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

特許第5710852号
(P5710852)

(45)発行日　平成27年4月30日(2015.4.30)　(24)登録日　平成27年3月13日(2015.3.13)

(51)Int.Cl.		F I		
G06F 9/44	(2006.01)	G06F	9/06	620B
G06F 9/45	(2006.01)	G06F	9/44	322E
G06Q 10/06	(2012.01)	G06F	17/60	162C

請求項の数 16　(全 34 頁)

(21)出願番号	特願2005-267392 (P2005-267392)
(22)出願日	平成17年9月14日 (2005.9.14)
(65)公開番号	特開2006-107481 (P2006-107481A)
(43)公開日	平成18年4月20日 (2006.4.20)
審査請求日	平成20年9月16日 (2008.9.16)
審判番号	不服2013-4213 (P2013-4213/J1)
審判請求日	平成25年3月4日 (2013.3.4)
(31)優先権主張番号	60/615,547
(32)優先日	平成16年10月1日 (2004.10.1)
(33)優先権主張国	米国(US)
(31)優先権主張番号	11/047,275
(32)優先日	平成17年1月31日 (2005.1.31)
(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　100140109
弁理士　小野　新次郎
(74)代理人　100075270
弁理士　小林　泰
(74)代理人　100101373
弁理士　竹内　茂雄
(74)代理人　100118902
弁理士　山本　修
(74)代理人　100153028
弁理士　上田　忠

最終頁に続く

(54)【発明の名称】設計時および実行時にワークフローを継ぎ目なくオーサリングし編集するためのフレームワーク

(57)【特許請求の範囲】

【請求項１】

ワークフローに関係するユーザコードを参照しつつ前記ワークフローを実行する、コンピュータにより実装される方法であって、

ユーザ入力に基づいて、複数のアクティビティを有するコンポーネント化され視覚的にモデル化されたワークフローを構成するステップと、

前記ワークフローをシリアライズするために、宣言的表現でシリアライズされたワークフローのコンパイルされていない表現を作成するステップと、

ユーザから、前記ワークフローの複数のアクティビティの１つ以上との関連についての前記ワークフローのユーザコードを受信するステップであって、前記アクティビティはそれぞれメタデータ、インスタンスデータおよび実行ロジックの少なくとも３つの部分を持ち、前記メタデータは前記ワークフローの構成に関するデータプロパティを定義し、前記複数のアクティビティは１つ以上の意味論及び動作に関連する、ステップと、

前記ユーザコードをコンパイルするステップと、

前記構成されたワークフローを、前記ワークフローの作成されたコンパイルされていない表現及び前記コンパイルされたユーザコードの機能として実行するステップであって、前記メタデータは１つ以上の所定のフィールドに格納され、前記複数のアクティビティのインスタンス間で共有される、ステップと、

前記ワークフローモデルの実行に影響を及ぼすために、前記構成されたワークフローが実行され続けている間に、前記コンパイルされていないワークフローの表現を前記ワーク

フローのコンパイルを行わずに動的に修正するステップであって、前記コンパイルされていない表現の修正が前記ワークフローと前記ユーザコードにリアルタイムで直接反映されるステップ、と
を備えたことを特徴とするコンピュータにより実装される方法。
【請求項２】
前記各アクティビティは、該アクティビティに関連付けられたアクティビティ実行プログラムを有し、前記構成されたワークフローを実行するステップは、前記コンパイルされたユーザコードを参照しつつ前記複数のアクティビティのそれぞれについて前記アクティビティ実行プログラムを実行するステップを含むことを特徴とする請求項１に記載のコンピュータにより実装される方法。
【請求項３】
前記修正されたワークフローを、前記ワークフローの修正された、コンパイルされていない表現及び前記コンパイルされたユーザコードとして実行するステップをさらに備えたことを特徴とする請求項１に記載のコンピュータにより実装される方法。
【請求項４】
前記構成されたワークフローを実行するステップ、および前記修正されたワークフローを実行するステップは、前記構成されたワークフロー及び前記修正されたワークフローの前記コンパイルされていない表現をコンパイルせずに実行するステップを含むことを特徴とする請求項１に記載のコンピュータにより実装される方法。
【請求項５】
前記ユーザコードは、前記ワークフローの実行に影響を及ぼす１つ以上のビジネスルールを表すことを特徴とする請求項１に記載のコンピュータにより実装される方法。
【請求項６】
前記ユーザコードをコンパイルするステップは、前記複数のアクティビティのそれぞれに関連付けられた前記ユーザコードをコンパイルするステップを含むことを特徴とする請求項１に記載のコンピュータにより実装される方法。
【請求項７】
前記構成されたワークフローを実行するステップは、デッドロック検出および同時実行制御を課すステップを含むことを特徴とする請求項１に記載のコンピュータにより実装される方法。
【請求項８】
１つ以上のコンピュータ読み取り可能記録媒体が、請求項１に記載の前記方法を実行するコンピュータ実行可能命令を格納することを特徴とする請求項１に記載のコンピュータにより実装される方法。
【請求項９】
ワークフローに関係するユーザコードを参照しつつ前記ワークフローを実行するためのコンピュータ実行可能コンポーネントを格納する１つ以上のコンピュータ読み取り可能記録媒体であって、前記コンポーネントは、
視覚的にモデル化されコンポーネント化されたワークフローの前記ユーザコードを実行可能オブジェクトコードに翻訳するコンパイラコンポーネントであって、ユーザから受け取られつつある前記ユーザコードが、前記ワークフローの複数のアクティビティの１つ以上と関連付けることを定義され、前記アクティビティはそれぞれメタデータ、インスタンスデータおよび実行ロジックの少なくとも３つの部分を持ち、前記メタデータは前記ワークフローの構成に関するデータプロパティを定義し、前記複数のアクティビティは１つ以上の意味論および動作に関連する、コンパイラコンポーネントと、
ワークフローをシリアライズするために、宣言的表現でシリアライズされたワークフローのコンパイルされていない表現の機能及び前記コンパイラコンポーネントからの前記ワークフローに関連する前記実行可能オブジェクトコードとして前記ワークフローを実行するワークフローコンポーネントであって、前記メタデータは、１つ以上の所定のフィールドに格納され、複数のアクティビティのインスタンス間で共有される、ワークフローコン

ポーネントと、
前記ワークフローモデルの実行に影響を及ぼすために、前記ワークフローが前記ワークフローコンポーネントにより実行され続けている間に、前記ユーザが前記コンパイルされていないワークフローの表現を前記ワークフローのコンパイルを行わずに動的に修正できるようにするデザイナコンポーネントであって、前記コンパイルされていない表現の修正が前記ワークフローと前記ユーザコードにリアルタイムで直接反映されるもの、と
を備えたことを特徴とするコンピュータ読み取り可能記録媒体。
【請求項１０】
前記ワークフローコンポーネントは、前記修正されたコンパイルされていないワークフローの表現及び前記コンパイルされたユーザコードに基づいて前記修正されたワークフローをさらに実行することを特徴とする請求項９に記載のコンピュータ読み取り可能記録媒体。
【請求項１１】
前記ワークフローに関連するアクティビティのそれぞれは、該アクティビティに関連付けられたアクティビティ実行プログラムを有し、前記ワークフローコンポーネントは、前記アクティビティのそれぞれに対するアクティビティ実行プログラムによって前記ワークフローを実行することを特徴とする請求項１０に記載のコンピュータ読み取り可能記録媒体。
【請求項１２】
ワークフローに関係するユーザコードを参照しつつ前記ワークフローを実行するシステムであって、
コンパイルされていないコンポーネント化されたワークフローの表現および前記ワークフローのユーザコードを格納するメモリ領域と、
ユーザから、前記ワークフローの複数のアクティビティの１つ以上との関連についての前記ユーザコードを受信するためのインターフェースであって、前記アクティビティはそれぞれメタデータ、インスタンスデータおよび実行ロジックの少なくとも３つの部分を持ち、前記メタデータは前記ワークフローの構成に関するデータプロパティを定義し、前記複数のアクティビティは１つ以上の意味論及び動作に関連するものと、
プロセッサであって、
ユーザ入力に基づいて、複数のアクティビティを有するコンポーネント化され視覚的にモデル化されたワークフローを構成するためのコンピュータ実行可能命令と、
ワークフローをシリアライズするために、宣言的表現でワークフローのコンパイルされていない表現を作成するためのコンピュータ実行可能命令と、
前記メモリ領域内に格納された前記ユーザコードをコンパイルするためのコンピュータ実行可能命令と、
前記構成されたワークフローを、前記ワークフローのコンパイルされていない表現及び前記ワークフローに関連する前記メモリ領域内に格納された前記コンパイルされたユーザコードの機能として実行するためのコンピュータ実行可能命令であって、前記メタデータは１つ以上の所定のフィールドに格納され、前記複数のアクティビティのインスタンス間で共有される、コンピュータ実行可能命令と、
前記ワークフローモデルの実行に影響を及ぼすために、前記構成されたワークフローが実行され続けている間に、前記コンパイルされていないワークフローの表現を前記ワークフローのコンパイルを行わずに動的に修正できるようにするためのコンピュータ実行可能命令であって、前記コンパイルされていない表現の修正が前記ワークフローと前記ユーザコードにリアルタイムで直接反映されるもの、を実行するように構成されたプロセッサと
を備えたことを特徴とするシステム。
【請求項１３】
前記複数のアクティビティの各々は、それに関連付けられたアクティビティ実行プログラムを有し、前記プロセッサは、前記アクティビティのそれぞれについて前記アクティビ

ティ実行プログラムを実行することにより前記構成されたワークフローを実行するように構成されていることを特徴とする請求項１２に記載のシステム。
【請求項１４】
前記メモリ領域に格納されている前記ユーザコードをコンパイルするための手段をさらに備えたことを特徴とする請求項１２に記載のシステム。
【請求項１５】
前記構成されたワークフローを、前記ワークフローの修正された、コンパイルされていない表現及び前記コンパイルされたユーザコードの機能として実行するための手段をさらに備えることを特徴とする請求項１４に記載のシステム。
【請求項１６】
前記構成されたワークフローが実行され続けている間に、前記ユーザが前記ワークフローのコンパイルされていない表現を動的に修正できるようにするための手段をさらに備えたことを特徴とする請求項１５に記載のシステム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、ワークフローモデリングの分野に関し、より詳細には、コンポーネント化された拡張可能ワークフローモデル（componentized and extensible workflow model）に関する。
【背景技術】
【０００２】
既存のシステムでは、ビジネス問題をモデル化することによりビジネス問題を高水準のワークフローにマッピングすることを試みる。しかし、現実世界のワークフローは、（ａ）実行およびモデリングの複雑さ、（ｂ）設計時のフローの構造に関する知識、（ｃ）静的に定義された、またはアドホック／動的な特性、（ｄ）ライフサイクルにおけるさまざまな時点でのフローのオーサリングおよび編集の容易さ、（ｅ）ビジネスロジックとコアワークフロープロセスとの弱いまたは強い関連などのさまざまな次元において異なる。既存のモデルでは、これらすべての要因に対応することはできない。
【０００３】
さらに、ほとんどの既存のワークフローモデルは、言語ベースのアプローチ（例えば、ＢＰＥＬ４ＷＳ、ＸＬＡＮＧ／Ｓ、およびＷＳＦＬ）またはアプリケーションベースのアプローチに基づいている。言語ベースのアプローチは、定義済み言語構文の閉じた集合を持つ高水準のワークフロー言語であり、これにより、ユーザ／プログラマに対するワークフロープロセスのモデル化が容易になる。ワークフロー言語は、言語構文の閉じた集合に対するすべての意味情報を備え、これによりユーザはワークフローモデルを構築できる。しかし、言語は、開発者側で拡張することはできず、ワークフローモデルを構成するプリミティブの閉じた集合となっている。言語は、ワークフローシステムベンダが出荷する言語コンパイラに結び付けられている。ワークフローシステム製品ベンダのみが、将来の製品バージョンにおいて新しい言語構文の集合を加えて言語を拡張することによりモデルを拡張することができる。このため、多くの場合、言語に関連付けられたコンパイラのアップグレードが必要となる。
【０００４】
アプリケーションベースのアプローチは、アプリケーション内に領域特有の問題を解決するためのワークフロー機能を備えるアプリケーションである。これらのアプリケーションは、本当に拡張可能であるとはいえず、またプログラム可能なモデルも持たない。
【０００５】
既存のアプローチでは、複雑さ、先見、動的ワークフロー、オーサリングの容易さ、およびビジネスロジックとコアワークフローとの関連度の強さの課題が適切に扱えない。さまざまなクラスのワークフローをモデル化するビジュアルワークフローデザイナ（visual workflow designers）を構築するために利用できる、拡張可能、カスタマイズ可能、お

よび再ホスト可能な（re-hostable）ワークフローデザイナフレームワークはない。既存のシステムでは、ユーザがグラフィックスを利用してワークフロープロセスの設計を行い、開発者が選択したプログラミング言語によるビジネスロジックを関連付けることができるアプリケーション短期開発（ＲＡＤ）スタイルのワークフロー設計の経験を欠いている。さらに、インク対応ワークフローデザイナ（ink-enabled workflow designer）もない。

【０００６】

さらに、既存システムは、ワークフローの実行のための継ぎ目のないアドホックな、または動的な編集機能を備えていない。ワークフロープロセスは、本質的に動的であり移動性を有し、その形態は設計時には全く予見できない。ワークフロープロセスは、構造化様式で始まり、最終的には、その実行存続期間の過程で発展し変化する。ワークフロービルダが設計時にさまざまな種類のワークフローモデルをオーサリングするだけでなく、継ぎ目のない方式でアドホックな、または動的な変更を実行中のワークフローに加えられるようにするワークフローオーサリングフレームワーク（workflow authoring framework）が必要である。ワークフロープロセスが展開された後、実行されていても、ビジネス要件に変更があれば、多くの場合、現在実行中のワークフロープロセスを変更または編集せざるを得ない。ワークフロープロセスの実行時オーサリングを行えるシステムが必要である。

【０００７】

さらに、ワークフロープロセスでは、ワークフロープロセスの複数のステップにまたがる分野横断的に直交する、かつ入り組んだ諸問題を取り扱う。例えば、ワークフロープロセスの一部が長い実行トランザクションに参加するように設計されているが、同じプロセスの他の部分は同時実行用に設計されている。同じワークフロープロセスのさらに他の部分では、追跡を必要とするが、他の部分ではビジネスまたはアプリケーションレベルの例外を扱う。特定の動作をワークフロープロセスの１つまたは複数の部分に適用する必要がある。

【０００８】

いくつかのワークフローモデリングアプローチは、すべての例外および人間介入を含むビジネスプロセス全体の完全なフローベースの記述を必要とするので実用的でない。これらのアプローチのいくつかは、例外が発生したときの追加機能を用意しているが、他のアプローチでは、ビジネスプロセスをモデル化するフローベースのアプローチの代わりに制約ベースのアプローチのみを採用する。既存システムでは、フローベースまたは制約ベースのアプローチのいずれかを実装する。このようなシステムは、柔軟性が低すぎて、ビジネスの数多くの一般的状況をモデル化できない。

【発明の開示】

【発明が解決しようとする課題】

【０００９】

したがって、これらの欠点およびその他の欠点の１つまたは複数を解決する、コンポーネント化された拡張可能なワークフローモデルが望まれる。

【課題を解決するための手段】

【００１０】

本発明のいくつかの実施形態は、コンポーネント化されたワークフローモデルを構築する拡張可能なフレームワークを実現する。特に、ワークフロープロセスの各ステップは、ワークフローステップの設計時の態様、コンパイル時の態様、および実行時の態様を記述する関連コンポーネントモデルを持つ。さらに、開発者は、これらのコンポーネントをオーサリングすることによりコアワークフローモデルを拡張することができる。本発明は、高度に形式的なマシン同士のプロセス、制約ベースのアドホックなヒューマンワークフロー、およびフローベースのアプローチおよび制約ベースのアプローチの混合を持つワークフローを含む、さまざまな種類のワークフローの実行を調整するのに十分柔軟であり強力なワークフローエンジンを含む。ワークフローエンジンでは、実行中ワークフローに対するアクティベーション、実行、クエリ、および制御の機能を使用できる。例えば、本発明

