシップは、ソースアイテムに対してターゲットアイテムの名前を定義する「Ｎａｍｅ」プロパティを含む。この相対名は、所与のアイテムから供給されたすべての保留リレーションシップについて一意である。ルートアイテムから開始して所与のアイテムまでのこの相対名の順序付きリストは、アイテムへのフルネームを形成する。

【０１２９】

保留リレーションシップは非循環有向グラフ（ＤＡＧ）を形成する。保留リレーション

シップが作成される場合、システムは、周期が作成されないことを保証し、したがって、アイテムネームスペースがＤＡＧを形成することを保証する。

【０１３０】

保留リレーションシップはターゲットアイテムのライフタイムをコントロールするが、ターゲットエンドポイントアイテムのオペレーショナルな一貫性をコントロールしない。ターゲットアイテムは、保留リレーションシップを通じてそれを所有するアイテムから、オペレーション上は独立している。保留リレーションシップのソースであるアイテムにおけるＣｏｐｙ、Ｍｏｖｅ、Ｂａｃｋｕｐおよび他のオペレーションは、同じリレーションシップのターゲットであるアイテムに影響を及ぼさず、例えばすなわち、フォルダアイテムをバックアップしても、そのフォルダ内のすべてのアイテム（ＦｏｌｄｅｒＭｅｍｂｅｒリレーションシップのターゲット）を自動的にバックアップしない。

【０１３１】

以下は保留リレーションシップの一実施例である。

【０１３２】

【表２】

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

【０１３３】

ＦｏｌｄｅｒＭｅｍｂｅｒｓリレーションシップは、アイテムの汎用コレクションとしてのフォルダの概念を可能にする。

【０１３４】

埋め込みリレーションシップ

埋め込みリレーションシップは、ターゲットアイテムのライフタイムの排他的コントロールの概念をモデリングする。このリレーションシップは、複合アイテムの概念を可能にする。

【０１３５】

埋め込みリレーションシップインスタンス、および、それがターゲットにしているアイテムの作成は、アトミックオペレーションである。アイテムを、ゼロまたは複数の埋め込みリレーションシップのソースにすることができる。しかし、アイテムを、ただ１つの埋め込みリレーションシップのターゲットにすることができる。埋め込みリレーションシップのターゲットであるアイテムを、保留リレーションシップ内のターゲットにすることはできない。

【０１３６】

ターゲットエンドポイント参照タイプはＩｔｅｍＩＤＲｅｆｅｒｅｎｃｅでなければならず、リレーションシップインスタンスと同じデータストア内のアイテムを参照しなければならない。

【０１３７】

リレーションシップ宣言で規定されたエンドポイントアイテムのタイプは一般に、リレーションシップのインスタンスが作成される場合に実施されるようになる。エンドポイントアイテムのタイプを、リレーションシップが確立された後に変更することはできない。

【０１３８】

埋め込みリレーションシップは、ターゲットエンドポイントのオペレーショナルな一貫性をコントロールする。例えば、アイテムの直列化のオペレーションは、そのアイテムならびにそれらのターゲットのすべてから供給するすべての埋め込みリレーションシップの直列化を含むことができ、アイテムをコピーすることはまたそのすべての埋め込みアイテムをもコピーする。

【０１３９】
以下は一実施例の宣言である。
【０１４０】
【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>
```

【０１４１】
参照リレーションシップ
参照リレーションシップは、それが参照するアイテムのライフタイムをコントロールしない。さらに、参照リレーションシップは、ターゲットの存在を保証せず、リレーションシップ宣言で規定されたターゲットのタイプも保証しない。これは、参照リレーションシップがダングリングになる可能性があることを意味する。また、参照リレーションシップは他のデータストア内のアイテムを参照することができる。参照リレーションシップを、ウェブページ内のリンクに類似した概念と見なすことができる。
【０１４２】
以下は参照リレーションシップ宣言の一実施例である。
【０１４３】
【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>
```

【０１４４】
いかなる参照タイプもターゲットポイントにおいて許可される。参照リレーションシップに参加するアイテムは、いかなるアイテムタイプであってもよい。
【０１４５】
参照リレーションシップは、複数のアイテムの間の非ライフタイム管理リレーションシップをモデリングするために使用される。ターゲットの存在が実施されないので、参照リレーションシップは、疎結合リレーションシップをモデリングするために好都合である。参照リレーションシップを、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アイテムをターゲットにするために使用することができる。
【０１４６】
ルールおよび制約
以下の追加のルールおよび制約がリレーションシップに対して適用される。
【０１４７】
・アイテムは（厳密に１つの埋め込みリレーションシップ）または（１つまたは複数の

保留リレーションシップ）のターゲットでなければならない。１つの例外はルートアイテムである。アイテムをゼロまたは複数の参照リレーションシップのターゲットにすることができる。

【０１４８】

・埋め込みリレーションシップのターゲットであるアイテムを、保留リレーションシップのソースにすることはできない。このアイテムを参照リレーションシップのソースにすることができる。

【０１４９】

・アイテムがファイルからプロモートされる場合、このアイテムを保留リレーションシップのソースにすることはできない。このアイテムを埋め込みリレーションシップおよび参照リレーションシップのソースにすることができる。

【０１５０】

・ファイルからプロモートされるアイテムを、埋め込みリレーションシップのターゲットにすることはできない。

【０１５１】

リレーションシップの順序付け

少なくとも１つの実施形態では、本発明のストレージプラットフォームは、リレーションシップの順序付けをサポートする。この順序付けは、ベースリレーションシップ定義内の「Ｏｒｄｅｒ」という名前のプロパティを通じて達成される。Ｏｒｄｅｒフィールド上に一意性の制約はない。同じ「ｏｒｄｅｒ」プロパティ値を有するリレーションシップの順序は保証されないが、これらのリレーションシップを、より低い「ｏｒｄｅｒ」値を有するリレーションシップの後、および、より高い「ｏｒｄｅｒ」フィールド値を有するリレーションシップの前に順序付けすることができることが保証される。

【０１５２】

アプリケーションは、組合せ（ＳｏｕｒｃｅＩｔｅｍＩＤ、ＲｅｌａｔｉｏｎｓｈｉｐＩＤ、Ｏｒｄｅｒ）において順序付けることによって、デフォルト順序におけるリレーションシップを得ることができる。所与のアイテムから供給されたすべてのリレーションシップインスタンスは、コレクション内のリレーションシップのタイプにかかわらず、単一のコレクションとして順序付けされる。これはしかし、所与のタイプ（例えば、ＦｏｌｄｅｒＭｅｍｂｅｒｓ）のすべてのリレーションシップが所与のアイテムのためのリレーションシップコレクションの順序付きサブセットであることを保証する。

【０１５３】

リレーションシップを操作するためのデータストアＡＰＩ３１２は、リレーションシップの順序付けをサポートするオペレーションのセットを実装する。以下の用語は、これらのオペレーションを説明する助けとなるために導入される。

【０１５４】

・ＲｅｌＦｉｒｓｔは、順序付きコレクション内で順序値ＯｒｄＦｉｒｓｔを有する最初のリレーションシップである。

【０１５５】

・ＲｅｌＬａｓｔは、順序付きコレクション内で順序値ＯｒｄＬａｓｔを有する最後のリレーションシップである。

【０１５６】

・ＲｅｌＸは、コレクション内で順序値ＯｒｄＸを有する所与のリレーションシップである。

【０１５７】

・ＲｅｌＰｒｅｖは、コレクション内で、ＯｒｄＸより小さい順序値ＯｒｄＰｒｅｖを有する、ＲｅｌＸに最も近いリレーションシップである。

【０１５８】

・ＲｅｌＮｅｘｔは、コレクション内で、ＯｒｄＸより大きい順序値ＯｒｄＮｅｘｔを有する、ＲｅｌＸに最も近いリレーションシップである。

【０１５９】
これらのオペレーションには、制限なしに以下が含まれる。
【０１６０】
・ＩｎｓｅｒｔＢｅｆｏｒｅＦｉｒｓｔ（ＳｏｕｒｃｅＩｔｅｍＩＤ、Ｒｅｌａｔｉｏｎｓｈｉｐ）は、リレーションシップをコレクション内の最初のリレーションシップとして挿入する。新しいリレーションシップの「Ｏｒｄｅｒ」プロパティの値は、ＯｒｄＦｉｒｓｔより小さくてもよい。
【０１６１】
・ＩｎｓｅｒｔＡｆｔｅｒＬａｓｔ（ＳｏｕｒｃｅＩｔｅｍＩＤ、Ｒｅｌａｔｉｏｎｓｈｉｐ）は、リレーションシップをコレクション内の最後のリレーションシップとして挿入する。新しいリレーションシップの「Ｏｒｄｅｒ」プロパティの値は、ＯｒｄＬａｓｔより大きくてもよい。
【０１６２】
・ＩｎｓｅｒｔＡｔ（ＳｏｕｒｃｅＩｔｅｍＩＤ、ｏｒｄ、Ｒｅｌａｔｉｏｎｓｈｉｐ）は、「Ｏｒｄｅｒ」プロパティに対して規定された値を有するリレーションシップを挿入する。
【０１６３】
・ＩｎｓｅｒｔＢｅｆｏｒｅ（ＳｏｕｒｃｅＩｔｅｍＩＤ、ｏｒｄ、Ｒｅｌａｔｉｏｎｓｈｉｐ）は、リレーションシップを、所与の順序値を有するリレーションシップの前に挿入する。新しいリレーションシップに、非包含的にＯｒｄＰｒｅｖとｏｒｄの間である「Ｏｒｄｅｒ」値を割り当てることができる。
【０１６４】
・ＩｎｓｅｒｔＡｆｔｅｒ（ＳｏｕｒｃｅＩｔｅｍＩＤ、ｏｒｄ、Ｒｅｌａｔｉｏｎｓｈｉｐ）は、リレーションシップを、所与の順序値を有するリレーションシップの後に挿入する。新しいリレーションシップに、非包含的にｏｒｄとＯｒｄＮｅｘｔの間である「Ｏｒｄｅｒ」値を割り当てることができる。
【０１６５】
・ＭｏｖｅＢｅｆｏｒｅ（ＳｏｕｒｃｅＩｔｅｍＩＤ、ｏｒｄ、ＲｅｌａｔｉｏｎｓｈｉｐＩＤ）は、所与のリレーションシップＩＤを有するリレーションシップを、規定された「Ｏｒｄｅｒ」値を有するリレーションシップの前に移動する。このリレーションシップに、非包含的にＯｒｄＰｒｅｖとｏｒｄの間である新しい「Ｏｒｄｅｒ」値を割り当てることができる。
【０１６６】
・ＭｏｖｅＡｆｔｅｒ（ＳｏｕｒｃｅＩｔｅｍＩＤ、ｏｒｄ、ＲｅｌａｔｉｏｎｓｈｉｐＩＤ）は、所与のリレーションシップＩＤを有するリレーションシップを、規定された「Ｏｒｄｅｒ」値を有するリレーションシップの後に移動する。このリレーションシップに、非包含的にｏｒｄとＯｒｄＮｅｘｔの間である新しい順序値を割り当てることができる。
【０１６７】
前述のように、あらゆるアイテムはアイテムフォルダのメンバでなければならない。リレーションシップに関して、あらゆるアイテムはアイテムフォルダとのリレーションシップを有し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、あるリレーションシップは、複数のアイテムの間に存在するリレーションシップによって表現される。
【０１６８】
本発明の様々な実施形態について実装されるように、リレーションシップは、あるアイテム（ソース）によって別のアイテム（ターゲット）に「拡張」される、有向の２項リレーションシップを提供する。リレーションシップはソースアイテム（それを拡張したアイテム）によって所有され、したがってリレーションシップは、ソースが除去される場合に除去される（例えば、リレーションシップは、ソースアイテムが削除される場合、削除される）。また、ある場合では、リレーションシップは（共同所有）ターゲットアイテムの

所有権を共有することができ、このような所有権は、リレーションシップのＩｓＯｗｎｅｄプロパティ（またはその同等物）内で反映される可能性がある（図７のように、Ｒｅｌａｔｉｏｎｓｈｉｐプロパティタイプについて）。これらの実施形態では、新しいＩｓＯｗｎｅｄリレーションシップの作成は自動的にターゲットアイテムにおける参照カウントを増分し、このようなリレーションシップの削除はターゲットアイテムにおける参照カウントを減分することができる。これらの特定の実施形態では、アイテムは、ゼロより大きい参照カウントを有する場合に存在し続け、カウントがゼロに達する場合に自動的に削除される。この場合も、アイテムフォルダは、他のアイテムへのリレーションシップのセットを有する（または、有することができる）アイテムであり、これらの他のアイテムは、アイテムフォルダのメンバシップを備える。リレーションシップの他の実際の実装は、本明細書で説明する機能性を達成するために本発明によって可能であり、予想される。

【０１６９】

実際の実装にかかわらず、リレーションシップはあるオブジェクトから別のオブジェクトへの選択可能な接続である。アイテムが複数のアイテムフォルダに、ならびに、１つまたは複数のカテゴリに属するための能力、および、これらのアイテム、フォルダおよびカテゴリがパブリックであるかプライベートであるかは、アイテムベースの構造内の存在（またはその欠如）に与えられた意味によって決定される。これらの論理的リレーションシップは、本明細書で説明する機能性を達成するために特に使用される物理的実装にかかわらず、リレーションシップのセットに割り当てられた意味である。論理的リレーションシップは、アイテムとそのアイテムフォルダまたはカテゴリ（およびその逆）の間で確立され、これは本質的に、アイテムフォルダおよびカテゴリはそれぞれアイテムの特殊タイプであるからである。したがって、アイテムフォルダおよびカテゴリを他のいかなるアイテムとも同じように操作することができ、すなわち、限定なしに、コピー、電子メールメッセージに追加、ドキュメントに埋め込むことなどができ、アイテムフォルダおよびカテゴリを、他のアイテム用と同じメカニズムを使用して直列化および非直列化（インポートおよびエクスポート）することができる。（例えば、ＸＭＬでは、すべてのアイテムは直列化フォーマットを有することができ、このフォーマットはアイテムフォルダ、カテゴリおよびアイテムに等しく適用される）。

【０１７０】

前述の関係は、アイテムとそのアイテムフォルダの間の関係を表現し、論理的にアイテムからアイテムフォルダへ、アイテムフォルダからアイテムへ、またはその両方に拡張することができる。アイテムからアイテムフォルダへ論理的に拡張するリレーションシップは、アイテムフォルダがそのアイテムに対してパブリックであり、そのメンバシップ情報をそのアイテムと共有することを示し、反対に、アイテムからアイテムフォルダへの論理的リレーションシップの欠如は、アイテムフォルダがそのアイテムに対してプライベートであり、そのメンバシップ情報をそのアイテムと共有しないことを示す。同様に、アイテムフォルダからアイテムへ論理的に拡張するリレーションシップは、アイテムがそのアイテムフォルダに対してパブリックおよび共有可能であることを示すのに対して、アイテムフォルダからアイテムへの論理的リレーションシップの欠如は、アイテムがプライベートおよび非共有可能であることを示す。したがって、アイテムフォルダが別のシステムにエクスポートされる場合、新しいコンテキストにおいて共有されるのは「パブリック」アイテムであり、アイテムがそのアイテムフォルダを他の共有可能なアイテムについて検索する場合、それに属する共有可能なアイテムに関する情報をアイテムに提供するのは、「パブリック」アイテムフォルダである。

【０１７１】

図９は、アイテムフォルダ（この場合もアイテム自体である）、そのメンバアイテム、およびアイテムフォルダとそのメンバアイテムの間の相互接続リレーションシップを例示するブロック図である。アイテムフォルダ９００は、メンバとして複数のアイテム９０２、９０４および９０６を有する。アイテムフォルダ９００は、それ自体からアイテム９０２へのリレーションシップ９１２を有し、これはアイテム９０２が、アイテムフォルダ９

００、そのメンバ９０４および９０６、ならびに、アイテムフォルダ９００にアクセスすることができる他のいかなるアイテムフォルダ、カテゴリまたはアイテム（図示せず）に対してもパブリックおよび共有可能であることを示す。しかし、アイテム９０２からアイテムフォルダ９００へのリレーションシップはなく、これは、アイテムフォルダ９００がアイテム９０２に対してプライベートであり、そのメンバシップ情報をアイテム９０２と共有しないことを示す。アイテム９０４は、他方では、それ自体からアイテムフォルダ９００へのリレーションシップ９２４を有し、これはアイテムフォルダ９００がパブリックであり、そのメンバシップ情報をアイテム９０４と共有することを示す。しかし、アイテムフォルダ９００からアイテム９０４へのリレーションシップはなく、これは、アイテム９０４がアイテムフォルダ９００、その他のメンバ９０２および９０６、ならびに、アイテムフォルダ９００にアクセスすることができる他のいかなるアイテムフォルダ、カテゴリまたはアイテム（図示せず）に対してもプライベートであり、共有可能でないことを示す。アイテム９０２および９０４へのそのリレーションシップ（またはその欠如）とは対照的に、アイテムフォルダ９００はそれ自体からアイテム９０６へのリレーションシップ９１６を有し、アイテム９０６はアイテムフォルダ９００に戻るリレーションシップ９２６を有し、これは共に、アイテム９０６がアイテムフォルダ９００、そのメンバ９０２および９０４、ならびに、アイテムフォルダ９００にアクセスすることができる他のいかなるアイテムフォルダ、カテゴリまたはアイテム（図示せず）に対してもパブリックおよび共有可能であること、および、アイテムフォルダ９００がパブリックであり、そのメンバシップ情報をアイテム９０６と共有することを示す。

【０１７２】

前述のように、アイテムフォルダ内のアイテムは共通性を共有する必要はなく、これはアイテムフォルダが「記述」されないからである。カテゴリは、他方では、そのメンバアイテムのすべてに共通である共通性によって記述される。したがって、カテゴリのメンバシップは本質的に、記述された共通性を有するアイテムに制限され、ある実施形態では、カテゴリの記述を満たすすべてのアイテムは自動的にそのカテゴリのメンバにされる。このように、アイテムフォルダはトリビアルなタイプ構造をそれらのメンバシップによって表現することを可能にするのに対して、カテゴリは、定義された共通性に基づいたメンバシップを可能にする。

【０１７３】

言うまでもなく、カテゴリ記述は性質において論理的であり、したがってカテゴリをいかなる論理表現のタイプ、プロパティおよび／または値によっても記述することができる。例えば、カテゴリのための論理表現をそのメンバシップにして、２つのプロパティの一方または両方を有するアイテムを備えるようにすることができる。カテゴリについて記述されたこれらのプロパティが「Ａ」および「Ｂ」である場合、カテゴリメンバシップは、プロパティＡを有するがＢを有していないアイテム、プロパティＢを有するがＡを有していないアイテム、および、プロパティＡおよびＢを共に有するアイテムを備えることができる。このプロパティの論理表現は論理演算子「ＯＲ」によって記述され、カテゴリによって記述されたメンバのセットは、プロパティＡ　ＯＲ　Ｂを有するアイテムである。類似の論理オペランド（制限なしに「ＡＮＤ」、「ＸＯＲ」および「ＮＯＴ」を単独または組合せで含む）もまた、カテゴリを記述するために使用することができ、これは当業者には理解されよう。

【０１７４】

アイテムフォルダ（記述されない）とカテゴリ（記述される）の間の区別にもかかわらず、アイテムへのカテゴリリレーションシップ、およびカテゴリへのアイテムリレーションシップは本質的に、本明細書で本発明の多数の実施形態におけるアイテムフォルダおよびアイテムについて上記で開示した物と同様である。

【０１７５】

図１０は、カテゴリ（この場合もアイテム自体である）、そのメンバアイテム、およびカテゴリとそのメンバアイテムの間の相互接続リレーションシップを例示するブロック図

である。カテゴリ１０００は、メンバとして複数のアイテム１００２、１００４および１００６を有し、そのすべては、カテゴリ１０００によって記述された共通プロパティ、値またはタイプのある組合せ１００８（共通性記述１００８’）を共有する。カテゴリ１０００は、それ自体からアイテム１００２へのリレーションシップ１０１２を有し、これはアイテム１００２が、カテゴリ１０００、そのメンバ１００４および１００６、ならびに、カテゴリ１０００にアクセスすることができる他のいかなるカテゴリ、アイテムフォルダまたはアイテム（図示せず）に対してもパブリックおよび共有可能であることを示す。しかし、アイテム１００２からカテゴリ１０００へのリレーションシップはなく、これは、カテゴリ１０００がアイテム１００２に対してプライベートであり、そのメンバシップ情報をアイテム１００２と共有しないことを示す。アイテム１００４は、他方では、それ自体からカテゴリ１０００へのリレーションシップ１０２４を有し、これはカテゴリ１０００がパブリックであり、そのメンバシップ情報をアイテム１００４と共有することを示す。しかし、カテゴリ１０００からアイテム１００４に拡張されたリレーションシップはなく、これは、アイテム１００４がカテゴリ１０００、その他のメンバ１００２および１００６、ならびに、カテゴリ１０００にアクセスすることができる他のいかなるカテゴリ、アイテムフォルダまたはアイテム（図示せず）に対してもプライベートであり、共有可能でないことを示す。アイテム１００２および１００４へのそのリレーションシップ（またはその欠如）とは対照的に、カテゴリ１０００はそれ自体からアイテム１００６へのリレーションシップ１０１６を有し、アイテム１００６はカテゴリ１０００に戻るリレーションシップ１０２６を有し、これは共に、アイテム１００６がカテゴリ１０００、そのアイテムメンバ１００２および１００４、ならびに、カテゴリ１０００にアクセスすることができる他のいかなるカテゴリ、アイテムフォルダまたはアイテム（図示せず）に対してもパブリックおよび共有可能であること、および、カテゴリ１０００がパブリックであり、そのメンバシップ情報をアイテム１００６と共有することを示す。

【０１７６】

最後に、カテゴリおよびアイテムフォルダはそれら自体がアイテムであり、アイテムは互いにリレーションシップすることができるので、カテゴリはアイテムフォルダにリレーションシップすることができ、その逆も可能であり、ある代替実施形態では、カテゴリ、アイテムフォルダおよびアイテムはそれぞれ他のカテゴリ、アイテムフォルダおよびアイテムにリレーションシップすることができる。しかし、様々な実施形態では、アイテムフォルダ構造および／またはカテゴリ構造は、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は、サイクルを含むことを禁止される。アイテムフォルダおよびカテゴリ構造が有向グラフに類似しているところでは、サイクルを禁止する実施形態は非循環有向グラフ（ＤＡＧ）に類似しており、このグラフは、グラフ理論の技術分野における数学的定義によって、パスが同じ頂点で開始および終了しない有向グラフである。

【０１７７】

６．拡張可能性

ストレージプラットフォームは、上述のように、スキーマの初期セット３４０を備えるように意図される。加えて、しかし、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は、独立系ソフトウェアベンダー（ＩＳＶ）を含む顧客が新しいスキーマ３４４（すなわち、新しいＩｔｅｍおよびＮｅｓｔｅｄ　Ｅｌｅｍｅｎｔタイプ）を作成できるようにする。このセクションは、スキーマの初期セット３４０内で定義されたＩｔｅｍタイプおよびＮｅｓｔｅｄ　Ｅｌｅｍｅｎｔタイプ（または単に「Ｅｌｅｍｅｎｔ」タイプ）を拡張することによって、このようなスキーマを作成するためのメカニズムを扱う。

【０１７８】

好ましくは、ＩｔｅｍおよびＮｅｓｔｅｄ　Ｅｌｅｍｅｎｔタイプの初期セットのエクステンションは、以下のように制約される。

【０１７９】

・ＩＳＶは新しいＩｔｅｍタイプ、すなわちサブタイプＢａｓｅ．Ｉｔｅｍを導入する

ことを許可される。

【０１８０】

・ＩＳＶは新しいＮｅｓｔｅｄ　Ｅｌｅｍｅｎｔタイプ、すなわちサブタイプＢａｓｅ．ＮｅｓｔｅｄＥｌｅｍｅｎｔを導入することを許可される。

【０１８１】

・ＩＳＶは新しいエクステンション、すなわちサブタイプＢａｓｅ．ＮｅｓｔｅｄＥｌｅｍｅｎｔを導入することを許可されるが、

・ＩＳＶは、ストレージプラットフォームスキーマの初期セット３４０によって定義されたいかなるタイプ（Ｉｔｅｍ、Ｎｅｓｔｅｄ　ＥｌｅｍｅｎｔまたはＥｘｔｅｎｓｉｏｎタイプ）もサブタイプ定義することはできない。

【０１８２】

ストレージプラットフォームスキーマの初期セットによって定義されたＩｔｅｍタイプまたはＮｅｓｔｅｄ　Ｅｌｅｍｅｎｔタイプは、ＩＳＶのアプリケーションの必要性に厳密に一致しない場合があるので、ＩＳＶがタイプをカスタマイズすることを許可することが必要である。これは、エクステンションの概念により可能にされる。エクステンションは強く型付け（ｔｙｐｉｎｇ）されたインスタンスであるが、（ａ）エクステンションは独立して存在することはできず、（ｂ）エクステンションはアイテムまたはネストされた要素にアタッチされなければならない。

【０１８３】

スキーマ拡張可能性の必要性に対処することに加えて、エクステンションはまた、「マルチタイプ定義（ｍｕｌｔｉ－ｔｙｐｉｎｇ）」の問題に対処するようにも意図される。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複数の継承またはサブタイプのオーバーラップをサポートしない場合があるので、アプリケーションはエクステンションを、タイプインスタンスのオーバーラップをモデリングするための方法として使用することができる（例えば、Ｄｏｃｕｍｅｎｔは法的ドキュメントならびにセキュアドキュメントである）。

【０１８４】

アイテムエクステンション

アイテム拡張可能性を提供するために、データモデルはさらに、Ｂａｓｅ．Ｅｘｔｅｎｓｉｏｎという名前の抽象タイプを定義する。これは、エクステンションタイプの階層のためのルートタイプである。アプリケーションはＢａｓｅ．Ｅｘｔｅｎｓｉｏｎをサブタイプ定義して、特定のエクステンションタイプを作成することができる。

【０１８５】

Ｂａｓｅ．Ｅｘｔｅｎｓｉｏｎタイプは、ベーススキーマ内で以下のように定義される。

【０１８６】

【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>
```

【０１８７】

ＩｔｅｍＩＤフィールドは、エクステンションが関連付けられるアイテムのＩｔｅｍＩＤを含む。このＩｔｅｍＩＤを有するアイテムが存在しなければならない。所与のＩｔｅｍＩＤを有するアイテムが存在しない場合、エクステンションを作成することはできない

。アイテムが削除される場合、同じＩｔｅｍＩＤを有するすべてのエクステンションは削除される。タプル（ＩｔｅｍＩＤ、ＥｘｔｅｎｓｉｏｎＩＤ）はエクステンションインスタンスを一意に識別する。

【０１８８】

エクステンションタイプの構造は、以下のようにアイテムタイプの構造に類似している。

【０１８９】

・エクステンションタイプはフィールドを有する。

【０１９０】

・フィールドは、プリミティブまたはネストされた要素のタイプにすることができる。

【０１９１】

・エクステンションタイプをサブタイプ定義することができる。

【０１９２】

以下の制限はエクスションタイプに適用される。

【０１９３】

・エクステンションをリレーションシップのソースおよびターゲットにすることはできない。

【０１９４】

・エクステンションタイプインスタンスはアイテムから独立して存在することはできない。

【０１９５】

・エクステンションタイプを、ストレージプラットフォームタイプ定義内でフィールドタイプとして使用することはできない。

【０１９６】

所与のアイテムタイプに関連付けることができるエクステンションのタイプに対する制約はない。いかなるエクステンションタイプがいかなるアイテムタイプを拡張することも許可される。複数のエクステンションインスタンスが１つのアイテムにアタッチされる場合、これらのエクステンションインスタンスは構造およびビヘイビアにおいて互いに独立している。

【０１９７】

エクステンションインスタンスは、アイテムとは分離して格納およびアクセスされる。すべてのエクステンションタイプインスタンスは、グローバルエクステンションビューからアクセス可能である。それらが関連付けられるアイテムが何のタイプかにかかわらず、所与のタイプのエクステンションのすべてのインスタンスを戻す、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、アイテムにおいてエクステンションを格納、検索および修正することができる、プログラミングモデルを提供する。

【０１９８】

エクステンションタイプを、ストレージプラットフォームの単一の継承モデルを使用してサブタイプ定義されたタイプにすることができる。エクステンションタイプから派生することは、新しいエクステンションタイプを作成する。エクステンションの構造またはビヘイビアは、アイテムタイプ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換することはできない。アイテムタイプに類似して、エクステンションタイプインスタンスには、エクステンションタイプに関連付けられたビューを通じて直接アクセスすることができる。エクステンションのＩｔｅｍＩＤは、それらがどのアイテムに属するかを示し、対応するアイテムオブジェクトをグローバルアイテムビューから検索するために使用することができる。エクステンションは、オペレーショナルな一貫性のためにアイテムの一部と見なされる。Ｃｏｐｙ／Ｍｏｖｅ、Ｂａｃｋｕｐ／Ｒｅｓｔｏｒｅ、および、ストレージプラットフォームが定義する他の共通オペレーションは、アイテムの一部としてエクステンションに操作を加えることができる。

【０１９９】

以下の実施例を考察する。ＣｏｎｔａｃｔタイプはＷｉｎｄｏｗｓ（登録商標）Ｔｙｐｅセット内で定義される。

【０２００】

【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>
```

【０２０１】

あるＣＲＭアプリケーション開発者は、ＣＲＭアプリケーションエクステンションを、ストレージプラットフォーム内に格納された連絡先にアタッチすることを望む。このアプリケーション開発者は、アプリケーションが操作することができる追加のデータ構造を含むであろうＣＲＭエクステンションを定義するようになる。

【０２０２】

【表７】

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

【０２０３】

あるＨＲアプリケーション開発者もまた、追加のデータをＣｏｎｔａｃｔにアタッチすることを望む場合がある。このデータはＣＲＭアプリケーションデータから独立している。この場合も、このアプリケーション開発者はエクステンションを作成することができる。

【０２０４】

【表８】

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

【０２０５】

ＣＲＭＥｘｔｅｎｓｉｏｎおよびＨＲＥｘｔｅｎｓｉｏｎは、Ｃｏｎｔａｃｔアイテムにアタッチすることができる２つの独立したエクステンションである。これらのエクステンションは、互いに独立して作成およびアクセスされる。

【０２０６】

上記の実施例では、ＣＥＭＥｘｔｅｎｓｉｏｎタイプ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はできない。ＣＲＭＥｘｔｅｎｓｉｏｎタイプのインスタンスをＣｏｎｔａｃｔ以外のアイテムタイプにアタッチすることができることに留意されたい。

【０２０７】
Ｃｏｎｔａｃｔアイテムが検索される場合、そのアイテムエクステンションは自動的に検索されない。Ｃｏｎｔａｃｔアイテムが与えられると、その関係付けられたアイテムエクステンションには、同じＩｔｅｍＩｄを有するエクステンションについてグローバルエクステンションビューをクエリすることによってアクセスすることができる。
【０２０８】
システム内のすべてのＣＲＭＥｘｔｅｎｓｉｏｎエクステンションに、それらがどのアイテムに属するかにかかわらず、ＣＲＭＥｘｔｅｎｓｉｏｎタイプ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エクステンションは、同じアイテムｉｄを共有する。上記の実施例では、ＣｏｎｔａｃｔアイテムインスタンスならびにアタッチされたＣＲＭＥｘｔｅｎｓｉｏｎおよびＨＲＥｘｔｅｎｓｉｏｎは、同じＩｔｅｍＩＤを有する。
【０２０９】
以下の表は、Ｉｔｅｍ、ＥｘｔｅｎｓｉｏｎおよびＮｅｓｔｅｄＥｌｅｍｅｎｔタイプの間の類似性および違いを要約する。
【０２１０】

【表９】

Ｉｔｅｍ対Ｉｔｅｍ　Ｅｘｔｅｎｓｉｏｎ対ＮｅｓｔｅｄＥｌｅｍｅｎｔ

	Item	Itemエクステンション	NestedElement
アイテムID	それ自体のアイテムidを有する。	アイテムのアイテムidを共有する。	それ自体のアイテムidを有していない。ネストされた要素はアイテムの一部である。
ストレージ	アイテム階層はそれ自体のテーブル内に格納される。	アイテムエクステンション階層はそれ自体のテーブル内に格納される。	アイテムと共に格納される。
クエリ/検索	アイテムテーブルをクエリすることができる。	アイテムエクステンションテーブルをクエリすることができる。	一般に、含むアイテムコンテキスト内のみでクエリされることが可能である。
クエリ/検索範囲	アイテムタイプのすべてのインスタンスにわたって検索することができる。	アイテムエクステンションタイプのすべてのインスタンスにわたって検索することができる。	一般に、単一の（含む）アイテムのネストされた要素タイプのインスタンス内でのみ検索することができる。
リレーションシップのセマンティックス	アイテムへのリレーションシップを有することができる。	アイテムエクステンションへのリレーションシップはない。	ネストされた要素へのリレーションシップはない。
アイテムへの関連付け	保留、埋め込みおよびソフトリレーションシップを介して他のアイテムに関係付けることができる。	一般にエクステンションを介してのみ関係付けることができる。エクステンションのセマンティックスは埋め込みアイテムのセマンティックスに類似している。	フィールドを介してアイテムに関係付けられる。ネストされた要素はアイテムの一部である。

【０２１１】

ＮｅｓｔｅｄＥｌｅｍｅｎｔタイプの拡張

ネストされた要素タイプは、アイテムタイプと同じメカニズムにより拡張されない。ネストされた要素のエクステンションは、ネストされた要素タイプのフィールドと同じメカニズムにより格納およびアクセスされる。

【０２１２】

データモデルは、Ｅｌｅｍｅｎｔという名称のネストされた要素タイプのためのルートを定義する。

【０２１３】

【表１０】

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

【０２１４】

ＮｅｓｔｅｄＥｌｅｍｅｎｔタイプはこのタイプから継承する。ＮｅｓｔｅｄＥｌｅｍｅｎｔ要素タイプは加えて、Ｅｌｅｍｅｎｔのマルチセットであるフィールドを定義する。

【０２１５】

【表１１】

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

【０２１６】

ＮｅｓｔｅｄＥｌｅｍｅｎｔエクステンションは、以下のようにアイテムエクステンションとは異なる。

【０２１７】

・ネストされた要素エクステンションはエクステンションタイプではない。これらのエクステンションは、Ｂａｓｅ．Ｅｘｔｅｎｓｉｏｎタイプ内でルートされるエクステンションタイプ階層に属していない。

【０２１８】

・ネストされた要素エクステンションは、アイテムの他のフィールドと共に格納され、グローバルにアクセス可能ではなく、所与のエクステンションタイプのすべてのインスタンスを検索するクエリを作成することはできない。

【０２１９】

・これらのエクステンションは、（そのアイテムの）他のネストされた要素が格納される方法と同じ方法で格納される。他のネストされたセットと同様に、ＮｅｓｔｅｄＥｌｅｍｅｎｔエクステンションはＵＤＴにおいて格納される。これらは、ネストされた要素タイプのＥｘｔｅｎｓｉｏｎｓフィールドを通じてアクセス可能である。

【０２２０】

・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はまた、タイプエクステンションのセットを介してアクセスおよび反復するためにも使用される。

【０２２１】

以下の表は、ＩｔｅｍエクステンションおよびＮｅｓｔｅｄＥｌｅｍｅｎｔエクステンションを要約し比較する。

【０２２２】

【表１２】

ＩｔｅｍエクステンションＮｅｓｔｅｄＥｌｅｍｅｎｔエクステンション

	Itemエクステンション	NestedElement エクステンション
ストレージ	アイテムエクステンション階層はそれ自体のテーブル内に格納される。	ネストされた要素のように格納される。
クエリ/検索	アイテムエクステンションテーブルをクエリすることができる。	一般に、含むアイテムコンテキスト内のみでクエリされることが可能である。
クエリ/検索範囲	アイテムエクステンションタイプのすべてのインスタンスにわたって検索することができる。	一般に、単一の（含む）アイテムのネストされた要素タイプのインスタンス内でのみ検索することができる。
プログラム可能性	特殊なエクステンションAPI、および、エクステンションテーブルにおける特殊なクエリを必要とする。	NestedElementエクステンションは、ネストされた要素の他のあらゆる多値フィールドとも同様であり、通常のネストされた要素タイプ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。	ビヘイビアは許可されない(?)
リレーションシップのセマンティクス	アイテムエクステンションへのリレーションシップはない。	NestedElementエクステンションへのリレーションシップはない。
アイテムID	アイテムのアイテムIDを共有する。	それ自体のアイテムIDを有していない。NestedElementエあクステンションはアイテムの一部でる。

【０２２３】

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で実装される。本発明では、データベースエンジンは、Ｍｉｃｒｏｓｏｆｔ　ＳＱＬ　Ｓｅｒｖｅｒエンジンなど、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、オブジェクトリレーショナルエクステンションと共に備える。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を説明し、本実施形態による、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についての情報を提供する。しかし、異なるデータベースエンジンが使用される場合、異なるマッピングを使用することができることは理解されよう。実際に、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、例えばオブジェクト指向およびＸＭＬデータベースなど、他のタイプのデータベース上でも実装することができる。

【０２２４】

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））のための永続性およびトランザクションを提供する。「アイテム」のストレージプラットフォーム概念は、オブジェクト指向システムの「オブジェクト」に十分にマップするが、埋め込みコレクションはオブジェクトに追加されなければならなくなる。継承およびネストされた要素タイプのように、他のストレージプラットフォームタイプの概念もまた、オブジェクト指向タイプのシステムをマップする。オブジェクト指向システムは通常すでにオブジェクト識別をサポートし、よって、アイテム識別をオブジェクト識別にマップさせることができる。アイテムのビヘイビア（オペレーション）はオブジェクトのメソッドに十分にマップする。しかし、オブジェクト指向システムは通常、編成機能が欠けており、検索において不十分である。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ためのサポートを提供しない。本明細書で説明す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、リレーションシップ、フォルダおよびエクステンションのような概念が、オブジェクトデータモデルに追加されることが必要となる。加えて、プロモーション、同期化、通知およびセキュリティのようなメカニズムが実装されることが必要となる。

【０２２５】

オブジェクト指向システムに類似して、ＸＭＬデータベースは、ＸＳＤ（ＸＭＬスキーマ定義）に基づいて、単一の継承ベースのタイプシステムをサポートする。本発明のアイテムタイプシステムをＸＳＤタイプモデルにマップさせることができる。ＸＳＤはまた、ビヘイビアのためのサポートを提供しない。アイテムのためのＸＳＤは、アイテムビヘイビアにより増補されなければならなくなる。ＸＭＬデータベースは単一のＸＳＤドキュメントを処理し、編成および幅広い検索機能に欠けている。オブジェクト指向データベースと同様に、本明細書で説明するデータモデルをサポートするため、リレーションシップのような他の概念、およびフォルダがこのようなＸＭＬデータベースに組み込まれることが必要となり、また、同期化、通知およびセキュリティのようなメカニズムもまた実装されることが必要となる。

【０２２６】

以下のサブセクションに関して、開示された全体的な情報を実施するために少数の例示が提供され、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共に新しいレコードを同じＢツリーに挿入中である、一実施例を例示する図である。図１５は、データ変更検出プロセスを例示する。図１６は、例示的ディレクトリツリーを例示する。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データストアに移動される、一実施例を示す。

【０２２７】

１．ＵＤＴを使用したデータストア実装

この実施形態では、一実施形態ではＭｉｃｒｏｓｏｆｔ　ＳＱＬ　Ｓｅｒｖｅｒエンジンを備えるリレーショナルデータベースエンジン３１４は、組み込みスカラ型をサポートする。組み込みスカラ型は「ネイティブ」および「単純」である。組み込みスカラ型は、ユーザがそれら自体のタイプを定義することができないという意味でネイティブであり、複合構造をカプセル化することができない点で単純である。ユーザ定義タイプ（以下、ＵＤＴ）は、ユーザが複合の構造化タイプを定義することによってタイプシステムを拡張できるようにすることによって、ネイティブのスカラ型システム以上のタイプ拡張性のためのメカニズムを提供する。ユーザによって定義された後、ＵＤＴを、タイプシステム内で組み込みスカラ型が使用される可能性のあるいかなる所で使用することもできる。

【０２２８】

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クラスにマップされる。データストアアイテムは、Ｂａｓｅ．Ｉｔｅｍタイプから派生するＵＤＴクラスにマップされる。アイテムと同様に、エクステンションもまたＵＤＴクラスにマップされ、継承を使用する。ルートのＥｘｔｅｎｓｉｏｎタイプはＢａｓｅ．Ｅｘｔｅｎｓｉｏｎであり、そこからすべてのＥｘｔｅｎｓｉｏｎタイプが派生される。

【０２２９】
ＵＤＴはＣＬＲクラスであり、状態（すなわち、データフィールド）およびビヘイビア（すなわち、ルーチン）を有する。ＵＤＴは、管理された言語、すなわちＣ＃、ＶＢ．ＮＥＴなどのいずれかを使用して定義される。ＵＤＴメソッドおよびオペレータをＴ－ＳＱＬ内で、そのタイプのインスタンスに対して呼び出すことができる。ＵＤＴを、行内の列のタイプ、Ｔ－ＳＱＬ内のルーチンのパラメータのタイプ、または、Ｔ－ＳＱＬ内の変数のタイプにすることができる。
【０２３０】
ＵＤＴクラスへのストレージプラットフォームスキーマのマッピングは、高レベルで極めて直接的である。一般に、ストレージプラットフォームのスキーマはＣＬＲネームスペースにマップされる。ストレージプラットフォームのタイプはＣＬＲクラスにマップされる。ＣＬＲクラス継承は、ストレージプラットフォームのタイプ継承をミラーリングし、ストレージプラットフォームプロパティはＣＬＲクラスプロパティにマップされる。
【０２３１】
２．アイテムマッピング
アイテムがグローバルに検索可能であることが望ましいこと、および、継承およびタイプ置換可能性のためのこの実施形態のリレーショナルデータベースにおけるサポートが与えられると、データベースストアにおけるアイテム格納のための１つの可能な実施態様は、タイプＢａｓｅ．Ｉｔｅｍの列を有する単一のテーブル内にすべてのアイテムを格納することになる。タイプ置換可能性を使用すると、すべてのタイプのアイテムを格納することができ、検索をアイテムタイプおよびサブタイプによって、Ｙｕｋｏｎの「ｉｓ　ｏｆ（タイプ）」演算子を使用してフィルタリングすることができる。
【０２３２】
しかし、このような手法に関連付けられたオーバーヘッドについての懸念により、この実施形態では、アイテムはトップレベルタイプによって分割され、各タイプ「ファミリ」のアイテムは別々のテーブル内に格納されるようになる。このパーティショニングスキームの下で、テーブルが、Ｂａｓｅ．Ｉｔｅｍから直接継承する各アイテムタイプについて作成される。これらの下で継承するタイプは、上述のようにタイプ置換可能性を使用して適切なタイプファミリテーブル内に格納される。Ｂａｓｅ．Ｉｔｅｍからの継承の最初のレベルのみが特別に扱われる。
【０２３３】
「シャドー」テーブルは、すべてのアイテムのためのグローバルに検索可能なプロパティのコピーを格納するために使用される。このテーブルを、ストレージプラットフォームＡＰＩのＵｐｄａｔｅ（）メソッドによって維持することができ、これを通じてすべてのデータ変更が行われる。タイプファミリテーブルとは異なり、このグローバルアイテムテーブルは、アイテムのトップレベルスカラプロパティのみを含み、完全なＵＤＴアイテムオブジェクトは含まない。グローバルアイテムテーブルは、タイプファミリテーブル内に格納されたアイテムオブジェクトへのナビゲーションを、ＩｔｅｍＩＤおよびＴｙｐｅＩＤをエクスポーズすることによって可能にする。ＩｔｅｍＩＤは一般に、データストア内のアイテムを一意に識別するようになる。ＴｙｐｅＩＤを、ここでは説明しないメタデータを使用して、タイプ名、およびアイテムを含むビューにマップすることができる。アイテムをそのＩｔｅｍＩＤによって発見することは、グローバルアイテムテーブルのコンテキストおよびそうでない場合の両方で、共通のオペレーションである場合があるので、ＧｅｔＩｔｅｍ（）ファンクションが、アイテムのＩｔｅｍＩＤが与えられたアイテムオブジェクトを検索するために提供される。
【０２３４】
好都合なアクセスのため、および、可能な範囲内で実装詳細を隠すために、アイテムのすべてのクエリを、上述のアイテムテーブル上に構築されたビューに対するようにすることができる。具体的には、ビューを各アイテムタイプについて、適切なタイプファミリテーブルに対して作成することができる。これらのタイプビューは、サブタイプを含む、関

連付けられたタイプのすべてのアイテムを選択することができる。便宜上、ＵＤＴオブジェクトに加えて、ビューは、継承されたフィールドを含む、そのタイプのトップレベルフィールドのすべてのための列をエクスポーズすることができる。

【０２３５】

３．エクステンションのマッピング

エクステンションはアイテムに大変類似しており、同じ要件のいくつかを有する。継承をサポートするもう１つのルートタイプとして、エクステンションは、ストレージにおける同じ考慮事項およびトレードオフの多数を受ける。このため、類似のタイプのファミリマッピングは、単一テーブル手法ではなくエクステンションに適用される。言うまでもなく、他の実施形態では、単一テーブル手法を使用することができる。この実施形態では、エクステンションは、ＩｔｅｍＩＤによって厳密に１つのアイテムに関連付けられ、アイテムのコンテキストにおいて一意であるＥｘｔｅｎｓｉｏｎＩＤを含む。アイテムと同様に、その識別が与えられたエクステンションを検索するためにファンクションが提供される場合があり、この識別はＩｔｅｍＩＤおよびＥｘｔｅｎｓｉｏｎＩＤのペアからなる。アイテムタイプビューに類似して、ビューが各エクステンションタイプについて作成される。

【０２３６】

４．ネストされた要素のマッピング

ネストされた要素は、アイテム、エクステンション、リレーションシップ、または他のネストされた要素に埋め込んで、深くネストされた構造を形成することができるタイプである。アイテムおよびエクステンションのように、ネストされた要素はＵＤＴとして実装されるが、アイテムおよびエクステンション内に格納される。したがって、ネストされた要素はそれらのアイテムおよびエクステンションコンテナのストレージマッピングを越えたストレージマッピングは有していない。すなわち、システム内にＮｅｓｔｅｄＥｌｅｍｅｎｔタイプのインスタンスを直接格納するテーブルはなく、ネストされた要素に特に専用のビューはない。

【０２３７】

５．オブジェクト識別

データモデル内の各エンティティ、すなわち、各アイテム、エクステンションおよびリレーションシップは、一意のキー値を有する。アイテムはそのＩｔｅｍＩＤによって一意に識別される。エクステンションは、（ＩｔｅｍＩＤ、ＥｘｔｅｎｓｉｏｎＩｄ）の複合キーによって一意に識別される。リレーションシップは、複合キー（ＩｔｅｍＩＤ、ＲｅｌａｔｉｏｎｓｈｉｐＩｄ）によって識別される。ＩｔｅｍＩｄ、ＥｘｔｅｎｓｉｏｎＩｄおよびＲｅｌａｔｉｏｎｓｈｉｐＩｄは、ＧＵＩＤ値である。

【０２３８】

６．ＳＱＬオブジェクトの名前付け

データストア内で作成されたすべてのオブジェクトを、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スキーマ名において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のベーススキーマ（しばしば「ベース」と呼ばれる）は、「［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．Ｉｔｅｍ」など、「［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］」ＳＱＬスキーマ内でタイプを作成することができる。生成された名前に、修飾子によってプレフィックスが付けられて、名前付けの競合が排除される。適切な場合には、感嘆符文字（！）が名前の各論理部のためのセパレータとして使用される。以下の表は、データストア内のオブジェクトについて使用される名前付け規則を概説する。各スキーマ要素（アイテム、エクステンション、リレーションシップおよびビュー）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た、修飾された名前付け規則と共にリストされる。

【０２３９】

【表１３】

オブジェクト	名前修飾	説明	例
マスタアイテム検索ビュー	Master!Item	現在のアイテムドメイン内のアイテムの概要を提供する。	[System.Storage].[Master!Item]
型付けアイテム検索ビュー	ItemType	アイテムおよびいかなる親タイプからのすべてのプロパティデータを提供する。	[AcmeCorp.Doc].[OfficeDoc]
マスタエクステンション検索ビュー	Master!Extension	現在のアイテムドメイン内のすべてのエクステンションの概要を提供する。	[System.Storage].[Master!Extension]
型付けエクステンション検索ビュー	Extension!extensionType	エクステンションについてのすべてのプロパティデータを提供する。	[AcmeCorp.Doc].[Extension!StickyNote]
マスタリレーションシップビュー	Master!Relationship	現在のアイテムドメイン内のすべてのリレーションシップの概要を提供する。	[System.Storage].[Master!Relationship]
リレーションシップビュー	Relationship!relationshipName	所与のリレーションシップに関連付けられたすべてのデータを提供する。	[AcmeCorp.Doc].[Relationship!AuthorsFromDocument]
ビュー	View!viewName	スキーマビュー定義に基づいた列/タイプを提供する。	[AcmeCorp.Doc].[View!DocumentTitles]

【０２４０】

７．列の名前付け

いかなるオブジェクトモデルをストアにマップする場合にも、名前付けの衝突の可能性は、アプリケーションオブジェクトと共に格納された追加の情報により発生する。名前付けの衝突を回避するために、すべてのタイプ固有でない列（タイプ宣言内で名前付きプロパティに直接マップしない列）には、下線（＿）文字でプレフィックスが付けられる。この実施形態では、下線（＿）文字は、いかなる識別子プロパティの開始文字としても許可されない。さらに、ＣＬＲとデータストアの間の名前付けを単一化するために、ストレージプラットフォームタイプまたはスキーマ要素のすべてのプロパティ（リレーションシップなど）は、最初の文字を大文字で有するべきである。

【０２４１】

８．検索ビュー

ビューは、格納されたコンテンツを検索するためにストレージプラットフォームによって提供される。ＳＱＬビューは、各アイテムおよびエクステンションタイプについて提供される。さらに、ビューは、リレーションシップおよびビュー（データモデルによって定義される）をサポートするために提供される。ストレージプラットフォーム内のすべてのＳＱＬビューおよび下にあるテーブルは、読み取り専用である。データを、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、これを以下でより十分に説明する。

【０２４２】

ストレージプラットフォームスキーマ内で明示的に定義された各ビュー（スキーマデザイナによって定義され、ストレージプラットフォームによって自動的に生成されない）には、名前付きＳＱＬビュー［＜スキーマ名＞］．［Ｖｉｅｗ！＜ビュー名＞］によって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］」を使用してアクセス可能となる。ビューの出力フォーマットはビュー毎にカスタムである（ビューを定義するパーティによって提供された任意のクエリによって定義される）ので、列はスキーマビュー定義に基づいて直接マップされる。

【０２４３】

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

【０２４４】

・ＩｔｅｍＩｄ、ＥｌｅｍｅｎｔＩｄ、ＲｅｌａｔｉｏｎｓｈｉｐＩｄなど、ビュー結果の論理「キー」列。

【０２４５】

・ＴｙｐｅＩｄなど、結果のタイプについてのメタデータ情報。

【０２４６】

・ＣｒｅａｔｅＶｅｒｓｉｏｎ、ＵｐｄａｔｅＶｅｒｓｉｏｎなど、変更追跡列。

【０２４７】

・タイプ固有の列（宣言されたタイプのプロパティ）。

【０２４８】

・タイプ固有のビュー（ファミリビュー）もまた、オブジェクトを戻すオブジェクト列を含む。

【０２４９】

各タイプファミリのメンバは、一連のアイテムビューを使用して検索可能であり、データストア内にアイテムタイプにつき１つのビューがある。図２８は、アイテム検索ビューの概念を例示する図である。

【０２５０】

アイテム

各アイテム検索ビューは、固有のタイプまたはそのサブタイプのアイテムの各インスタンスのための行を含む。例えば、Ｄｏｃｕｍｅｎｔのためのビューは、Ｄｏｃｕｍｅｎｔ、ＬｅｇａｌＤｏｃｕｍｅｎｔおよびＲｅｖｉｅｗＤｏｃｕｍｅｎｔのインスタンスを戻すことができる。この例を考えると、アイテムビューを図２９のように概念化することができる。

【０２５１】

（１）マスタアイテム検索ビュー

ストレージプラットフォームデータストアの各インスタンスは、マスタアイテムビューと呼ばれる特殊アイテムビューを定義する。このビューは、データストア内の各アイテムについての概要情報を提供する。このビューは、アイテムタイププロパティにつき１つの列、アイテムのタイプを記述した列、ならびに、変更追跡および同期情報を提供するために使用されるいくつかの列を提供する。マスタアイテムビューはデータストア内で、名前

「［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．［Ｍａｓｔｅｒ！Ｉｔｅｍ］」を使用して識別される。

【０２５２】

【表１４】

列	タイプ	説明
ItemId	ItemId	アイテムのストレージプラットフォーム識別。
_TypeId	TypeId	アイテムのTypeIdであり、アイテムの正確なタイプを識別し、Metadataカタログを使用してタイプについての情報を検索するために使用することができる。
_RootItemId	ItemId	このアイテムのライフタイムをコントロールする最初の非埋め込み上位のItemId。
〈グローバル変更追跡〉	...	グローバル変更追跡情報。
〈アイテムプロパティ〉	適用なし	アイテムタイププロパティにつき1つの列。

【０２５３】

（２）型付けアイテム検索ビュー

各アイテムタイプはまた検索ビューをも有する。ルートのアイテムビューに類似するが、このビューはまた「_Ｉｔｅｍ」列を介してアイテムオブジェクトへのアクセスをも提供する。各型付けアイテム検索ビューはデータストア内で、名前［ｓｃｈｅｍａＮａｍｅ］．［ｉｔｅｍＴｙｐｅＮａｍｅ］を使用して識別される。例えば、［ＡｃｍｅＣｏｒｐ．Ｄｏｃ］．［ＯｆｆｉｃｅＤｏｃ］である。

【０２５４】

【表１５】

列	タイプ	説明
ItemId	ItemId	アイテムのストレージプラットフォーム識別。
〈タイプ変更追跡〉	...	タイプ変更追跡情報。
〈親プロパティ〉	〈プロパティ固有〉	親プロパティにつき1つの列。
〈アイテムプロパティ〉	〈プロパティ固有〉	このタイプの排他的プロパティにつき1つの列。
_Item	アイテムのCLRタイプ	CLRオブジェクト、宣言されたアイテムのタイプ。

【０２５５】

９．アイテムエクステンション

ＷｉｎＦＳ　Ｓｔｏｒｅにおけるすべてのアイテムエクステンションはまた、検索ビューを使用してもアクセス可能である。

【０２５６】

（１）マスタエクステンション検索ビュー

データストアの各インスタンスは、マスタエクステンションビューと呼ばれる特殊エクステンションビューを定義する。このビューは、データストア内の各エクステンションに

ついての概要情報を提供する。このビューは、エクステンションプロパティにつき１つの列、エクステンションのタイプを記述する列、ならびに、変更追跡および同期情報を提供するために使用されるいくつかの列を有する。マスタエクステンションビューはデータストア内で、名前「［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．［Ｍａｓｔｅｒ！Ｅｘｔｅｎｓｉｏｎ］」を使用して識別される。

【０２５７】

【表１６】

列	タイプ	説明
ItemId	ItemId	このエクステンションが関連付けられるアイテムのストレージプラットフォーム識別。
ExtensionId	ExtensionId (GUID)	このエクステンションインスタンスのId。
_TypeId	TypeId	エクステンションのTypeIdであり、エクステンションの正確なタイプを識別し、Metadataカタログを使用してエクステンションについての情報を検索するために使用することができる。
〈グローバル変更追跡〉	...	グローバル変更追跡情報。
〈エクステンション プロパティ〉	〈プロパティ固有〉	エクステンションタイププロパティにつき1つの列。

【０２５８】

１０．

（１）型付けエクステンション検索ビュー

各エクステンションタイプはまた検索ビューをも有する。マスタエクステンションビューに類似するが、このビューはまた＿Ｅｘｔｅｎｓｉｏｎ列を介してアイテムオブジェクトへのアクセスをも提供する。各型付けエクステンション検索ビューはデータストア内で、名前［ｓｃｈｅｍａＮａｍｅ］．［Ｅｘｔｅｎｓｉｏｎ！ｅｘｔｅｎｓｉｏｎＴｙｐｅＮａｍｅ］を使用して識別される。例えば、［ＡｃｍｅＣｏｒｐ．Ｄｏｃ］．［Ｅｘｔｅｎｓｉｏｎ！ＯｆｆｉｃｅＤｏｃＥｘｔ］である。

【０２５９】

【表１７】

列	タイプ	説明
ItemId	ItemId	このエクステンションが関連付けられるアイテムのストレージプラットフォーム識別。
ExtensionId	ExtensionId (GUID)	このエクステンションインスタンスのId。
〈タイプ変更追跡〉	...	タイプ変更追跡情報。
〈親プロパティ〉	〈プロパティ固有〉	親プロパティにつき1つの列。
〈エクステンションプロパティ〉	〈プロパティ固有〉	このタイプの排他的プロパティにつき1つの列。
_Extension	エクステンションインスタンスのCLRタイプ	CLRオブジェクト、宣言されたエクステンションのタイプ。

【０２６０】

ネストされた要素

すべてのネストされた要素は、アイテム、エクステンションまたはリレーションシップインスタンス内に格納される。したがって、これらの要素は、適切なアイテム、エクステンションまたはリレーションシップ検索ビューをクエリすることによってアクセスされる。

【０２６１】

リレーションシップ

上述のように、リレーションシップは、ストレージプラットフォームデータストア内の複数のアイテムの間でリンクする基本単位を形成する。

【０２６２】

（２）マスタリレーションシップ検索ビュー

各データストアは、マスタリレーションシップビューを提供する。このビューは、データストア内のすべてのリレーションシップインスタンスについての情報を提供する。マスタリレーションシップビューはデータストア内で、名前「［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．［Ｍａｓｔｅｒ！Ｒｅｌａｔｉｏｎｓｈｉｐ］」を使用して識別される。

【０２６３】

【表１８】

列	タイプ	説明
ItemId	ItemId	ソースエンドポイントの識別（ItemId）。
RelationshipId	RelationshipId（GUID）	リレーションシップインスタンスのid。
_RelTypeId	RelationshipTypeId	リレーションシップのRelTypeIdであり、Metadataカタログを使用してリレーションシップインスタンスのタイプを識別する。
<グローバル変更追跡>	...	グローバル変更追跡情報。
TargetItemReference	ItemReference	ターゲットエンドポイントの識別。
_Relationship	Relationship	このインスタンスのためのリレーションシップオブジェクトのインスタンス。