では、実行ワークフローにアドホックな動的変更を加えることができる。ワークフローエンジンは、サーバおよびクライアント環境の両方を含むさまざまなホスト環境において再ホスト可能または埋め込み可能である。それぞれの特定のホスト環境では、ワークフローエンジンをサービスプロバイダ群に結合する。サービスプロバイダの集約機能により、特定のホスト環境で実行できるワークフローの種類が決定される。

【００１１】

本発明の他の実施形態では、ワークフローモデルをシリアライゼーションするために拡張オーケストレーションマークアップ言語（extensible orchestration markup language）（ＸＯＭＬ）などの宣言形式を備える。宣言形式を使用すると、ユーザはコンポーネントの集合を書くことによりワークフローモデルを拡張することができる。ワークフロープロセスのさまざまなステップに対応する意味論は、コンパイル時に与えられたコンポーネントの意味論の妥当性を確認し、それを強制するアクティビティバリデータコンポーネント（activity validator component）内にカプセル化される。本発明の宣言形式の実施態様では、さらに、データの宣言およびデータとワークフローモデルのさまざまな要素との関連付けが可能である。宣言形式では、ワークフローを通じてデータを変換する操作をサポートする。例えば、この形式は、ワークフローモデル内のデータベースまたはファイル、コードスニペット、およびビジネスルールなどの外部データソースを宣言的に表す。

【００１２】

本発明の一実施態様では、さまざまなクラスのワークフローをモデル化するグラフィカル／ビジュアルワークフローデザイナを構築する拡張可能、カスタマイズ可能、および再ホスト可能なワークフローデザイナフレームワークを提供する。本発明の他の実施態様では、ユーザが任意のプログラミング言語によりグラフィックスを利用してワークフロープロセスの設計を行い、ビジネスロジックの関連付けを行うことができるようにするアプリケーション短期開発スタイルのワークフロー設計のエクスペリエンスをサポートする。本発明の実施態様は、さらに、ペンおよびタブレット技術を使用してインクもサポートする。本発明は、ユーザにより描画されたワークフローが内部表現に変換されるフリーフォームドローイングサーフェス（free form drawing surface）を備える。本発明は、既存のドローイングサーフェス上のインク編集（例えば、追加／削除アクティビティ）、および既存のワークフローのインク注釈（例えば、デザインサーフェス（design surface）に手書きされたコメント、提案、または注意喚起）を介してワークフローの作成および修正をサポートする。

【００１３】

本発明のさらに他の実施態様では、宣言的方法により分野横断的な動作を捕捉し、その動作をワークフローモデルの選択された部分に適用するためのコンポーネントを備える。本発明の他の実施態様は、それに関連付けられている動作の背景状況においてワークフローモデルの選択された部分を実行する。本発明の実施態様は、フレームワーク、再利用可能コンポーネント、およびワークフロープロセスモデルの複数のステップにまたがる分野横断的に直交する、かつ入り組んだ諸問題を扱うための言語を提供する。

本発明の一態様によれば、コンピュータにより実装される方法は、ワークフローに関係するユーザコードを参照しつつワークフローを実行する。このコンピュータにより実装される方法は、ユーザコードをコンパイルすることを含む。この方法は、さらに、コンパイルされていないワークフローをコンパイルされたコードで実行することも含む。この方法は、さらに、コンパイルされていないワークフローが実行されている間にユーザがコンパイルされていないワークフローを動的に修正できるようにすることも含む。

【００１４】

本発明の他の態様によれば、１つまたは複数のコンピュータ読み取り可能媒体は、ワークフローに関係するユーザコードを参照しつつワークフローを実行するためのコンピュータ実行可能コンポーネントを格納する。これらのコンポーネントは、ユーザコードを実行可能オブジェクトコードに翻訳するためのコンパイラコンポーネントを含む。これらのコンポーネントは、さらに、コンパイルされていないワークフローをコンパイラコンポーネ

ントから得られた実行可能オブジェクトコードで実行するためのワークフローコンポーネントも含む。これらのコンポーネントは、さらに、コンパイルされていないワークフローがワークフローコンポーネントにより実行されている間にユーザがコンパイルされていないワークフローを動的に修正できるようにするデザイナコンポーネントも含む。

【００１５】

本発明のさらに他の態様によれば、システムは、ワークフローに関係するユーザコードを参照しつつワークフローを実行する。このシステムは、コンパイルされていないワークフローおよびユーザコードを格納するメモリ領域を備える。このシステムは、さらに、メモリ領域内に格納されているユーザコードをコンパイルするためのコンピュータ実行可能命令と、コンパイルされていないワークフローをコンパイルされたコードで実行するためのコンピュータ実行可能命令と、コンパイルされていないワークフローが実行されている間に、ユーザがコンパイルされていないワークフローを動的に修正できるようにするためのコンピュータ実行可能命令を実行するように構成されたプロセッサを備える。

【００１６】

それとは別に、本発明は、他のさまざまな方法および装置を含むことができる。

【００１７】

他の特徴は、一部は明白であり、また一部は以下で指摘される。

【発明を実施するための最良の形態】

【００１８】

本発明のいくつかの実施形態では、ビジネスプロセスなどのプロセスを表すワークフローをモデル化する。ビジネスプロセスとは、予測可能で再現可能な成果が得られる従属する、順序付けられたタスク、アクティビティなどのことである。組織の運営手順、制度に関する実務知識、および情報資源を含めて、ビジネスプロセスは、効率よく、時機を逃すことなく定義済みのビジネス目的を達成するように設計される。効率的な環境では、プロセスの機能コンポーネントは、絶え間なく変化する企業要件に対処するために、容易に識別し、適合し、展開することができる。ワークフローは、ビジネスプロセス内のタスクとやり取りするエンドユーザ体験である。タスクは、アクティビティ、コンポーネントなどとしてモデル化され、それぞれ人またはマシンにより実行される一単位の作業である。一実施形態では、複数のアクティビティが一人のユーザに提示される。ユーザは、ワークフローを作成する複数のアクティビティを選択して編成する。作成されたワークフローは、実行され、ビジネスプロセスをモデル化する。図１を参照すると、ワークフロー１００の実施例は、タスクおよび制御フロー複合アクティビティを含んでいる。

【００１９】

一実施例では、オーケストレーションエンジンワークフローモデルは、さまざまなクラスのワークフローのモデリング、オーサリング、および実行をサポートする。実施例は、整然とした順序で、または非同期イベントの集合として実行される構造化されたステップの集合に関して与えられた問題をモデル化することを含む。オーケストレーションエンジンは、スケジュールの実行を調整する。スケジュールは、ツリー構造で階層的に配列されたアクティビティの整理された集合である。実行アクティビティの実行コンテキストおよび実行アクティビティから見える共有データは、スコープにより規定される。それぞれのアクティビティは、ワークフロープロセス内のステップに対するメタデータをカプセル化するコンポーネントを表す。アクティビティは、ワークフローモデルでの実行の基本単位であり、関連付けられたプロパティ、ハンドラ、制約、およびイベントを持つ。それぞれのアクティビティは、任意のプログラミング言語のユーザコードにより構成することができる。例えば、ユーザコードは、共通言語ランタイム（ＣＬＲ）言語で書かれたビジネスもしくはアプリケーションロジックまたはルールを表すことができる。それぞれのアクティビティは、ユーザコードでの実行へのインターセプト前フックおよびインターセプト後フック（pre-interception hooks and post-interception hooks）をサポートする。それぞれのアクティビティは、関連付けられたランタイム実行意味論および動作（例えば、状態管理、トランザクション、イベント処理、および例外処理）を持つ。アクティビティは

、他のアクティビティと状態を共有できる。アクティビティは、プリミティブなアクティビティであるか、または複合アクティビティにグループ化することができる。プリミティブつまり基本アクティビティは、下位構造（例えば、子アクティビティ）を持たず、したがって、ツリー構造内の葉ノードである。複合アクティビティは、下位構造を含む（例えば、これは、１つまたは複数の子アクティビティの親である）。

【００２０】

一実施形態では、アクティビティは、単純アクティビティ、コンテナアクティビティ、およびルートアクティビティの３種類がある。この実施形態では、モデルには１つのルートアクティビティがあり、ルートアクティビティの内側に単純アクティビティまたはコンテナアクティビティは全くないか、またはいくらかある。コンテナアクティビティは、単純またはコンテナアクティビティを含むことができる。ワークフロープロセス全体は、高位のワークフロープロセスを構築するアクティビティとして使用することができる。さらに、アクティビティは、中断可能であるか、または非中断可能とすることができる。非中断可能複合アクティビティは、中断可能アクティビティを含まない。非中断可能アクティビティには、アクティビティにブロックさせるサービスはない。

【００２１】

オーケストレーションエンジンは、アクティビティの集まりの例を示している。図２を参照すると、アクティビティ継承ツリーにアクティビティの実施例が示されている。図２に一覧として示されているアクティビティの実施例は、付録Ａで詳しく説明する。さらに、ユーザは、ワークフローモデルを拡張するために１つまたは複数のアクティビティを書くことができる。例えば、ユーザは、特定のビジネス問題、領域、ワークフロー標準（例えば、ビジネスプロセス実行言語）、またはターゲットプラットフォームに対するアクティビティを書くことができる。オーケストレーションエンジンは、例えば、コードを分析するサービス、タイプ解決およびタイプシステム、シリアライゼーションするサービス、およびレンダリングを含むアクティビティを書くためのさまざまなサービスの集合をユーザに提供することができる。

【００２２】

一実施形態では、それぞれのアクティビティは、メタデータ、インスタンスデータ、および実行ロジックの少なくとも３つの部分を持つ。アクティビティのメタデータは、構成することができるデータプロパティを定義する。例えば、いくつかのアクティビティは、アクティビティ抽象基本クラスで定義されているメタデータの共通集合を共有することができる。それぞれのアクティビティは、このクラスを拡張することによりそのニーズに応じて独自の追加メタデータプロパティを宣言する。

【００２３】

メタデータプロパティの値は、アクティビティが構成されたスケジュールのいくつかのインスタンスの範囲において、そのアクティビティのすべてのインスタンスにより共有される。例えば、ユーザがスケジュールＡを作成し、送信アクティビティをそれに追加した場合、送信アクティビティはそのメタデータの一部として識別情報（例えば、「００１」）を与えられる。スケジュールに追加される第２の送信アクティビティは、その独自の一意的な識別情報（例えば、「００２」）を受け取ることになる。スケジュールＡの複数のインスタンスが作成され、実行されると、送信「００１」のすべてのインスタンスはメタデータ値を共有する。対照的に、アクティビティのインスタンスデータは、実行スケジュールインスタンス内のアクティビティのインスタンスに特有のデータの集合を定義する。例えば、遅延アクティビティは、遅延アクティビティのタイムアウト値を表す日時値であるインスタンスデータに関する読み取り専用プロパティを与えることができる。この値は、遅延アクティビティが実行を開始した後利用可能になり、遅延アクティビティの１つ１つのインスタンスについて十中八九異なる。参照を「インスタンス」で修飾せずに、スケジュールのインスタンス、特にアクティビティおよびタスクのインスタンスを参照するのがふつうである。

【００２４】

複合アクティビティは、その子アクティビティの集合を他の要素として持つ。子アクティビティは、一実施形態ではメタデータと考えられる。オーケストレーションエンジンモデルでは、スケジュールのインスタンス内で実行時にこのメタデータを操作できることを明示している。新しい子アクティビティを実行スケジュールインスタンスの一部である複合アクティビティに追加し、そのスケジュールインスタンスに対するメタデータ（アクティビティツリー）のみが影響を受けるようにすることが可能である。

【００２５】

次に図３を参照すると、それぞれのアクティビティは、そのアクティビティに対するコンポーネントモデルを形成する関連付けられたコンポーネントの集合を持つ。関連付けられたコンポーネントの集合は、アクティビティエグゼキュータ（activity executor）、アクティビティデザイナ、アクティビティシリアライザ（activity serializer）、アクティビティバリデータ（例えば、意味チェッカー（semantic checker））、およびアクティビティコードジェネレータを含む。アクティビティエグゼキュータは、アクティビティに対する実行意味論を実装するステートレスコンポーネントである。アクティビティエグゼキュータは、アクティビティを実装するためのアクティビティのメタデータを操作する。コアスケジューラは、アクティビティエグゼキュータのサービスプロバイダとして機能し、アクティビティエグゼキュータにサービスを提供する。

【００２６】

アクティビティデザイナは、アクティビティ設計時のビジュアル表現を視覚的に表示する。アクティビティデザイナは、デザイナ階層内の１つのノードであり、テーマまたはスキンを作成して設定できる。アクティビティデザイナは、設計環境（例えば、アプリケーションプログラム）でホスティングされ、サービスを介してホスト設計環境とやり取りする。アクティビティバリデータでは、コンパイル時だけでなく実行時にもアクティビティ意味論を課す。アクティビティバリデータは、ワークフローモデルのコンテキストに作用し、環境が提供するサービス（例えば、コンパイラ、デザイナ、またはランタイム）を使用する。妥当性検証は、ワークフローのライフサイクルの各時点で実行される。構造準拠検査は、ワークフローのシリアライゼーション表現を作成するときに、コンパイルするときに、およびユーザの要求に対する応答として実行される。意味検査は、コンパイル時に実行するよりも実行時の方が強く、このため実行インスタンスのアクティビティツリー内のアクティビティの追加または置換などの実行時オペレーションの安全性を保証することができる。本発明では、例えば、定義済みインターフェイス要件への適合または準拠に関してアクティビティのそれぞれに関連付けられた意味論を評価する。

【００２７】

アクティビティシリアライザは、アクティビティのメタデータをシリアライゼーションするコンポーネントである。アクティビティシリアライザは、さまざまなモデル／フォーマットシリアライザから呼び出される。ワークフローモデル全体が、拡張可能スキーマに基づいて宣言的マークアップ言語内にシリアライゼーションされ、さらに、望むとおりに他のワークフロー言語に変換することができる。

【００２８】

一実施形態では、アクティビティのコンポーネントモデルは、コンピュータ読み取り可能媒体上にデータ構造体として格納される。データ構造体において、アクティビティデザイナは、アクティビティを視覚的に表現するためデータ（例えば、アイコン）を格納する画像フィールドにより表される。さらに、１つまたは複数の作者時刻フィールドでは、アクティビティに関連付けられているプロパティ、メソッド、およびイベントを定義するメタデータを格納する。アクティビティシリアライザは、作者時刻フィールドに格納されているメタデータをアクティビティの宣言的表現に変換するためのデータを格納するシリアライザフィールドにより表される。アクティビティジェネレータは、作者時刻フィールドに格納されているメタデータに関連付けられているソフトウェアコードを格納するビジネスロジックフィールドにより表される。アクティビティエグゼキュータは、ビジネスロジックフィールドに格納されているソフトウェアコードを実行するためのデータを格納する

エグゼキュータフィールドにより表される。

【００２９】

（スコープとスケジュール）

実行アクティビティの実行コンテキストおよび実行アクティビティから見える共有データは、スコープにより規定される。スコープとは、コアアクティビティのうちの１つである。スコープは、変数および長時間実行しているサービスの状態をトランザクション意味論、エラー処理意味論、補正、イベントハンドラ、およびデータ状態管理とともにひとまとめにするための統一言語構文である。スコープは、関連付けられた例外およびイベントハンドラを持つことができる。一実施形態では、スコープは、トランザクション、アトミック、長時間実行、または同期化のスコープとすることができる。ユーザ変数に対するｒｅａｄ－ｗｒｉｔｅまたはｗｒｉｔｅ－ｗｒｉｔｅ　アクセスが衝突する場合にユーザのために同時実行制御が用意される。スコープは、さらに、トランザクション境界、例外処理境界、および補正境界でもある。スコープはスケジュール内でネストすることができるので、さらに、名前が衝突することなく異なるスコープ（スコープがネストしているとしても）内で同じ名前により変数、メッセージ、チャネル、および相関関係集合を宣言することが可能である。