【０２６４】

（３）リレーションシップインスタンス検索ビュー

各宣言されたリレーションシップはまた、特定のリレーションシップのすべてのインスタンスを戻す検索ビューをも有する。マスタリレーションシップビューに類似するが、このビューはまた、リレーションシップデータの各プロパティのための名前付き列をも提供する。各リレーションシップインスタンス検索ビューはデータストア内で、名前［ｓｃｈｅｍａＮａｍｅ］．［Ｒｅｌａｔｉｏｎｓｈｉｐ！ｒｅｌａｔｉｏｎｓｈｉｐＮａｍｅ］を使用して識別される。例えば、［ＡｃｍｅＣｏｒｐ．Ｄｏｃ］．［Ｒｅｌａｔｉｏｎｓｈｉｐ！ＤｏｃｕｍｅｎｔＡｕｔｈｏｒ］である。

【０２６５】

【表１９】

列	タイプ	説明
ItemId	ItemId	ソースエンドポイントの識別（ItemId）。
RelationshipId	RelationshipId（GUID）	リレーションシップインスタンスのid。
〈タイプ変更追跡〉	...	タイプ変更追跡情報。
TargetItemReference	ItemReference	ターゲットエンドポイントの識別。
〈ソース名〉	ItemId	ソースエンドポイント識別の名前付きプロパティ（ItemIdのためのエイリアス）。
〈ターゲット名〉	ItemReferenceまたは派生クラス	ターゲットエンドポイント識別の名前付きプロパティ（TargetItemReferenceのためのエイリアスおよびキャスト）。
〈リレーションシッププロパティ〉	〈プロパティ固有〉	リレーションシップ定義のプロパティにつき1つの列。
_Relationship	リレーションシップインスタンスのCLRタイプ	CLRオブジェクト、宣言されたリレーションシップのタイプ。

【０２６６】

１１．アップデート

ストレージプラットフォーム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、エクステンションまたはリレーションシップ）の新しいインスタンスを作成するため、または、既存のインスタンスをアップデートするために、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、データストアによって定義された単一のストアドプロシージャであり、実行されるアクションを詳述する「オペレーション」を消費する。ＰｒｏｃｅｓｓＵｐｄａｔｅｇｒａｍメソッドは、「ｕｐｄａｔｅｇｒａｍ」として知られる、実行されるアクションのセットを集合的に詳述するオペレーションの順序付きセットを取る、ストアドプロシージャである。

【０２６７】

オペレーションのフォーマットは拡張可能であり、スキーマ要素を介して様々なオペレーションを提供する。いくつかの共通オペレーションには以下が含まれる。

【０２６８】

１．アイテムオペレーション

ａ．ＣｒｅａｔｅＩｔｅｍ（新しいアイテムを、埋め込みまたは保留リレーションシップのコンテキストで作成する）

ｂ．ＵｐｄａｔｅＩｔｅｍ（既存のアイテムをアップデートする）

２．リレーションシップオペレーション

ａ．ＣｒｅａｔｅＲｅｌａｔｉｏｎｓｈｉｐ（参照または保留リレーションシップのインスタンスを作成する）

ｂ．ＵｐｄａｔｅＲｅｌａｔｉｏｎｓｈｉｐ（リレーションシップインスタンスをア

ップデートする）
ｃ．ＤｅｌｅｔｅＲｅｌａｔｉｏｎｓｈｉｐ（リレーションシップインスタンスを除去する）
３．エクステンションオペレーション
ａ．ＣｒｅａｔｅＥｘｔｅｎｓｉｏｎ（エクステンションを既存のアイテムに追加する）
ｂ．ＵｐｄａｔｅＥｘｔｅｎｓｉｏｎ（既存のエクステンションをアップデートする）
ｃ．ＤｅｌｅｔｅＥｘｔｅｎｓｉｏｎ（エクステンションを削除する）

【０２６９】

１２．変更追跡および廃棄（Ｔｏｍｂｓｔｏｎｅｓ）

変更追跡および廃棄サービスはデータストアによって提供され、これを以下でより十分に論じる。このセクションは、データストア内でエクスポーズされた変更追跡情報の概要を提供する。

【０２７０】

変更追跡

データストアによって提供された各検索ビューは、変更追跡情報を提供するために使用された列を含み、これらの列はすべてのアイテム、エクステンションおよびリレーションシップビューにわたって共通である。ストレージプラットフォームのスキーマビューは、スキーマデザイナによって明示的に定義され、変更追跡情報を自動的に提供せず、このような情報は、それにおいてビュー自体が構築される検索ビューを通じて間接的に提供される。

【０２７１】

データストア内の各要素について、変更追跡情報は２つの場所、すなわち、「マスタ」要素ビューおよび「型付け」要素ビューから入手可能である。例えば、ＡｃｍｅＣｏｒｐ．Ｄｏｃｕｍｅｎｔ．Ｄｏｃｕｍｅｎｔアイテムタイプについての変更追跡情報は、マスタアイテムビュー「［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．［Ｍａｓｔｅｒ！Ｉｔｅｍ］」および型付けアイテム検索ビュー［ＡｃｍｅＣｏｒｐ．Ｄｏｃｕｍｅｎｔ］．［Ｄｏｃｕｍｅｎｔ］から入手可能である。

【０２７２】

（１）「マスタ」検索ビューにおける変更追跡

マスタ検索ビューにおける変更追跡情報は、要素の作成およびアップデートバージョンについての情報、どの同期パートナーが要素を作成したか、どの同期パートナーが最後に要素をアップデートしたか、ならびに、作成およびアップデートのための各パートナーからのバージョン番号についての情報を提供する。同期関係におけるパートナー（後述）は、パートナーキーによって識別される。タイプ［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ．Ｓｔｏｒｅ］．ＣｈａｎｇｅＴｒａｃｋｉｎｇＩｎｆｏの_ＣｈａｎｇｅＴｒａｃｋｉｎｇＩｎｆｏという名前の単一のＵＤＴオブジェクトは、すべてのこの情報を含む。タイプはＳｙｓｔｅｍ．Ｓｔｏｒａｇｅスキーマ内で定義される。_ＣｈａｎｇｅＴｒａｃｋｉｎｇＩｎｆｏは、アイテム、エクステンションおよびリレーションシップのためのすべてのグローバル検索ビューで使用可能である。ＣｈａｎｇｅＴｒａｃｋｉｎｇＩｎｆｏのタイプ定義は以下の通りである。

【０２７３】

【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>
```

【０２７４】

これらのプロパティは以下の情報を含む。

【０２７５】

【表２１】

列	説明
_CreationLocalTS	ローカルマシンによる作成タイムスタンプ。
_CreatingPartnerKey	このエンティティを作成したパートナーのPartnerKey。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ーで作成された時間のタイムスタンプ。
_LastUpdateLocalTS	ローカルマシンでのアップデート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アップデートしたパートナーのPartnerKey。エンティティへの最後のアップデートがローカルで行われた場合、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ーでアップデートされた時間のタイムスタンプ。

【０２７６】

（２）「型付け」検索ビューにおける変更追跡

グローバル検索ビューと同じ情報を提供することに加えて、各型付け検索ビューは、同期トポロジにおける各要素の同期状態を記録する追加の情報を提供する。

【０２７７】

【表２２】

列	タイプ	説明
〈グローバル変更追跡〉	...	グローバル変更追跡からの情報。
_ChangeUnitVersions	MultiSet<ChangeUnitVersion>	特定の要素内の変更単位のバージョン番号の説明。
_ElementSyncMetadata	ElementSyncMetadata	同期化ランタイムにとってのみ関心のある、このアイテムについての追加のバージョン独立のメタデータ。
_VersionSyncMetadata	VersionSyncMetadata	同期化ランタイムにとってのみ関心のある、このバージョンについての追加のバージョン固有のメタデータ。

【０２７８】

廃棄

データストアは、アイテム、エクステンションおよびリレーションシップのための廃棄情報を提供する。廃棄ビューは、ライブおよび廃棄されたエンティティ（アイテム、エクステンションおよびリレーションシップ）についての情報を１つの場所で提供する。アイテムおよびエクステンション廃棄ビューは、対応するオブジェクトへのアクセスを提供しないが、リレーションシップ廃棄ビューは、リレーションシップオブジェクトへのアクセスを提供する（廃棄されたリレーションシップの場合、リレーションシップオブジェクトはＮＵＬＬである）。

【０２７９】

（１）アイテム廃棄

アイテム廃棄はシステムから、ビュー［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．［Ｔｏｍｂｓｔｏｎｅ！Ｉｔｅｍ］を介して検索される。

【０２８０】

【表２３】

列	タイプ	説明
ItemId	ItemId	アイテムの識別。
_TypeID	TypeId	アイテムのタイプ。
〈アイテムプロパティ〉	...	すべてのアイテムについて定義されたプロパティ。
_RootItemId	ItemId	このアイテムを含む、最初の非埋め込みアイテムのItemId。
_ChangeTrackingInfo	タイプChangeTrackingInfoのCLRインスタンス	このアイテムについての変更追跡情報。
_IsDeleted	BIT	これはフラグであり、ライブアイテムでは0であり、廃棄されたアイテムでは1である。
_DeletionWallclock	UTCDATETIME	アイテムを削除したパートナーによるUTC壁時計日付時刻。アイテムがライブの場合、NULLである。

【０２８１】
（２）エクステンション廃棄
エクステンション廃棄はシステムから、ビュー［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．［Ｔｏｍｂｓｔｏｎｅ！Ｅｘｔｅｎｓｉｏｎ］を介して検索される。エクステンション変更追跡情報は、ＥｘｔｅｎｓｉｏｎＩｄプロパティを追加して、アイテムについて提供された情報に類似している。
【０２８２】
【表２４】

列	タイプ	説明
ItemId	ItemId	エクステンションを所有するアイテムの識別。
ExtensionId	ExtensionId	エクステンションのExtensionId。
_TypeID	TypeId	エクステンションのタイプ。
_ChangeTrackingInfo	タイプChangeTrackingInfoのCLRインスタンス	このエクステンションについての変更追跡情報。
_IsDeleted	BIT	これはフラグであり、ライブアイテムでは0であり、廃棄されたエクステンションでは1である。
_DeletionWallclock	UTCDATETIME	エクステンションを削除したパートナーによるUTC壁時計日付時刻。エクステンションがライブの場合、NULLである。

【０２８３】
（３）リレーションシップ廃棄
リレーションシップ廃棄はシステムから、ビュー［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．［Ｔｏｍｂｓｔｏｎｅ！Ｒｅｌａｔｉｏｎｓｈｉｐ］を介して検索される。リレーションシップ廃棄情報は、エクステンションについて提供された情報に類似している。しかし、リレーションシップインスタンスのターゲットＩｔｅｍＲｅｆについての追加の情報が提供される。加えて、リレーションシップオブジェクトもまた選択される。
【０２８４】

【表２５】

列	タイプ	説明
ItemId	ItemId	リレーションシップを所有したアイテムの識別（リレーションシップソースエンドポイントの識別）。
RelationshipId	RelationshipId	リレーションシップのRelationshipId。
_TypeID	TypeId	リレーションシップのタイプ。
_ChangeTrackingInfo	タイプChangeTrackingInfoのCLRインスタンス	このリレーションシップについての変更追跡情報。
_IsDeleted	BIT	これはフラグであり、ライブアイテムでは0であり、廃棄されたリレーションシップでは1である。
_DeletionWallclock	UTCDATETIME	リレーションシップを削除したパートナーによるUTC壁時計日付時刻。リレーションシップがライブの場合、NULLである。
_Relationship	リレーションシップのCLRインスタンス	これは、ライブリレーションシップのためのリレーションシップオブジェクトである。廃棄されたリレーションシップではNULLである。
TargetItemReference	ItemReference	ターゲットエンドポイントの識別。

【０２８５】

（４）廃棄クリーンアップ

廃棄情報の限りない増大を防止するため、データストアは廃棄クリーンアップタスクを提供する。このタスクは、廃棄情報を廃棄することができる場合を決定する。このタスクはローカル作成／アップデートバージョンにおける境界を計算し、次いで、すべてのより以前の廃棄バージョンを廃棄することによって廃棄情報を切り捨てる。

【０２８６】

１３．ヘルパーＡＰＩおよびファンクション

ベースマッピングはまたいくつかのヘルパーファンクションをも提供する、これらのファンクションは、データモデル上の共通オペレーションを助けるために供給される。

【０２８７】

```
ファンクション［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．ＧｅｔＩｔｅｍ
／／ＩｔｅｍＩｄが与えられたアイテムオブジェクトを戻す
／／
Item GetItem (ItemId ItemId)
ファンクション［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．ＧｅｔＥｘｔｅｎｓｉｏｎ
／／ＩｔｅｍＩｄおよびＥｘｔｅｎｓｉｏｎＩｄが与えられたエクステンションオブジェクトを戻す
／／
Extension GetExtension (ItemId ItemId, ExtensionId ExtensionId)
ファンクション［Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ］．ＧｅｔＲｅｌａｔｉｏｎｓｈｉｐ
／／ＩｔｅｍＩｄおよびＲｅｌａｔｉｏｎｓｈｉｐＩｄが与えられたリレーションシッ
```

プオブジェクトを戻す
／／
Relationship GetRelationship (ItemId ItemId, RelationshipId RelationshipId)
１４．メタデータ
ストア内に表現された２つのタイプのメタデータ、すなわち、インスタンスメタデータ（アイテムのタイプなど）およびタイプメタデータがある。
【０２８８】
スキーマメタデータ
スキーマメタデータはデータストア内で、メタスキーマからのアイテムタイプのインスタンスとして格納される。
【０２８９】
インスタンスメタデータ
インスタンスメタデータは、アプリケーションによってアイテムのタイプについてクエリするために使用され、アイテムに関連付けられたエクステンションを発見する。アイテムのためのＩｔｅｍＩｄが与えられると、アプリケーションはグローバルアイテムビューをクエリして、アイテムのタイプを戻し、この値を使用してＭｅｔａ．Ｔｙｐｅビューをクエリして、アイテムの宣言されたタイプについての情報を戻す。例えば、以下の通りである。
／／所与のアイテムインスタンスについてのメタデータアイテムオブジェクトを戻す
／／
SELECT m._Item AS metadataInfoObj
FROM [System.Storage].[ITEM] i INNER JOIN [Meta].[Type] m ON i._TypeId = m.ItemId
WHERE i.ItemId = @ItemId
【０２９０】
Ｅ．セキュリティ
一般に、すべての保護可能なオブジェクトはそれらのアクセス権を、図２６に示すアクセスマスクフォーマットを使用して構成する。このフォーマットでは、下位の１６ビットはオブジェクト固有のアクセス権のためであり、次の７ビットは、オブジェクトの大部分のタイプに適用される標準アクセス権のためであり、高位の４ビットは、各オブジェクトが標準およびオブジェクト固有の権利のセットにマップすることができる、汎用アクセス権を規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にアクセスするための権利に対応する。
【０２９１】
図２６のアクセスマスク構造では、アイテム固有の権利は、オブジェクト固有の権利のセクション（下位１６ビット）に配置される。この実施形態では、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ために２つのセットのＡＰＩ、すなわち、Ｗｉｎ３２およびストレージプラットフォームＡＰＩをエクスポーズするので、ファイルシステムオブジェクト固有の権利は、ストレージプラットフォームオブジェクト固有の権利の設計を促すために考慮されなければならない。
【０２９２】
本発明のストレージプラットフォームのためのセキュリティモデルは、本明細書で以前に参照により組み込まれた関連出願で十分に説明される。これについて、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り、新しい等しく保護されたセキュリティ領域が既存のセキュリティ領域から切り開かれることを示す図である。
【０２９３】
Ｆ．通知および変更追跡
本発明のもう１つの態様によれば、ストレージプラットフォームは、アプリケーションがデータ変更を追跡できるようにする通知機能を提供する。この機能は主として、揮発性状態を維持し、あるいはデータ変更イベントにおいてビジネスロジックを実行するアプリ

ケーションのために意図される。アプリケーションは、アイテム、アイテムエクステンションおよびアイテムリレーションシップについての通知のために登録する。通知は、データ変更がコミットされた後、非同期的に配信される。アプリケーションは通知をアイテム、エクステンションおよびリレーションシップタイプ、ならびにオペレーションのタイプによってフィルタリングすることができる。

【０２９４】

一実施形態によれば、ストレージプラットフォームＡＰＩ３２２は通知用の２種類のインターフェースを提供する。第１に、アプリケーションは、アイテム、アイテムエクステンションおよびアイテムリレーションシップへの変更によってトリガされた単純なデータ変更イベントのために登録する。第２に、アプリケーションは、アイテム、アイテムエクステンション、および、アイテムの間のリレーションシップのセットを監視するための「ウォッチャー」オブジェクトを作成する。ウォッチャーオブジェクトの状態を保存し、システム障害の後、または、システムが延長された期間にわたってオフラインになっていた後に再作成することができる。単一の通知は複数のアップデートを反映することができる。

【０２９５】

この機能性についての追加の詳細は、本明細書で以前に参照により組み込まれた関連出願で見ることができる。

【０２９６】

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、少なくともいくつかの実施形態では、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一体部分として実施されるように意図される。例えば、本発明のストレージプラットフォームを、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、オペレーティングシステムの一体部分として実施することができる。その能力において、ストレージプラットフォームＡＰＩは、オペレーティングシステムＡＰＩの一部となり、これを通じてアプリケーションプログラムはオペレーティングシステムと対話する。このように、ストレージプラットフォームは、それを通じてアプリケーションプログラムが情報をオペレーティングシステム上に格納する手段となり、ストレージプラットフォームのアイテムベースのデータモデルはしたがって、このようなオペレーティングシステム従来のファイルシステムに取って代わる。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実施される場合、ストレージプラットフォームは、そのオペレーティングシステムで実装されたＮＴＦＳファイルシステムに取って代わる可能性がある。現在、アプリケーションプログラムはＮＴＦＳファイルシステムのサービスに、Ｗｉｎｄｏｗｓ（登録商標）ファミリのオペレーティングシステムによってエクスポーズされたＷｉｎ３２　ＡＰＩを通じてアクセスする。

【０２９７】

しかし、完全に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の記録が必要となること、および、このような記録は望ましくない場合があることを認識すると、本発明のストレージプラットフォームが、ＮＴＦＳなど既存のファイルシステムとのある相互運用性を提供することが有益となる。本発明の一実施形態では、したがって、ストレージプラットフォームは、Ｗｉｎ３２プログラミングモデルに依拠するアプリケーションプログラムが、ストレージプラットフォームのデータストアならびに従来のＮＴＦＳファイルシステムの両方のコンテンツにアクセスできるようにする。このために、ストレージプラットフォームは、Ｗｉｎ３２名前付け規則のスーパーセットである名前付け規則を使用して、容易な相互運用性を実施する。さらに、ストレージプラットフォームは、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たファイルおよびディレクトリにアクセスすることをサポートする。

【０２９８】

この機能性についての追加の詳細は、本明細書で以前に参照により組み込まれた関連出願で見ることができる。

【０２９９】

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ＡＰＩを備え、ＡＰＩは、アプリケーションプログラムが上述のストレージプラットフォームの特徴および機能にアクセスすること、およびデータストア内に格納されたアイテムにアクセスすることを可能にする。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の一実施形態を説明する。この機能性についての詳細は、本明細書で以前に参照により組み込まれた関連出願で見ることができ、この情報のいくつかを便宜上、以下に要約する。

【０３００】

図１８を参照すると、包含フォルダは、他のアイテムへの保留リレーションシップを含むアイテムであり、ファイルシステムフォルダの共通概念に相当する物である。各アイテムは少なくとも１つの包含フォルダ内に「包含」される。

【０３０１】

図１９は、この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は、ＳＱＬクライアント１９００を使用してローカルデータストア３０２と通信し、またＳＱＬクライアント１９００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）と通信することもできる。ローカルストア３０２もまたリモートデータストア３４０と、ＤＱＰ（分散クエリプロセッサ）を使用して、あるいは、後述のストレージプラットフォーム同期化サービス（「Ｓｙｎｃ」）を通じて通信することができる。ストレージプラットフォームＡＰＩ３２２はまた、データストア通知のためのブリッジＡＰＩとしての機能も果たし、アプリケーションの購読を通知エンジン３３２に渡し、上述にもあるように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ＡＰＩ３２２はまた制限された「プロバイダ」アーキテクチャを定義し、Ｍｉｃｒｏｓｏｆｔ　Ｅｘｃｈａｎｇｅおよびアクティブディレクトリにおけるデータにアクセスできるようにすることもできる。

【０３０２】

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概略的に表現する。ストレージプラットフォームＡＰＩは以下のコンポーネントからなり、すなわち、（１）ストレージプラットフォーム要素およびアイテムタイプを表現するデータクラス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）ＣＬＲクラスをストレージプラットフォームスキーマから生成するために使用されるツール２００８である。

【０３０３】

所与のスキーマから生じるクラスの階層は、そのスキーマ内のタイプの階層を直接反映する。一例として、図２１Ａおよび図２１Ｂに示すようなＣｏｎｔａｃｔｓスキーマ内で定義されたアイテムタイプを考察されたい。

【０３０４】

図２２は、オペレーション中のランタイムフレームワークを例示する図である。ランタイムフレームワークは以下のように動作する。

【０３０５】

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。

【０３０６】

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、これをアプリケーションに戻す。

【０３０７】

３．アプリケーションはＦｉｎｄをこのＩｔｅｍＣｏｎｔｅｘｔ上でサブミットして、

アイテムのコレクションを得る。戻されたコレクションは概念的にオブジェクトグラフ２２０４である（リレーションシップによる）。

【０３０８】

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。

【０３０９】

５．アプリケーションは、Ｕｐｄａｔｅ（）メソッドをコールすることによって、変更を保存する。

【０３１０】

図２３は、「ＦｉｎｄＡｌｌ」オペレーションの実行を例示する。

【０３１１】

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スキーマから生成されるプロセスを例示する。

【０３１２】

図２５は、ファイルＡＰＩがそれに基づくスキーマを例示する。ストレージプラットフォームＡＰＩには、ファイルオブジェクトを処理するためのネームスペースが含まれる。このネームスペースは、Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ．Ｆｉｌｅｓと呼ばれる。Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ．Ｆｉｌｅｓ内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内に格納された情報を直接反映し、この情報はファイルシステムオブジェクトから「プロモート」されるか、あるいは、Ｗｉｎ３２　ＡＰＩを使用してネイティブで作成することができる。Ｓｙｓｔｅｍ．Ｓｔｏｒａｇｅ．Ｆｉｌｅｓネームスペースには２つのクラス、すなわちＦｉｌｅＩｔｅｍおよびＤｉｒｅｃｔｏｒｙＩｔｅｍがある。これらのクラスのメンバおよびそのメソッドを、図２５のスキーマ図を見ることによって容易に推測することができる。ＦｉｌｅＩｔｅｍおよびＤｉｒｅｃｔｏｒｙＩｔｅｍは、ストレージプラットフォームＡＰＩから読み取り専用である。これらを修正するためには、Ｗｉｎ３２　ＡＰＩまたはＳｙｓｔｅｍ．ＩＯ内のクラスを使用しなければならない。

【０３１３】

ＡＰＩについては、プログラミングインターフェース（またはより単純には、インターフェース）を、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って提供された機能性と通信すること、またはその機能性にアクセスすることを可能にするための、いかなるメカニズム、プロセス、プロトコルと見なすこともできる。代替として、プログラミングインターフェースを、他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、ファンクションコール、モジュールなどに通信的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、ファンクションコール、モジュール、オブジェクトなどと見なすことができる。前文の「コードのセグメント」という用語は、適用された専門用語にかかわらず、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にかかわらず、または、コードセグメントがソースとして提供されるか、中間物として提供されるか、オブジェクトコードとして提供されるかにかかわらず、コードセグメトがランタイムシステムで利用されるか、プロセスで使用されるかにかかわらず、これらが同じマシン上に位置するか、異なるマシン上に位置するか、複数のマシンにわたって分散されるかにかかわらず、または、コードのセグメントによって表現された機能性が全体としてソフトウェアで実装されるか、全体としてハードウェアで実装されるか、ハードウェアおよびソフトウェアの組合せで実装されるかにかかわらず、１つまたは複数の命令またはコードの行を含むように意図され、例えば、コードモジュール、オブジェクト、サブルーチン、ファンクションなどを含む。

【０３１４】

概念上、プログラミングインターフェースを一般的に図３０Ａまたは図３０Ｂのように見なすことができる。図３０Ａはインターフェースであるインターフェース１を、それを通じて第１および第２のコードセグメントが通信するコンジットとして例示する。図３０Ｂはインターフェースを、システムの第１および第２のコードセグメントがメディアＭを介して通信することを可能にする、インターフェースオブジェクトＩ１およびＩ２（これ

は第１および第２のコードセグメントの一部であってもそうでなくてもよい）を備えるとして例示する。図３０Ｂの図では、インターフェースオブジェクトＩ１およびＩ２を同じシステムの別々のインターフェースと見なすことができ、また、オブジェクトＩ１およびＩ２にメディアＭを加えた物がインターフェースを備えると見なすこともできる。図３０Ａおよび３０Ｂは、双方向の流れ、および、流れの両側のインターフェースを示すが、ある実施態様は、情報の一方向の流れをのみを有する（または、後述のように情報の流れがない）場合があり、あるいは、一方にのみインターフェースオブジェクトを有する場合がある。例として、限定ではなく、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、ファンクション、サブルーチン、リモートプロシージャコール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。

【０３１５】

このようなプログラミングインターフェースの態様には、それにより第１のコードセグメントが情報（ただし、「情報」はその最も幅広い意味で使用され、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メソッド、それにより第２のコードセグメントが情報を受信するメソッド、および、情報の構造、シーケンス、構文、編成、スキーマ、タイミングおよびコンテンツが含まれる場合がある。これについては、メディアが有線であるか無線であるか、その組合せであるかにかかわらず、インターフェースによって定義された方法で情報がトランスポートされる限り、基礎となるトランスポートメディア自体はインターフェースのオペレーションにとって重要でない場合がある。ある状況では、情報は従来の意味において１方向または両方向で渡されない場合があり、これは情報転送が別のメカニズムを介する場合がある（例えば、複数のコードセグメントの間の情報の流れから分離した、バッファ、ファイルなどに配置された情報）か、または存在しない場合があるからであり、これは、あるコードセグメントが単に、第２のコードセグメントによって実行された機能性にアクセスする場合などである。これらの態様のいずれかまたはすべては、例えば、コードセグメントが疎結合構成においてシステムの一部であるか、密結合構成においてシステムの一部であるかに応じて、所与の状況において重要である場合があり、そのため、このリストは例示的であり非制限的であると見なされるべきである。

【０３１６】

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ための他の方法があり、明示的に除外しない限り、これらもまた、本明細書の最後に示す特許請求の範囲によって包含されるように意図される。このような他の方法は、図３０Ａおよび３０Ｂの単純化したビューよりも高度または複雑に見える場合があるが、これらの他の方法はそれにもかかわらず、類似のファンクションを実行して、同じ全体的な結果を実施する。ここで簡単に、プログラミングインターフェースのいくつかの例示的代替実施態様を説明する。

【０３１７】

ファクタリング：あるコードセグメントから別のコードセグメントへの通信を、この通信を複数の離散的通信に分割することによって間接的に実施することができる。これを図３１Ａおよび３１Ｂに概略的に示す。図のように、いくつかのインターフェースを、分割可能な機能性のセットに関して説明することができる。このように、図３０Ａおよび３０Ｂのインターフェース機能性を、同じ結果を達成するようにファクタリング（因数分解）することができ、ちょうど数学的に２４、または、２掛ける２掛ける３掛ける２を出すことができることと同様である。したがって、図３１Ａに例示するように、インターフェースであるインターフェース１によって提供されたファンクションを細分して、インターフェースの通信を複数のインターフェースであるインターフェース１Ａ、インターフェース１Ｂ、インターフェース１Ｃなどに変換し、同じ結果を達成することができる。図３１Ｂに例示するように、インターフェースＩ１によって提供されたファンクションを、複数の

インターフェースＩ１ａ、Ｉ１ｂ、Ｉ１ｃなどに細分し、同じ結果を達成することができる。同様に、情報を第１のコードセグメントから受信する第２のコードセグメントのインターフェースＩ２を、複数のインターフェースＩ２ａ、Ｉ２ｂ、Ｉ２ｃなどにファクタリングすることができる。ファクタリングする場合、第１のコードセグメントと共に含まれたインターフェースの数は、第２のコードセグメントと共に含まれたインターフェースの数と一致する必要はない。図３１Ａおよび３１Ｂの場合のいずれでも、インターフェースであるインターフェース１およびＩ１の機能的な精神は、それぞれ図３０Ａおよび３０Ｂと同じままで残る。インターフェースのファクタリングはまた、連想、通信および他の数学的プロパティにも従うことができ、ファクタリングを認識することが困難な場合があるようにすることもできる。例えば、オペレーションの順序付けは重要でない場合があり、したがって、あるインターフェースによって実行されるファンクションを、そのインターフェースに達するより前に、コードまたはインターフェースの別の部分によってうまく実行することができ、あるいは、システムの別のコンポーネントによって実行することができる。また、同じ結果を達成する、異なるファンクションコールを行う様々な方法があることは、当業者には理解されよう。

【０３１８】

再定義：いくつかの場合、なお所期の結果を実施しながら、プログラミングインターフェースのあるアスペクト（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である場合がある。これを図３２Ａおよび３２Ｂに例示する。例えば、図３０Ａのインターフェースであるインターフェース１がファンクションコールＳｑｕａｒｅ（入力、精度、出力）を含むと仮定し、これは、３つのパラメータである入力、精度および出力を含むコールであり、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発行される。図３２Ａのように、中央のパラメータである精度が所与のシナリオにおいて関係がない場合、このパラメータをちょうどよく無視することができ、あるいは（この状況では）無意味なパラメータで置き換えることもできる。関係のない追加のパラメータを追加することもできる。いずれの場合にも、入力が第２のコードセグメントによって二乗された後で出力が戻される限り、二乗の機能性を達成することができる。精度は、あるダウンストリームまたはコンピューティングシステムの他の部分にとって有意味パラメータとなる場合が十分にあるが、精度が二乗を計算する狭い目的には必要でないと認識された後、置換または無視される場合がある。例えば、有効な精度の値を渡すのではなく、誕生日などの無意味な値を、結果に悪影響を及ぼすことなく渡すことができる。同様に、図３２Ｂに示すように、インターフェースＩ１はインターフェースＩ１’によって置き換えられ、インターフェースに対してパラメータを無視または追加するように再定義される。インターフェースＩ２は同様にインターフェースＩ２’として再定義され、不要なパラメータ、または他のどこかで処理される可能性のあるパラメータを無視するように再定義される場合がある。ここでのポイントは、いくつかの場合にプログラミングインターフェースには、パラメータなど、ある目的のために必要とされないアスペクトが含まれる場合があり、そのためこれらのアスペクトは無視または再定義されるか、あるいは他のどこかで他の目的のために処理される場合があることである。

【０３１９】

インラインコーディング：２つの別々のコードモジュールの機能性のいくつかまたはすべてをマージして、それらの間の「インターフェース」が形態を変更するようにすることが実行可能である場合もある。例えば、図３０Ａおよび３０Ｂの機能性を、図３３Ａおよび図３３Ｂの機能性にそれぞれ変換することができる。図３３Ａで、図３０Ａの以前の第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、これらのコードセグメントはなお互いに通信中であることが可能であるが、インターフェースを、単一のモジュールにより適切である形態に適合させることができる。したがって、例えば、形式的なＣａｌｌおよびＲｅｔｕｒｎステートメントはもはや必要でない可能性があるが、インターフェースであるインターフェース１による類似の処理または応答はなお実施されることが可能である。同様に、図３３Ｂのように、図３０Ｂ

からのインターフェースＩ２の一部（またはすべて）をインターフェースＩ１にインラインで書き込んで、インターフェースＩ１”を形成することができる。例示のように、インターフェースＩ２はＩ２ａおよびＩ２ｂに分割され、インターフェース部Ｉ２ａはインラインでインターフェースＩ１と共にコーディングされて、インターフェースＩ１”が形成されている。具体的な例では、図３０ＢからのインターフェースＩ１がファンクションコールｓｑｕａｒｅ（入力、出力）を実行し、これがインターフェースＩ２によって受信され、インターフェースＩ２は、入力により渡された値を第２のコードセグメントによって処理（してこれを二乗）した後、二乗された結果を出力により戻すことを考えてみる。このような場合、第２のコードセグメントによって実行された処理（入力を二乗すること）を、第１のコードセグメントによって、インターフェースへのコールなしに実行することができる。

【０３２０】

分離（Ｄｉｖｏｒｃｅ）：あるコードセグメントから別ンコードセグメントの通信を、この通信を複数の離散的通信に分割することによって間接的に実施することができる。これを図３４Ａおよび３４Ｂに概略的に示す。図３４Ａのように、１つまたは複数のミドルウェア（分離インターフェース、機能性および／またはインターフェースファンクションを元のインターフェースから分離させるため）が提供されて、第１のインターフェースであるインターフェース１における通信が変換されて、これらが異なるインターフェース、この場合はインターフェースであるインターフェース２Ａ、インターフェース２Ｂおよびインターフェース２Ｃに適合される。これは例えば、インターフェース１プロトコルに従って、例えばオペレーティングシステムと通信するように設計されたアプリケーションのインストールされたベースがあるところで行われる場合があるが、次いでオペレーティングシステムは、異なるインターフェース、この場合はインターフェースであるインターフェース２Ａ、インターフェース２Ｂおよびインターフェース２Ｃを使用するように変更される。ポイントは、第２のコードセグメントによって使用された元のインターフェースが変更されて、もはや、第１のコードセグメントによって使用されたインターフェースと互換性がないようになり、そのため、古いインターフェースおよび新インターフェースが互換性を有するようにするために中間物が使用されることである。同様に、図３４Ｂのように、第３のコードセグメントを分離インターフェースＤＩ１により導入して、インターフェースＩ１からの通信を受信することができ、分離インターフェースＤＩ２により、インターフェース機能性を、例えば、ＤＩ２と共に動作するように再設計されたインターフェースＩ２ａおよびＩ２ｂへ送信することができるが、同じ機能的結果を提供することができる。同様に、ＤＩ１およびＤＩ２は共に図３０Ｂのインターフェース１およびＩ２の機能性を新しいオペレーティングシステムに変換するように動作することができるが、同じまたは類似の機能的結果を提供することができる。

【０３２１】

書き換え：さらにもう１つの可能な変形形態は、コードを動的に書き換えて、インターフェース機能性を、他の物だが同じ全体的な結果を達成する物で置き換えることである。例えば、中間言語（例えば、Ｍｉｃｒｏｓｏｆｔ　ＩＬ、Ｊａｖａ（登録商標）ＢｙｔｅＣｏｄｅなど）で表現されたコードセグメントが、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タイプの環境によって提供された実行環境など）内で、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提供されるシステムがある場合がある。ＪＩＴコンパイラを、第１のコードセグメントから第２のコードセグメントへの通信を動的に変換するように、すなわち、これらの通信を、第２のコードセグメント（元の、または異なる第２のコードセグメントのいずれか）によって必要とされる可能性のある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。これを図３５Ａおよび３５Ｂに示す。図３５Ａを見るとわかるように、この手法は、上述の分離シナリオに類似している。これは例えば、アプリケーションのインストールされたベースがインターフェース１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るところで行われる場合があるが、次いでオペレーテ

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。ＪＩＴコンパイラを使用して、インストールベースのアプリケーションからのオンザフライの通信を、オペレーティングシステムの新しいインターフェースに適合させることができる。図３５Ｂに示すように、インターフェースを動的に書き換えるこの手法を、インターフェースを動的にファクタリングするように、またはそうでない場合は変更するように適用させることもできる。

【０３２２】

代替実施形態を介してあるインターフェースと同じまたは類似の結果を達成するための上述のシナリオをまた様々な方法で、直列および／または並列に、または他の介入コードと共に組み合わせることもできることに留意されたい。このように、上記で提示された代替実施形態は相互排他的ではなく、これらの代替実施形態を混合、マッチおよび組み合わせて、図３０Ａおよび３０Ｂに提示された汎用シナリオと同じかあるいはそれに相当するシナリオを作成することができる。また、大部分のプログラミング構造と同様に、本明細書で説明されない可能性のあるインターフェースの同じまたは類似の機能性を達成する他の類似の方法があるが、それにもかかわらず、これらの他の類似の方法は、本発明の精神および範囲によって表現されることにも留意されたい。すなわち、インターフェースの価値の基礎となる物は、少なくとも部分的には、インターフェースによって表された機能性、および、インターフェースによって可能にされた有利な結果であることに留意されたい。

【０３２３】

ＩＩＩ．同期ＡＰＩ

同期へのいくつかの手法が、アイテム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可能である。

【０３２４】

Ａ．同期化の概観

本発明のいくつかの実施形態では、また図３については、ストレージプラットフォームは同期化サービス３３０を提供し、このサービスは、（ｉ）ストレージプラットフォームの複数のインスタンスが（それぞれそれ自体のデータストア３０２と共に）柔軟なセットのルールに従ってそれらのコンテンツの部分を同期化することを可能にし、（ｉｉ）サードパーティが本発明のストレージプラットフォームのデータストアを、メーカ独自のプロトコルを実装する他のデータソースと同期化させるためのインフラストラクチャを提供する。

【０３２５】

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の同期化は、「レプリカ」に参加するグループの間で発生する。例えば、図３を参照すると、ストレージプラットフォーム３００のデータストア３０２と別のリモートデータストア３３８の間の同期化を、場合によっては異なるコンピュータシステム上で実行するストレージプラットフォームの別のインスタンスのコントロール下で提供することが望ましい場合がある。このグループの全体のメンバシップは必ずしも、いかなる所与の時間にいかなる所与のレプリカにも知られるとは限らない。

【０３２６】

異なるレプリカは独立して（すなわち、同時に）変更を行うことができる。同期化のプロセスは、他のレプリカによって行われた変更をあらゆるレプリカに認識させることとして定義される。この同期化機能は本質的にマルチマスタ（すなわち、ピアツーピア）である。

【０３２７】

本発明の同期化機能により、レプリカは以下を行うことができる。

【０３２８】

・どの変更を別のレプリカが認識するかを決定すること。

【０３２９】

・このレプリカが認識しない変更についての情報を要求すること。
【０３３０】
・他のレプリカが認識しない変更についての情報を搬送すること。
【０３３１】
・２つの変更が互いに競合する場合を決定すること。
【０３３２】
・変更をローカルで適用すること。
【０３３３】
・競合解決を他のレプリカに搬送して収束を保証すること。
【０３３４】
・競合を、競合解決のための規定されたポリシーに基づいて解決すること。
【０３３５】
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の同期化
本発明のストレージプラットフォームの同期化サービス３３０の主な応用例は、ストレージプラットフォームの複数のインスタンスを（それぞれそれ自体のデータストアと共に）同期化することである。同期化サービスは、ストレージプラットフォームスキーマ（データベースエンジン３１４の下にあるテーブルではない）のレベルで動作する。したがって、例えば、「Ｓｃｏｐｅｓ」は、後述のように同期化を定義するために使用される。
【０３３６】
同期化サービスは「純変化」（ｎｅｔ　ｃｈａｎｇｅｓ）の原則に基づいて動作する。個々のオペレーションを記録および送信すること（トランザクショナルレプリケーションによるなど）ではなく、同期化サービスはこれらのオペレーションの最終結果を送信し、したがってしばしば、複数のオペレーションの結果を単一の結果の変更に統合する。
【０３３７】
同期化サービスは一般に、トランザクション境界を尊重しない。すなわち、２つの変更がストレージプラットフォームデータストアに対して単一のトランザクションにおいて行われる場合、これらの変更がすべての他のレプリカにアトミックに適用される保証はなく、一方は現れるが他方はない場合がある。この原理の例外は、２つの変更が同じアイテムに対して同じトランザクションにおいて行われる場合、これらの変更は他のレプリカにアトミックに送信および適用されるように保証されることである。このように、アイテムは同期化サービスの一貫性単位である。
【０３３８】
同期化（Ｓｙｎｃ）コントロールアプリケーション
いかなるアプリケーションも同期化サービスに接続し、ｓｙｎｃオペレーションを開始することができる。このようなアプリケーションは、同期化を実行するために必要とされたパラメータのすべてを提供する（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を本明細書でＳｙｎｃコントロールアプリケーション（ＳＣＡ）と称する。
【０３３９】
２つのストレージプラットフォームインスタンスを同期化する場合、ｓｙｎｃは一方の側でＳＣＡによって開始される。そのＳＣＡはローカル同期化サービスに通知して、リモートパートナーと同期化する。他方の側では、同期化サービスは、同期化サービスによって発信側マシンから送信されたメッセージによってアウェイクされる。これは、宛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。同期化サービスをスケジュール通りに、またはイベントに応答して実行させることができる。これらの場合、スケジュールを実施する同期化サービスはＳＣＡとなる。
【０３４０】
同期化を可能にするため、２つのステップを取る必要がある。第１に、スキーマデザイナはストレージプラットフォームスキーマに、適切なｓｙｎｃセマンティクスにより注釈を付けなければならない（後述のように変更単位を指定する）。第２に、同期化は、同期

化に参加することになっているストレージプラットフォームのインスタンスを有するマシンのすべてにおいて適切に構成されなければならない（後述の通り）。

【０３４１】

スキーマのアノテーション

同期化サービスの基本的概念は、変更単位の概念である。変更単位は、ストレージプラットフォームによって個別に追跡されるスキーマの最小部分である。あらゆる変更単位について、同期化サービスは、変更単位が前回のｓｙｎｃ以来変化したか、変化しなかったかを判定することができる場合がある。

【０３４２】

変更単位をスキーマにおいて指定することは、いくつかの目的にかなう。最初に、同期化サービスがオンザワイヤでどれだけチャッティ（ｃｈａｔｔｙ）であるかを決定する。変更が変更単位の内側で行われる場合、変更単位の全体が他のレプリカに送信され、これは、同期化サービスが、変更単位のどの部分が変更されたかを知らないからである。第２に、競合検出の細分性を決定する。２つの同時変更（これらの用語を詳細に後続のセクションで定義する）が同じ変更単位に行われる場合、同期化サービスは競合を引き起こし、他方では、同時変更が異なる変更単位に行われる場合、競合は引き起こされず、これらの変更は自動的にマージされる。第３に、システムによって保持されたメタデータの量に強く影響を与える。同期化サービスメタデータの多くは変更単位毎に保持され、したがって、変更単位をより小さくすることで、ｓｙｎｃのオーバーヘッドが増す。

【０３４３】

変更単位の定義には、正しいトレードオフを発見することが必要である。そのため、同期化サービスは、スキーマデザイナがプロセスに参加することを可能にする。

【０３４４】

一実施形態では、同期化サービスは、要素より大きい変更単位をサポートしない。しかし、同期化サービスは、スキーマデザイナが要素より小さい変更単位を規定する、すなわち、１つの要素の複数の属性を別の変更単位にグループ化するための能力をサポートする。その実施形態では、これは以下の構文を使用して実施される。

【０３４５】

【表２６】

```
<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>
```

【０３４６】

Ｓｙｎｃ構成

それらのデータのある部分を同期して保持することを望むストレージプラットフォームパートナーのグループを、ｓｙｎｃコミュニティと称する。コミュニティのメンバは同期した状態であることを望むが、これらのメンバは必ずしもデータをまったく同じ方法で表現するとは限らず、すなわち、ｓｙｎｃパートナーは、それらが同期中であるデータを変換する場合がある。

【０３４７】

ピアツーピアのシナリオでは、ピアがそれらのパートナーのすべてのための変換マッピングを維持することは、非現実的である。その代わりに、同期化サービスは「コミュニティフォルダ」を定義する手法を取る。コミュニティフォルダは、それと共にすべてのコミュニティメンバが同期中である仮想「共有フォルダ」を表す抽象化である。

【０３４８】

この考えは一例によって最良に例示される。Ｊｏｅが自分のいくつかのコンピュータのＭｙ　Ｄｏｃｕｍｅｎｔｓフォルダを同期して保持することを望む場合、Ｊｏｅは、たとえばＪｏｅｓＤｏｃｕｍｅｎｔｓと呼ばれるコミュニティフォルダを定義する。次いで、あらゆるコンピュータ上で、Ｊｏｅは、仮想ＪｏｅｓＤｏｃｕｍｅｎｔｓフォルダとローカルのＭｙ　Ｄｏｃｕｍｅｎｔｓフォルダの間のマッピングを構成する。この点から先は、Ｊｏｅのコンピュータが互いに同期化し、それら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通信する。このように、すべてのＪｏｅのコンピュータは、他のコンピュータが何者であるかを知る必要なしに互いを理解し、コミュニティフォルダはｓｙｎｃコミュニティの共通語となる。

【０３４９】

同期化サービスの構成は３つのステップからなり、すなわち（１）ローカルフォルダと

コミュニティフォルダの間のマッピングを定義すること、（２）何が同期化されるか（例えば、何者と同期化するか、および、どのサブセットが送信され、どれが受信されるべきであるか）を決定するｓｙｎｃプロファイルを定義すること、および（３）どの異なるｓｙｎｃプロファイルが実行するべきであるかについてのスケジュールを定義すること、またはそれらを手動で実行することである。

【０３５０】

（１）コミュニティフォルダーマッピング

コミュニティフォルダマッピングは、ＸＭＬ構成ファイルとして個々のマシン上に格納される。各マッピングは以下のスキーマを有する。

【０３５１】

/mappings/communityFolder

この要素は、このマッピングが対象とするコミュニティフォルダを名前付けする。名前はフォルダの構文規則に従う。

【０３５２】

/mappings/localFolder

この要素は、マッピングが変換される先のローカルフォルダを名前付けする。名前はフォルダの構文規則に従う。フォルダは、マッピングが有効とな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、このマッピング毎の同期化について考慮される。

【０３５３】

/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、およびその逆に変換する方法を定義する。不在または空である場合、変換は実行されない。具体的には、これはＩＤがマップされないことを意味する。この構成は主として、フォルダのキャッシュを作成するために有用である。

【０３５４】

/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、新たに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プ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ランタイムは、アイテムを前後に変換するためにＩＤマッピングを維持するようになる。

【０３５５】

/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、特定されたルートの子にされることを要求する。

【０３５６】

/mappings/runAs

この要素は、誰の権限の下でこのマッピングに対する要求が処理されるかをコントロールする。不在の場合、ｓｅｎｄｅｒが仮定される。

【０３５７】

/mappings/runAs/sender

この要素の存在は、このマッピングへのメッセージの送信者が成りすまされなければならず、要求がその者のクレデンシャルの下で処理されなければならないことを示す。

【０３５８】

（２）プロファイル

Ｓｙｎｃプロファイルは、同期化を開始するために必要とされたパラメータの全体セットである。ＳｙｎｃプロファイルはＳＣＡによってＳｙｎｃランタイムに、ｓｙｎｃを開始するために供給される。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の同期化のためのＳｙｎｃプロファイルは、以下の情報を含む。

【０３５９】

・変更のためのソースおよび宛先としての機能を果たすためのローカルフォルダ。
【０３６０】
・共に同期化するためのリモートフォルダ名であり、このフォルダはリモートパートナーから、上記で定義されたマッピングを介してパブリッシュされなければならない。
【０３６１】
・方向－同期化サービスは送信のみ、受信のみ、および送受信ｓｙｎｃをサポートする。
【０３６２】
・ローカルフィルター何のローカル情報をリモートパートナーに送信するかを選択する。ローカルフォルダを介したストレージプラットフォームクエリとして表現される。
【０３６３】
・リモートフィルター何のリモート情報をリモートパートナーから検索するかを選択し、コミュニティフォルダを介したストレージプラットフォームクエリとして表現される。
【０３６４】
・変換－アイテムをローカルフォーマットへ、およびそれから変換する方法を定義する。
【０３６５】
・ローカルセキュリティーリモートエンドポイントから検索された変更がリモートエンドポイント（成りすまされた）の許可の下で適用されるか、ｓｙｎｃをローカルで開始するユーザの許可の下で適用されるかを規定する。
【０３６６】
・競合解決ポリシー－競合が拒否されるべきか、ログされるべきか、自動的に解決されるべきかを規定し、後者の場合、どの競合リゾルバを使用するか、ならびにそのための構成パラメータを規定する。
【０３６７】
同期化サービスは、Ｓｙｎｃプロファイルの単純な構築を可能にするランタイムＣＬＲクラスを提供する。プロファイルをまた、（しばしばスケジュールと並行して）容易な格納のためにＸＭＬファイルへ、またはそれから直列化することもできる。しかし、ストレージプラットフォーム内に、すべてのプロファイルが格納される標準の場所はなく、ＳＣＡは、それを持続することもなく直ちにプロファイルを自由に構築してよい。ｓｙｎｃを開始するためにローカルマッピングを有する必要はないことに留意されたい。すべてのｓｙｎｃ情報をプロファイル内で規定することができる。しかし、マッピングは、リモート側によって開始されたｓｙｎｃ要求に応答するために必要とされる。
【０３６８】
（３）スケジュール
一実施形態では、同期化サービスはそれ自体のスケジューリングインフラストラクチャを提供しない。その代わりに、同期化サービスは、別のコンポーネントに依拠してこのタスクを実行し、このコンポーネントはＭｉｃｒｏｓｏｆｔ　Ｗｉｎｄｏｗｓ（登録商標）オペレーティングシステムにより使用可能なＷｉｎｄｏｗｓ（登録商標）Ｓｃｈｅｄｕｌｅｒである。同期化サービスにはコマンドラインユーティリティが含まれ、このユーティリティはＳＣＡの機能を果たし、ＸＭＬファイル内に保存されたｓｙｎｃプロファイルに基づいて同期化をトリガする。このユーティリティにより、Ｗｉｎｄｏｗｓ（登録商標）Ｓｃｈｅｄｕｌｅｒを、スケジュール通りに、またはユーザのログオンもしくはログオフなどのイベントに応答して同期化を実行するように構成することが、大変容易になる。
【０３６９】
競合処理
同期化サービスにおける競合処理は３つの段階に分割され、すなわち（１）変更適用時間で発生する競合検出－このステップは、変更を安全に適用することができるかどうかを判定する、（２）自動競合解決およびロギング－このステップ（競合が検出された直後に起こる）中に、自動競合リゾルバ（または「競合ハンドラ」）は、競合を解決することが

できるかどうかを確かめるために調べられ、そうでない場合、競合をオプショナルでログすることができる、および（３）競合検査および解決－このステップは、いくつかの競合がログされている場合に起こり、ｓｙｎｃセッションのコンテキストの外側で発生し、この時間に、ログされた競合を解決してログから除去することができる。本発明の様々な実施形態は競合処理を対象とし、これを以下のセクションＩＶでより詳細に論じる。

【０３７０】

２．非ストレージプラットフォームデータストアへの同期化

本発明のストレージプラットフォームのもう１つの態様によれば、ストレージプラットフォームは、ＩＳＶがＳｙｎｃアダプタを実装するためのアーキテクチャを提供し、Ｓｙｎｃアダプタは、ストレージプラットフォームがＭｉｃｒｏｓｏｆｔ　Ｅｘｃｈａｎｇｅ、ＡＤ、Ｈｏｔｍａｉｌなどのレガシーシステムに同期化できるようにする。Ｓｙｎｃアダプタは、後述のように、同期化サービスによって提供された多数のＳｙｎｃサービスから利益を得る。

【０３７１】

その名前にもかかわらず、Ｓｙｎｃアダプタは、プラグインとしてあるストレージプラットフォームアーキテクチャに実装される必要はない。望まれる場合、「ｓｙｎｃアダプタ」を単に、同期化サービスランタイムインターフェースを利用して変更エニュメレーションおよび適用などのサービスを得る、いかなるアプリケーションにすることもできる。

【０３７２】

他者が所与のバックエンドへの同期化を構成および実行することをより単純にするために、Ｓｙｎｃアダプタライタは、上述のようにＳｙｎｃプロファイルが与えられたｓｙｎｃを実行する、標準のＳｙｎｃアダプタインターフェースをエクスポーズすることが推奨される。プロファイルは構成情報をアダプタに提供し、そのいくつかをアダプタはＳｙｎｃランタイムに渡してランタイムサービスをコントロールする（例えば、同期化するためのフォルダ）。

【０３７３】

Ｓｙｎｃサービス

同期化サービスはいくつかのｓｙｎｃサービスをアダプタライタに提供する。このセクションの残りでは、その上でストレージプラットフォームが「クライアント」として同期化を実行中であるマシン、および、アダプタが「サーバ」として通信中である非ストレージプラットフォームバックエンドに言及することが好都合である。

【０３７４】

（１）変更エニュメレーション

同期化サービスによって維持された変更追跡データに基づいて、変更エニュメレーションは、ｓｙｎｃアダプタが、前回このパートナーとの同期化が試みられて以来、データストアフォルダに発生している変更を、容易にエニュメレートすることを可能にする。

【０３７５】

変更は、「アンカー」の概念に基づいてエニュメレートされ、アンカーは、最後の同期化についての情報を表現する不透明な構造である。アンカーは、先のセクションで説明する、ストレージプラットフォームのナレッジの形態を取る。変更エニュメレーションサービスを利用するＳｙｎｃアダプタは、２つの幅広いカテゴリに入り、すなわち、「格納されたアンカー」を使用するアダプタに対して「供給されたアンカー」を使用するアダプタである。

【０３７６】

この区別は、最後のｓｙｎｃについての情報がどこに、クライアント上に格納されるか、サーバ上に格納されるかに基づく。アダプタがこの情報をクライアント上に格納することはしばしばより容易であり、バックエンドはしばしばこの情報を好都合に格納することはできない。他方では、複数のクライアントが同じバックエンドに同期化する場合、この情報をクライアント上に格納することは非効率的であり、いくつかの場合には不正確であり、これは、あるクライアントを、他のクライアントがすでにサーバにプッシュしている

変更を認識しないようにさせる。アダプタが、サーバに格納されたアンカーを使用することを望む場合、アダプタはこのアンカーを、変更エニュメレーションの時間にストレージプラットフォームに戻すように供給する必要がある。

【０３７７】

ストレージプラットフォームがアンカーを（ローカルまたはリモートストレージのために）維持するために、ストレージプラットフォームは、サーバで適用が成功された変更を認識するようにされる必要がある。これら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。変更エニュメレーション中に、Ｓｙｎｃアダプタは肯定応答インターフェースを使用して、どの変更の適用が成功されたかをレポートする。同期化の終了で、供給されたアンカーを使用するアダプタは新しいアンカー（適用が成功された変更のすべてを組み込む）を読み取り、それらのバックエンドに送信しなければならない。

【０３７８】

しばしば、アダプタは、アダプタ固有のデータを、ストレージプラットフォームデータストアに挿入するアイテムと共に格納する必要がある。このようなデータの共通の例は、リモートＩＤおよびリモートバージョン（タイムスタンプ）である。同期化サービスは、このデータを格納するためのメカニズムを提供し、変更エニュメレーションは、この余分のデータを、戻される変更と共に受信するためのメカニズムを提供する。これにより、大抵の場合、アダプタがデータベースを再クエリする必要はなくなる。

【０３７９】

（２）変更適用

変更適用は、Ｓｙｎｃアダプタが、それらのバックエンドから受信され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を可能にする。アダプタは、ストレージプラットフォームスキーマへの変更を変換すると期待される。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スキーマから生成されるプロセスを例示する。

【０３８０】

変更適用の主なファンクションは、自動的に競合を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のｓｙｎｃの場合のように、競合は２つのオーバーラップする変更が、互いのナレッジなしに行われることとして定義される。アダプタが変更適用を使用する場合、アダプタは、それについて競合検出が実行されるアンカーを特定しなければならない。アダプタのナレッジによってカバーされないオーバーラップするローカル変更が検出される場合、変更適用は競合を引き起こす。変更エニュメレーションに類似して、アダプタは、格納されたアンカーまたは供給されたアンカーを使用することができる。変更適用は、アダプタ固有のメタデータの効率的な格納をサポートする。このようなデータをアダプタによって、適用される変更にアタッチすることができ、同期化サービスによって格納される場合がある。データは次の変更エニュメレーションにおいて戻される場合がある。

【０３８１】

（３）競合解決

セクションＩＶで後述する競合解決メカニズム（ロギングおよび自動解決オプションを含む）も、ｓｙｎｃアダプタにとって使用可能である。Ｓｙｎｃアダプタは、変更を適用する場合、競合解決のためのポリシーを規定することができる。規定される場合、競合を、規定された競合ハンドラ上に渡し、（可能な場合は）解決することができる。競合をまたログすることもできる。ローカル変更をバックグラウンドに適用しようと試みる場合、アダプタが競合を検出する場合があることは可能である。このような場合、アダプタはなお競合をＳｙｎｃランタイム上に渡して、ポリシーに従って解決させることができる。加えて、Ｓｙｎｃアダプタは、同期化サービスによって検出されたいかなる競合も、処理のためにＳｙｎｃアダプタに戻すように送信されることを要求する場合がある。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特に好都合である。

【０３８２】

アダプタ実装
いくつかの「アダプタ」は単に、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であるが、アダプタは標準アダプタインターフェースを実装することが推奨される。これらのインターフェースにより、Ｓｙｎｃコントロールアプリケーションは、アダプタが所与のＳｙｎｃプロファイルに従って同期化を実行することを要求すること、進行中の同期化をキャンセルすること、および、進行中のｓｙｎｃについての進行レポート（完成率）を受信することが可能となる。
【０３８３】
３．セキュリティ
同期化サービスは、可能な限りわずかな物を、ストレージプラットフォームによって実装されたセキュリティモデルに導入するように努める。同期化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。具体的には以下の通りである。
【０３８４】
・データストアアイテムを読むことができるいかなる者も、そのアイテムへの変更をエニュメレートすることができる。
【０３８５】
・データストアアイテムに書き込むことができるいかなる者も、変更をそのアイテムに適用することができる。
【０３８６】
・データストアアイテムを拡張することができるいかなる者も、ｓｙｎｃメタデータをそのアイテムに関連付けることができる。
【０３８７】
同期化サービスは、安全なアンカーシップ情報を維持しない。変更がレプリカＡでユーザＵによって行われ、レプリカＢに転送される場合、変更が最初にＡで（またはＵによって）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に転送する場合、これはＡの権限ではなくＢの権限の下で行われる。これは以下の制限につながる。すなわち、レプリカがそれ自体の変更をアイテムに行うために信頼されない場合、レプリカは他者によって行われた変更を転送することはできない。
【０３８８】
同期化サービスが開始される場合、Ｓｙｎｃコントロールアプリケーションによって行われる。同期化サービスはＳＣＡの識別に成りすまし、すべてのオペレーションをその識別の下で（ローカルおよびリモートで）実行する。例えば、ユーザＵが、ローカル同期化サービスに、ユーザＵが読み取りアクセスを有していないアイテムに対するリモートストレージプラットフォームからの変更を検索させることはできないことを認められたい。
【０３８９】
４．管理可能性
レプリカの分散コミュニティを監視することは複雑な問題である。同期化サービスは「スイープ」アルゴリズムを使用して、レプリカの状況についての情報を収集および配信することができる。スイープアルゴリズムのプロパティは、すべての構成されたレプリカについての情報が最終的に収集されること、および、障害のある（非反応）レプリカが検出されることを保証する。
【０３９０】
このコミュニティ全体のモニタリング情報は、あらゆるレプリカで使用可能にされる。モニタリングツールを、任意に選択されたレプリカで実行して、このモニタリング情報を検査し、管理的決定を行うことができる。いかなる構成変更も、影響を受けるレプリカに直接行われなければならない。
【０３９１】
Ｂ．同期化ＡＰＩ概観
ますます分散されるデジタル世界では、個人およびワークグループはしばしば情報およびデータを様々な異なるデバイスおよびロケーションに格納する。これは、これらの別々

の、しばしば異種のデータストアにおける情報を常に、最小限のユーザ介入により同期化させて保持することができる、データ同期化サービスの開発を促している。

【０３９２】

本発明の同期化プラットフォームは、本明細書のセクションＩＩで説明した豊富なストレージプラットフォーム（別名「ＷｉｎＦＳ」）の一部であり、以下の３つの主な目的に対処する。

【０３９３】

・アプリケーションおよびサービスが、異なる「ＷｉｎＦＳ」ストアの間でデータを効率的に同期化することを可能にすること。

【０３９４】

・開発者が、「ＷｉｎＦＳ」および非「ＷｉｎＦＳ」ストアの間でデータを同期化するための豊富な解決法を構築することを可能にすること。

【０３９５】

・開発者に、同期化ユーザ体験をカスタマイズするための適切なインターフェースを提供すること。

【０３９６】

１．一般的な専門用語

本明細書の以下は、本明細書のこのセクションＩＩＩ．Ｂの後の考察に関する、いくつかのさらに微細な定義および重要な概念である。

【０３９７】

Ｓｙｎｃレプリカ：大部分のアプリケーションは、ＷｉｎＦＳストア内の所与のアイテムのサブセットに対する変更を追跡、エニュメレートおよび同期化することにのみ関心を有する。同期化オペレーションに参加するアイテムのセットは、同期化レプリカと称される。レプリカは、所与のＷｉｎＦＳ包含階層（通常はフォルダアイテムでルートとされる）内に含まれたアイテムに関して定義される。すべての同期化サービスは、所与のレプリカのコンテキスト内で実行される。ＷｉｎＦＳ　Ｓｙｎｃは、レプリカを定義、管理およびクリーンアップするためのメカニズムを提供する。あらゆるレプリカは、所与のＷｉｎＦＳストア内でレプリカを一意に識別するＧＵＩＤ識別子を有する。

【０３９８】

Ｓｙｎｃパートナー：ｓｙｎｃパートナーは、ＷｉｎＦＳアイテム、エクステンションおよびリレーションシップにおける変更に影響を及ぼすことができるエンティティとして定義される。したがって、あらゆるＷｉｎＦＳストアをｓｙｎｃパートナーと称することができる。非ＷｉｎＦＳストアと同期化する場合、外部データソース（ＥＤＳ）もまたｓｙｎｃパートナーと称される。あらゆるパートナーは、それを一意に識別するＧＵＩＤ識別子を有する。

【０３９９】

Ｓｙｎｃコミュニティ：同期コミュニティは、ピアツーピア同期化オペレーションによって同期して保持される複数の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ストア内、異なるＷｉｎＦＳストア内にある場合があり、あるいはそれら自体を非ＷｉｎＦＳストアにおける仮想レプリカとして明示する場合もある。ＷｉｎＦＳ　ｓｙｎｃは、コミュニティのためのいかなる特定のトポロジも規定または命令せず、これは特に、コミュニティ内のｓｙｎｃオペレーションのみがＷｉｎＦＳ　Ｓｙｎｃサービス（ＷｉｎＦＳアダプタ）を通る場合にそうである。同期化アダプタ（以下で定義）はそれら自体のトポロジ制限を導入することができる。

【０４００】

変更追跡、変更単位およびバージョン：あらゆるＷｉｎＦＳストアは、すべてのローカルのＷｉｎＦＳアイテム、エクステンションおよびリレーションシップへの変更を追跡する。変更は、スキーマ内で定義された変更単位細分性のレベルで追跡される。いかなるアイテム、エクステンションおよびリレーションシップタイプのトップレベルフィールドも、スキーマデザイナによって変更単位に細分することができ、最小の細分性は１つのトッ

プレベルフィールドになる。変更追跡のために、あらゆる変更単位にバージョンが割り当てられ、バージョンは、ｓｙｎｃパートナーｉｄおよびバージョン番号のペアである（バージョン番号はパートナー固有の単調増加の数である）。バージョンは、変更がストア内でローカルに発生する場合、または変更が他のレプリカから得られる場合、アップデートされる。

【０４０１】

Ｓｙｎｃナレッジ：ナレッジは、いずれかの時間での所与のｓｙｎｃレプリカの状態を表現し、すなわち、ローカルでも他のレプリカからでも、所与のレプリカが認識するすべての変更についてのメタデータをカプセル化する。ＷｉｎＦＳ　ｓｙｎｃは、ｓｙｎｃレプリカについてのナレッジをｓｙｎｃオペレーションにわたって維持およびアップデートする。留意するべき重要なことは、ナレッジ表現が、ナレッジが格納される特定のレプリカに対してだけでなく、コミュニティ全体に対して解釈されることを可能にすることである。

【０４０２】

Ｓｙｎｃアダプタ：同期化アダプタは、ＳｙｎｃランタイムＡＰＩを通じてＷｉｎＦＳ　Ｓｙｎｃサービスにアクセスする、管理されたコードアプリケーションであり、ＷｉｎＦＳデータの、非ＷｉｎＦＳデータストアへの同期化を可能にする。シナリオの要件に応じて、どのＷｉｎＦＳデータのサブセットおよび何のＷｉｎＦＳデータタイプを同期化するかについては、アダプタ開発者次第である。アダプタはＥＤＳとの通信を担い、ＷｉｎＦＳスキーマとＥＤＳによりサポートされたスキーマとの間で変換し、それ自体の構成およびメタデータを定義および管理する。アダプタは、ＷｉｎＦＳ　ＳｙｎｃアダプタＡＰＩを実装して、ＷｉｎＦＳ　Ｓｙｎｃチームによって提供されたアダプタのための共通の構成およびコントロールインフラストラクチャを活用することが強く推奨される。より詳細については、ＷｉｎＦＳ　ＳｙｎｃアダプタＡＰＩ仕様［ＳＡＤＰ］およびＷｉｎＦＳ　ＳｙｎｃコントローラＡＰＩ［ＳＣＴＲＬ］仕様を参照されたい。

【０４０３】

ＷｉｎＦＳデータを外部の非ＷｉｎＦＳストアに同期化し、ＷｉｎＦＳフォーマットにおけるナレッジを作成または維持することができないアダプタでは、ＷｉｎＦＳ　Ｓｙｎｃは、後続の変更エニュメレーションまたは適用オペレーションのために使用することができるリモートナレッジを得るためのサービスを提供する。バックエンドストアの機能に応じて、アダプタはこのリモートナレッジをバックエンド上またはローカルＷｉｎＦＳストア上に格納することを望むことができる。

【０４０４】

簡単にするために、同期化「レプリカ」は、単一の論理ロケーション内に存在する「ＷｉｎＦＳ」ストア内のデータのセットを表現する構造であるのに対して、非「ＷｉｎＦＳ」ストア上のデータは「データストア」と呼ばれ、一般にアダプタの使用を必要とする。

【０４０５】

リモートナレッジ：所与のｓｙｎｃレプリカは、別のレプリカから変更を得ることを望む場合、それ自体のナレッジをベースラインとして提供し、このベースラインに対して他のレプリカは変更をエニュメレートする。同様に、所与のレプリカは、変更を別のレプリカに送信することを望む場合、それ自体のナレッジをベースラインとして提供し、このベースラインをリモートレプリカによって、競合を検出するために使用することができる。ｓｙｎｃ変更エニュメレーションおよび適用中に提供される他のレプリカについてのこのナレッジは、リモートナレッジと称される。

【０４０６】

２．同期化ＡＰＩプリンシパル

ある実施形態では、同期化ＡＰＩは２つの部分、すなわち、同期化構成ＡＰＩおよび同期化コントローラＡＰＩに分かれる。同期化構成ＡＰＩは、アプリケーションが同期化を構成すること、および、２つのレプリカの間の特定の同期化セッションのためのパラメータを規定することを可能にする。所与の同期化セッションでは、構成パラメータには、同

期化されるアイテムのセット、同期化のタイプ（一方向または双方向）、リモートデータソースについての情報、および競合解決ポリシーが含まれる。同期化コントローラＡＰＩは同期化セッションを開始し、同期化をキャンセルし、進行中の同期化についての進行およびエラー情報を受信する。また、同期化が所定のスケジュール通りに実行される必要がある特定の実施形態では、このようなシステムには、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が含まれる場合がある。

【０４０７】

本発明のいくつかの実施形態は、「ＷｉｎＦＳ」と非「ＷｉｎＦＳ」データソースの間で情報を同期化するための同期化アダプタを使用する。アダプタの例には、「ＷｉｎＦＳ」のｃｏｎｔａｃｔｓフォルダと非ＷｉｎＦＳのｍａｉｌｂｏｘの間でアドレス帳情報を同期化するアダプタが含まれる。これらの例では、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の間のスキーマ変換コードを開発するために、「ＷｉｎＦＳ」同期化プラットフォームによって提供されたサービスにアクセスするために、本明細書で説明する「ＷｉｎＦＳ」同期化コアサービスＡＰＩを使用することができる。加えて、アダプタ開発者は、非「ＷｉｎＦＳ」データソースと変更を通信するためのプロトコルサポートを提供する。同期化アダプタは、同期化コントローラＡＰＩを使用することによって呼び出され、コントロールされ、このＡＰＩを使用して進行およびエラーをレポートする。

【０４０８】

しかし、本発明のある実施形態では、「ＷｉｎＦＳ」データストアを別の「ＷｉｎＦＳ」データストアと同期化する場合、「ＷｉｎＦＳ」と「ＷｉｎＦＳ」の同期化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で統合される場合、同期化アダプタは不要である場合がある。いずれにしても、いくつかのこのような実施形態は、「ＷｉｎＦＳ」と「ＷｉｎＦＳ」、および同期化アダプタ解決のための同期化サービスのセットを提供し、このサービスには以下が含まれる。

【０４０９】

・「ＷｉｎＦＳ」アイテム、エクステンションおよびリレーションシップへの変更の追跡。

【０４１０】

・所与の過去の状態以来の効率的な増分変更エニュメレーションのためのサポート。

【０４１１】

・「ＷｉｎＦＳ」への外部変更の適用。

【０４１２】

・変更適用中の競合処理。

【０４１３】

図３６を参照して、共通データストアの３つのインスタンスおよびそれらを同期化するためのコンポーネントを例示する。第１のシステム３６０２はＷｉｎＦＳデータストア３６１２を有し、ＷｉｎＦＳデータストア３６１２は、ＷｉｎＦＳ－ＷｉｎＦＳ　Ｓｙｎｃサービス３６２２およびコアＳｙｎｃサービス３６２４を備え、コアＳｙｎｃサービス３６２４はＷｉｎＦＳと非ＷｉｎＦＳ同期化用であり、３６４６でＳｙｎｃ　ＡＰＩ３６５２を利用のためにエクスポーズする。第１のシステム３６０２と同様に、第２のシステム３６０４はＷｉｎＦＳデータストア３６１４を有し、ＷｉｎＦＳデータストア３６１４は、ＷｉｎＦＳ－ＷｉｎＦＳ　Ｓｙｎｃサービス３６３２およびコアＳｙｎｃサービス３６３４を備え、コアＳｙｎｃサービス３６３４はＷｉｎＦＳと非ＷｉｎＦＳ同期化用であり、３６４６’でＳｙｎｃ　ＡＰＩ３６５２を利用のためにエクスポーズする。第１のシステム３６０２および第２のシステム３６０４は３６４２で、それらの各ＷｉｎＦＳ－ＷｉｎＦＳ　Ｓｙｎｃサービス３６２２および３６３２を介して同期化する。第３のシステム３６０６はＷｉｎＦＳシステムではなく、ＷｉｎＦＳ　Ｓｙｎｃを使用するためのアプリケーション３６６６を有して、ＷｉｎＦＳレプリカとのｓｙｎｃコミュニティ内のデータソースを維持する。このアプリケーションは、ＷｉｎＦＳ　Ｓｙｎｃ構成／コントロール

サービス３６６４を利用して、ＷｉｎＦＳ－ＷｉｎＦＳ　Ｓｙｎｃｈサービス３６２２を介して（ＷｉｎＦＳ－ＷｉｎＦＳ　Ｓｙｎｃサービス３６２２がそのようにそれ自体をＷｉｎＦＳデータストアとして仮想化することができる場合）、３６４４でＷｉｎＦＳデータストア３６１２と直接インターフェースするか、あるいは、３６４８でＳｙｎｃ　ＡＰＩ３６５２とインターフェースするＳｙｎｃアダプタ３６６２を介して、インターフェースすることができる。

【０４１４】

この図に例示するように、第１のシステム３６０２は、第２のシステム３６０４および第３のシステム３６０６を認識し、これらと直接同期化する。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、それらの変更を直接互いに同期化しないが、その代わりに、１つのシステム上で発生する変更は第１のシステム３６０２を通じて伝搬しなければならない。

【０４１５】

Ｃ．同期化ＡＰＩサービス

本発明のいくつかの実施形態は、２つの基本的サービスである変更エニュメレーションおよび変更適用を備える同期化サービスを対象とする。

【０４１６】

１．変更エニュメレーション

本明細書で以前に論じたように、変更エニュメレーションは、ｓｙｎｃアダプタが、前回このパートナーとの同期化が試みられて以来、データストアフォルダに発生している変更を、同期化サービスによって維持された変更追跡データに基づいて、容易にエニュメレートすることを可能にする。変更エニュメレーションに関して、本発明のいくつかの実施形態は以下を対象とする。

【０４１７】

・特定されたナレッジインスタンスに対する、所与のレプリカにおけるアイテム、エクステンションおよびリレーションシップへの変更の効率的なエニュメレーション。

【０４１８】

・ＷｉｎＦＳスキーマ内で規定された変更単位細分性のレベルでの変更のエニュメレーション。

【０４１９】

・複合アイテムに関する、エニュメレートされた変更のグループ化。複合アイテムは、アイテム、すべてのそのエクステンション、アイテムへのすべての保留リレーションシップ、および、その埋め込みアイテムに対応するすべての複合アイテムからなる。複数のアイテムの間の参照リレーションシップへの変更は別々にエニュメレートされる。

【０４２０】

・変更エニュメレーションにおけるバッチ。バッチの細分性は、複合アイテムまたはリレーションシップ変更である（参照リレーションシップに関して）。

【０４２１】

・変更エニュメレーション中のレプリカにおけるアイテム上のフィルタの仕様、例えば、レプリカは所与のフォルダ内のすべてのアイテムからなるが、この特定の変更エニュメレーションでは、アプリケーションは、ファーストネームが「Ａ」で開始するすべてのＣｏｎｔａｃｔアイテムへの変更のみをエニュメレートすることを望む。（このサポートにはＢ後のマイルストン（ｐｏｓｔ　ｂ－ｍｉｌｅｓｔｏｎｅ）が追加される）。

【０４２２】

・個々の変更単位（または、アイテム、エニュメレーションもしくはリレーションシップ全体）を同期化に失敗したとしてナレッジ内で記録して、これらの変更単位を次回に再エニュメレートさせるようにするための能力と共に、エニュメレートされた変更についてのリモートナレッジを使用すること。

【０４２３】

・変更エニュメレーション中にメタデータを変更と共に戻すことによって、ＷｉｎＦＳ

Ｓｙｎｃメタデータを理解することができる可能性のある高度なアダプタの使用。

【０４２４】

２．変更適用

本明細書で以前に論じたように、変更適用によってＳｙｎｃアダプタが、それらのバックエンドから受信され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を可能にし、これはアダプタがストレージプラットフォームスキーマへの変更を変換すると期待されるからである。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。

【０４２５】

・他のレプリカ（または非ＷｉｎＦＳストア）からの、対応するアップデートによる、ＷｉｎＦＳ変更メタデータへの増分変更の適用。

【０４２６】

・変更単位細分性での変更適用における競合の検出。

【０４２７】

・変更適用における個々の変更単位レベルでの成功、失敗および競合をレポートして、アプリケーション（アダプタおよびｓｙｎｃコントロールアプリケーションを含む）がその情報を進行、エラーおよび状況レポートのために、ならびに、それらのバックエンド状態がある場合にアップデートするために使用することができるようにすること。

【０４２８】

・変更適用中にリモートナレッジをアップデートして、次の変更エニュメレーションオペレーション中に、アプリケーションにより供給された変更の「反映」を防止するようにすること。

【０４２９】

・ＷｉｎＦＳ　Ｓｙｎｃメタデータを変更と共に理解および提供することができる、高度のアダプタの使用。

【０４３０】

３．サンプルコード

以下は、ＦＯＯ　ＳｙｎｃアダプタがＳｙｎｃランタイムと対話することができる方法についてのコードサンプルである（すべてのアダプタ固有のファンクションには、プレフィックスＦＯＯが付けられる）。

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

//レプリカアイテムｉｄおよびリモートパートナーｉｄをプロファイルから得る。
//大部分のアダプタはこの情報をｓｙｎｃプロファイルから得るようになる。

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

//
//上記のようなｓｔｏｒｅｄＫｎｏｗｌｅｄｇｅＩｄを使用して、ストア内に格納されたナレッジをルックアップする。
//
ReplicaKnowledge remoteKnowledge = ...;

//
//ＲｅｐｌｉｃａＳｙｎｃｈｒｏｎｉｚｅｒを開始する。
//
ctx.ReplicaSynchronizer = new ReplicaSynchronizer(replicaItemId,remotePartnerI
d);
```

```
ctx.ReplicaSynchronizer.RemoteKnowledge = remoteKnowledge;
ChangeReader reader = ctx.ReplicaSynchronizer.GetChangeReader();
／／
／／変更をエニュメレートして処理する。
／／
bool bChangesToRead = true;
while (bChangesToRead)
 {
  ChangeCollection<object> changes = null;
  bChangesToRead = reader.ReadChanges(10,out changes);

  foreach (object change in changes)
   {
    ／／エニュメレートされた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マッピングを行う。このためにＣｔｘから追加のオブジェクトを検索する
こともでき、変更がリモートストアに適用された後にアダプタメタデータを修正するこ
もできる。
    ChangeStatus status = FOOProcessAndApplyToRemoteStore(change);

    ／／学習されたナレッジを状況によりアップデートする。
    reader.AcknowledgeChange (changeStatus);
  }
}

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

／／
／／アップデートされたナレッジ、およびある場合はアダプタメタデータを保存する。
／／
ctx.Update();

／／
／／変更適用のためにサンプリングし、最初にリモートナレッジを、前のようなｓｔｏ
ｒｅｄＫｎｏｗｌｅｄｇｅＩｄを使用して初期化する。
／／
remoteKnowledge = ...;

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

／／
／／変更をリモートストアから得る。アダプタは、そのバックエンド固有のメタデータ
をストアから検索することを担う。これをレプリカ上のエクステンションにすることがで
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)
 {
  //アダプタはＩＤマッピングを行うことを担う。
  Guid localID = FOO_MapRemoteId(change);

  //仮にＰｅｒｓｏｎオブジェクトを同期化中であるとする。
  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。
  //この一部としてアダプタは、リモートオブジェクトのためのアイテムレベルのバックエンド固有のメタデータに変更を行うこともできる。
  //
  FOO_TransformRemoteToLocal(remoteChange,person);
 }

ctx.Update();

//
//新しいリモートアンカーを保存する（これをレプリカ上のエクステンションにすることができる）。
//
Foo_SaveRemoteAnchor();

//
//これは、リモートナレッジが同期化されないので、通常のＷｉｎＦＳ　ＡＰＩ保存である。
//
remoteKnowledge = ctx.ReplicaSynchronizer.GetUpdatedRemoteKnowledge();
ctx.Update();
ctx.Close();

//
//アプリケーションステータスコールバックを処理するためのアダプタコールバック。
//
void FOO_OnEntitySaved(object sender,ChangeStatusEventArgs args)
 {
remoteAnchor.AcceptChange(args.ChangeStatus);
 }
```