【００３０】

スケジュール内にネストされているスコープは、そのスケジュールのコンテキスト内でのみ実行可能である。スケジュールは、アプリケーション（例えば、スタンドアロンの実行可能エンティティ）として、またはライブラリ（例えば、他のスケジュールから呼び出すため）としてコンパイルすることができる。ライブラリとしてコンパイルされたスケジュールはすべて、実際に、他のスケジュール内から呼び出すことができる新しいアクティビティ型を構成する。スケジュールのメタデータは、パラメータの宣言を含む。

【００３１】

スケジュールが作成された後、作成されたスケジュールのインスタンスを実行できる。スケジュールインスタンスをアクティブにし、制御するプロセスは、オーケストレーションエンジンが埋め込まれているホスト環境に応じて異なる。オーケストレーションエンジンは、スケジュールをテストするために使用できる余計な機能のない「単純なホスト」を提供する。さらに、オーケストレーションエンジンは、サービス環境（つまり、ホスト）とやり取りするため、エンジンおよび外部アプリケーションにより同様に使用される「サービスプロバイダ」モデル（例えば、アプリケーションプログラミングインターフェイス）の標準化を推進するアクティベーションサービスを提供する。アクティベーションサービスは、特定のスケジュール型のスケジュールインスタンスを作成し、その際に、任意選択で、パラメータを受け渡す。スケジュールインスタンスは、本質的に、実行スケジュールインスタンスのプロキシであり、インスタンス、スケジュールのメタデータ（アクティビティツリー）への参照、およびインスタンスのサスペンド、レジューム、および終了を実行するメソッドを一意に識別する識別子を含む。アクティベーションサービスは、さらに、与えられたスケジュールインスタンス識別子に基づいてスケジュールインスタンスを見つける操作もサポートする。

【００３２】

（コードビサイド）

スコープアクティビティは、スコープアクティビティのビジネスロジックを含む関連するコードビサイドクラスを持つことができる。スケジュールはそれ自体スコープであるため、スケジュールはコードビサイドクラスも持つことができる。スケジュール内にネストされているスコープも、それ独自のコードビサイドクラスを持つことができる。スコープ内でネストされているアクティビティは、共有データ状態およびビジネスロジックのコンテナとして働くスコープのコードビサイドクラスを共有する。例えば、コードアクティビティのメタデータは、コードビサイド内に特定のシグネチャを持つメソッドへの参照を含む。他の実施例では、送信アクティビティのメタデータは、特定のシグネチャのコードビサイドメソッドへのオプションの参照に加えてメッセージ宣言およびチャネル宣言への必

須参照を含む。

【００３３】

コードビサイドの使用例としては、変数、メッセージ、チャネル、および相関関係集合の宣言、ｉｎ／ｏｕｔ／ｒｅｆパラメータの宣言、追加カスタムプロパティの宣言、送信するメッセージの準備、受信されたメッセージの処理、論理値を返すコードで表現されたルールの実装、ローカルで定義されている変数の操作、アクティビティメタデータおよびインスタンスデータの読み取り、アクティビティインスタンスデータの書き込み（例えば、実行されようとしているアクティビティに関するプロパティの設定）、イベントの発生、例外のスロー、横断的なネストされているスコープおよびスケジュール呼び出し境界を含む実行スケジュールインスタンスのアクティビティツリー内のアクティビティの階層の列挙およびナビゲート、実行スケジュールインスタンス内で複合アクティビティへの新規アクティビティの追加、実行スケジュールインスタンス内のアクティビティに関連付けられている宣言的ルールの変更、ならびに他の実行スケジュールインスタンスへの参照の取得および他の実行スケジュールインスタンスの操作が挙げられる。

【００３４】

図４を参照すると、ブロック図はコンポーネントモデルのライフサイクルの例を示す。ユーザは、１つまたは複数のコンピュータ読み取り可能媒体に格納されているコンピュータ実行可能コンポーネントを対話形式で操作し、ワークフローに関係するユーザコードを参照しつつワークフローを実行する。コンピュータ実行可能コンポーネントは、コンパイラコンポーネント４０２、ワークフローコンポーネント４０４、デザイナコンポーネント４０６、およびインターフェイスコンポーネント４０８を含む。インターフェイスコンポーネント４０８は、ユーザからユーザコードを受け取る。コンパイラコンポーネント４０２は、ユーザコードを実行可能オブジェクトコードにコンパイルを行うか、または他の何らかの方法で翻訳する。ワークフローコンポーネント４０４は、コンパイルされていないワークフローをコンパイラコンポーネント４０２から得られた実行可能オブジェクトコードで実行する。デザイナコンポーネント４０６を使用すると、ユーザはコンパイルされていないワークフローを動的に修正することができる。つまり、デザイナコンポーネント４０６を使用すると、コンパイルされていないワークフローがワークフローコンポーネント４０４により実行されている間に、ユーザがコンパイルされていないワークフローを修正できるということである。さらに、本発明では、修正されたワークフローを遂行または実行する。コンパイルされていないワークフローを実行し、修正されたワークフローを実行することは、ワークフローをコンパイルせずに実行することを含む。

【００３５】

一実施形態では、ワークフローは、それに関連付けられたアクティビティエグゼキュータをそれぞれ備える複数のアクティビティを含む。このような一実施形態では、ワークフローコンポーネント４０４は、（例えば、コンパイルされたユーザコードを参照しつつ）アクティビティのそれぞれについてアクティビティエグゼキュータを実行することによりコンパイルされていないワークフローを実行する。一実施形態では、１つまたは複数のコンピュータ読み取り可能媒体は、本明細書で説明されているメソッドを実行するコンピュータ実行可能命令を格納する。

【００３６】

図４に示されているようなシステムは、ワークフローに関係するユーザコードを参照しつつワークフローを実行する。特に、このシステムは、コンパイルされていないワークフローおよびユーザコードを格納するためのメモリ領域を備える。さらに、このシステムは、メモリ領域内に格納されているユーザコードをコンパイルするためのコンピュータ実行可能命令と、コンパイルされていないワークフローをコンパイルされたコードで実行するためのコンピュータ実行可能命令と、コンパイルされていないワークフローが実行されている間にユーザがコンパイルされていないワークフローを動的に修正できるようにするためのコンピュータ実行可能命令を実行するように構成されたプロセッサを備える。図４のハードウェア、ソフトウェア、およびシステム例は、メモリ領域内に格納されているユー

ザコードをコンパイルする手段の実施例と、コンパイルされていないワークフローをコンパイルされたコードで実行する手段の実施例と、コンパイルされていないワークフローが実行されている間にユーザがコンパイルされていないワークフローを動的に修正できるようにする手段の実施例の構成要素となる。

【００３７】

（ワークフローステンシル（Workflow Stencils））

ワークフローステンシル（例えば、ワークフローテンプレートまたはアクティビティパッケージ）は、ルートアクティビティおよびアクティビティの集合を含む。ステンシルは、領域および／またはホスト固有である。前者の実施例としては、構造化ワークフローステンシル、ヒューマンワークフローステンシル、および非構造化ワークフローステンシルがある。いくつかのステンシルは、場合によっては特定のホスト環境内で連携するように設計されている１つまたは複数のルートを含むアクティビティの集合として「閉じて」いてもよい。他のステンシルは、さまざまな程度で「開いて」いてもよい。ステンシルは、その拡張性ポイントを定める。例えば、開発者は、ＣｕｓｔｏｍＲｏｏｔおよび新しい抽象ＣｕｓｔｏｍＡｃｔｉｖｉｔｙを書いて、パッケージがＣｕｓｔｏｍＲｏｏｔであり、さらにＣｕｓｔｏｍＡｃｔｉｖｉｔｙから派生したアクティビティであると宣言する。

【００３８】

ＢＰＥＬまたはＸＬＡＮＧ／Ｓステンシルの実施例は、状態管理およびトランザクションに参加する、関連付けられたイベントおよび例外ハンドラを備える、コントラクトファーストモデル（contract first model）をサポートする、分析できる、正しく定義されたアクティベーションおよび終了動作を持つ、という特性を有するルートアクティビティを含む。ステンシルの実施例は、さらに、メッセージング特有のアクティビティの集合（例えば、ＳｅｎｄおよびＲｅｃｅｉｖｅおよびその変種）およびＳｃｏｐｅ、Ｌｏｏｐ、Ｃｏｎｄｉｔｉｏｎ、Ｌｉｓｔｅｎ、およびＴｈｒｏｗなどの他の構造化アクティビティを含む。

【００３９】

Ｈａｌｉｆａｘ　Ｓｔｅｎｃｉｌの実施例は、暗黙の状態管理関連の例外ハンドラ（０～ｎ）、イベントベースのモデルをサポートする、適切に定義されたアクティベーション動作を持つ、および未定義の終了がある、という特性を備えるルートアクティビティを含む。ルートアクティビティは、０～ｎのＥｖｅｎｔＤｒｉｖｅｎアクティビティを含む。それぞれのＥｖｅｎｔＤｒｉｖｅｎ　Ａｃｔｉｖｉｔｙは、Ｈａｌｉｆａｘ　Ａｃｔｉｏｎを表す。それぞれのＥｖｅｎｔＤｒｉｖｅｎ　Ａｃｔｉｖｉｔｙは、関連付けられた状態管理プロトコルを有し、アトミックスコープ内で実行される。

【００４０】

（デザイナフレームワーク（ユーザインターフェイス））

オーケストレーションエンジンは、ＷＹＳＷＹＧ方式でさまざまなクラスのワークフローモデルを設計するためのフレームワークを備える。例えば、図５を参照すると、ワークフローオーサリングするための高水準アプリケーションユーザインターフェイスは、ワークフローの指定をウィザードに依存している。このフレームワークは、開発者がビジュアルワークフローデザイナを書けるようにするサービスおよび動作の集合を含む。これらのサービスは、ワークフロープロセスのレンダリング、フローを描画するためのインク／タブレットのサポート、および元に戻す／繰り返し、ドラッグ／ドロップ、切り取り／コピー／貼り付け、ズーム、パン、検索／置換、ブックマーク、装飾、妥当性検証エラー用のスマートタグ、アクティビティ用の有効なドロップターゲットインジケータ、自動レイアウト、ビューページ付け、ナビゲーションマーカー、ドラッグインジケータ、ヘッダ／フッタ付き印刷およびプレビューなどの効率的方法を提供する。このようなユーザインターフェイスを使用することで、タスクおよび制御フロー複合アクティビティ（例えば、シーケンス、パラレル、および条件付き）を含む単純なワークフローを構成することができる。ルール指定（例えば、条件付き分岐ロジック、ｗｈｉｌｅループロジック）、またはデータフロー指定（例えば、タスクＡの出力がタスクＢへの入力である）のいずれにも、コ

ードの入力（または既存のコンパイル済みコードへの依存）は不要である。（ルールおよびデータフローを含む）スケジュールのシリアライゼーションされた表現は、コードビサイドが必要ないいくつかのシナリオにおいて自己充足しており、完全である。

【００４１】

本発明のデザイナフレームワークを使用することで、本発明のオーケストレーションエンジンは、ソフトウェアコードをビジュアルな形でワークフローモデルに関連付ける機能をサポートするアプリケーション短期開発（ＲＡＤ）スタイルのビジュアルワークフローデザイナを含む。ワークフロー内のそれぞれのアクティビティは、関連付けられたアクティビティデザイナを備える。それぞれのアクティビティデザイナは、フレームワークサービスに関して作成される。本発明のフレームワークは、さらに、ビジュアルデザイナモデルも含む。ビジュアルデザイナモデルは、ワークフローモデルで記述された関係を介して互いにリンクされているアクティビティデザイナの集合を含む。図６は、ワークフローデザイナの実施例を示す。本発明は、ユーザコードとワークフローモデルとの間のラウンドトリッピングをリアルタイムで可能にする「Ｃｏｄｅ－Ｂｅｓｉｄｅ」、「Ｃｏｄｅ－Ｗｉｔｈｉｎ」、および「Ｃｏｄｅ－Ｏｎｌｙ」を含むコードをワークフローモデルに関連付けるさまざまなモードを含む。本発明は、さらに、ユーザがワークフロー構築している最中にリアルタイムで意味論的エラーを提示する。

【００４２】

一実施形態では、本発明は、デザイナフレームワークのユーザインターフェイス内の複数のアクティビティを識別するパッケージをユーザに提示する。本発明は、さらに、ユーザから、提示されたアクティビティの選択および階層的編成結果を受け取る。本発明では、受け取ったアクティビティをシリアライゼーションし、ワークフローの永続的表現を作成する。本発明は、さらに、ユーザから、ワークフロー内の複数のアクティビティのうちの１つとの関連付けのためのビジネスロジックを表すソフトウェアコードを受け取る。本発明は、さらに、関連付けられている１つまたは複数の意味論を持つユーザ定義アクティビティも受け取ることができる。本発明は、定義済みインターフェイス要件への適合に関して意味論を評価する意味チェッカーまたはバリデータを含む。意味論が定義済みインターフェイス要件に適合している場合、本発明では、ユーザ定義アクティビティを複数のアクティビティのうちの１つとして提示する。本発明は、さらに、そのソフトウェアコードをコンパイルして１つまたは複数のバイナリファイルを作成する。例えば、本発明は、シリアライゼーションされたワークフロー表現およびソフトウェアコードをワークフローの実行可能表現を含む単一のアセンブリにコンパイルする。本発明は、作成されたワークフローを実行する。一実施形態では、１つまたは複数のコンピュータ読み取り可能媒体は、メソッドを実行するコンピュータ実行可能命令を持つ。

【００４３】

オーケストレーションエンジンデザイナによって、ユーザは、他の作成済みスケジュールを使用し、それらを使用して上位のスケジュールを再帰的に作成することができる。スケジュールのインライン展開では、ユーザはスケジュールコンテンツをインラインで表示し、コンテンツを切り取ったりコピーしたりすることができる。スケジュールのインライン展開を有効にし、スケジュールを読み取り専用にするには、インラインスケジュールに対して別のデザインサーフェスおよびデザイナホストを作成する。さらに、複合スケジュールデザイナはそれ独自の階層を持つ。呼び出されたスケジュールはロードされ、ユーザによってデザイナが展開されたときに表示される。一実施形態では、デザイナは、アクティビティがデザインサーフェス上でアクティビティドロップまたはコピーされた場合に縮小する。プロパティは、呼び出し側アクティビティデザイナをホスティングされたスケジュールのルートデザイナに連鎖する。下記の関数は、デザイナからアクティビティの追加および削除を行うことを禁止する。

```
internal static bool AreAllComponentsInWritableContext(ICollection components)
internal static bool IsContextReadOnly(IServiceProvider serviceProvider)
```

【００４４】
これらの関数は、アクティビティが挿入されるコンテキストが書き込み可能かどうかをチェックするためにインフラストラクチャにより呼び出される。ホスティングされたデザイナでは、これらの関数は偽を返す。さらに、プロパティの修正が禁止される。他の関数は、適切なコンポーネントからアクティビティデザイナをフェッチする。

```
internal static ServiceDesigner GetSafeRootDesigner(IServiceProvider service
Provider)
internal static ICompositeActivityDesigner GetSafeParentDesigner(object obj)
internal static IActivityDesigner GetSafeDesigner(object obj)
```

【００４５】
一実施例において、ユーザは、スケジュールを作成し、それをアクティビティとしてコンパイルする。コンパイルに成功すると、スケジュールはツールボックス上に表示される。ユーザは、コンパイルされたスケジュールの使用が望ましいスケジュールを開くか、または作成する。ユーザは、ツールボックスからコンパイル済みスケジュールをドラッグ＆ドロップする。縮小されたスケジュールデザイナがデザインサーフェスに示される。ユーザがドロップされたコンパイル済みスケジュールのコンテンツを表示したい場合、ユーザはそのスケジュールデザイナを展開して、呼び出されたスケジュールのコンテンツを読み取り専用状態でインライン表示する。呼び出されたスケジュールのインライン化により、ユーザはスケジュールデザイナ同士を切り換えなくても呼び出されたスケジュールを表示することができる。この機能は、既存のスケジュールを再利用して上位スケジュールを作成する開発者に有用な機能である。