【０４３１】

４．ＡＰＩ同期化のメソッド

本発明の一実施形態では、ＷｉｎＦＳストアと非ＷｉｎＦＳストアの間の同期化は、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ってエクスポーズされた同期化ＡＰＩを介して実施され、可能である。

【０４３２】

一実施形態では、すべての同期化アダプタは、共通言語ランタイム（ＣＬＲ）により管理されたＡＰＩである同期化アダプタＡＰＩを実装することが必要とされ、これらのアダプタを一貫して配置、初期化およびコントロールすることができるようにする。アダプタＡＰＩは以下を提供する。

【０４３３】

・アダプタをハードウェア／ソフトウェアインターフェースシステム同期化フレームワークに登録するための標準メカニズム。

【０４３４】

・アダプタがそれらの機能、および、アダプタを初期化するために必要とされた構成情報のタイプを宣言するための、標準メカニズム。

【０４３５】

・初期化情報をアダプタに渡すための標準メカニズム。

【０４３６】

・アダプタが進行状況を、同期化を呼び出すアプリケーションに戻すようにレポートするためのメカニズム。

【０４３７】

・同期化中に発生するいかなるエラーをもレポートするためのメカニズム。

【０４３８】

・進行中の同期化オペレーションのキャンセルを要求するためのメカニズム。

【０４３９】

シナリオの要件に応じて、アダプタのための２つの潜在的なプロセスモデルがある。アダプタは、アダプタを呼び出すアプリケーションと同じプロセススペース内で、または、別のプロセス内でそれだけで実行することができる。それ自体の別のプロセス内で実行するには、アダプタはそれ自体のファクトリクラスを定義し、このクラスはアダプタをインスタンス化するために使用される。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内でアダプタのインスタンスを戻すことができ、あるいは、異なるＭｉｃｒｏｓｏｆｔ共通言語ランタイムアプリケーションドメイン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戻すことができる。デフォルトファクトリ実装が提供され、これは同じプロセス内でアダプタをインスタンス化する。実際には、多数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内で実行するようになる。プロセス外のモデルは通常、以下の理由の一方または両方のために必要とされる。

【０４４０】

・セキュリティの目的。アダプタはあるプロセスまたはサービスのプロセススペース内で実行しなければならない。

【０４４１】

・アダプタは、呼び出し側アプリケーションからの要求の処理に加えて、他のソースからの要求、例えば、入力ネットワーク要求を処理しなければならない。

【０４４２】

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、またはその関連付けられたメタデータがどのように交換されるかを認識しない、単純なアダプタを仮定する。この実施形態では、同期化はレプリカによって、それと共にレプリカが同期化することを望むデータソースに関して達成され、これは最初にステップ３７０２で、最後に前記データストアと同期化して以来どの変更が発生しているかを決定することによって行い、次いでレプリカは、この最後の同期化以来発生している増分変更を、その現在の状態情報に基づいて送信し、この現在の状態情報および増分変更はアダプタを介してデータソースへ向けられる。ステップ３７０４で、アダプタは、前のステップで変更データを

レプリカから受信すると、データソースへのできるだけ多くの変更を実施し、どの変更が成功し、どれが失敗するかを追跡し、成功および失敗情報を（レプリカの）ＷｉｎＦＳに戻すように送信する。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、成功および失敗情報をレプリカから受信すると、次いでデータソースについての新しい状態情報を計算し、この情報をそのレプリカによる将来の使用のために格納し、この新しい状態情報をデータソースへ、すなわちアダプタへ、アダプタによる格納および後続の使用のために送信する。

【０４４３】

Ｄ．同期化階層

本明細書で以前に論じたように、各レプリカ（ならびに、データソースおよび／またはアダプタ）は、その変更の増分および順次エニュメレーションを維持し、このような各変更には、対応する増分および順次変更番号が割り当てられる（すなわち、最初の変更は１であり、２番目の変更は２であり、３番目の変更は３である、など）。また、各レプリカは、そのｓｙｎｃコミュニティ内の他の既知のレプリカ（ｓｙｎｃパートナー）についての状態情報も維持し、これは、レプリカがどの変更をこれらの他のレプリカから受信しているかを追跡するためである。第２のレプリカから来た、第１のレプリカに適用された最後の変更の変更番号を知ることによって、第１のレプリカは次いでこの番号をその後に使用して、この最後に適用された変更の番号より大きい変更のみを要求、受信または処理することができる。図３８Ａ～Ｄは、この順次変更エニュメレーション方法を使用して、変更がどのように追跡、エニュメレートおよび同期化されるかを例示する。

【０４４４】

図３８Ａでは、ｓｙｎｃパートナーＡおよびＢは、共通のｓｙｎｃコミュニティ内のレプリカであり、それらの現在状態において示され、この状態は、変更がまだ行われていないので、各レプリカについてゼロの変更番号と同じであり、例えば、各レプリカについてそれぞれＡ０およびＢ０である。（この実施形態では、一意の変更番号が、初期状態を反映するために使用される。）各レプリカは、それ自体の状態を認識し、そのｓｙｎｃパートナーの状態を追跡し、ここで示すようにこの情報をその「ベクトル」内に反映させる（ベクトルは、例示するように、最初にレプリカ自体の状態と、続いてそのパートナーの各々の最後の既知の状態を、最後の同期化またはこの場合は初期化に基づいてリストする）。レプリカＡのための初期ベクトルは「［Ａ０，Ｂ０］」であり、レプリカＢのための初期ベクトルは「［Ｂ０，Ａ０］」であり、これら２つのレプリカは現在完全に同期している。

【０４４５】

図３８Ｂで、レプリカＡは変更を行い、その変更に一意の増分変更番号Ａ１を割り当てる（この変更番号は、レプリカ自体のための一意の識別「Ａ」、ならびに、そのレプリカにおける変更のための一意の増分された番号「１」を備える）。レプリカＢは、他方では、２つの変更を行い、それらの変更に一意の増分変更番号Ｂ１およびＢ２をそれぞれ割り当てる。この時点で、次の同期化に先立って、これらのレプリカは現在同期しておらず、レプリカＡのためのベクトルは現在［Ａ１，Ｂ０］であり、レプリカＢのためのベクトルは［Ｂ２，Ａ０］である（この場合も、既知の最後の変化を反映する）。

【０４４６】

図３８Ｃで、レプリカＡは、レプリカＢにその現在のベクトルを送信して変更を要求すること（ステップ１）によって、レプリカＢと同期化する。レプリカＢは、レプリカＡのベクトルを受信し、変更Ｂ１およびＢ２を両方ともレプリカＡに送信するために必要とする計算を行い、したがってそのように行うために進む（ステップ２）。レプリカＡは、Ｂ１およびＢ２として識別された、レプリカＢの変更（変更単位）を受信し、これらの変更を適用し、それ自体のベクトルを［Ａ１，Ｂ２］にアップデートする（ステップ３）。

【０４４７】

図３８Ｄに例示する代替実施形態では、レプリカＢは、レプリカＡへの正しい変更を計算および送信する（ステップ２）と共に、また、レプリカＡのベクトルに基づいて、レプ

リカＢが有していない変更がレプリカＡに行われていると判定し、したがって、レプリカＢもまたそれ自体のベクトルおよび変更のための要求をレプリカＡに送信する（ステップ２’）。次いで、レプリカＡがレプリカＢの変更を受信し、これらの変更を適用し、それ自体のベクトルを［Ａ１，Ｂ２］にアップデートする場合（ステップ３の間）、レプリカＡはまたその変更のどれをレプリカＢに送信するかを計算し、これらも送信する（ステップ３’）。レプリカＢはこの情報を受信すると、変更を行い、そのベクトルを［Ｂ２，Ａ１］にアップデートする（ステップ４）。

【０４４８】

前述の実施例に関して、競合がいくつか状況で生じる場合があることが可能である。例えば、Ａ１およびＢ２は同じ変更単位に行われている場合があり、あるいは、Ａ１は、Ｂ２が修正中であった同じ変更単位への削除である場合がある。これらの競合のいくつかを、本明細書で以前に論じた競合解決オプションを使用して解決することができるが、ある競合は特に困難な課題を提供し、これらの課題およびそれらの解決法を本明細書で以下に、この実施例を踏まえて論じる。

【０４４９】

１．以前の「範囲外」変更の同期化

本発明のある実施形態では、レプリカの範囲が静的でない場合がある。したがって、レプリカＡは、その範囲内であるアイテムとその範囲内でないアイテムの間の新しい関係を作成する変更により、その範囲を効果的に増すことができる。しかし、範囲外であるアイテムのための変更単位がレプリカＡおよびＢの間で同期化されていない（それらのレプリカのための同期化の範囲外であったため）と仮定すると、同期化の不整合が、その特定のアイテムのためのバージョンパスに関して発生する可能性がある。この問題の解決法は、レプリカＡがレプリカＢに、範囲外のアイテムに行われたすべての変更を、レプリカＡにおいて範囲内アイテムと範囲外アイテムの間の関係を作成する特定の変更と共に送信することである。

【０４５０】

２．親子順序逆転（Ｄｉｓｏｒｄｅｒｉｎｇ）の同期化

本発明のある実施形態では、同期化について、親アイテムが常に子アイテムより前に送信されることが一般原則である（例えば、子であるアイテムＫが、親であるアイテムＪに埋め込まれる場合、アイテムＫを、アイテムＪが送信される前に送信することはできない）。しかし、レプリカＡでは、同期化の間で、アイテムＪおよびＫが変更されるが、子アイテムＫが親アイテムＪよりも低いソート番号を有し（例えば、その識別番号の順次的先行に基づく）、したがって、通常は先に送信されるようになることが可能である。本発明の様々な実施形態における同期化のためのこの問題の１つの解決法は、これらの変更を２つのグループ、すなわちアイテムＫに行われた変更のみを反映するグループ、および、アイテムＪに行われた変更のみを反映する第２のグループに分割し、これらのグループを正しい順序で送信する（すなわち、子アイテムＫのための変更のグループを、親アイテムＪのための変更のグループを送信した後に送信する）ことである。

【０４５１】

３．廃棄伝搬

本明細書で以前に論じたように、廃棄は、同期化のために、削除された変更単位にマークを付けるために使用される。しかし、同期化はｓｙｎｃコミュニティ内の複数のベクトルについて非同期であるので、これらの廃棄はデータプラットフォーム全体にわたって伝搬しなければならない。問題は、廃棄伝搬の責任を負うことなく、レプリカＡはアイテムを作成し、レプリカＢとのｓｙｎｃ中にそのアイテムをレプリカＢに送信することができることである。レプリカＡは次いでこのアイテムを削除する場合があり、レプリカＣとのｓｙｎｃ中に、送信する物がない（アイテムが削除されたので）ためにこのアイテムに関して何も送信しないようになる。次いで、レプリカＢおよびレプリカＣがｓｙｎｃしようと試みる場合、レプリカＣはレプリカＢから、Ｂの上で永続するこのアイテムを受信するようになる。

【０４５２】
本発明の様々な実施形態でのこの問題の解決法は、レプリカＡが、削除されたアイテムに廃棄によりマークを付けることである。次いで、レプリカＡがアイテムを削除する場合、レプリカＣとのｓｙｎｃ中に、レプリカＡは廃棄をレプリカＢに送信する。レプリカＢおよびレプリカＣがｓｙｎｃしようと試みる場合、レプリカＢはこの廃棄も受信し、このアイテムは直ちに完全にｓｙｎｃコミュニティから排除される。
【０４５３】
４．ルート廃棄伝搬
Ｐ１で、アイテムＸが複数の埋め込みアイテムＡ、Ｂ、Ｃ、ＤおよびＥを有する場合、Ｐ１が最初にこれらの子アイテムを、第２に親アイテムＸを、同期化の間で削除する場合（すなわち、６つの変更としてｄｅｌ　Ａ、ｄｅｌ　Ｂ、ｄｅｌ　Ｃ、ｄｅｌ　Ｄ、ｄｅｌ　Ｅおよびｄｅｌ　Ｘ）、興味深いシナリオが生じ、これは、Ｐ１が単に親Ｘを削除した場合（１つの変更）に同じ最終結果が起こっていたであろうからであり、この場合、埋め込みアイテムも自動的に削除されるようになる。これに関して、本発明のいくつかの実施形態は、同期化の上で、Ｘを削除することが実際には６つの別々の削除イベントに相当する物となることを認識することによって、効率を得て、このようにＰ１はＰ２に、Ｘの削除に対応する変更単位のみを送信し、この削除が必然的にＰ２内のＸの埋め込みアイテムに伝搬することを可能にするようになる。
【０４５４】
５．リレーションシップ名のスワッピング
前述のように、リレーションシップは名前を有し、したがって、１つのレプリカ（Ｐ１）が、２つのリレーションシップ（Ｒ１およびＲ２）のための名前を、一時名前要素（Ｘ）の使用を通じてスワップすることが可能であり、すなわち、Ｒ１の名前がＸにコピーされ、Ｒ２の名前が次いでＲ１にコピーされ、次いでＸはＲ２にコピーされ、最後にＸが削除される。しかし、パートナーレプリカ（Ｐ２）は、一時名前要素Ｘについて知らないので、同期化中にエラーが発生するようになり、これは、Ｒ１が新しい名前を有することを認識すると、Ｐ２によりこの名前を変更するための試みは、Ｒ１およびＲ２の両方で同じ名前を使用するためにエラーの結果となるからである。本発明の様々な実施形態でのこの問題の１つの解決法は、Ｐ２が、この同じ名前エラーを受信または認識すると、可能性のある名前スワップシナリオを仮定し、自動的にそれ自体の一時名前要素（Ｙ）を作成することであり、後続の変更が実際に、Ｒ２の名前をＸにおける名前に変更することを含む場合、Ｐ２はスワップを完了する（そうでない場合、このシナリオを通常の競合イベントとして生成する）。
【０４５５】
６．参照リレーションシップ
レプリカＰ１（ＷｉｎＦＳシステム上で実行する）とデータソースＰ２（非ＷｉｎＦＳシステム上で実行中である）の間の同期化では、ダングリングリレーションシップ（ＷｉｎＦＳによってサポートされる）が非ＷｉｎＦＳシステムによってサポートされないコンテキストにおいて、問題が生じる。この問題は、２つのアイテムＡおよびＢがＰ１においてリレーションシップＲを有し、Ｐ１がこれらのアイテムを Ａ（変更単位Ｐ１－２１として）、次いでＲ（変更単位Ｐ１－２２として）、次いでＢ（変更単位Ｐ１－２３として）の順序で作成する場合に生じる。Ｒが作成される（Ｐ１－２２）場合、Ｒはダングリングリレーションシップであり、そのためＰ２がこれらの変更を順番に適用する場合、許されないダングリングリレーションシップエラーの結果となる。本発明のいくつかの実施形態でのこの問題の解決法は、その代わりに、すべての他の変更がＰ１からＰ２に送信された後で、すべての参照リレーションシップ（例えば、Ｒ）が送信されるように、これらの変更を再順序付けすることであり、このようにこの問題は、最初にアイテムＡおよびＢを作成すること、および次いでこれらのアイテムをＲにより互いに関係付けることによって、完全に回避される。
【０４５６】

Ｅ．ＳＹＮＣスキーマの追加の態様
以下は、本発明の様々な実施形態のための同期化スキーマの追加の（またはより特定の）態様である。
【０４５７】
・各レプリカは、データストアの全体からのデータの定義された同期化サブセットであり、複数のインスタンスを有するデータのスライスである。
【０４５８】
・ｓｙｎｃスキーマのルートは、一意のＩＤを有するルートフォルダ（実際にはルートアイテム）を定義するためのベースタイプを有するレプリカであり、一意のＩＤは、それがメンバであるｓｙｎｃコミュニティのためのＩＤであり、どんなフィルタおよび他の要素でも、特定のレプリカのために必要または望ましい。
【０４５９】
・各レプリカの「マッピング」はレプリカ内で維持され、したがって、いかなる特定のレプリカのためのマッピングも、このようなレプリカが知っている他のレプリカに制限される。このマッピングはｓｙｎｃコミュニティ全体のサブセットのみを備える場合があるが、前記レプリカへの変更はなおｓｙｎｃコミュニティ全体に、（いずれかの特定のレプリカが、どの他のレプリカを未知のレプリカと共通して共有中であるかを認識しないとしても）共通して共有されたレプリカを介して伝搬するようになる。
【０４６０】
・ｓｙｎｃスキーマおよびレプリカの使用は、真の分散ピアツーピアのマルチマスタ同期コミュニティを可能にする。また、ｓｙｎｃコミュニティタイプはないが、ｓｙｎｃ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内の値として存在する。
【０４６１】
・あらゆるレプリカは、増分変更エニュメレーションを追跡するため、および、ｓｙｎｃコミュニティ内で知られる他のレプリカについての状態情報を格納するための、それ自体のメタデータを有する。
【０４６２】
・変更単位はそれら自体のメタデータを有し、このメタデータは、パートナーキーに加えてパートナー変更番号を備えるバージョン、各変更単位のためのアイテム／エクステンション／リレーションシップのバージョン管理、レプリカがｓｙｎｃコミュニティから見ている／受信している変更に関するナレッジ、ＧＵＩＤおよびローカルＩＤ構成、ならびに、クリーンアップのために参照リレーションシップ上に格納されたＧＵＩＤを備える。
【０４６３】
ＩＶ．同期化－競合処理
本明細書で前述したように、同期化サービスにおける競合処理は３つの段階に分割され、すなわち（１）変更適用時間で発生する競合検出－このステップは、変更を安全に適用することができるかどうかを判定する、（２）自動競合解決およびロギング－このステップ（競合が検出された直後に起こる）中に、自動競合ハンドラは、競合を解決することができるかどうかを確かめるために調べられ、そうでない場合、競合をオプショナルでログすることができる、および（３）競合検査および解決－このステップは、いくつかの競合がログされている場合に起こり、ｓｙｎｃセッションのコンテキストの外側で発生し、この時間に、ログされた競合を解決することができ、ログから除去することができる。
【０４６４】
Ａ．競合処理スキーマ
本発明の様々な実施形態は特に、ピアツーピア同期化システムで発生する競合のための競合処理（本明細書で上述した同期化システム用など）を対象とする。競合を正確および効率的に処理するための能力は、よい有用性を保持しながらデータ損失を最小限にし、同期化中のユーザ介入の必要性を低減する。本発明のいくつかの実施形態は、以下の競合処理要素のうち１つまたは複数を備える競合処理スキーマを対象とし、これらの要素は（ａ）競合のスキーマ化された表現、（ｂ）競合の検出、（ｃ）永続的ストアへの競合のロギ

ング、（ｄ）柔軟性があり構成可能な競合解決ポリシーによる競合の自動解決、（ｅ）競合をフィルタリングおよび解決するための作成可能および拡張可能な競合ハンドラ、（ｆ）古くなった競合の自動検出および除去、ならびに（ｇ）プログラム的な競合解決である。また、競合処理スキーマとは分離して、これらの各競合処理要素はそれ自体で本発明の追加の実施形態を表す。

【０４６５】

１．競合タイプ

一般に、競合は、同期化オペレーション中にデータを同期化することができないこと（「変更適用障害」）がある場合は常に生じる。これらの障害は様々な理由のために発生する場合があるが、一般に競合を、制約競合およびナレッジ競合の２つのカテゴリに分けることができる。

【０４６６】

ナレッジベースの競合

ナレッジベースの競合は、２つのレプリカが独立した変更を同じ変更単位に行う場合に発生する。２つの変更は、互いのナレッジなしに行われる場合、独立してコールされ、すなわち、第１のバージョンは第２のナレッジによってカバーされず、その逆も同様である。同期化サービスはすべてのこのような競合を、上述のレプリカのナレッジに基づいて自動的に検出し、これらの競合を、本明細書で以下に説明するように処理する。ナレッジ競合のいくつかの特定のタイプには、アップデート－削除、削除－アップデート、およびアップデート－アップデート競合が含まれる（各名前は、ローカルアクションおよびリモートアクションを順番に指し、例えば、アップデート－削除競合は、同じデータに対するローカルアップデートとリモート削除による）。

【０４６７】

競合を変更単位のバージョン履歴におけるフォークとして考えることは有用であることがある。競合が変更単位のライフ内で発生しない場合、そのバージョン履歴は単純なチェーンであり、各変更は前の変更の後に発生する。ナレッジベースの競合の場合、２つの変更は並行して発生し、チェーンを分割させ、バージョンツリーにさせる。

【０４６８】

要約すると、ナレッジ競合は、ナレッジおよびバージョン処理の結果として発生する。ナレッジ競合は、データベース内に格納された情報への競合するバージョンを有する変更が適用される場合、ＷｉｎＦＳ同期化によって作成される。これらの競合は、競合する変更情報、ならびにバージョン情報を含む必要がある。ナレッジ競合のための大部分の要件は、制約競合のための要件でもある。しかし、ナレッジ競合を、ｓｙｎｃバージョンおよびナレッジのみに基づいて検出することができる。

【０４６９】

制約ベースの競合

複数の独立した変更が共に適用される場合、完全性制約に違反する場合がある。例えば、２つのレプリカがファイルを同じ名前で同じディレクトリ内に作成することで、このような競合を発生させる可能性があり、システム内の制約（フォルダ内の一意のアイテム名の実施など）は、このタイプの制約ベースの競合を起こす。

【０４７０】

一般に制約ベースの競合は、まさにナレッジベースの競合の場合のように、２つの独立した変更を含むが、制約ベースの競合が備える変更は、同じ変更単位に影響を与えないが、その代わりに、それらの間に存在する制約により異なる変更単位に影響を与える。制約ベースの競合は単一の変更の結果生じる場合があり、これは、一方が制約を有し、他方が制約を有していない、２つの異なるシステムの間で同期化する場合などである。例えば、システムが、最大ファイル名が８文字の長さであるという制約を有する場合、および、そのシステムが、このような制約を有していない別のシステムからの、ファイルへの変更を受信し、この変更がファイル名を８文字より長くすることである場合、制約競合の結果となる（単一のマシン上の単一の変更から発生した）。

【０４７１】
制約競合の特定のタイプには、限定なしに以下が含まれる。
【０４７２】
・挿入－挿入競合：これは、２つの同期化パートナーがそれぞれ、同じ名前を有するファイルなど、同じ論理識別子を有するオブジェクトを作成する場合に発生する。
【０４７３】
・親なし競合：これは、作成される入力オブジェクトの親が存在しない場合に発生する。一例は、ファイルがその親フォルダの前に受信される場合である。
【０４７４】
・未定義タイプ競合：これは、入力オブジェクトのスキーマがインストールされず、オブジェクトの作成が妨げられる場合に発生する。
【０４７５】
要約すると、制約競合は、様々な理由のために変更を適用する際の障害によって引き起こされる。このような障害は、最終的な収束の結果となる解決の形態において、または、ユーザ対話を通じて最終的な解決のためにログすることができる場合、これらの障害を有意味的に処理することができる場合に、制約競合と称される。レポートされる以外に有意味的に処理することができない障害は単に、変更適用エラーと呼ばれる。ある実施形態では、すべての変更適用障害がエラーとして扱われ、すなわち、認識された制約競合はない。また、ある実施形態では、送信同期化中に発生するすべての競合は無視され、これは、ナレッジ競合が次の受信同期化で再提示されると予想されるからである。（非収束につながる他の障害もまた無視される場合がある。）
【０４７６】
２．競合検出
同期化サービスは、制約違反を変更適用時間に検出し、制約ベースの競合を自動的に引き起こす。制約ベースの競合の解決には通常、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が必要となり、同期化サービスは、それを行うための汎用メカニズムを提供してもしなくてもよい。
【０４７７】
本発明の様々な実施形態では、ローカルのナレッジがリモートバージョンを認識するかどうか、およびその逆であるかどうかをチェックすることによって、競合が変更単位毎に検出される。ナレッジベースの競合では、以下の４つの競合検出シナリオがある。
【０４７８】
１．リモートバージョンを認識するローカルナレッジ、ローカルバージョンを認識しないリモートナレッジ。これは、入力変更が古くなっており、したがって廃棄されることを意味する。
【０４７９】
２．リモートバージョンを認識しないローカルナレッジ、ローカルバージョンを認識するリモートナレッジ。これは、入力変更がローカルバージョンよりも最近であり、したがって受け入れられることを意味する。
【０４８０】
３．リモートバージョンを認識するローカルナレッジ、ローカルバージョンを認識するリモートナレッジ。これは、両方のバージョンが等しい場合にのみ発生する可能性があり、そのため変更は適用されない。
【０４８１】
４．リモートバージョンを認識しないローカルナレッジ、ローカルバージョンを認識しないリモートナレッジ。これは、ローカルおよびリモートバージョンが競合しており、そのため競合が引き起こされることを意味する。
【０４８２】
３．競合処理
競合は、送信または受信同期化のいずれかの間に発生する場合があるが、一方向の同期

化オペレーション内の両方のパートナーが類似（２つの類似に構成されたＷｉｎＦＳストアなど）である場合、これらのシナリオは対称的であり、自動的に競合を同期的に解決すること、または、競合を非同期的解決のためにログすること（自動または手動）によって、受信側で最も容易に処理することができる。

【０４８３】

言うまでもなく、ＷｉｎＦＳ－非ＷｉｎＦＳ同期化における場合など、送信側パートナーが競合を処理する必要のある場合がある状況がある。このような場合、制約競合は、後続の受信同期化において送信側パートナーに戻らない可能性がある。さらに、受信側パートナーは競合ログを有していない場合があり、あるいは、管理を容易にするために送信側の競合ログを使用する必要がある場合がある。このような場合、変更を完全に拒否して、送信側に強制的に競合を解決させることができる（本明細書で以下に論じる）。

【０４８４】

Ｓｙｎｃ開始側は、競合解決をそれらのＳｙｎｃプロファイル内に構成する。同期化サービスは、複数の競合ハンドラを単一のプロファイル内で結合することをサポートする。競合処理メカニズムは拡張可能であるので、複数の競合ハンドラを結合するためのいくつかの方法がある。１つの特定の方法は、競合ハンドラの１つが成功するまで、次々に試行される競合ハンドラのリストを規定することを含む（本明細書で以下に説明する）。もう１つの方法は、競合ハンドラを競合タイプに関連付けることを含み、例えば、アップデート－アップデートのナレッジベースの競合をある競合ハンドラに向けて送り、他のすべての競合をログに向けて送るなどである。

【０４８５】

競合が検出される場合、同期化サービスは３つのアクション、すなわち（１）変更を拒否すること、（２）競合を自動的に解決すること、または（３）競合を競合ログにログすることのうち、１つを取ることができる（ｓｙｎｃ開始側によってＳｙｎｃプロファイル内で選択される）。

【０４８６】

変更の拒否

変更が拒否される場合、同期化サービスは、変更がレプリカに到着しなかったかのように動作し、否定応答を発信側に戻すように送信する。この解決ポリシーは主として、競合のロギングが実行可能でないヘッドレスレプリカ（ファイルサーバなど）において有用である。その代わりに、このようなレプリカはこれらの変更を拒否することによって、他者に強制的に競合を処理させる。

【０４８７】

自動競合解決

自動競合解決は、規定されたポリシーに従って同期的に競合を解決するプロセスである。ＷｉｎＦＳ同期化オペレーションでは、ポリシーを、送信オペレーションおよび受信オペレーションに対して独立して規定することができる。自動競合解決ポリシーは、同期化プロファイルを介して規定される。引き起こされる競合は、プロファイル内で規定されたトップレベルの競合ハンドラに渡される。この競合ハンドラは競合を解決し、ログし、または競合を別の競合ハンドラへ、競合処理パイプラインに沿ったさらなる処理のために渡すことができる。

【０４８８】

図３９Ａは、本発明のいくつかの実施形態のための競合処理パイプラインを例示する。この図では、競合が発生する場合、競合ハンドラリスト（または「リスト」）３９１０は競合アイテム３９０２を受信し、競合をパイプラインの第１のパス上で第１のハンドラ３９１２に渡し、この場合、ハンドラ３９１２はフィルタである。フィルタ３９１２はゲートキーパーであり、競合３９０２を評価し、これを次のハンドラ３９１４に渡すか、あるいはリスト３９１０に戻すように拒否し、次いでリスト３９１０はこれをフィルタ３９１２に戻すように渡し、フィルタ３９１２はこれをパイプライン内の次のパス上の第１のハンドラ３９２２に渡す。競合３９０２が第１のフィルタ３９１２によって、この場合はリ

ゾルバである第２のハンドラ３９１４に渡される場合、競合は、可能である場合はリゾルバ３９１４によって解決され、あるいは可能でない場合は競合は拒否されて第１のハンドラ３９１２に戻され、リスト３９１０に戻され、次いでパイプライン内の次のパス上でリゾルバ３９２２に渡される。次いで競合は、（ａ）パイプライン内のハンドラのうち１つによって解決されるまで、（ｂ）例えばロガー３９３６など、「ロガー」として知られる特殊な競合ハンドラによって、明示的に競合ログへログされるまで（すなわち、競合がフィルタ３９３４を通過する場合）、または（ｃ）完全にパイプラインの外に戻すように渡され、デフォルトにより競合ログ（破線でロガー３９４４として論理的に図示）に送信されるまで、パイプライン中を進行し続ける。

【０４８９】

図３９Ｂは、図３９Ａに例示されたパイプラインの論理的トラバーサルを例示する流れ図である。図３９Ｂで、また図３９Ａをも参照すると、ステップ３９５０で、競合３９０２はパイプラインに、競合ハンドラリスト３９１０で入り、ステップ３９５２で、最初にフィルタ３９１２に送信される。ステップ３９５４で、競合３９０２がこのフィルタ３９１２を通過する場合、ステップ３９５６で、競合３９０２はリゾルバ３９１４に進行し、ステップ３９５８で、リゾルバ３９１４は競合３９０２を解決しようと試みる。成功する場合、ステップ３９９８でプロセスが戻り、そうでない場合はステップ３９６０で、競合はリゾルバ３９２２に進行し、ステップ３９６２で、リゾルバ３９２２は競合３９０２を解決しようと試みる。成功する場合、ステップ３９９８でプロセスが戻り、そうでない場合はステップ３９６４で、競合はリスト３９３２に進行し、そこからステップ３９６６でフィルタ３９３４に進行し、ステップ３９６８で、競合３９０２がこのフィルタ３９３４を通過する場合、ステップ３９７０で、競合３９０２はロガー３９３６によって競合ログ（図示せず）にログされ、ステップ３９９８でプロセスが戻り、そうでない場合はステップ３９７２で、競合３９０２はフィルタ３９３８に送信され、ステップ３９７４で、競合３９０２がこのフィルタ３９３８を通過する場合、ステップ３９７６で競合３９０２はリゾルバ３９４０に進行し、ステップ３９７８で、リゾルバ３９４０は競合３９０２を解決しようと試みる。成功する場合、ステップ３９９８でプロセスが戻り、そうでない場合はステップ３９８０で、競合はリゾルバ３９４２に進行し、ステップ３９８２で、リゾルバ３９４２は競合３９０２を解決しようと試みる。成功する場合、ステップ３９９８でプロセスが戻り、そうでない場合はステップ３９８４で、競合３９０２はロガー３９４４によって競合ログ（図示せず）にログされ、ステップ３９９８でプロセスが戻る。

【０４９０】

図３９Ａおよび３９Ｂに示さないが、連続競合リゾルバのパスもまた構築することができ、競合をあるリゾルバで解決できない場合、その競合が次のリゾルバに渡され、次のリゾルバが次いで競合を解決しようと試みるなどとなることに留意されたい。競合が未解決で残る場合、次のパスに進行するためにその競合がパスを後退してリストに渡されるのは、パスの最後のみである。同様に、リストのためのパスのすべてが使い尽くされており、競合が未解決で残った後、次いでリストは、次のリストに到達するまでそのパスを後退して競合を渡すなどとなる。

【０４９１】

また、パイプラインが必ずしもリストで開始しなければならないとは限らず、それどころか、例えばフィルタなど、いかなるタイプの競合ハンドラで開始することもできることにも留意されたい。しかし、それにもかかわらず、競合がパスを後退してパイプライン内で第１の競合ハンドラに渡され、その競合ハンドラが試行するための追加のパスを有していない場合（すべてのパスが試行されていない競合ハンドラリストの場合のみとなる）、競合はパイプラインの外に通過して、自動的にデフォルトにより競合ログにログされる。

【０４９２】

ＣｏｎｆｌｉｃｔＨａｎｄｌｅｒタイプは、競合ハンドラのためのベースタイプであり、競合ハンドラリスト、競合ログおよび競合フィルタ、ならびに他のタイプの競合ハンドラが含まれる。加えて、同期化サービスはまたいくつかのデフォルト競合ハンドラをも提

供することができ、これらのハンドラには以下が含まれるがそれだけに限定されない。
【０４９３】
・ローカル勝利：ローカルに格納されたデータを、入力データを超える勝者として選択することによって、競合を解決する。
【０４９４】
・リモート勝利：入力データを、ローカルに格納されたデータを超える勝者として選択することによって、競合を解決する。
【０４９５】
・最終書き込み側勝利：ローカル勝利またはリモート勝利のいずれかを変更単位毎に、変更単位のタイムスタンプに基づいて選択する（同期化サービスは一般にクロック値には依拠せず、この競合リゾルバはそのルールの唯一の例外であることに留意されたい）。
【０４９６】
・決定性：すべてのレプリカ上で同じであるが、そうでない場合は有意味であることが保証される方法で勝者を選択し、同期化サービスの一実施形態は、パートナーＩＤの辞書式比較を使用してこの機能を実施することができる。
【０４９７】
たとえば、競合ハンドラは以下のように、アップデート－削除競合に対しては、ローカル勝利の解決が適用されるべきであること、および、他のすべての競合に対しては、最終書き込み側勝利の解決が適用されるべきであることを規定することができる。
【０４９８】
【表２７】

```
<conflictHandlerList xmlns="http://schemas.microsoft.com/winfs.2003/10/conflicts">
 <conflictFilter xmlns="http://schemas.microsoft.com/winfs.2003/10/conflicts">
  <conflictType>UpdateDeleteConflict</conflictType>
  <conflictResolver><ResolutionType>LocalWins</ResolutionType></conflictResolver>
 </conflictFilter>
 <conflictResolver><ResolutionType>LastWriterWins</ResolutionType></conflictResolver>
</conflictHandlerList>
```

【０４９９】
言うまでもなく、競合ハンドラが特定されない場合、または、特定された競合ハンドラのいずれによっても競合が処理されない場合、競合は競合ログに配置される。ある実施形態では、競合ログもまた競合ハンドラである。
【０５００】
本発明の様々な実施形態では、ＩＳＶはそれら自体の競合ハンドラを実装およびインストールすることができる。カスタム競合ハンドラは構成パラメータを受け入れることができるが、このようなパラメータはＳｙｎｃプロファイルの競合解決セクション内でＳＣＡによって規定されなければならない。
【０５０１】
競合リゾルバが競合を処理する場合、（競合する変更の代わりに）実行される必要があるオペレーションのリストをランタイムに戻す。次いで、同期化サービスはこれらのオペレーションを適用し、競合ハンドラが何を検討したかを含むために、適切に調整されたリモートナレッジを有する。
【０５０２】
解決を適用中に、もう１つの競合が検出されることが可能である。このような場合、最初の処理が再開する前に、新しい競合が解決またはログされなければならない。
【０５０３】
競合をアイテムのバージョン履歴内のブランチとして考える場合、競合解決を、２つのブランチを結合して単一のポイントを形成する結合と見なすことができる。したがって、競合解決は、バージョン履歴を非循環有向グラフ（ＤＡＧ）にする。
【０５０４】

競合ロギング

いくつかのレポートされた競合は、自動競合解決を使用して同期的に解決される可能性があるが、他のレポートされた競合は、後のプログラム的解決のためにログされる場合がある。競合ロギングは、競合解決プロセスが非同期的に進行することを可能にし、すなわち、競合は検出される時間に解決される必要はないが、将来の解決のためにログされる場合がある。例えば、競合ビューアアプリケーションは、ユーザがログされた競合を事後に検査して手動で解決することを、可能にすることができる。

【０５０５】

本発明のいくつかの実施形態では、非常に特殊な種類の競合ハンドラは、競合ロガー（またはより単純に「ロガー」）である。同期化サービスは競合を競合ログ内に、タイプＣｏｎｆｌｉｃｔＲｅｃｏｒｄのアイテムとして（または、代替実施形態では、単にタイプＣｏｎｆｌｉｃｔとして）ログする。これらのレコードは、競合内にあるアイテムに戻るように関係付けられる（アイテム自体が削除されてしまうまで）。ある実施形態では、各競合レコードは、競合を引き起こした入力変更、競合のタイプ（例えば、アップデート－アップデート、アップデート－削除、削除－アップデート、挿入－挿入、または制約）、ならびに、入力変更のバージョン、およびそれを送信するレプリカのナレッジを含む。本発明のある代替実施形態では、このような各競合アイテムは、競合する変更データおよびメタデータ、競合の記述、ならびに、変更適用側情報、エンリストメントデータおよびリモートパートナー名などの他のコンテキスト情報を含む。加えて、変更データは、変更を適用するために使用することができるフォーマット内で格納される。さらに、本発明の様々な実施形態では、競合から派生された各タイプは、競合のそのタイプに関する新しいフィールドを追加することができる。例えば、ＩｎｓｅｒｔＩｎｓｅｒｔＣｏｎｆｌｉｃｔタイプは、一意性制約が違反されることを引き起こしたアイテムのアイテムＩＤを追加する。