【００４６】
（テーマ／スキンを使用したデザイナフレームワークのカスタマイズのサポート）
デザイナフレームワークを使用して書かれたワークフローデザイナは、ワークフローテーマを使用してカスタマイズすることができる。これらは、デザイナのさまざまな態様を宣言的に記述する拡張マークアップ言語（ＸＭＬ）ファイルとすることができる。ワークフローデザイナは、アクティビティを拡張するパートナのサポートをウィザードで対応している。ワークフローデザイナによりサポートされるユーザインターフェイス機能の例として、限定はしないが、元に戻す／繰り返し、ドラッグ／ドロップ、切り取り／コピー／貼り付け、ズーム、パン、検索／置換、ブックマーク、装飾、妥当性検証エラー用のスマートタグ、アクティビティ用の有効なドロップターゲットインジケータ、自動レイアウト、ビューページ付け、ナビゲーションマーカー、ドラッグインジケータ、ヘッダ／フッタ付き印刷およびプレビュー、およびドキュメントアウトライン統合がある。ワークフローデザイナは、ＸＭＬメタデータを使用してデザイナのルック＆フィールをカスタマイズできるカスタムデザイナテーマ／スキンをサポートする。ワークフローデザイナでは、バックグラウンドコンパイルをサポートする。一実施例では、スケジュールを設計しながら妥当性検証エラーを調べるスマートタグおよびスマートアクションが用意される。ワークフローデザイナは、任意のコンテナ（例えば、アプリケーションプログラム、シェルなど）でホスティングすることができる。

【００４７】
オーケストレーションエンジンプログラムの一実施例は、送信アクティビティが後に続く受信アクティビティを含む。このプロセスでは、メッセージを受信し、それを送出する。ユーザは、「Ｈｅｌｌｏ　Ｗｏｒｌｄ」と呼ばれるプロジェクトを作成し、オーケストレーションアイテムをプロジェクトに追加する。ユーザは、その後、スコープアクティビティをデザインサーフェス上にドラッグ＆ドロップする。次に、ユーザは、送信アクティビティが後に続く受信アクティビティをスコープ上にドロップする。図６は、デザイナ内の結果として得られるワークフロー７００を例示している。それぞれのアクティビティデザイナは、オブジェクトモデル上のユーザインターフェイス表現を提供する。開発者は、オブジェクトモデルを直接プログラムして、アクティビティのプロパティを設定するか、またはデザイナを使用することができる。オーケストレーションエンジンデザイナを使用

することにより、開発者はツールボックスからアクティビティを選択し、それをデザイナサーフェス上にドラッグすることができる。アクティビティがすでにスケジュールに入れられており、移動する必要がある場合、開発者は、それを選択して（それをクリックすることにより）、そのアクティビティを入れる必要のあるスケジュールの領域にドラッグすることができる。開発者がコントロールキーを押しながらドラッグ＆ドロップを行うと、選択されたアクティビティのコピーが作成される。

【００４８】

アクティブな配置では、可能なドロップポイント（ターゲット）をデザインサーフェス上にビジュアルインジケータとして用意する。自動スクロール機能も、ドラッグ＆ドロップのコンテキスト内で関与する。大きなスケジュールを取り扱う場合、現在ビューポート内にないデザイナの領域へのナビゲーションは、入れるスケジュールの領域に向かってアクティビティをドラッグすることによりアクセス可能である。

【００４９】

ドラッグ＆ドロップは、同じプロジェクト内のスケジュール間で、また同じソリューション内の他のプロジェクト内のスケジュール間でサポートされる。アクティビティがデザインサーフェス上に配置された後、開発者はそのアクティビティの構成を設定する。各アクティビティは、スケジュールが有効なものとなるように開発者が構成する一連のプロパティを持つ。これらのプロパティは、プロパティブラウザで編集可能である。すべてのアクティビティは、プロパティブラウザ内にどのようなプロパティを表示できるかを制御する。開発者がさまざまなアクティビティを構成するのを補助するため、デザイナは、さまざまなダイアログまたは「サブデザイナ」を備えている。ダイアログの各々は、アクティビティのさまざまなプロパティについて呼び出される。

【００５０】

オーケストレーションエンジンは、ツールボックス内に提示されるアクティビティをカスタマイズすることができる。開発者がカスタムアクティビティまたはスケジュールを作成する場合、最終結果はアセンブリである。ダイアログを使用することにより、開発者は、アセンブリロケーションに移動してブラウズし、そのアセンブリを選択してオーケストレーションエンジンアクティビティとして現れるようにすることができる。それとは別に、開発者は、オーケストレーションエンジンのインストールパス内にそのアセンブリを置くことができ、オーケストレーションエンジンアクティビティとして存在することになる。

【００５１】

（アプリケーションプログラミングインターフェイス（ＡＰＩ））

他の実施形態では、本発明は、さまざまなワークフローオペレーションを実行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イス（ＡＰＩ）を備える。本発明は、ワークフローをオーサリングするためのデザインアプリケーションプログラミングインターフェイスを含む。デザインアプリケーションプログラミングインターフェイスは、ワークフローをオーサリングする手段およびアクティビティのうちの１つまたは複数を選択してワークフローを作成する手段を備える。本発明は、さらに、デザインアプリケーションプログラミングインターフェイスを介してオーサリングされたワークフローをコンパイルするためのコンパイルアプリケーションプログラミングインターフェイスも含む。コンパイルアプリケーションプログラミングインターフェイスは、ワークフローをシリアライゼーションする手段、ワークフローの視覚的な表示をカスタマイズする手段、デザインアプリケーションプログラミングインターフェイスを介してオーサリングされたワークフローをコンパイルする手段、ワークフローの妥当性確認を行う手段を含む。

【００５２】

本発明は、さらに、型をワークフロー内のアクティビティのそれぞれに関連付けるためのタイププロバイダアプリケーションプログラミングインターフェイスも含む。タイププロバイダアプリケーションプログラミングインターフェイスは、型をワークフロー内のアクティビティのそれぞれに関連付ける手段および型をワークフロー内のアクティビティの

それぞれに関連付ける手段を含む。

【００５３】

１つまたは複数のアプリケーションプログラミングインターフェイスは、ワークフローをオーサリングする手段の実施例、アクティビティの１つまたは複数を選択してワークフローを作成する手段の実施例、ワークフローをシリアライズする手段の実施例、ワークフローの視覚的表示をカスタマイズする手段の実施例、ワークフローの妥当性確認を行う手段の実施例、ワークフローをコンパイルする手段の実施例、および型をワークフロー内のアクティビティのそれぞれに関連付ける手段の実施例を含む。

【００５４】

（アクティビティ実行フレームワーク）

スケジュールおよびスコープを除き、エンジンは、アクティビティを抽象エンティティとして表示し、特定のアクティビティの特定のデータまたは意味論を知らずにアクティビティの実行を簡単に調整する。一実施形態では、４つのエンティティは、アクティビティ自体、実行中のアクティビティの親アクティビティ、実行中のアクティビティを取り囲むスコープ、およびオーケストレーションエンジンの実行中にやり取りをする。それぞれのエンティティは異なる関数を持つ。

【００５５】

完了をアクティビティコーディネータに知らせることなくアクティビティの実行メソッドが戻る場合、そのアクティビティは、論理的待ち状態にあると識別される。このようなアクティビティは、オーケストレーションエンジンによりキャンセルするか、または継続することができる（例えば、待っているアイテムまたはイベントが利用可能になるか、または実行され、エンジンによりこれにアクティビティが通知された後）。

【００５６】

論理的待ち状態に決して入らないいくつかのアクティビティは、決してキャンセルできない。実施例は、送信アクティビティおよびコードアクティビティを含むが、これらは外部イベントまたはサブスクリプションへの要求なしで実行されるからである。スレッドを渡された後（つまり、実行メソッドがオーケストレーションエンジンにより呼ばれた後）、これらのアクティビティは完了するまで動作し続ける。完了を通知するまでスレッドを返さないので、オーケストレーションエンジンにはキャンセルする機会が決して与えられない。

【００５７】

オーケストレーションエンジンのランタイムは、ルールを使用して、オーケストレーションエンジンアクティビティが実行されるイベントをトリガする。オーケストレーションエンジンデザイナは、関連付けられたルールを実行時に評価し、イベントを取り出すユーザ機能を提供する。オーケストレーションエンジンデザイナを利用することで、ユーザは、拡張性アーキテクチャを用意することによりさまざまな種類のルール技術を使用できる。デザイナは、使用されるルールの型には関知しない。

【００５８】

一実施形態では、デザイナは、ルールをアクティビティに関連付ける手段として論理式ハンドラをサポートしている。これは、ユーザコードファイル内で、ユーザが真または偽の値を返すメソッドを書くことを意味し、それに基づいてルールがトリガされる。現在、Ｉｎｆｏ ＡｇｅｎｔおよびＢｕｓｉｎｅｓｓ Ｒｕｌｅｓ Ｅｎｇｉｎｅ（ＢＲＥ）を含むルールを評価するために使用することもできる複数の技術がある。これを達成するために、デザイナは、ルール技術開発者がデザイナでカスタムユーザインターフェイスをホスティングできるようにする拡張性アーキテクチャを備える。デザイナは、コードステートメントコレクションの形でルールをシリアライゼーションするカスタムユーザインターフェイスライター向けの手段を用意している。デザイナは、コードステートメントコレクションが中に挿入されているユーザコードファイル内にブールハンドラを生成し出力する。オーケストレーションエンジンは、ルールライターにより使用することもできる既定のユーザインターフェイスを含む。ルール技術プロバイダは、カスタムルール宣言を作成し

、そのカスタムルール宣言に関連付けられたユーザインターフェイスタイプエディタを書き、ルールユーザインターフェイスをホスティングするカスタムユーザインターフェイスを作成し、保存後にコードステートメントを生成することによりオーケストレーションエンジンデザイナにルールを追加する。

【００５９】

一実施例では、ユーザは、ルールがアタッチされる必要のあるアクティビティデザイナを選択し、プロパティブラウザでルールプロパティを特定し、ドロップダウン内の「ＲｕｌｅＥｘｐｒｅｓｓｉｏｎＨａｎｄｌｅｒ」を選択し（「Ｓｔａｔｅｍｅｎｔｓ」プロパティがユーザインターフェイス内のＲｕｌｅ　Ｐｒｏｐｅｒｔｙの下に表示されるようにする）、「Ｓｔａｔｅｍｅｎｔｓ」プロパティ内のユーザコードメソッド名を指定し、ユーザインターフェイスタイプエディタを呼び出してルール特有のユーザインターフェイスをホスティングするダイアログを呼び出し、新しい述語行を作成しそれらをグループ化することによりダイアログ内でルールを定義する。ユーザインターフェイスは、ユーザコードファイルにメソッドを生成して出力する。メソッド名は、プロパティブラウザでユーザが指定したのと同じものである。ルールを作成することに相当するコードステートメントが、ルール用のユーザコードメソッド内に挿入される。

【００６０】

（実行時のメッセージング）

実行ワークフローでは、スケジュールに送られるメッセージは、特定のスケジュールインスタンス向けである。例えば、発注書＃１２３に対するインボイスをその発注書を発信した（例えば、送出した）のと同じスケジュールインスタンスに送り返さなければならない。受信メッセージと適切なスケジュールインスタンスとを照合するために、メッセージおよびスケジュールインスタンスは相関関係集合を共有する。相関関係集合は、メッセージ内の識別子フィールドがスケジュールインスタンスにより保持される同じ型の識別子と突き合わせて照合されることを意味する一価相関関係集合でよい。多プロパティ相関関係集合も可能であり、データベーステーブル内の複数列主キーに類似している。

【００６１】

スケジュールインスタンスにより保持されている相関関係集合の値は、スケジュールインスタンスがメッセージを送出するか（例えば、値は送信発注書の識別子フィールドから取り出せる）、またはメッセージを受信するときに初期化される。その後、この相関関係集合値は、そのスケジュールインスタンスの状態の一部となる。後続の受信メッセージが到着すると、スケジュールインスタンス状態で保持されている相関関係集合値は、予想される型の受信メッセージにより保持されている識別子と突き合わせて照合される。一致が見つかった場合、相関関係集合は満たされ、メッセージがスケジュールインスタンスに配送される。

【００６２】

相関関係集合の実装はオーケストレーションエンジンおよびホスト環境に左右されるが、一実施形態では、ユーザは相関関係集合を宣言してスケジュールインスタンスを正しく動作するようにする。他の実施形態では、いくつかのアクティビティ（例えば、ＳｅｎｄＲｅｑｕｅｓｔ／ＲｅｃｅｉｖｅＲｅｓｐｏｎｓｅアクティビティおよびＲｅｃｅｉｖｅＲｅｑｕｅｓｔ／ＳｅｎｄＲｅｓｐｏｎｓｅアクティビティ）は、ユーザと無関係に相関関係集合をセットアップする。送信および受信アクティビティにより広範にわたる妥当性チェックが実行され、相関関係集合が正しく初期化され、追随されるようにする。

【００６３】

（実行ワークフローの動的編集）

オーケストレーションエンジンは、さまざまな種類のワークフローをオーサリングする（そして、その後、ビジュアル化し、実行する）ためのフレームワークを提供する。実施例としては、ｅｖｅｎｔ－ｃｏｎｄｉｔｉｏｎ－ａｃｔｉｏｎ（ＥＣＡ）スタイルのワークフローまたは構造化フローまたはルール駆動フローがある。さらに、ワークフローのモデル化方法に関係なく、フレームワークにより、ユーザは設計時と同じ方法で、またはワ

ークフロープロセスを再コンパイルしなくてもワークフロープロセスが実行しているときであってもワークフローをオーサリングまたは編集することができる。フレームワークを使用すると、ユーザはランタイムと高忠実度の設計時表現との間のラウンドトリッピングが可能になる。アドホックな変更は、プロセスモデルに対し実行時に加えられる変更である。ユーザは、そのスケジュールモデルを実行インスタンスに問い合わせ、モデルに変更を加えることができる。例えば、ユーザは、バッチ式でアクティビティの追加、削除、または置換を行うことができ、その後、バッチ変更をコミットまたはロールバックすることができる。一実施形態では、モデルは更新の後に妥当性確認が行われる。本発明の多くのワークフローシナリオでは、「設計時オーサリング」および「ランタイム実行」との間の分離のぼかし、またはさらには除去がある。

【００６４】

スケジュールインスタンスは、実際に、それらのインスタンスのスケジュール型について定義されているアクティビティ型（メタデータ）ツリーを他のインスタンスと共有する。しかし、スケジュールインスタンスは、実行を開始した後、新しいアクティビティの追加または宣言的ルールの操作を介してオンザフライで変更できる。そのような修正されたスケジュールインスタンスを取り出し、新しいスケジュール型として「名前を付けて保存」するか、またはより一般的に、インスタンスからシリアライゼーションされた表現を単純に復元することが可能である。つまり、実行スケジュールインスタンスを、シリアライゼーションし、その後、デザイナ（例えば、オーサリング環境）またはランタイムビジュアル化ツールに持ち込むことができる。