【０５０６】

本発明のいくつかの実施形態では、ログされる競合アイテムはまたターゲットアイテムのコピーを、競合アイテムへのエクステンションとして、または単に、それと競合アイテム自体の間で定義されたリレーションシップと共に同じく競合ログ内に格納された別のアイテムとして、あるいは代替として、競合アイテム自体の一部（プロパティ値のペアのセットなど）として含むようにもなる。競合ログ（永続的データストア上で保持される）内の競合アイテムの一部として、あるいはそれと共に格納されるこのターゲットアイテムは、まず第１に競合を起こした特定の変更を反映するようになる。図４０は、一実施例のｃｏｎｔａｃｔアイテムを使用したこの手法を例示するブロック図である。この実施例では、ｃｏｎｔａｃｔアイテム４００２（「ターゲットアイテム」）は名前フィールド４００４を備え、このフィールドは、最後に同期化が成功した時点で最初に「Ｊｏｈｎ」に設定される。このフィールド４００４は次いで、ローカルシステムによってローカルで「Ｂｏｂ」に変更される。後続の同期化中に、この同じフィールド４００４を「Ｊａｎｅ」に変更しようとする試みは競合の結果となり、これはローカルシステムが、「Ｂｏｂ」または「Ｊａｎｅ」のどちらの名前変更が適用されるべきであるかを確認することができないからであり、次いで、ローカル変更（「Ｂｏｂ」）が保持され、競合４００６は競合ログ４００８内に、競合につながった変更（「Ｊａｎｅ」）の適用を反映するｃｏｎｔａｃｔアイテムのコピー４００２’と共にログされる。このように、競合ログは、競合を起こした完全ターゲットアイテムを備え、この特定のターゲットアイテムは、競合につながったアイテムにおいて行われるように試みられた変更を反映するように、アップデートされる。

【０５０７】

競合を競合ログに追加するために、ログは最初に検索されて、同じ変更単位上に他の競合があるかどうかが判定される。同じ変更単位上にいずれかの既存の競合がある場合、これらの競合は可能な除去のために検査される。既存の競合は、その変更認識が新しい競合の変更認識によって包含される場合、除去される。他方では、新しい競合の変更認識が既存のログされた競合の変更認識によって包含される場合、新しい競合が廃棄され、逆もま

た同じである（すなわち、競合は、その認識がストアの認識によって包含される場合に古くされ、これは、その認識が競合の認識を包含する変更をストアが受信し、適用に成功する場合などである）。２つの変更認識のいずれもが他方を包含しない第３の場合、新しい競合はログに追加され、同じ変更単位に対応する両方の競合は、後に手動または自動で解決されるまで、ログ内に存在する。

【０５０８】

競合の検査および解決

同期化サービスは、アプリケーションが競合ログを検査し、その中の競合の解決を提案するためのＡＰＩを提供する。このＡＰＩは、アプリケーションがすべての競合または所与のアイテムに関係付けられた競合をエニュメレートすることを可能にする。このＡＰＩはまた、このようなアプリケーションが、ログされた競合を以下の３つの方法のうち１つで解決することも可能にし、すなわち（１）リモート勝利－ログされた変更を受け入れ、競合するローカル変更を上書きすること、（２）ローカル勝利－ログされた変更の競合部分を無視すること、および（３）新しい変更の提案－アプリケーションが、その意見において、競合を解決するマージを提案することである。競合がアプリケーションによって解決された後、同期化サービスは競合をログから除去する。

【０５０９】

レプリカの収束および競合解決の伝搬

複雑な同期化シナリオでは、同じ競合が複数のレプリカで検出される可能性がある。これが起こる場合、いくつかのことが発生する可能性があり、すなわち（１）競合をあるレプリカ上で解決することができ、この解決を他のレプリカに送信することができる、（２）競合が両方のレプリカ上で自動的に解決される、または（３）競合が両方のレプリカ上で手動で解決される（競合検査ＡＰＩを通じて）。

【０５１０】

収束を保証するため、同期化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到着する場合、同期化サービスは自動的に、ログ内でこのアップデートによって解決されるいかなる競合レコードをも発見し、これらの競合レコードを排除する。この意味で、１つのレプリカでの競合解決は、他のすべてのレプリカにおいてバインディングである。

【０５１１】

異なる勝者が異なるレプリカによって同じ競合に対して選択される場合、同期化サービスは競合解決をバインディングする原理を適用し、２つの解決のうち一方を、他の解決を超えて勝利するように自動的に選択する。勝者は、常に同じ結果を生じることが保証される決定的な方法で選択される（一実施形態はレプリカＩＤの辞書式比較を使用する）。

【０５１２】

異なる「新しい変更」が異なるレプリカによって同じ競合に対して提案される場合、同期化サービスはこの新しい競合を特殊な競合として扱い、競合ロガーを使用して、競合が他のレプリカに伝搬しないようにする。このような状況は一般に、手動の競合解決により起こる。

【０５１３】

Ｂ．競合処理スキーマの追加の態様

以下は、本発明の様々な実施形態のための競合処理スキーマの追加（またはより特定）の態様である。

【０５１４】

・競合解決ポリシーは各レプリカ（およびアダプタ／データソースの組合せ）によって個別に処理され、すなわち、各レプリカは競合をそれ自体の基準および競合解決スキーマに基づいて解決することができる。また、データストアの各インスタンスにおける違いが追加の将来の競合の結果となり、これにつながる可能性があるが、アップデートされた状態情報において反映される競合の増分および順次エニュメレーションは、アップデートされた状態情報を受信する他のレプリカにとって不可視である。

【０５１５】
・ｓｙｎｃスキーマには、すべてのレプリカにとって使用可能な複数の事前定義された競合ハンドラ、ならびに、ユーザ／開発者により定義されたカスタム競合ハンドラのための能力の両方が含まれる。スキーマはまた３つの特殊な「競合ハンドラ」をも備えることができ、すなわち（ａ）例えば（ｉ）同じ変更単位が２つの場所で変更した場合の処理方法、（ｉｉ）変更単位が１つの場所で変更されるが別の場所で削除される場合の処理方法、および（ｉｉｉ）２つの異なる変更単位が同じ名前を２つの異なるロケーション内で有する場合の処理方法に基づいて、異なる競合を異なる方法で解決する競合「フィルタ」、（ｂ）リストの各要素が、競合がうまく解決されるまで順番に試みるための一連のアクションを規定する、競合「ハンドラリスト」、および（ｃ）競合を追跡するが、ユーザ介入なしにさらなるアクションを取らない、「何もしない」ログである。

【０５１６】
Ｖ．結論
前述が例示するように、本発明は、データを編成、検索および共有する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てデータストレージの概念を拡張および拡大し、リレーショナル（テーブル）データ、ＸＭＬ、および、アイテムと呼ばれる新しい形式のデータなど、構造化、非構造化、または半構造化データを含む、すべてのタイプのデータのためのストアとなるように設計される。その共通ストレージ基盤およびスキーマ化されたデータを通じて、本発明のストレージプラットフォームは、消費者、知識労働者および企業のためのより効率的なアプリケーション開発を可能にする。本発明のストレージプラットフォームは、そのデータモデル内に固有の機能を使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の包含および拡張をも行う、豊富で拡張可能なアプリケーションプログラミングインターフェースを提供する。その幅広い発明の概念から逸脱することなく、上述の実施形態に変更を行うことができることを理解されたい。したがって、本発明は、開示された特定の実施形態に限定されないが、付属の特許請求の範囲によって定義されるような本発明の精神および範囲内であるすべての修正を包含するように意図される。

【０５１７】
上記から明らかであるように、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を、プログラムコード（すなわち、命令）の形態において実施することができる。このプログラムコードをコンピュータ可読メディア上に格納することができ、コンピュータ可読メディアは、磁気、電気、または光ストレージメディアなどであり、限定なしに、フロピーディスク、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、または他のいかなるマシン可読ストレージメディアもが含まれ、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によってロードおよび実行される場合、マシンは本発明を実施するための装置となる。本発明はまた、電気配線またはケーブリングを介して、光ファイバーを通じて、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を介して、あるいは、他のいかなる形態の伝送をも介してなど、ある伝送メディアを介して送信されるプログラムコードの形態においても実施することができ、プログラムコードが、コンピュータなどのマシンによって受信され、ロードされ、実行される場合、マシンは本発明を実施するための装置となる。汎用プロセッサ上で実装される場合、プログラムコードはプロセッサと結合して、特定の論理回路と同じように動作する一意の装置を提供する。

【図面の簡単な説明】

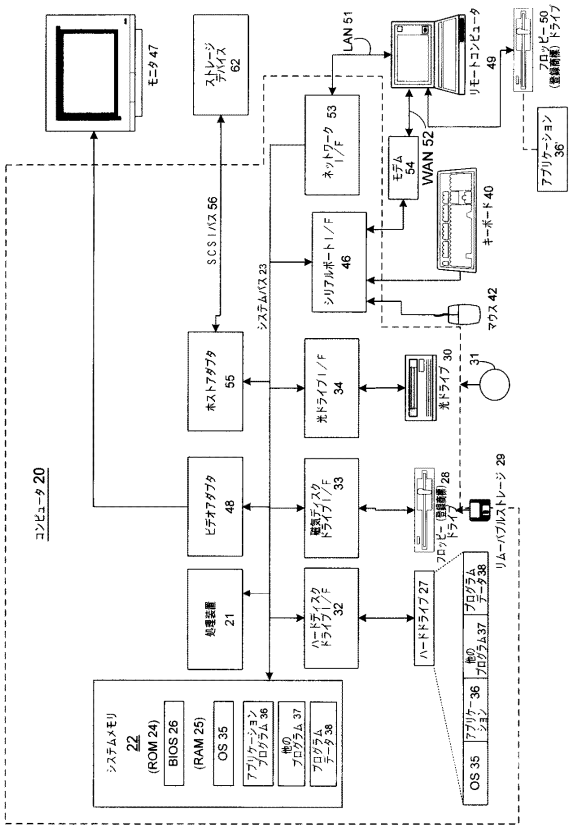
【０５１８】
【図１】本発明の態様を組み込むことができるコンピュータシステムを表すブロック図である。
【図２】３つのコンポーネントグループ、すなわち、ハードウェアコンポーネント、ハードウェア／ソフトウェアインターフェースシステムコンポーネント、およびアプリケーシ

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内のディレクトリ内のフォルダにグループ化されたファイルのため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】アイテム、アイテムフォルダおよびカテゴリの間の構造リレーションシップを例示する図である。
【図５Ａ】アイテムの構造を例示するブロック図である。
【図５Ｂ】図５Ａのアイテムの複合プロパティタイプを例示するブロック図である。
【図５Ｃ】「Ｌｏｃａｔｉｏｎ」アイテムを例示するブロック図であり、その複合タイプがさらに説明される（明示的にリストされる）図である。
【図６Ａ】アイテムを、ベーススキーマ内で発見されたアイテムのサブタイプとして例示する図である。
【図６Ｂ】図６Ａのサブタイプアイテムを例示するブロック図であり、その継承タイプが（その中間プロパティに加えて）明示的にリストされる図である。
【図７】その２つのトップレベルクラスタイプであるＩｔｅｍおよびＰｒｏｐｅｒｔｙＢａｓｅ、ならびに、それらから派生された追加のベーススキーマタイプを含む、ベーススキーマを例示するブロック図である。
【図８Ａ】コアスキーマ内のアイテムを例示するブロック図である。
【図８Ｂ】コアスキーマ内のプロパティタイプを例示するブロック図である。
【図９】アイテムフォルダ、そのメンバアイテム、および、アイテムフォルダとそのメンバアイテムの間の相互接続リレーションシップを例示するブロック図である。
【図１０】カテゴリ（これもまたアイテム自体である）、そのメンバアイテム、および、カテゴリとそのメンバアイテムの間の相互接続リレーションシップ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タイプ階層を例示する図である。
【図１２】どのようにリレーションシップが分類されるか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共に新しいレコードを同じＢツリーに挿入中である、一実施例を例示する図である。
【図１５】データ変更検出プロセスを例示する図である。
【図１６】例示的ディレクトリツリー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データストアに移動される、一実施例を示す図である。
【図１８】包含フォルダ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を概略的に表現する図である。
【図２１Ａ】例示的Ｃｏｎｔａｃｔアイテムスキーマの図表現の図である。
【図２１Ｂ】図２１Ａの例示的Ｃｏｎｔａｃｔアイテムスキーマのための要素の図表現の図である。
【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スキーマから生成されるプロセスを例示する図である。
【図２５】ファイルＡＰＩがそれに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ために使用されたアクセスマスクフォーマットを例示す

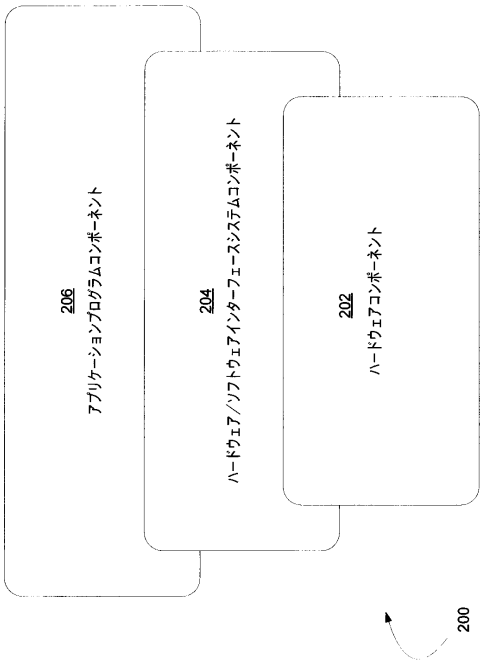
る図である。
【図２７】新しい等しく保護されたセキュリティ領域が既存のセキュリティ領域から切り開かれることを示す（パートａ、ｂおよびｃ）の図である。
【図２８】アイテム検索ビューの概念を例示する図である。
【図２９】例示的アイテム階層を例示する図である。
【図３０Ａ】インターフェースであるインターフェース１を、それを通じて第１および第２のコードセグメントが通信するコンジットとして例示する図である。
【図３０Ｂ】インターフェースを、システムの第１および第２のコードセグメントがメディアＭを介して通信することを可能にする、インターフェースオブジェクトＩ１およびＩ２を備えるとして例示する図である。
【図３１Ａ】インターフェースであるインターフェース１によって提供されたファンクションをどのように細分して、インターフェースの通信を複数のインターフェースであるインターフェース１Ａ、インターフェース１Ｂ、インターフェース１Ｃに変換することができるかを例示する図である。
【図３１Ｂ】インターフェースＩ１によって提供されたファンクションをどのように複数のインターフェースＩ１ａ、Ｉ１ｂ、Ｉ１ｃに細分することができるかを例示する図である。
【図３２Ａ】無意味なパラメータ精度を無視するか、あるいは任意のパラメータにより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】インターフェースが、パラメータを無視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インターフェースによって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のコードセグメントが、それらを両方とも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】１つのインターフェースの一部または全部をインラインで別のインターフェースに書き込んで、マージされたインターフェースを形成することができるシナリオを例示する図である。
【図３４Ａ】１つまたは複数のミドルウェアがどのように第１のインターフェース上の通信を変換して、それらを１つまたは複数の異なるインターフェースに適合させることができるかを例示する図である。
【図３４Ｂ】１つのインターフェースからの通信を受信するが、機能性を第２および第３のインターフェースに送信するために、どのようにコードセグメントをインターフェースにより導入することができるか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）がどのようにあるコードセグメントから別のコードセグメントへ通信を変換することができるか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メソッドをどのように、前記インターフェースを動的にファクタリングまたはそうでない場合は変更するように適用することができるか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を同期化するためのコンポーネントを例示する図である。
【図３７】どのように状態が計算されるか、またはその関連付けられたメタデータが交換されるかに気付かない単純なアダプタを仮定する、本発明の一実施形態を例示する図である。
【図３８Ａ】例外およびそれに対する解決法を強調するために、順次変更エニュメレーション方法を使用して、変更がどのように追跡、エニュメレートおよび同期化されるかを例示する図である。
【図３８Ｂ】例外およびそれに対する解決法を強調するために、順次変更エニュメレーション方法を使用して、変更がどのように追跡、エニュメレートおよび同期化されるかを例示する図である。
【図３８Ｃ】例外およびそれに対する解決法を強調するために、順次変更エニュメレーシ

ョン方法を使用して、変更がどのように追跡、エニュメレートおよび同期化されるかを例示する図である。
【図３８Ｄ】例外およびそれに対する解決法を強調するために、順次変更エニュメレーション方法を使用して、変更がどのように追跡、エニュメレートおよび同期化されるかを例示する図である。
【図３９Ａ】本発明のいくつかの実施形態のための競合処理パイプラインを例示する図である。
【図３９Ｂ】図３９Ａに例示されたパイプラインの論理的トラバーサルを例示する流れ図である。
【図４０】本発明のいくつかの実施形態のための、競合アイテムがターゲットアイテムのコピーと共にログされる一実施例を例示するブロック図である。
【符号の説明】
【０５１９】
２０　コンピュータ
２１　処理装置
２２　システムメモリ
２３　システムバス
２７　ハードドライブ
２８　フロッピー（登録商標）ドライブ
２９　リムーバブルストレージ
３０　光ドライブ
３２　ハードディスクドライブＩ／Ｆ
３３　磁気ディスクドライブＩ／Ｆ
３４　光ドライブＩ／Ｆ
３６　アプリケーションプログラム
３６’　アプリケーション
３７　他のプログラム
３８　プログラムデータ
４０　キーボード
４２　マウス
４６　シリアルポートＩ／Ｆ
４７　モニタ
４８　ビデオアダプタ
４９　リモートコンピュータ
５０　フロッピー（登録商標）ドライブ
５３　ネットワークＩ／Ｆ
５４　モデム
５５　ホストアダプタ
５６　ＳＣＳＩバス
６２　ストレージデバイス

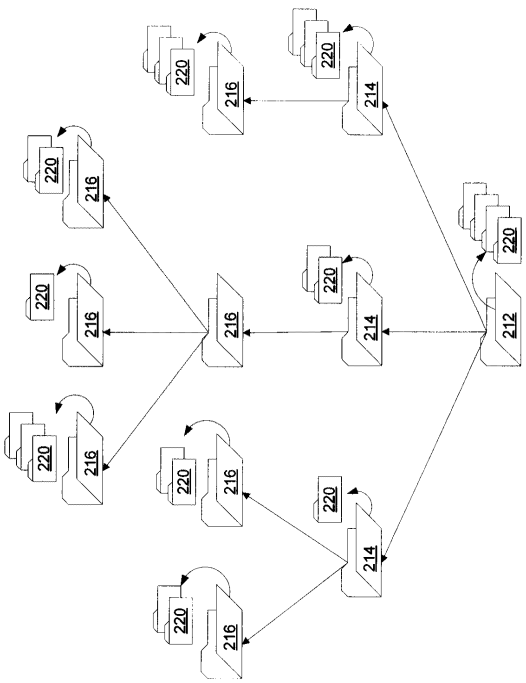
【図１】


コンピュータ 20
システムメモリ 22
(ROM 24)
BIOS 26
(RAM 25)
OS 35
アプリケーション プログラム 36
他の プログラム 37
プログラム データ 38
処理装置 21
システムバス 23
ビデオアダプタ 48
ハードディスク ドライブI/F 32
ハードドライブ 27
磁気ディスク ドライブI/F 33
プログラム データ 38
他の プログラム 37
OS 35
アプリケーション プログラム 36
リムーバブルストレージ 29
光ドライブI/F 34
光ドライブ 30
ホストアダプタ 55
SCSIバス 56
シリアルポートI/F 46
ネットワーク I/F 53
モデム 54
WAN 52
LAN 51
モニタ 47
ストレージ デバイス 62
リモートコンピュータ 49
キーボード 40
マウス 42
アプリケーション 36'

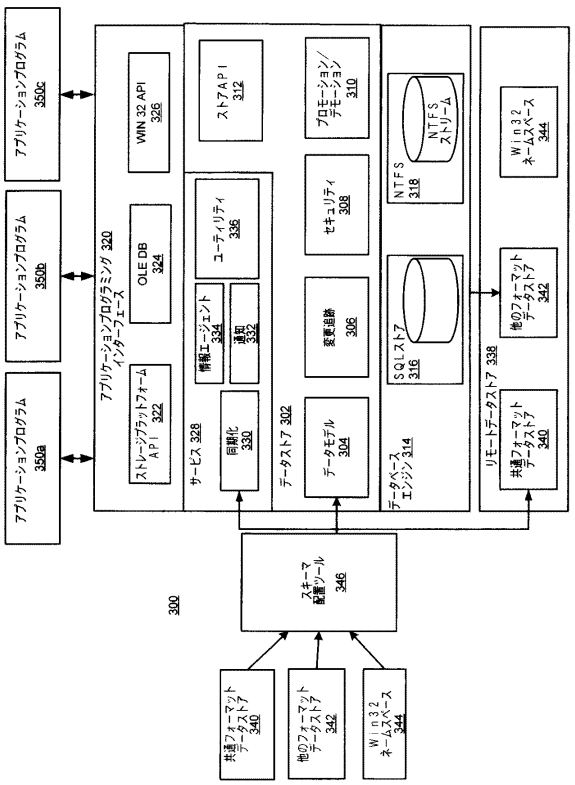
【図２】


206
アプリケーションプログラムコンポーネント
204
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
202
ハードウェアコンポーネント
200

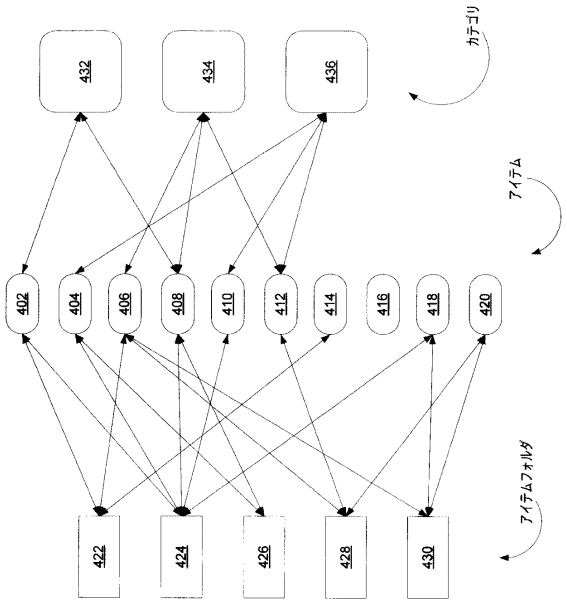
【図２Ａ】


220
216
214
212

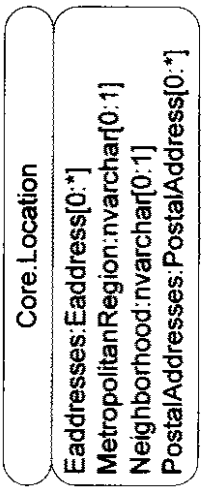
【図３】


アプリケーションプログラム 350a
アプリケーションプログラム 350b
アプリケーションプログラム 350c
アプリケーションプログラミング インターフェース 320
ストレージプラットフォーム API 322
OLE DB 324
WIN 32 API 326
サービス 328
同期化 330
情報エージェント 334
通知 332
ユーティリティ 336
ストアAPI 312
データストア 302
データモデル 304
変更追跡 306
セキュリティ 308
プロモーション／デモーション 310
データベース エンジン 314
SQLストア 316
NTFS 318
NTFS ストリーム
リモートデータストア 338
共通フォーマット データストア 340
他のフォーマット データストア 342
Win32 ネームスペース 344
スキーマ 配置ツール 346
300

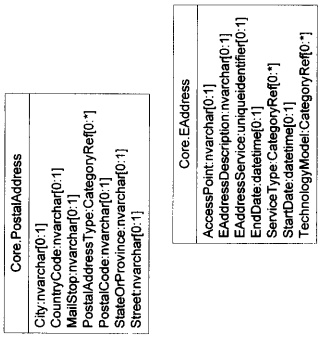
【図４】



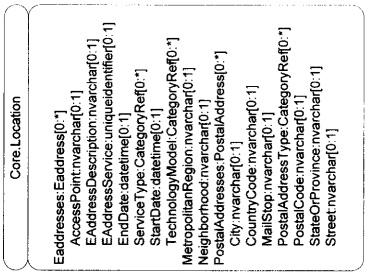
【図５Ａ】



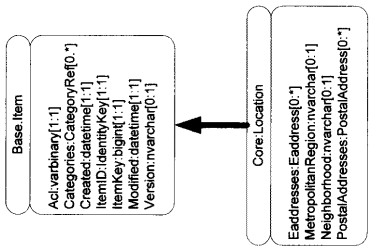
【図５Ｂ】



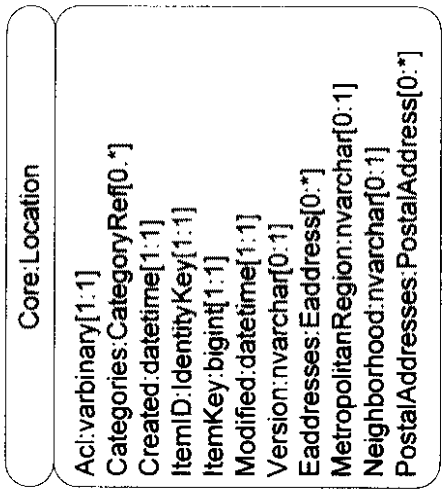
【図５Ｃ】



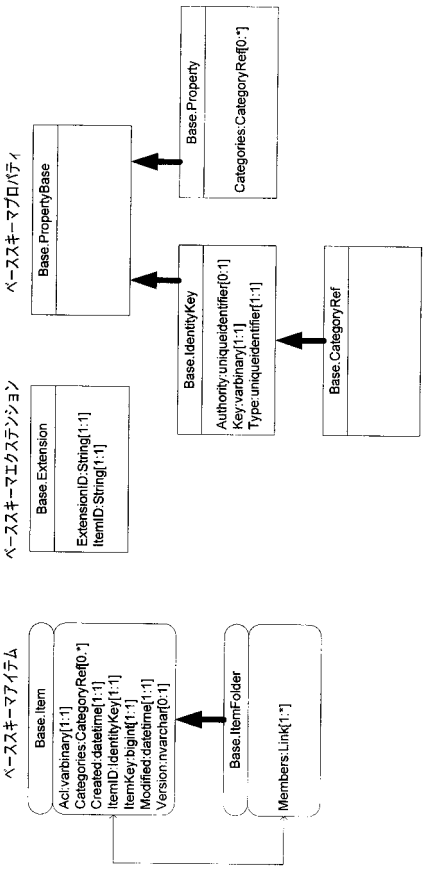
【図６Ａ】



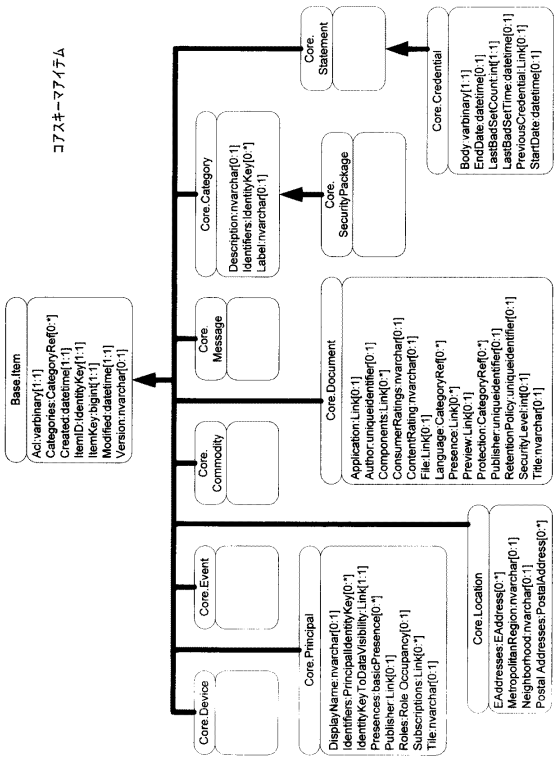
【図６Ｂ】



【図７】

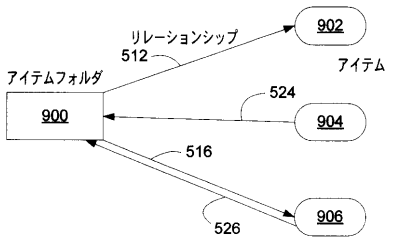


【図８Ａ】

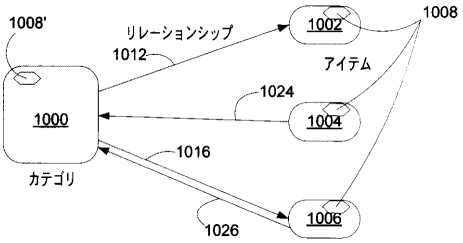


【図８Ｂ】

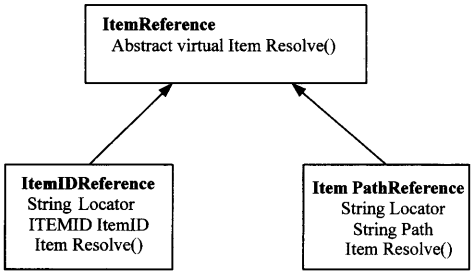
【図９】



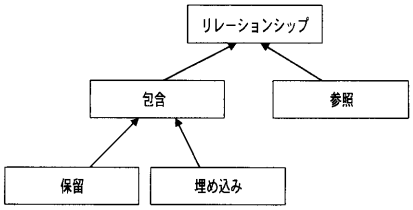
【図１０】



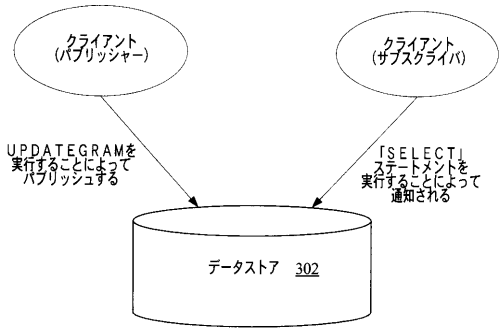
【図１１】



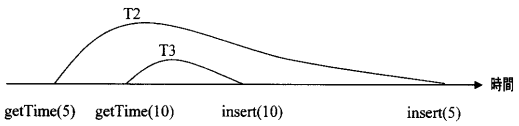
【図１２】



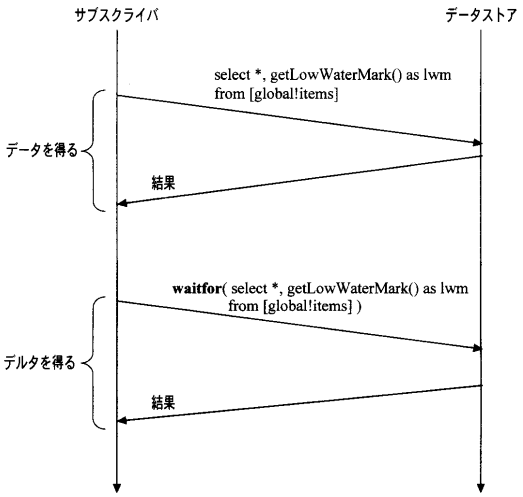
【図１３】



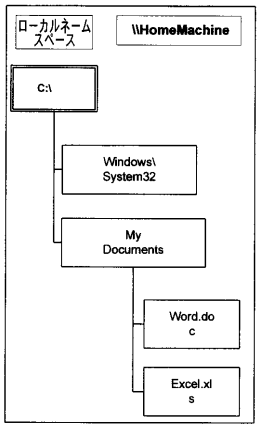
【図１４】



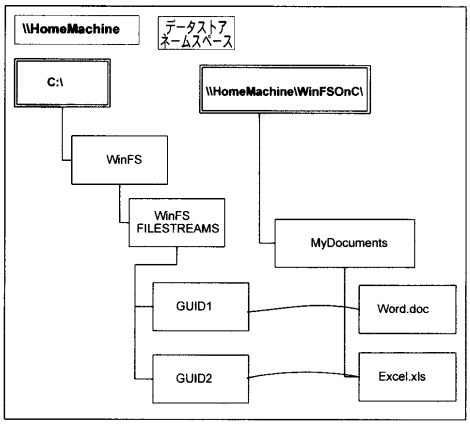
【図１５】



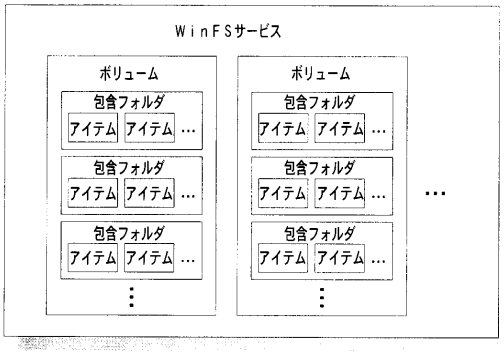
【図１６】



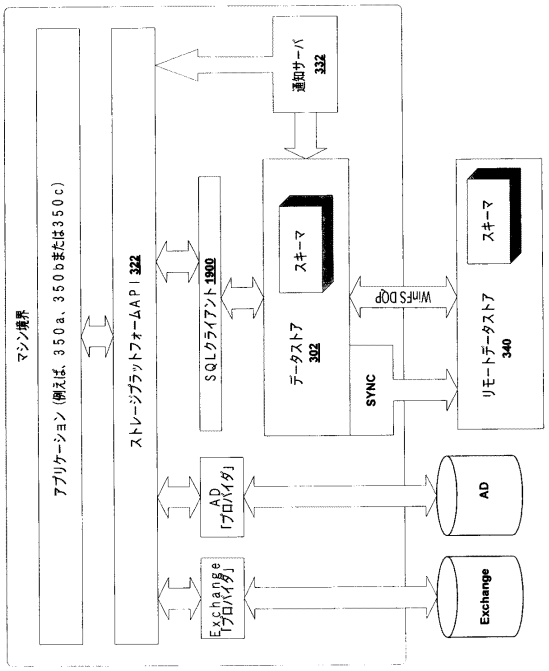
【図１７】



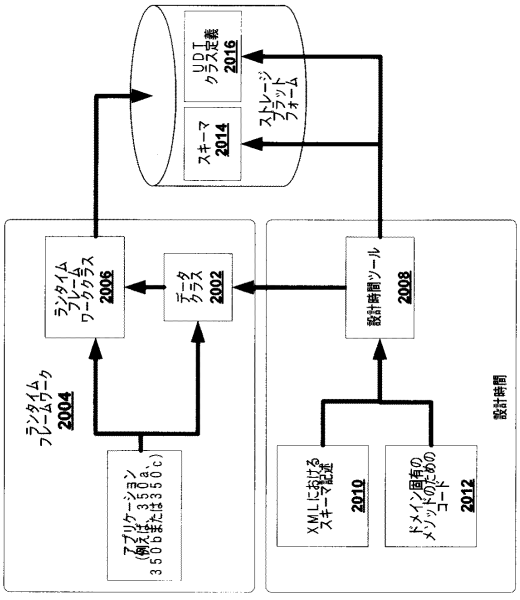
【図１８】



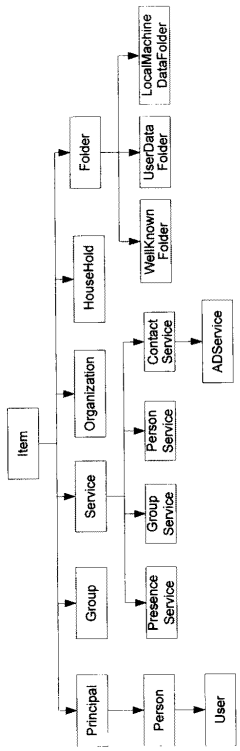
【図１９】



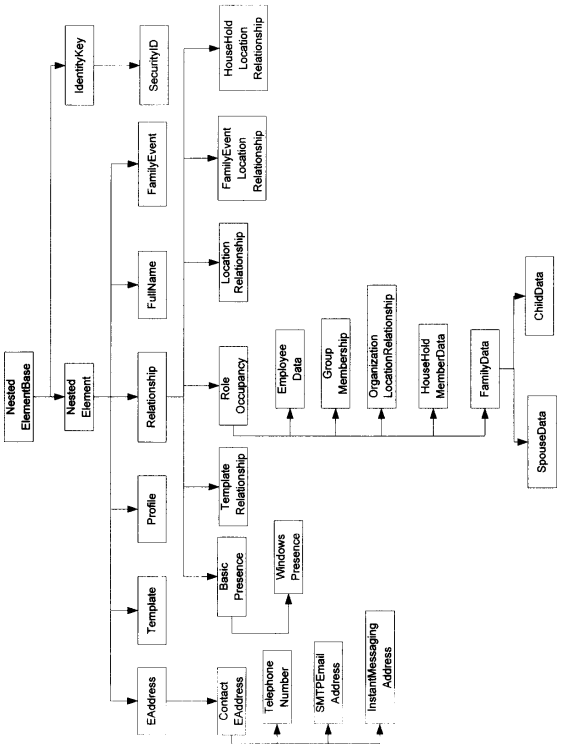
【図２０】



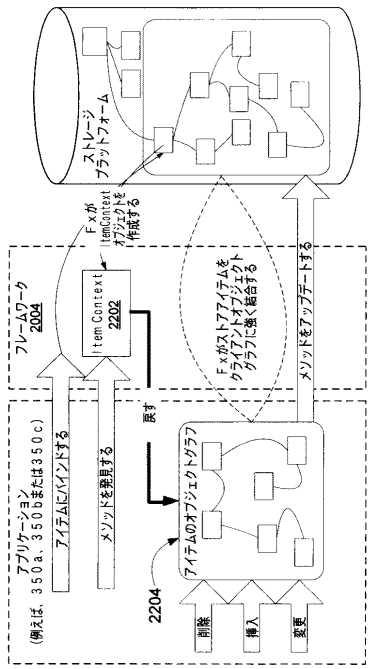
【図２１Ａ】



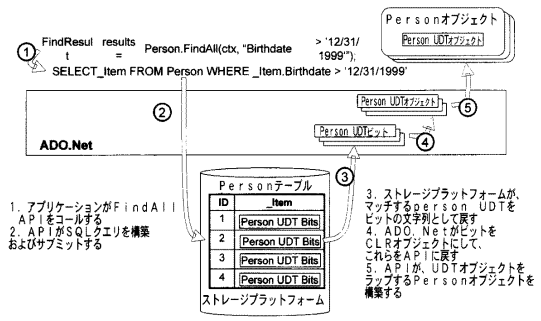
【図２１Ｂ】



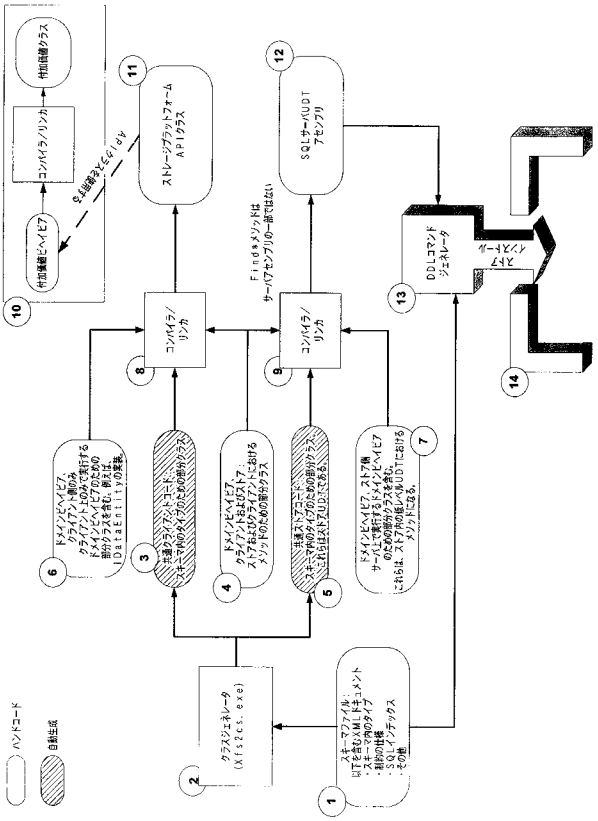
【図２２】



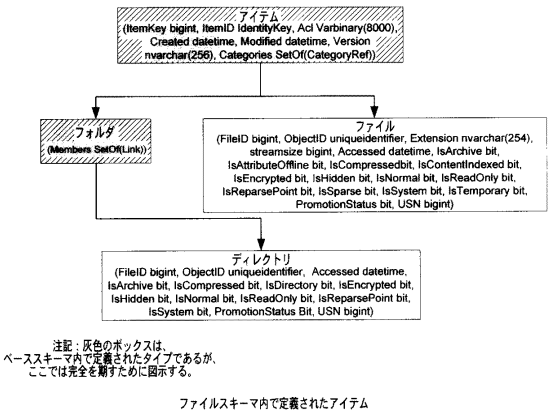
【図２３】



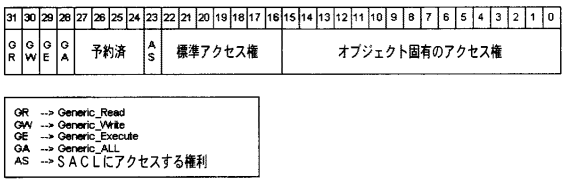
【図２４】



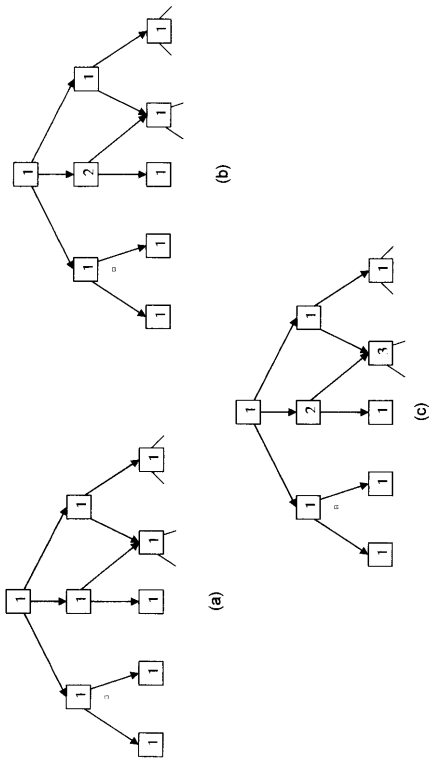
【図２５】



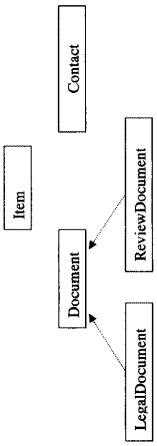
【図２６】



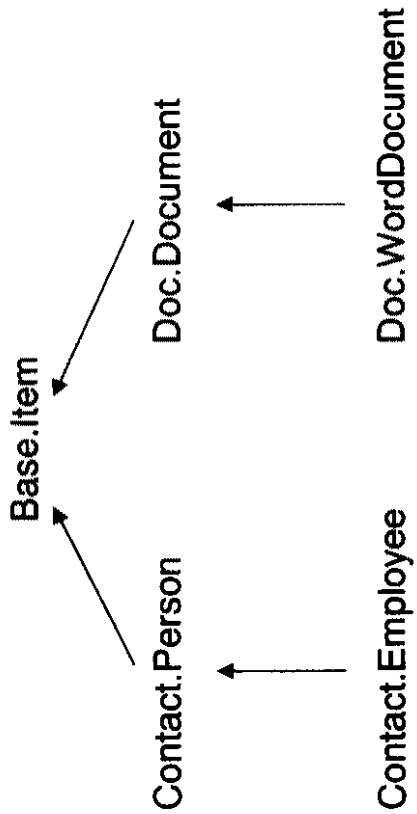
【図２７】



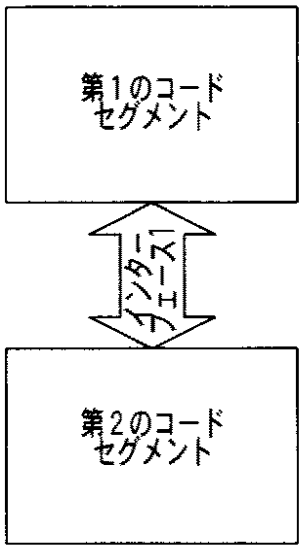
【図２８】



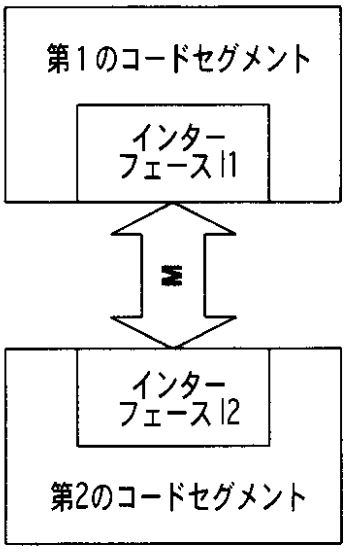
【図２９】



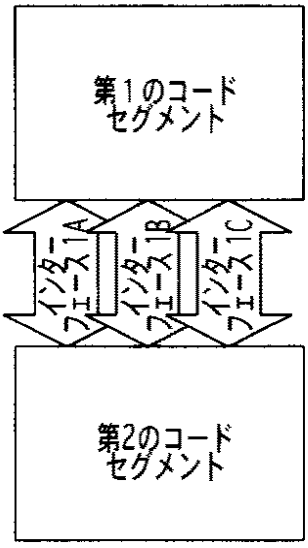
【図３０Ａ】



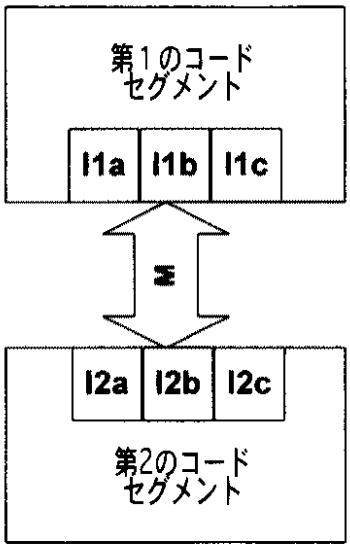
【図３０Ｂ】



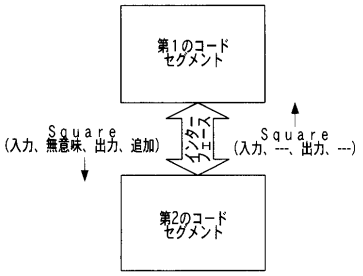
【図３１Ａ】



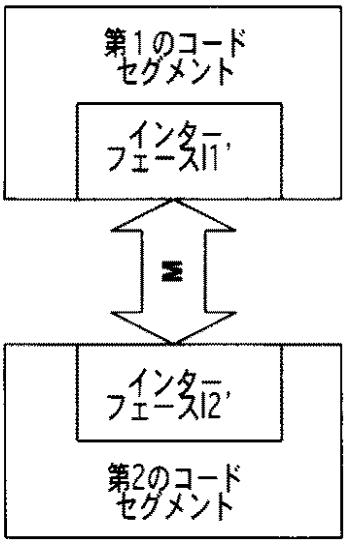
【図３１Ｂ】



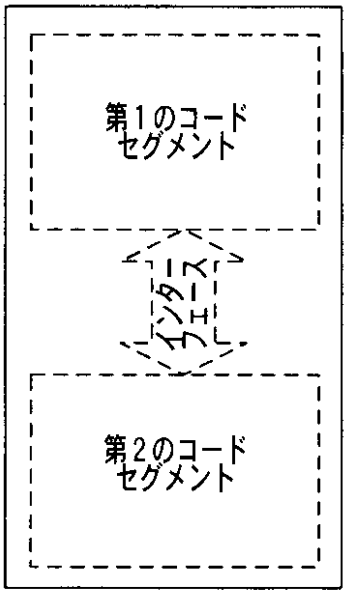
【図３２Ａ】



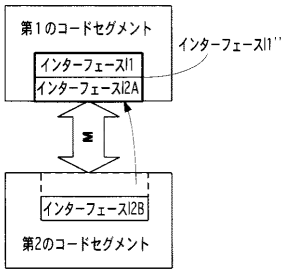
【図３２Ｂ】



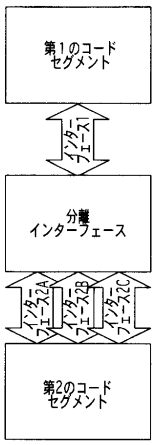
【図３３Ａ】



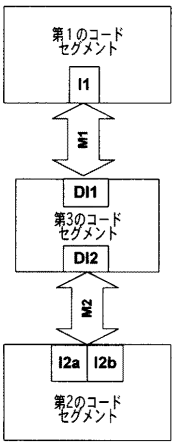
【図３３Ｂ】



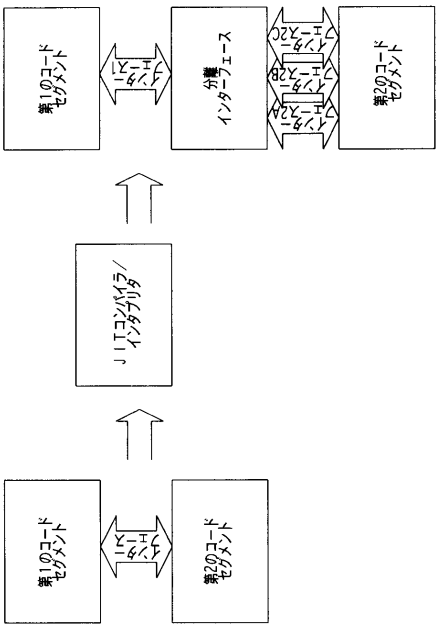
【図３４Ａ】



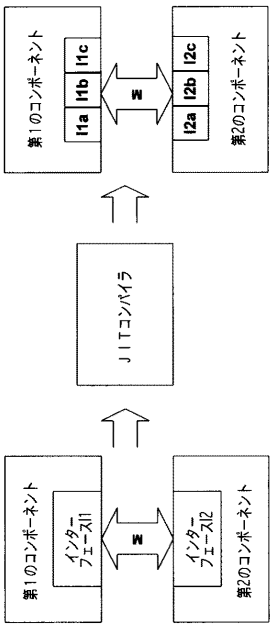
【図３４Ｂ】



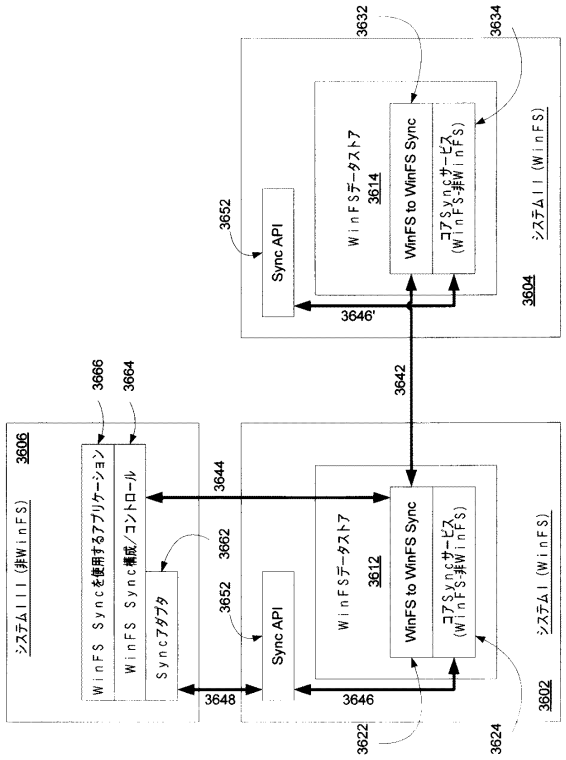
【図３５Ａ】



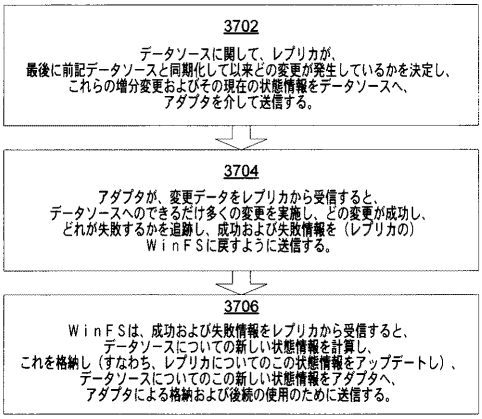
【図３５Ｂ】



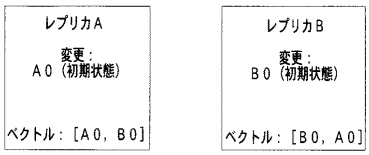
【図３６】



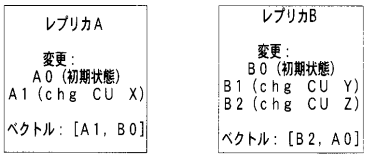
【図３７】



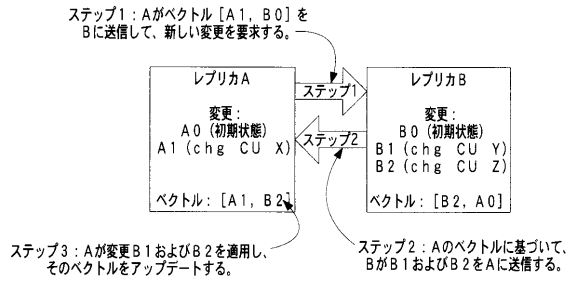
【図３８Ａ】



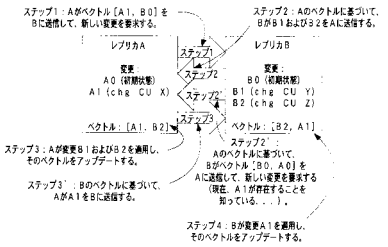
【図３８Ｂ】



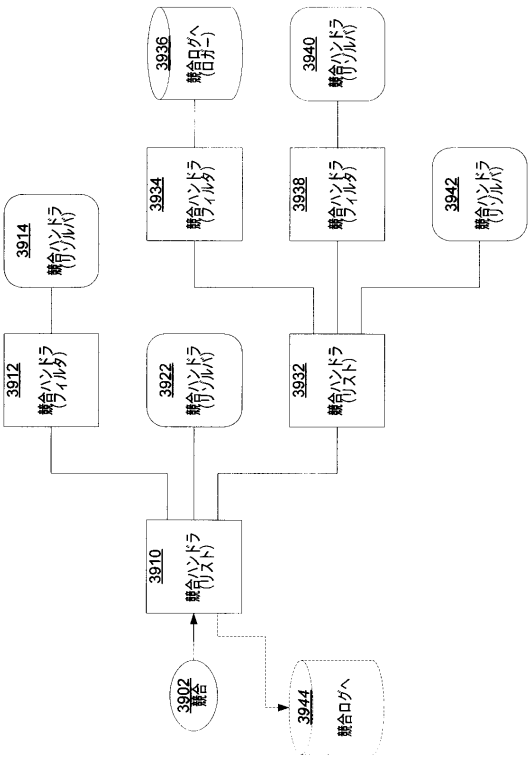
【図３８Ｃ】



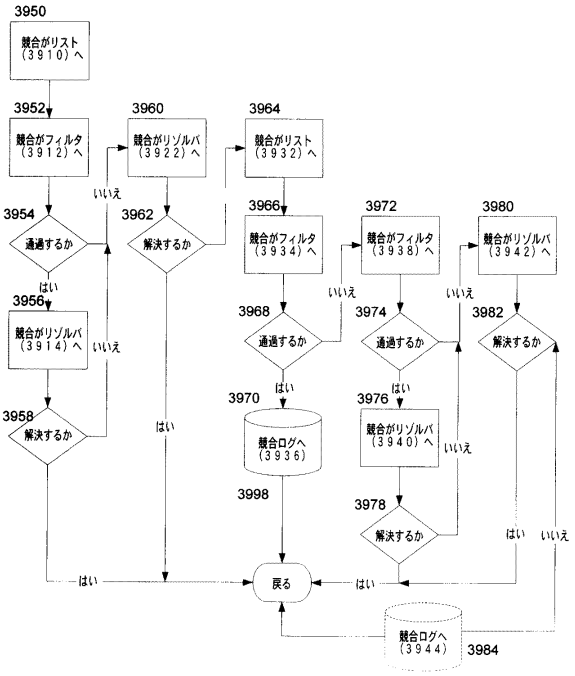
【図３８Ｄ】



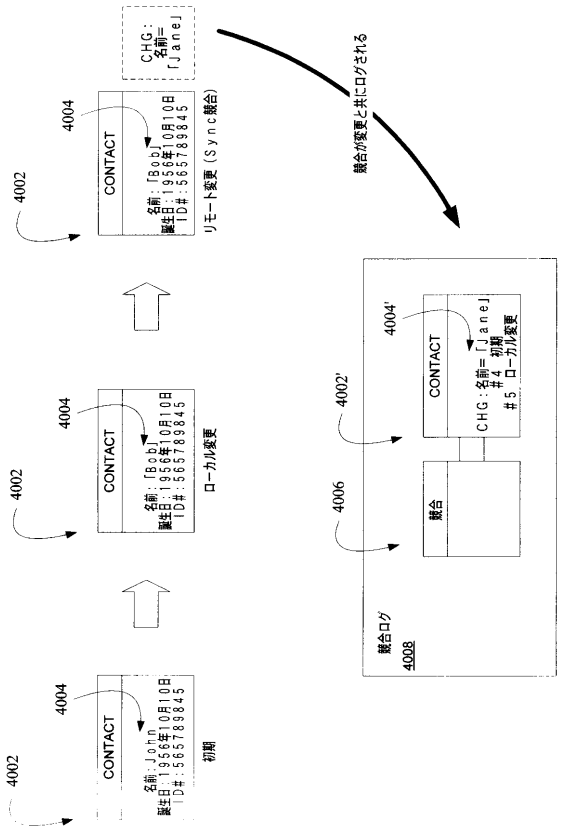
【図３９Ａ】



【図３９Ｂ】


3950
競合がリスト（３９１０）へ
3952
競合がフィルタ（３９１２）へ
3960
競合がリゾルバ（３９２２）へ
3964
競合がリスト（３９３２）へ
3954
通過するか
3962
解決するか
3966
競合がフィルタ（３９３４）へ
3972
競合がフィルタ（３９３８）へ
3980
競合がリゾルバ（３９４２）へ
はい
いいえ
3956
競合がリゾルバ（３９１４）へ
3968
通過するか
3974
通過するか
3982
解決するか
3958
解決するか
3970
競合ログへ（３９３６）
3976
競合がリゾルバ（３９４０）へ
3998
3978
解決するか
戻る
競合ログへ（３９４４）
3984

【図４０】


CHG：名前＝「Janel」
競合が変更と共にログされる
4004
CONTACT
名前：「Bob」
誕生日：1956年10月10日
ID#：565789845
4002
リモート変更（Sync競合）
4004
CONTACT
名前：「Bob」
誕生日：1956年10月10日
ID#：565789845
4002
ローカル変更
4004
CONTACT
名前：John
誕生日：1956年10月10日
ID#：565789845
4002
初期
4004'
CONTACT
CHG：名前＝「Janel」
#4 初期
#5 ローカル変更
4002'
4006
競合
競合ログ
4008

フロントページの続き

(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　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　ビベク　ジェイ．ジャベリ
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　木村　雅也

(56)参考文献　特開２０００－０５７０３２（ＪＰ，Ａ）
米国特許出願公開第２００２／０１７４１８０（ＵＳ，Ａ１）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　１３／００
Ｇ０６Ｆ　１２／００
Ｇ０６Ｆ　１５／００