【００６５】

さらに、上級開発者は、スケジュールを丸ごとソフトウェアコードとしてオーサリングすることも可能である。スケジュール型を直接オーサリングする場合も、開発者は、単に、スケジュールのコードビサイドクラスのソフトウェアコードにＩｎｉｔｉａｌｉｚｅＳｃｈｅｄｕｌｅＭｏｄｅｌという静的メソッドを入れ、このメソッドに［ＳｃｈｅｄｕｌｅＣｒｅａｔｏｒ］属性でマークするだけである。一実施形態では、静的メソッドは、パラメータを取らず、Ｓｃｈｅｄｕｌｅオブジェクトを返す。随伴するシリアライゼーションされたファイルはないが、スケジュールのシリアライゼーションされた表現は、作成されるＳｃｈｅｄｕｌｅオブジェクトから復元できる。これは、単一のソフトウェアコードファイルを使用してスケジュールを作成できることを意味しているが、妥当性チェックは、このファイルには実行されえない。オーケストレーションエンジンのコンパイルにより、スケジュール型の基礎となっているアクティビティツリーの構造上および意味論的な有効性が保証される。他の実施形態では、コンパイルおよび妥当性検証を内部で実行し、実行される実際の型を生成するが、コードの入力は不要である。スケジュール型のコンパイルは、コンパイル時オブジェクトモデルから実行時オブジェクトモデルへの変換がない分、超軽量なプロセスとなっている。本質的に、コンパイルは、単に、スケジュールのオブジェクトモデル表現をコードビサイドと組み合わせて新しい型を生成するだけである。一実施形態では、基本的に、コンパイル済みコードビサイドがオブジェクトモデル内のアクティビティにより要求される内容と一致するか、またはコードビサイドがコンパイル済みフォーム（アセンブリ）内にすでに存在している可能性がある場合には、特定のスケジュールについてコードビサイドを全く生成する必要はないと考えられる。

【００６６】

シリアライゼーションされたスケジュールをコンパイルする場合、そのスケジュールに対するコードビサイドとして実際に使用される既存のコンパイル済み型を指すことが可能である。このコンパイル済み型の派生型が作成され、この新しい型はコードビサイドとして使用され、一意的な型が新しいスケジュールを表すように作成されることが保証される。

【００６７】

（シリアライゼーションアーキテクチャ）

シリアライゼーションインフラストラクチャは、オーケストレーションエンジンのアク

ティビティツリーをシリアライゼーションする、モジュール方式の、形式に依存しない、容易に拡張可能なメカニズムを提供する。

【００６８】

特に、呼び出し側（例えば、アプリケーションプログラムまたはユーザ）は、シリアライゼーションマネージャからシリアライザにオブジェクト（またはアクティビティ）Ａを要求する。オブジェクトＡの型のメタデータ属性により、オブジェクトＡは要求された型のシリアライザにバインドされる。次に、呼び出し側は、オブジェクトＡをシリアライゼーションするようにシリアライザに要求する。続いて、オブジェクトＡのシリアライザはオブジェクトＡをシリアライゼーションする。シリアライザは、遭遇するオブジェクト毎に、シリアライゼーションしながら、シリアライゼーションマネージャに追加シリアライザを要求する。シリアライゼーションの結果が呼び出し側に返される。

【００６９】

オーケストレーションエンジンコンポーネントモデル内のすべてのアクティビティは、シリアライゼーションに参加することができる。シリアライザコンポーネントは、一実施形態ではアクティビティクラス自体の一部ではない。その代わりに、このコンポーネントは、アクティビティに関連付けられたクラス内のシリアライザ属性に注釈を入れることにより指定される。シリアライザ属性は、そのアクティビティ型のオブジェクトをシリアライゼーションするために使用されるクラスを指す。他の実施形態では、ある１つのアクティビティ型のプロバイダコンポーネントが、そのアクティビティにより与えられる既定のシリアライザをオーバーライドする。

【００７０】

デザイナシリアライゼーションは、メタデータ、シリアライザ、およびシリアライゼーションマネージャに基づく。メタデータ属性は、型をシリアライザに関係付けるために使用される。「ブートストラッピング」属性は、シリアライザを持たない型に対しシリアライザを提供するオブジェクトをインストールするために使用することができる。シリアライザは、特定の１つの型またはある範囲の複数の型をシリアライゼーションする方法を認識するオブジェクトである。それぞれのデータ形式に対して１つの基本クラスがある。例えば、オブジェクトをＸＭＬに変換する方法を認識するＸｍｌＳｅｒｉａｌｉｚｅｒ基本クラスが考えられる。本発明は、特定のシリアライゼーション形式に関係しない一般的なアーキテクチャである。シリアライゼーションマネージャは、オブジェクトグラフをシリアライゼーションするために使用されるさまざまなシリアライザに対する情報ストアとなるオブジェクトである。例えば、５０個のオブジェクトのグラフは、すべてそれ独自の出力を生成する５０個の異なるシリアライザを持つことができる。これらのシリアライザがシリアライゼーションマネージャを使用して、必要に応じて互いに通信し合うことができる。

【００７１】

一実施形態では、汎用オブジェクトメタデータを使用するシリアライザに結合されたシリアライゼーションプロバイダを使用することにより、与えられた型に対するシリアライザを用意する機会がオブジェクトに与えられるコールバックメカニズムを実現する。ＡｄｄＳｅｒｉａｌｉｚａｔｉｏｎＰｒｏｖｉｄｅｒなどのメソッドを通じて、シリアライゼーションマネージャにシリアライゼーションプロバイダを用意することができる。シリアライゼーションプロバイダは、ＤｅｆａｕｌｔＳｅｒｉａｌｉｚａｔｉｏｎＰｒｏｖｉｄｅｒＡｔｔｒｉｂｕｔｅなどの属性をシリアライザに追加することによりシリアライゼーションマネージャに自動的に追加できる。

【００７２】

一実施形態では、オブジェクトはｘｍｌ要素としてシリアライゼーションされる、オブジェクトのプロパティは単純プロパティ（例えば、ｘｍｌ属性としてシリアライゼーションされる）または複合プロパティ（子要素としてシリアライゼーションされる）として分類される、オブジェクトの子オブジェクトは子要素としてシリアライゼーションされる、というルールに従って形式が示される。子オブジェクトの定義は、オブジェクト毎に異な

ることがある。以下の実施例は、子オブジェクトの１つとしてＳｅｎｄアクティビティを持つ、ｗｈｉｌｅアクティビティのシリアライゼーションである。

【００７３】

【表１】

```
<While ID="while1">
  <ConditionRule>
    <CodeExpressionRuleDeclaration>
      <Expression Name="whileCondition" />
    </CodeExpressionRuleDeclaration>
  </ConditionRule>
  <Send HasTypedChannel="True" ID="send1">
    <Message Name="msg1" Type="System.UInt32" />
    <OnBeforeSend Name="onBeforeSend1" />
    <TypedChannel Type="System.Collections.IList"
Operation="AddIndex" Name="Foo" />
  </Send>
</While>
```

【００７４】

シリアライゼーションに使用される言語がＸＯＭＬである一実施形態では、それぞれのＸＯＭＬ要素は、スケジュールがコンパイルされるときにそれぞれのオブジェクトにシリアライゼーションされる。オブジェクトは、単純型と複合型の両方を含む。各アクティビティのＸＯＭＬ表現の間のマッピングとオーサリングオブジェクトモデルへのマッピングについて、次に説明する。ＸＯＭＬのシリアライゼーションは、プリミティブアクティビティと複合アクティビティとで異なる。

【００７５】

プリミティブアクティビティに対する単純型は、アクティビティ型に関する属性としてシリアライゼーションされる。プリミティブアクティビティに対する複合型は、子要素としてシリアライゼーションされる。例えば、以下に、Ｓｅｎｄアクティビティのｘｏｍｌ表現を示す。

【００７６】

【表２】

```
<Send ID="send1" HasTypedChannel="False">
  <Message Name="message1" Type="System.String" />
  <UntypedChannel Name="c1" />
</Send>
```

【００７７】

プリミティブ型のシリアライゼーションと同様に、複合アクティビティに対する単純型は、アクティビティ型に関する属性としてシリアライゼーションされる。しかし、定義により、複合アクティビティはネストされたアクティビティをカプセル化したものである。それぞれのネストされているアクティビティは、他の子要素としてシリアライゼーションされる。例えば、以下に、Ｗｈｉｌｅアクティビティのｘｏｍｌ表現を示す。

【００７８】

【表３】

```
<While ID="while1">
<ConditionRule>
  <CodeExpressionRule>
    <Expression Name="test" />
</CodeExpressionRule>
</ConditionRule>
</While>
```

【００７９】

プロセス／ワークフロービューとシリアライゼーションされた表現との間に強い関係が存在する。図８は、スケジュール定義およびビジュアルワークフローと、ワークフローのシリアライゼーションされた（例えば、ＸＯＭＬ）表現と、ワークフローのコードビサイドとの間の関係を示す図である。いずれかの表現でオーサリングする場合、他方は変更を招く。そのため、アクティビティ（または複合アクティビティの場合にはその構成要素部分）に対するＸＯＭＬを修正すると、開発者がプロセス／ワークフロービューの切り換えを行うときにプロセス／ワークフロービューに直接反映される。逆も言える。プロセス／ワークフロービュー内のアクティビティを修正すると、ＸＯＭＬ内に適切な修正が生じる。例えば、プロセス／ワークフロービュー内でアクティビティを削除すると、同じアクティビティについてＸＯＭＬ内のＸＭＬ要素も削除される。ラウンドトリッピングは、プロセス／ワークフロービューとコードビサイドとの間にも生じる。

【００８０】

ＸＯＭＬコードを作成するときに、ＸＯＭＬ定義が定義済みインターフェイス要件に適合していない場合、違反しているＸＭＬ要素は、下線が付けられるか、または開発者がそれとわかるように他の何らかの方法で視覚的に識別される。開発者がプロセスビューに切り換えた場合、ＸＯＭＬ内にエラーがあると警告され、デザイナがリンクを表示するので、開発者はそのリンクをクリックして、違反している要素にナビゲートする。この同じエラーは、タスクペインにも表示され、そのエラーをダブルクリックすると、開発者はＸＯＭＬの違反要素にナビゲートされる。

【００８１】

（ＸＯＭＬファイルからのアクティビティツリーの作成（デシリアライゼーション））

一実施形態では、ＣｒｅａｔｅＥｄｉｔｏｒＩｎｓｔａｎｃｅ（）関数が、ＤｅｓｉｇｎＳｕｒｆａｃｅオブジェクトを作成し、その後、ＤｅｓｉｇｎＳｕｒｆａｃｅオブジェクト上でＢｅｇｉｎＬｏａｄ（）関数を呼び出して実際のローダオブジェクトをその中に渡し、最後に、ＢｅｇｉｎＬｏａｄ（）でＤｅｓｉｇｎｅｒＬｏａｄｅｒ（）関数を呼び出す。ＰｅｒｆｏｒｍＬｏａｄ（）関数は、テキストバッファオブジェクトを読み込み、それをオーケストレーションエンジンコンポーネントモデル階層にデシリアライゼーションする。本発明は、階層内を辿り、アクティビティをデザインサーフェスに挿入し、コンポーネントをＶｉｓｕａｌ　Ｓｔｕｄｉｏにロードする。

【００８２】

本発明は、さらに、ＸＯＭＬの変更を監視し、階層およびアイテム識別の変更を追跡して、Ｖｉｓｕａｌ　Ｓｔｕｄｉｏのキャッシュ内の値を更新する。二次ドキュメントデータリストは、オーケストレーションエンジンデザイナが作用する、ユーザには見えない二次ドキュメントのリストを含む。例えば、ユーザはコードビサイドファイルを開いていないが、ユーザがオーケストレーションエンジンデザイナ内で変更を加えた場合に、コードビサイドファイルに変更が加えられる。このファイルはユーザには見えないので、ファイルは二次ドキュメントとして保持されるＸＯＭＬファイルが保存されると、必ず、二次ド

キュメントも自動的に保存される。これらのファイルの１つの名前が変更されるか、またはファイルが削除された場合、本発明では、それに応じて、対応する二次ドキュメントオブジェクトを更新する。

【００８３】

オブジェクトツリーのデシリアライゼーションガイドラインの例を以下に示す。ｘｍｌ要素は、まず、親オブジェクトのプロパティとして取り扱われる。親オブジェクトが要素のタグ名を持つプロパティを持たない場合、その要素は親オブジェクトの子オブジェクトとして取り扱われる。ｘｍｌ要素は、親オブジェクト上の単純プロパティとして取り扱われる。

【００８４】

上記のシリアライゼーションされたコードを使用するデシリアライゼーションの実施例では、＜Ｗｈｉｌｅ＞要素は、ｘｍｌ名前空間情報を使用して作成されたオブジェクトとして取り扱われる。＜ＣｏｎｄｉｔｉｏｎＲｕｌｅ＞要素は、Ｗｈｉｌｅアクティビティのプロパティとして取り扱われる。＜ＣｏｄｅＥｘｐｒｅｓｓｉｏｎＲｕｌｅＤｅｃｌａｒａｔｉｏｎ＞要素は、値がＣｏｎｄｉｔｉｏｎＲｕｌｅプロパティに適用されるオブジェクトとして取り扱われる。最初に、＜Ｓｅｎｄ＞要素がＷｈｉｌｅアクティビティのプロパティとして試みられるが、「Ｗｈｉｌｅ」アクティビティは、名前「Ｓｅｎｄ」のプロパティを持たず、したがって、＜Ｓｅｎｄ＞要素はオブジェクトとして取り扱われ、ｗｈｉｌｅアクティビティの子アクティビティとして取り扱われる。＜Ｍｅｓｓａｇｅ＞要素は、Ｓｅｎｄアクティビティのプロパティとして取り扱われる。Ｓｅｎｄ　上のＭｅｓｓａｇｅプロパティは読み取り専用なので、Ｍｅｓｓａｇｅ要素のコンテンツは、Ｍｅｓｓａｇｅオブジェクトのコンテンツと考えられる。同様のルールが、＜ＯｎＢｅｆｏｒｅＳｅｎｄ＞および＜ＴｙｐｅｄＣｈａｎｎｅｌ＞要素のデシリアライゼーションに適用される。

【００８５】

ＸＯＭＬコードがきちんとした形でない、ＸｏｍｌＤｏｃｕｍｅｎｔがＸＯＭＬコード内の第１の要素でなく、ＸＯＭＬコード内の第１のアクティビティがデシリアライザできないという第１の条件の下では、ＸＯＭＬデシリアライゼーションは決定的に失敗する。開発者に対して、ＸＯＭＬビューからプロセス／ワークフロービューへ切り換える場合に、違反している側のＸＭＬ要素にナビゲートするためのエラーメッセージが提示される。

【００８６】

（オーケストレーションエンジンデザイナのホスティング）

デザイナフレームワークは、どのようなアプリケーションプログラムでもホスティングできる。これは、サードパーティのアプリケーションでそれぞれの環境においてワークフローをレンダリングするのに非常に有用な機能である。またこれにより、サードパーティは、デザインサーフェスを再ホスティングおよびカスタマイズすることで、オーケストレーションエンジンデザイナ周辺のツールを開発することができる。本発明のフレームワークでは、ホスティングコンテナアプリケーション側でエディタおよび／またはテキストバッファなどのサービス群を用意することを想定している。

【００８７】

デザイナを再ホストする１ステップでは、ローダおよびデザインサーフェスを作成する。ローダは、ＸＯＭＬファイルをロードし、アクティビティホストインフラストラクチャを構築する役割を持つ。デザインサーフェスは、その中にデザイナホストインフラストラクチャを保持し、デザインサーフェスをホスティングし、それとやり取りするサービスを提供する。デザインサーフェスは、サービスコンテナだけでなくサービスプロバイダとしても機能する。一実施例では、以下のコードを実行して、ＸＯＭＬドキュメントをロードし、その中にアクティビティを保持するデザイナホストを構築する。

【００８８】

【表４】

```
this.loader.XomlFile = filePath;
if (this.surface.IsLoaded == false)
        this.surface.BeginLoad(this.loader);
```

【００８９】

以下のサービスでは、デザイナで異なる機能を使用できる。ＩＳｅｌｅｃｔｉｏｎＳｅｒｖｉｃｅ関数は、選択されたオブジェクトを保持する。ＩＴｏｏｌｂｏｘＳｅｒｖｉｃｅ関数は、ツールボックスとのやり取りを管理する。ＩＭｅｎｕＣｏｍｍａｎｄＳｅｒｖｉｃｅ関数は、メニューとのやり取りを管理する。ＩＴｙｐｅＰｒｏｖｉｄｅｒ関数を使用すると、タイプシステムを利用できる。さらに、高度なデザイナ機能を使用可能にするデザイナホスティング環境が提供する追加サービスもありえる。

【００９０】

タイプシステムは、本発明のコンポーネントモデルフレームワーク内の一コンポーネントである。デザイナがプロジェクトシステムの内側でホスティングされる場合、ＴｙｐｅＰｒｏｖｉｄｅｒオブジェクトがプロジェクト毎に作成される。プロジェクト内のアセンブリ参照は、タイププロバイダへプッシュされる。さらに、プロジェクト内のユーザコードファイルが解析され、単一コードコンパイル単位が作成され、タイププロバイダへプッシュされる。また、本発明は、タイプシステム内で型を変更させることが可能なプロジェクトシステム内のイベントを監視し、その変更に対する応答として型を再ロードする適切な呼び出しをタイププロバイダに対し実行する。

【００９１】

（元に戻す／繰り返し）

スケジュールを作成し、正しく構築した後、開発者は一連の実行済みオペレーションをロールバックしたい場合がある。本発明の「元に戻す」および「繰り返し」機能を使用すると、どのアクティビティが直接影響を受けているかを例示する視覚的フィードバックが得られる。例えば、アクティビティ上でプロパティの変更が元に戻される場合、影響を受けたアクティビティは選択状態になる。複数のオブジェクトの削除が元に戻される場合、関わっているすべてのオブジェクトが、スケジュールに復元されるときに選択状態になる。Ｕｎｄｏ／Ｒｅｄｏ（元に戻す／繰り返し）は、他の分野における多くのアプリケーション全体を通して使用される共通の機能であり、その意味はよく理解されている。オーケストレーションエンジンデザイナでは、Ｓａｖｅ後に、元に戻す／繰り返しアイテムがパージされることはない。さらに、元に戻す／繰り返しは、プロセス／ワークフロービューで、ＸＯＭＬビューで、開発者がビューを切り換えた場合に、およびコードビサイド内で実行することができる。

【００９２】

Ｕｎｄｏ／Ｒｅｄｏは、プロセス／ワークフロービュー内のアクション、つまり、アクティビティのドラッグ＆ドロップ（例えば、ツールボックスからデザインサーフェスにアクティビティをドラッグする、スケジュールの一部分から他の部分にアクティビティを移動する、一方のデザイナから他方のデザイナにアクティビティを移動する）、アクティビティの構成（例えば、アクティビティのプロパティを指定する）、および切り取り／コピー／貼り付け／削除のアクションに対し用意されている。

【００９３】

一実施形態では、シリアライゼーションされたビュー（例えば、ＸＯＭＬビュー）は、テキストエディタ標準の元に戻す／繰り返しオペレーションを備えるＸＭＬエディタである。本発明のデザイナは、プロセス／ワークフロービュー内で加えられた変更を示すフィードバックを開発者に提示する。その後、シリアライゼーションされたビュー内で元に戻した結果として、シリアライゼーションされたコードは失われる。開発者がプロセス／ワークフロービュー内のスケジュールの一部分を構築し、シリアライゼーションされたビュ

ーに切り換えて、その後、元に戻す／繰り返しオペレーションを実行することに決めた場合、警告が表示される。

【００９４】

（動作環境の例）

図９は、コンピュータ１３０の形態の汎用コンピューティング装置の一実施例を示している。本発明の一実施形態では、コンピュータ１３０などのコンピュータは、例示され、本明細書で説明されている他の図で使用するのに適している。コンピュータ１３０は、１つまたは複数のプロセッサまたは演算処理装置１３２およびシステムメモリ１３４を備える。例示されている実施形態では、システムバス１３６は、システムメモリ１３４を含むさまざまなシステムコンポーネントをプロセッサ１３２に結合する。バス１３６は、メモリバスまたはメモリコントローラ、周辺機器バス、アクセラレイティッドグラフィックスポート、およびさまざまなバスアーキテクチャのどれかを使用するプロセッサまたはローカルバスを含む数種類のバス構造のうちの１つまたは複数を表している。例えば、限定はしないが、このようなアーキテクチャとしては、Ｉｎｄｕｓｔｒｙ Ｓｔａｎｄａｒｄ Ａｒｃｈｉｔｅｃｔｕｒｅ（ＩＳＡ）バス、Ｍｉｃｒｏ Ｃｈａｎｎｅｌ Ａｒｃｈｉｔｅｃｔｕｒｅ（ＭＣＡ）バス、Ｅｎｈａｎｃｅｄ ＩＳＡ（ＥＩＳＡ）バス、Ｖｉｄｅｏ Ｅｌｅｃｔｒｏｎｉｃｓ Ｓｔａｎｄａｒｄｓ Ａｓｓｏｃｉａｔｉｏｎ（ＶＥＳＡ）ローカルバス、およびＭｅｚｚａｎｉｎｅバスとも呼ばれるＰｅｒｉｐｈｅｒａｌ Ｃｏｍｐｏｎｅｎｔ Ｉｎｔｅｒｃｏｎｎｅｃｔ（ＰＣＩ）バスがある。

【００９５】

コンピュータ１３０は、通常、少なくともある種の形態のコンピュータ読み取り可能媒体を備える。コンピュータ読み取り可能媒体は、揮発性および不揮発性媒体、取り外し可能および固定の媒体を含み、コンピュータ１３０によってアクセスできる媒体であればどのような媒体でもよい。例えば、限定はしないが、コンピュータ読み取り可能媒体は、コンピュータ記憶媒体および通信媒体を含む。コンピュータ記憶媒体は、コンピュータ読み取り可能命令、データ構造体、プログラムモジュール、またはその他のデータなどの情報を格納する方法または技術で実装される揮発性および不揮発性、取り外し可能な、および固定の媒体を含む。例えば、コンピュータ記憶媒体は、ＲＡＭ、ＲＯＭ、ＥＥＰＲＯＭ、フラッシュメモリまたはその他のメモリ技術、ＣＤ－ＲＯＭ、デジタル多目的ディスク（ＤＶＤ）またはその他の光ディスク記憶装置、磁気カセット、磁気テープ、磁気ディスク記憶装置またはその他の磁気記憶装置、または所望の情報を格納するために使用することができ、しかもコンピュータ１３０によりアクセスできるその他の媒体を含む。通信媒体は、通常、コンピュータ読み取り可能命令、データ構造体、プログラムモジュール、または搬送波もしくはその他のトランスポートメカニズムなどの変調データ信号によるその他のデータを具現するものであり、任意の情報配信媒体を含む。当業者は、信号内の情報を符号化するなどの方法により特性のうちの１つまたは複数が設定または変更される変調データ信号を熟知している。有線ネットワークまたは直接配線接続などの有線媒体、ならびに音響、ＲＦ、赤外線、およびその他の無線媒体などの無線媒体は、通信媒体のいくつかの実施例である。上記のいずれの組み合わせもコンピュータ読み取り可能媒体の範囲に含まれる。

【００９６】

システムメモリ１３４は、取り外し可能な、および／または固定の、揮発性および／または不揮発性メモリの形のコンピュータ記憶媒体を備える。例示されている実施形態では、システムメモリ１３４は、読み取り専用メモリ（ＲＯＭ）１３８およびランダムアクセスメモリ（ＲＡＭ）１４０を含む。起動時などにコンピュータ１３０内の要素間の情報伝送を助ける基本ルーチンを含む基本入出力システム１４２（ＢＩＯＳ）は、通常、ＲＯＭ１３８に格納される。通常、ＲＡＭ １４０は、演算処理装置１３２に直接アクセス可能な、および／または演算処理装置１３２によって現在操作されているデータおよび／またはプログラムモジュールを格納する。例えば、限定はしないが、図９は、オペレーティングシステム１４４、アプリケーションプログラム１４６、その他のプログラムモジュー

ル１４８、およびプログラムデータ１５０を例示している。

【００９７】

コンピュータ１３０はさらに、その他の取り外し可能な／固定の揮発性／不揮発性コンピュータ記憶媒体を備えることもできる。例えば、図９は、固定の不揮発性磁気媒体への読み書きを行うハードディスクドライブ１５４を例示している。図９は、さらに、取り外し可能不揮発性磁気ディスク１５８への読み書きを行う磁気ディスクドライブ１５６、およびＣＤ－ＲＯＭまたはその他の光媒体などの取り外し可能不揮発性光ディスク１６２への読み書きを行う光ディスクドライブ１６０を示している。動作環境の実施例で使用できる他の取り外し可能な／固定の揮発性／不揮発性コンピュータ記憶媒体としては、限定はしないが、磁気テープカセット、フラッシュメモリカード、デジタル多目的ディスク、デジタルビデオテープ、ソリッドステートＲＡＭ、ソリッドステートＲＯＭなどがある。ハードディスクドライブ１５４、および磁気ディスクドライブ１５６、および光ディスクドライブ１６０は、通常、インターフェイス１６６などの不揮発性メモリインターフェイスによりシステムバス１３６に接続される。

【００９８】

図９に例示されている上記のドライブまたは他の大容量記憶装置およびその関連コンピュータ記憶媒体は、コンピュータ１３０用のコンピュータ読み取り可能命令、データ構造体、プログラムモジュール、およびその他のデータを格納する機能を備える。例えば、図９では、ハードディスクドライブ１５４は、オペレーティングシステム１７０、アプリケーションプログラム１７２、その他のプログラムモジュール１７４、およびプログラムデータ１７６を格納するものとして例示されている。これらのコンポーネントは、オペレーティングシステム１４４、アプリケーションプログラム１４６、その他のプログラムモジュール１４８、およびプログラムデータ１５０と同じである場合もあれば異なる場合もあることに留意されたい。オペレーティングシステム１７０、アプリケーションプログラム１７２、その他のプログラムモジュール１７４、およびプログラムデータ１７６に対しては、ここで、異なる番号を割り当てて、最低でも、それらが異なるコピーであることを示している。

【００９９】

ユーザは、キーボード１８０、およびポインティング装置１８２（例えば、マウス、トラックボール、ペン、またはタッチパッド）などのなどの入力装置またはユーザインターフェイス選択装置を通じてコマンドおよび情報をコンピュータ１３０に入力することができる。他の入力装置（図に示されていない）としては、マイク、ジョイスティック、ゲームパッド、衛星放送受信アンテナ、スキャナなどがある。これらの入力装置およびその他の入力装置は、システムバス１３６に結合されているユーザ入力インターフェイス１８４を通じて演算処理装置１３２に接続されるが、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）などの他のインターフェイスおよびバス構造により接続することもできる。モニタ１８８またはその他の種類の表示装置も、ビデオインターフェイス１９０などのインターフェイスを介してシステムバス１３６に接続される。モニタ１８８のほかに、コンピュータはプリンタおよびスピーカなどの他の周辺出力装置（図に示されていない）を備えることが多く、これらは出力周辺機器インターフェイス（図に示されていない）を通じて接続することができる。

【０１００】

コンピュータ１３０は、リモートコンピュータ１９４などの１つまたは複数のリモートコンピュータへの論理接続を使用してネットワーク接続環境で動作することができる。リモートコンピュータ１９４は、パーソナルコンピュータ、サーバ、ルータ、ネットワークＰＣ、ピア装置、またはその他の共通ネットワークノードとすることができ、通常は、コンピュータ１３０に関して上で説明されている要素の多くまたはすべてを含む。図９に示されている論理接続は、ローカルエリアネットワーク（ＬＡＮ）１９６およびワイドエリアネットワーク（ＷＡＮ）１９８を含むが、他のネットワークを含むこともできる。ＬＡＮ　１９６および／またはＷＡＮ　１９８は、有線ネットワーク、無線ネットワーク、そ

れらの組み合わせなどとすることができる。このようなネットワーキング環境は、オフィス、企業規模のコンピュータネットワーク、イントラネット、および大域的コンピュータネットワーク（例えば、インターネット）では一般的である。

【０１０１】

ローカルエリアネットワーキング環境で使用される場合、コンピュータ１３０は、ネットワークインターフェイスまたはアダプタ１８６を介してＬＡＮ　１９６に接続される。ワイドエリアネットワーキング環境で使用される場合、コンピュータ１３０は、通常、インターネットなどのＷＡＮ　１９８上で通信を確立するためモデム１７８またはその他の手段を備える。モデム１７８は、内蔵でも外付けでもよいが、ユーザ入力インターフェイス１８４、またはその他の適切なメカニズムを介してシステムバス１３６に接続される。ネットワーク接続環境では、コンピュータ１３０またはその一部に関して示されているプログラムモジュールは、リモートメモリ記憶装置（図に示されていない）に格納されうる。例えば、限定はしないが、図９には、リモートアプリケーションプログラム１９２がメモリ装置に常駐しているように例示されている。図に示されているネットワーク接続は実施例であり、コンピュータ間の通信リンクを確立するのに他の手段が使用されうる。

【０１０２】

一般に、コンピュータ１３０のデータプロセッサは、コンピュータのさまざまなコンピュータ読み取り可能記憶媒体にさまざまな時点において格納された命令を使ってプログラムされる。プログラムおよびオペレーティングシステムは、通常、例えば、フロッピー（登録商標）ディスクまたはＣＤ－ＲＯＭで配布される。そこから、コンピュータの補助記憶装置にインストールまたはロードされる。実行時に、それらは少なくとも一部はコンピュータの主記憶装置にロードされる。本明細書で説明されている発明は、これらおよび他のさまざまな種類のコンピュータ読み取り可能記憶媒体を含むが、ただしそのような媒体にマイクロプロセッサまたはその他のデータプロセッサに関して以下で説明されるステップを実装する命令またはプログラムが格納されている場合である。本発明は、さらに、本明細書で説明されている方法および手法に従ってプログラムされた場合にコンピュータ自体も含む。

【０１０３】

例示の目的に関して、オペレーティングシステムなどのプログラムおよびその他の実行可能プログラムコンポーネントは、本明細書では異なるブロックとして例示されている。しかし、そのようなプログラムおよびコンポーネントは、コンピュータの異なる記憶装置コンポーネント内にさまざまな時点において常駐し、コンピュータの（複数の）データプロセッサにより実行されることは理解される。

【０１０４】

本発明は、コンピュータ１３０を含む、コンピューティングシステム環境例に関して説明されているが、他の多くの汎用または専用コンピューティングシステム環境または構成で動作する。このコンピューティングシステム環境は、本発明の用途または機能の範囲に関する制限を示唆する意図はない。さらに、コンピューティングシステム環境は、動作環境例に例示されている１つのコンポーネントまたは複数のコンポーネントの組み合わせに関係する何らかの依存関係または要求条件がその環境にあるものと解釈すべきでない。本発明とともに使用するのに適していると思われるよく知られているコンピューティングシステム、環境、および／または構成の例として、パーソナルコンピュータ、サーバコンピュータ、ハンドヘルドまたはラップトップ装置、マルチプロセッサシステム、マイクロプロセッサベースのシステム、セットトップボックス、プログラム可能な家電製品、携帯電話、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、上記システムまたは装置を含む分散コンピューティング環境などがある。

【０１０５】

本発明は、１つまたは複数のコンピュータまたはその他の装置により実行される、プログラムモジュールなどのコンピュータ実行可能命令の一般的状況において説明することができる。一般に、プログラムモジュールは、限定はしないが、特定のタスクを実行する、

または特定の抽象データ型を実装するルーチン、プログラム、オブジェクト、コンポーネント、およびデータ構造を含む。また、本発明は、通信ネットワークを通じてリンクされているリモート処理装置によりタスクが実行される分散コンピューティング環境で実施することもできる。分散コンピューティング環境では、プログラムモジュールをメモリ記憶装置などのローカルとリモートの両方のコンピュータ記憶媒体に配置できる。

【０１０６】

ソフトウェアアーキテクチャの背景状況におけるインターフェイスは、ソフトウェアモジュール、コンポーネント、コード部分、またはコンピュータ実行可能命令の他のシーケンスを含む。例えば、インターフェイスは、第１のモジュールの代わりにコンピューティングタスクを実行するために第２のモジュールにアクセスする第１のモジュールを含む。第１および第２のモジュールは、一実施例では、オペレーティングシステムによって提供されるようなアプリケーションプログラミングインターフェイス（ＡＰＩ）、コンポーネントオブジェクトモデル（ＣＯＭ）インターフェイス（例えば、ピアツーピアアプリケーション通信用）、および拡張マークアップ言語メタデータ交換形式（ＸＭＩ）インターフェイス（例えば、複数のＷｅｂサービス間の通信用）を含む。

【０１０７】

インターフェイスは、Ｊａｖａ（登録商標） ２ Ｐｌａｔｆｏｒｍ Ｅｎｔｅｒｐｒｉｓｅ Ｅｄｉｔｉｏｎ（Ｊ２ＥＥ）、ＣＯＭ、または分散ＣＯＭ（ＤＣＯＭ）の実施例などの密結合の同期実装であってよい。それとは別に、またはそれに加えて、インターフェイスは、Ｗｅｂサービスの場合のような疎結合の非同期実装とすることもできる（例えば、シンプルオブジェクトアクセスプロトコルを使用する）。一般に、インターフェイスは、密結合、疎結合、同期、および非同期という特性の任意の組み合わせを含む。さらに、インターフェイスは、標準プロトコル、専用プロトコル、または標準プロトコルと専用プロトコルとの任意の組み合わせに適合することができる。

【０１０８】

本明細書で説明されているインターフェイスは、すべて、単一インターフェイスの一部であるか、または別々のインターフェイスもしくはそれらの任意の組み合わせとして実装することができる。これらのインターフェイスは、機能を提供するためにローカルまたはリモートで実行できる。さらに、これらのインターフェイスが含む機能は、例示されている、または本明細書で説明されている機能よりも多い場合も少ない場合もある。

【０１０９】

例示され、本明細書で説明されている方法の実行または遂行の順序は、断りのない限り本質的ではない。つまり、これらの方法の要素は、断りのない限り任意の順序で実行することができ、それらの方法が含む要素は、本明細書で開示されている要素よりも多い場合も少ない場合もある。例えば、特定の要素を、他の要素の前に、他の要素と同時に、または他の要素の後に実行または遂行することは本発明の範囲であると考えられる。

【０１１０】

本発明または本発明の（複数の）実施形態の要素を導入する際に、英文中の「ａ」、「ａｎ」、「ｔｈｅ」、および「ｓａｉｄ」という冠詞、したがって和文中の「１つの」、「その」、「前記」は、それらの要素が１つまたは複数あることを意味することを意図している。「備える」、「含む」、および「持つ、格納する、含む、備える」という言葉は、包含的であることを意図し、一覧に示されている要素以外にさらに要素がありうることを意味する。

【０１１１】

上記の説明に照らして、本発明の複数の目的が達成され、他の有益な結果が得られることは理解されるであろう。

【０１１２】

本発明の範囲から逸脱することなく上記の構成、製品、および方法にさまざまな変更を加えることが可能であるので、上記の説明に含まれ、付属の図面に示されているすべての事柄は、例示しているのであって、限定する意味はないと解釈するものとする。

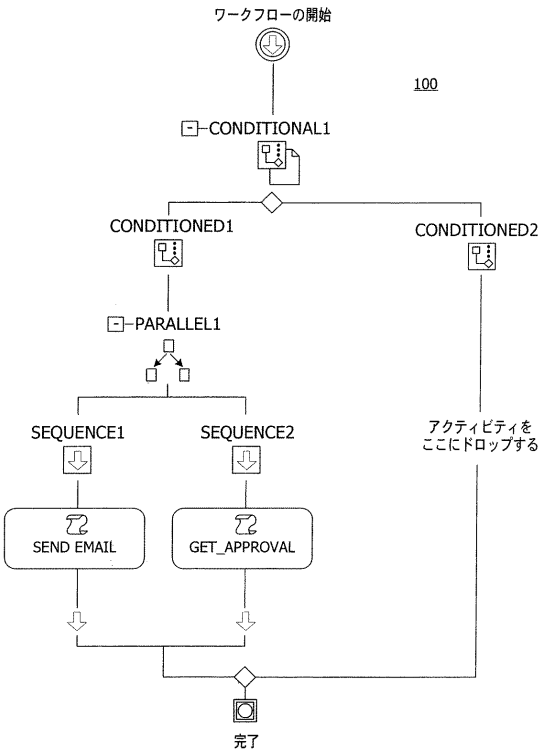
【０１１３】
付録Ａ
（アクティビティの例およびその実装例）
アクティビティの例としては、Ｓｅｎｄ、ＳｅｎｄＲｅｑｕｅｓｔ、ＳｅｎｄＲｅｓｐｏｎｓｅ、Ｒｅｃｅｉｖｅ、ＲｅｃｅｉｖｅＲｅｑｕｅｓｔ、ＲｅｃｅｉｖｅＲｅｓｐｏｎｓｅ、Ｃｏｄｅ、Ｄｅｌａｙ、Ｆａｕｌｔ、Ｓｕｓｐｅｎｄ、Ｔｅｒｍｉｎａｔｅ、ＩｎｖｏｋｅＳｃｈｅｄｕｌｅ、ＩｎｖｏｋｅＳｃｈｅｄｕｌｅｓ、Ｉｎｖｏｋｅ Ｗｅｂ Ｓｅｒｖｉｃｅ、ＤｏｔＮｅｔＥｖｅｎｔＳｏｕｒｃｅ、ＤｏｔＮｅｔＥｖｅｎｔＳｉｎｋ、Ｓｅｑｕｅｎｃｅ、Ｐａｒａｌｌｅｌ、Ｗｈｉｌｅ、ＣｏｎｄｉｔｉｏｎａｌＢｒａｎｃｈ、Ｃｏｎｄｉｔｉｏｎａｌ、Ｃｏｎｓｔｒａｉｎｅｄ、ＣｏｎｓｔｒａｉｎｅｄＡｃｔｉｖｉｔｙＧｒｏｕｐ（ＣＡＧ）、ＥｖｅｎｔＤｒｉｖｅｎ、Ｌｉｓｔｅｎ、ＥｖｅｎｔＨａｎｄｌｅｒｓ、ＥｘｃｅｐｔｉｏｎＨａｎｄｌｅｒ、ＥｘｃｅｐｔｉｏｎＨａｎｄｌｅｒｓ、Ｃｏｍｐｅｎｓａｔｅ、ＣｏｍｐｅｎｓａｔｉｏｎＨａｎｄｌｅｒ、Ｓｃｏｐｅ、およびＳｃｈｅｄｕｌｅがある。
【０１１４】
アクティビティの実施例はそれぞれ、メタデータが関連付けられている。メタデータは、アクティビティに関連付けられたシリアライザによってワークフローの宣言的表現に移される。例えば、メタデータは、オプションのコードビサイドメソッドおよびオプションの相関関係集合のコレクションを含むことができる。
【０１１５】
（Ｓｅｎｄアクティビティ）
オーケストレーションエンジンは、メッセージを送信する３つのアクティビティ（例えば、Ｓｅｎｄ、ＳｅｎｄＲｅｑｕｅｓｔ、およびＳｅｎｄＲｅｓｐｏｎｓｅ）を備え、それぞれ異なる使用事例を対象とする。さらに、これら３つのアクティビティは、何らかのメタデータを共有するので、抽象基本クラスが３つのすべての上位クラスとして定義され使用される。
【０１１６】
（Ｒｅｃｅｉｖｅアクティビティ）
オーケストレーションエンジンは、メッセージを受信する３つのアクティビティ（例えば、Ｒｅｃｅｉｖｅ、ＲｅｃｅｉｖｅＲｅｑｕｅｓｔ、およびＲｅｃｅｉｖｅＲｅｓｐｏｎｓｅ）を備え、それぞれ異なる使用事例を対象とする。さらに、これら３つのアクティビティは、何らかのメタデータを共有するので、抽象基本クラスが３つのすべての上位クラスとして定義され使用される。
【０１１７】
（Ｃｏｄｅ）
Ｃｏｄｅアクティビティは、メタデータで指示されているコードビサイドメソッドを実行する。
【０１１８】
（Ｄｅｌａｙ）
Ｄｅｌａｙアクティビティは、その必須コードビサイドメソッドを実行してＤａｔｅＴｉｍｅ値を生成する。そのインスタンスデータのＴｉｍｅｏｕｔＶａｌｕｅプロパティをこの値に内部的に設定する。ＤａｔｅＴｉｍｅが過去であれば、Ｄｅｌａｙは即座に完了する。そうでなければ、タイマーが作動したときにＤｅｌａｙに通知するようにタイマーサブスクリプションをセットアップする。タイマーが作動した場合、Ｄｅｌａｙは通知され、完了する。
【０１１９】
（Ｆａｕｌｔ）
Ｆａｕｌｔアクティビティは、その必須コードビサイドメソッドを実行してＥｘｃｅｐｔｉｏｎオブジェクトを生成する。その後、この例外をスローする。
【０１２０】

（Ｓｕｓｐｅｎｄ）
Ｓｕｓｐｅｎｄアクティビティは、現在のスケジュールインスタンスをサスペンドする。
【０１２１】
（Ｔｅｒｍｉｎａｔｅ）
Ｔｅｒｍｉｎａｔｅアクティビティは、現在のスケジュールインスタンスを終了する。
【０１２２】
（Ｉｎｖｏｋｅ　Ｓｃｈｅｄｕｌｅ）
ＩｎｖｏｋｅＳｃｈｅｄｕｌｅアクティビティは、スケジュールを呼び出す。
【０１２３】
（Ｉｎｖｏｋｅ　Ｗｅｂ　Ｓｅｒｖｉｃｅ）
プロキシクラスを介してＷｅｂサービスを呼び出して、パラメータを指定通り受け渡し、受信する。
【０１２４】
（ＤｏｔＮｅｔＥｖｅｎｔ　Ｓｉｎｋ）
すでに呼び出されているスケジュールインスタンスにより指定イベントが発生したという通知が来るのを待つブロック。
【０１２５】
（ＤｏｔＮｅｔＥｖｅｎｔ　Ｓｏｕｒｃｅ）
指定イベントを発生し、即座に実行を完了する。
【０１２６】
（Ｓｅｑｕｅｎｃｅ）
Ｓｅｑｕｅｎｃｅアクティビティは、一度に１つずつ、順序正しく子アクティビティの集合の実行を調整する。
【０１２７】
（Ｐａｒａｌｌｅｌ）
Ｐａｒａｌｌｅｌアクティビティは、子アクティビティの集合を同時実行する。
【０１２８】
（Ｗｈｉｌｅ）
子アクティビティを繰り返し実行する。
【０１２９】
（ＣｏｎｄｉｔｉｏｎａｌＢｒａｎｃｈ）
Ｓｅｑｕｅｎｃｅの意味論に従って、子アクティビティを実行する。
【０１３０】
（Ｃｏｎｄｉｔｉｏｎａｌ）
Ｃｏｎｄｉｔｉｏｎａｌアクティビティは、ＣｏｎｄｉｔｉｏｎａｌＢｒａｎｃｈアクティビティの順序付き集合を含む。
【０１３１】
（Ｃｏｎｓｔｒａｉｎｅｄ）
制約されているアクティビティの唯一の許容される親はＣＡＧである。ＣＡＧ自体は、制約されているアクティビティで使用可能および使用不可ルールを用いて、いつ実行するかを決定する。ＣｏｎｓｔｒａｉｎｅｄアクティビティがＣＡＧにより実行を指示された場合、それがラップするアクティビティを単に実行するだけである。
【０１３２】
（ＣＡＧ（Ｃｏｎｓｔｒａｉｎｅｄ　Ａｃｔｉｖｉｔｙ　Ｇｒｏｕｐ））
ＣＡＧは、制約アクティビティのみを含む。ＣＡＧは、実行されると、その使用可能および使用不可制約の評価に基づいて子アクティビティを実行（および再実行）する。
【０１３３】
（Ｔａｓｋ）
１つまたは複数のプリンシパルにより実行される外部作業単位をモデル化する。

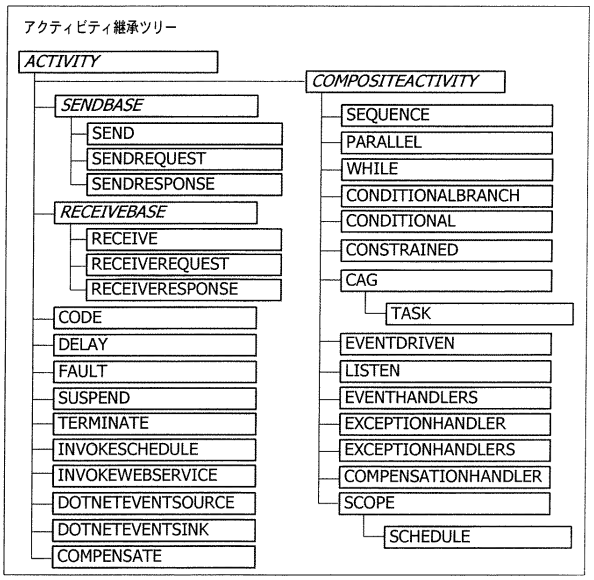
【０１３４】
（Ｅｖｅｎｔ　Ｄｒｉｖｅｎ）
実行が「イベント」アクティビティによりトリガされるアクティビティをラップする。
【０１３５】
（Ｌｉｓｔｅｎ）
ｎ個の子ＥｖｅｎｔＤｒｉｖｅｎアクティビティの１つを条件付きで実行する。
【０１３６】
（Ｅｖｅｎｔ　Ｈａｎｄｌｅｒｓ）
ＥｖｅｎｔＤｒｉｖｅｎアクティビティの集合をラップする。ＥｖｅｎｔＨａｎｄｌｅｒｓアクティビティは、単純に、ＥｖｅｎｔＤｒｉｖｅｎアクティビティの集合を保持し、これを関連するＳｃｏｐｅが使用する。
【０１３７】
（Ｅｘｃｅｐｔｉｏｎ　Ｈａｎｄｌｅｒ）
スコープに対するキャッチブロックを表すメタデータでアクティビティをラップする。
【０１３８】
（Ｅｘｃｅｐｔｉｏｎ　Ｈａｎｄｌｅｒｓ）
ＥｘｃｅｐｔｉｏｎＨａｎｄｌｅｒアクティビティの順序付き集合をラップする。
【０１３９】
（Ｃｏｍｐｅｎｓａｔｅ）
完了した子スコープを補正する。
【０１４０】
（Ｃｏｍｐｅｎｓａｔｉｏｎ　Ｈａｎｄｌｅｒ）
スコープに対する補正ハンドラとして定義されている子アクティビティをラップする。
【０１４１】
（Ｓｃｏｐｅ）
スコープは、トランザクション境界、例外処理境界、補正境界、イベント処理境界、およびメッセージ、変数、相関関係集合、およびチャネル宣言の境界（つまり、共有データ状態）である。Ｓｃｏｐｅ内のアクティビティの実行は順次であり、したがって、含まれているアクティビティは、Ｓｅｑｕｅｎｃｅの場合のように、スコープが構築されたときに明示的に順序付けられる。
【０１４２】
（Ｓｃｈｅｄｕｌｅ）
Ｓｃｈｅｄｕｌｅは、オーケストレーションエンジンが実行する唯一の最上位レベルのアクティビティである。
【０１４３】
（複合アクティビティ）
制御フローを使用可能にする複合アクティビティ型は、Ｓｅｑｕｅｎｃｅ、Ｐａｒａｌｌｅｌ、Ｃｏｎｓｔｒａｉｎｅｄ　Ａｃｔｉｖｉｔｙ　Ｇｒｏｕｐ、Ｃｏｎｄｉｔｉｏｎａｌ、Ｗｈｉｌｅ、Ｌｉｓｔｅｎである。さらに、ＳｃｏｐｅおよびＳｃｈｅｄｕｌｅは、中にあるアクティビティの暗黙のシーケンス動作を含むコンテナとして動作する複合アクティビティ型である。
【０１４４】
対応する参照文字は、図面全体を通して対応する部分を示す。
【図面の簡単な説明】
【０１４５】
【図１】タスクおよび制御フロー複合アクティビティを含むワークフローの実施例を示す図である。
【図２】アクティビティ継承ツリーの実施例を示す図である。
【図３】コンポーネントモデルの実施例を示す図である。
【図４】コンポーネントモデルのライフサイクルの例を示す図である。

【図５】ワークフローの指定にウィザードに依存する、ワークフローオーサリングするための高水準アプリケーションユーザインターフェイスを示す図である。
【図６】ワークフローデザイナの実施例を示す図である。
【図７】送信アクティビティが後に続く受信アクティビティを含むオーケストレーションプログラムを示す図である。
【図８】スケジュール定義およびビジュアルワークフローと、ワークフローのＸＯＭＬによるシリアライズされた表現と、ワークフローのコードビサイドとの間の関係を示す図である。
【図９】本発明を実装することができる１つの好適なコンピューティングシステム環境の一実施例を示すブロック図である。
【符号の説明】
【０１４６】
１３０　コンピュータ
１５４　ハードディスクドライブ
１５６　磁気ディスクドライブ
１６０　光ディスクドライブ
１７８　モデム
１８８　モニタ
１９４　リモートコンピュータ

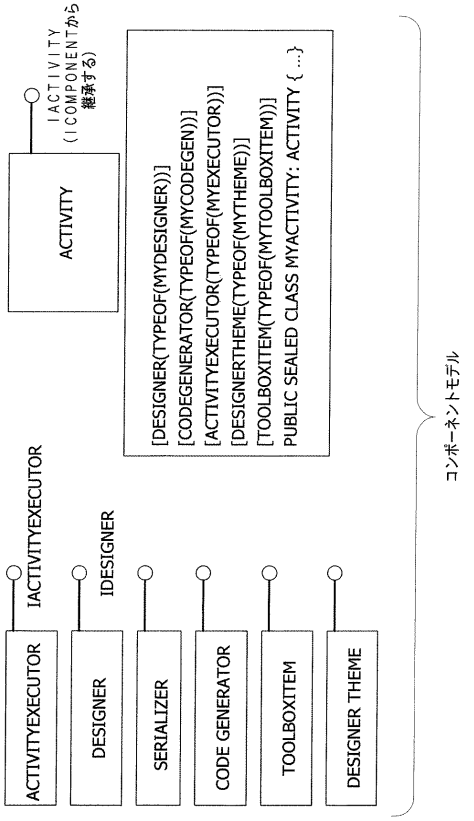
【図１】



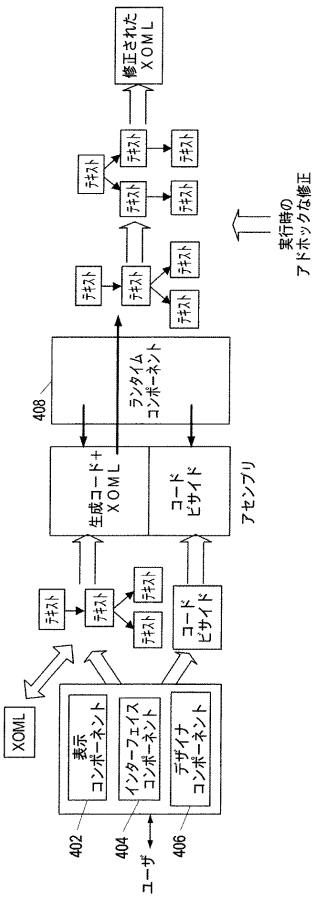
【図２】



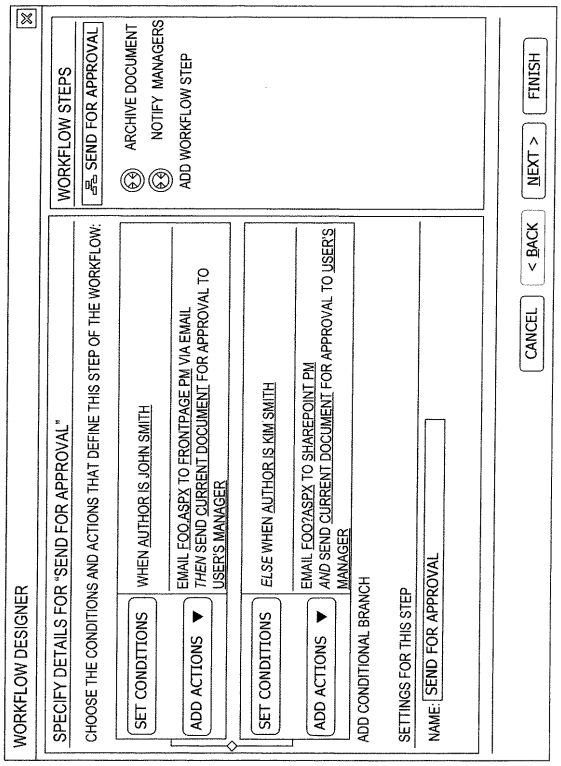
【図３】


IACTIVITYEXECUTOR
IDESIGNER
ACTIVITY
IACTIVITY
(ICOMPONENTから継承する)
ACTIVITYEXECUTOR
DESIGNER
SERIALIZER
CODE GENERATOR
TOOLBOXITEM
DESIGNER THEME
[DESIGNER(TYPEOF(MYDESIGNER))]
[CODEGENERATOR(TYPEOF(MYCODEGEN))]
[ACTIVITYEXECUTOR(TYPEOF(MYEXECUTOR))]
[DESIGNERTHEME(TYPEOF(MYTHEME))]
[TOOLBOXITEM(TYPEOF(MYTOOLBOXITEM))]
PUBLIC SEALED CLASS MYACTIVITY: ACTIVITY { ... }
コンポーネントモデル

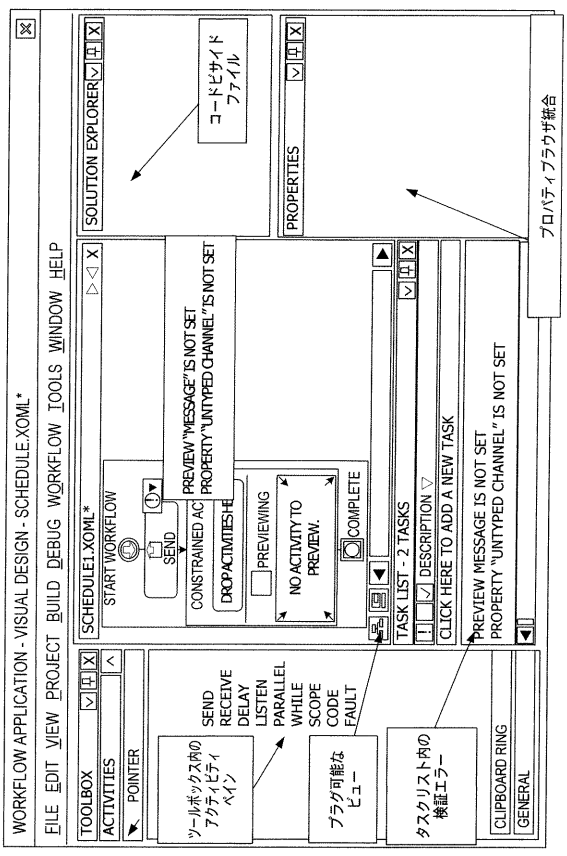
【図４】


XOML
表示
コンポーネント
インターフェイス
コンポーネント
デザイナ
コンポーネント
コード
ビサイド
生成コード+
XOML
ランタイム
コンポーネント
408
402
404
406
ユーザ
アセンブリ
テキスト
修正された
XOML
実行時の
アドホックな修正

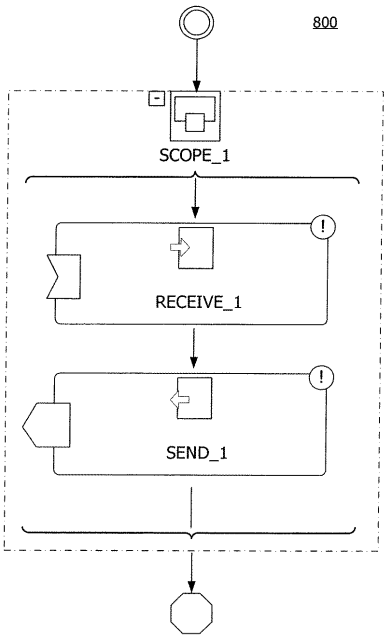
【図５】


WORKFLOW DESIGNER
SPECIFY DETAILS FOR "SEND FOR APPROVAL"
CHOOSE THE CONDITIONS AND ACTIONS THAT DEFINE THIS STEP OF THE WORKFLOW:
SET CONDITIONS
WHEN AUTHOR IS JOHN SMITH
ADD ACTIONS
EMAIL FOO.ASPX TO FRONTPAGE PM VIA EMAIL
THEN SEND CURRENT DOCUMENT FOR APPROVAL TO USER'S MANAGER
SET CONDITIONS
ELSE WHEN AUTHOR IS KIM SMITH
ADD ACTIONS
EMAIL FOO2.ASPX TO SHAREPOINT PM
AND SEND CURRENT DOCUMENT FOR APPROVAL TO USER'S MANAGER
ADD CONDITIONAL BRANCH
SETTINGS FOR THIS STEP
NAME: SEND FOR APPROVAL
WORKFLOW STEPS
SEND FOR APPROVAL
ARCHIVE DOCUMENT
NOTIFY MANAGERS
ADD WORKFLOW STEP
CANCEL
< BACK
NEXT >
FINISH

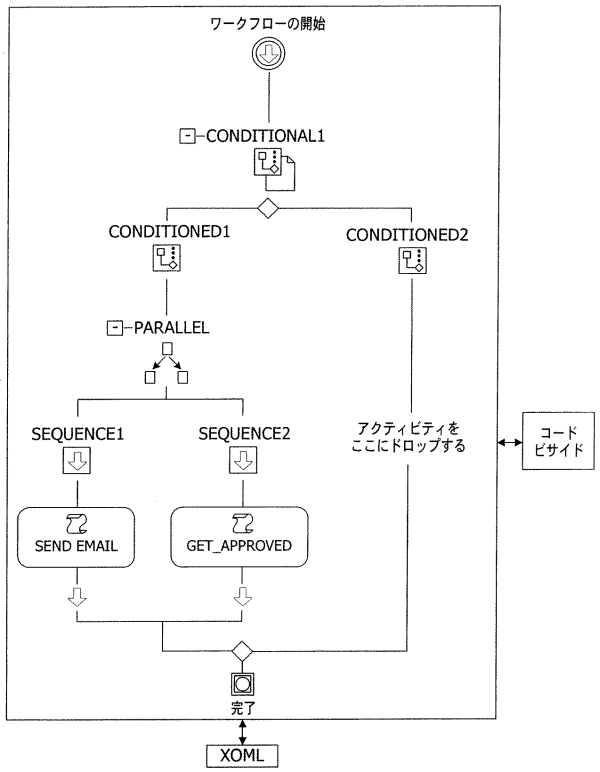
【図６】


WORKFLOW APPLICATION - VISUAL DESIGN - SCHEDULE.XOML*
FILE EDIT VIEW PROJECT BUILD DEBUG WORKFLOW TOOLS WINDOW HELP
TOOLBOX
ACTIVITIES
POINTER
SEND
RECEIVE
DELAY
LISTEN
PARALLEL
WHILE
SCOPE
CODE
FAULT
CLIPBOARD RING
GENERAL
SCHEDULE1.XOML*
START WORKFLOW
SEND
CONSTRAINED AC
DROP ACTIVITIES HE
PREVIEW "MESSAGE" IS NOT SET
PROPERTY "UNTYPED CHANNEL" IS NOT SET
PREVIEWING
NO ACTIVITY TO PREVIEW.
COMPLETE
TASK LIST - 2 TASKS
DESCRIPTION
CLICK HERE TO ADD A NEW TASK
PREVIEW MESSAGE IS NOT SET
PROPERTY "UNTYPED CHANNEL" IS NOT SET
SOLUTION EXPLORER
PROPERTIES
ツールボックス内の
アクティビティ
ペイン
プラグ可能な
ビュー
タスクリスト内の
検証エラー
コードビサイド
ファイル
プロパティブラウザ統合

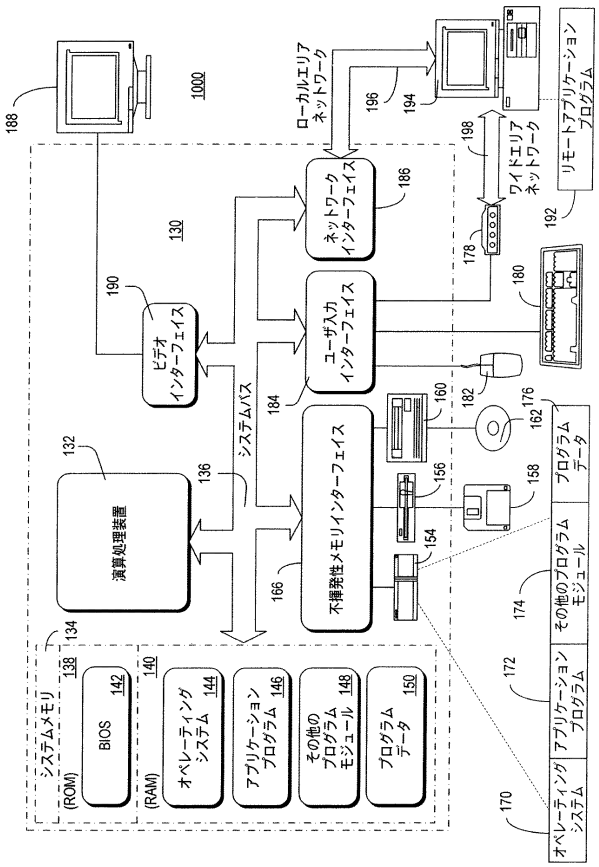
【図７】


800
SCOPE_1
RECEIVE_1
SEND_1

【図８】


ワークフローの開始
CONDITIONAL1
CONDITIONED1
CONDITIONED2
PARALLEL
SEQUENCE1
SEQUENCE2
アクティビティを
ここにドロップする
コード
ビサイド
SEND EMAIL
GET_APPROVED
完了
XOML

【図９】


188
1000
ローカルエリア
ネットワーク
196
194
198
ワイドエリア
ネットワーク
リモートアプリケーション
プログラム
192
130
ネットワーク
インターフェイス
186
178
180
190
ビデオ
インターフェイス
ユーザ入力
インターフェイス
システムバス
184
182
160
176
162
132
136
演算処理装置
不揮発性メモリインターフェイス
156
158
プログラム
データ
154
166
174
その他のプログラム
モジュール
134
138
システムメモリ
(ROM)
BIOS
142
140
(RAM)
オペレーティング
システム
144
アプリケーション
プログラム
146
その他の
プログラム
モジュール
148
プログラム
データ
150
172
アプリケーション
プログラム
170
オペレーティング
システム

フロントページの続き

(74)代理人 100120112
弁理士 中西 基晴
(74)代理人 100147991
弁理士 鳥居 健一
(74)代理人 100119781
弁理士 中村 彰吾
(74)代理人 100162846
弁理士 大牧 綾子
(74)代理人 100173565
弁理士 末松 亮太
(74)代理人 100138759
弁理士 大房 直樹
(74)代理人 100091063
弁理士 田中 英夫
(72)発明者 バリンダー エス. マルヒ
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 ダーマ ケー. シュクラ
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 クマースワミー ピー. バレゲレプラ
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 マヤンク メヘタ
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 ロバート ビー. シュミット
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

合議体
審判長 石井 茂和
審判官 西村 泰英
審判官 小林 大介

(56)参考文献 特開２００１－５６８０（ＪＰ，Ａ）
特開２００３－１７８１７０（ＪＰ，Ａ）
戸田保一（外３名）著，「ワークフロー」，１９９８年5月3日，株式会社日科技連出版社，第１３～３０頁
蒋海鷹（外２名），電子商取引プロセスにおける電子契約実行支援のためのメッセージ交換モデル」，ＤＥＷＳ２００３ ７－Ｂ－０４、第１４回データ工学ワークショップ（ＤＥＷＳ２００３）論文集

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 9/44