(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4828218号
(P4828218)
(45)発行日　平成23年11月30日(2011.11.30)　(24)登録日　平成23年9月22日(2011.9.22)

(51)Int.Cl.　F I
G06F 21/22 (2006.01)　G06F 9/06 66ON

請求項の数 15　外国語出願　(全 29 頁)

(21)出願番号　特願2005-354079(P2005-354079)
(22)出願日　平成17年12月7日(2005.12.7)
(65)公開番号　特開2006-164287(P2006-164287A)
(43)公開日　平成18年6月22日(2006.6.22)
審査請求日　平成20年11月21日(2008.11.21)
(31)優先権主張番号　11/007,808
(32)優先日　平成16年12月7日(2004.12.7)
(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　ベンジャミン　ゾーン
アメリカ合衆国　９８０５２　ワシントン州　レッドモンド　ワン　マイクロソフト　ウェイ　マイクロソフト　コーポレーション内

最終頁に続く

(54)【発明の名称】自己記述アーチファクトおよびアプリケーション抽象化

(57)【特許請求の範囲】
【請求項１】
プロセッサ実行可能命令を実行するように構成されたプロセッサと、
前記プロセッサに結合されたメモリと、
前記プロセッサに結合され、コンピューティングシステムのソフトウェアコンポーネントを表す複数の自己記述ソフトウェアアーチファクトを永続的に保存するように構成されたストレージサブシステムであって、各ソフトウェアアーチファクトが、実行可能なエンティティのコンテンツおよび構成を表すオフライン表現を備え、各実行可能なエンティティが、プロセス、アプリケーション、またはオペレーティングシステムのコンポーネントのうちのいずれか１つのソフトウェアコンポーネントである、ストレージサブシステムと
を備え、
前記複数の自己記述ソフトウェアアーチファクトのそれぞれは、関連する永続的に保存されたマニフェストを有し、
各マニフェストは、前記マニフェストのソフトウェアアーチファクトのメタデータ宣言型記述を含み、
各マニフェストは、関連する実行可能なエンティティのメタデータ宣言型記述を含み、
特定の実行可能なエンティティの複数の表現が存在する場合、別個のマニフェストが前記特定の実行可能なエンティティの各表現に関連付けられ、
各マニフェストは、２つの形態のうちの１つで存在し、
マニフェストが静的マニフェストの形態である場合、該静的マニフェストは、ソフト

ウェアアーチファクトに関連して格納され、
　マニフェストが動的マニフェストの形態である場合、該動的マニフェストは、ランタイムに入手可能な静的メタデータとランタイムに構築されて複数のランタイムシステム要素を結び付ける動的メタデータとを含むように、関連する実行可能なエンティティのそれぞれのランタイム中に使用される
ことを特徴とするコンピューティングシステム。
【請求項２】
　前記コンピューティングシステムの前記ソフトウェアコンポーネントは、前記コンピューティングシステム上にインストールされ、前記プロセッサ上で実行されるように構成されたオペレーティングシステム要素およびアプリケーションを含むことを特徴とする請求項１に記載のシステム。
【請求項３】
　前記ストレージサブシステムは、前記複数の自己記述ソフトウェアアーチファクトの宣言型記述を含むシステムマニフェストを永続的に保存することを特徴とする請求項１に記載のシステム。
【請求項４】
　前記自己記述ソフトウェアアーチファクトを管理するように構成されたアーチファクトマネージャをさらに備えたことを特徴とする請求項１に記載のシステム。
【請求項５】
　プロセッサによって実行されると、
　それぞれが、実行可能なエンティティのコンテンツおよび構成を表すオフライン表現を備え、コンピューティングシステム上にインストールされた動作可能なオペレーティングシステムコンポーネントまたは動作可能なアプリケーションを表す、複数の自己記述ソフトウェアアーチファクトを、前記コンピューティングシステム上に永続的に保存する動作と、
　前記複数の自己記述ソフトウェアアーチファクトにそれぞれ関連する複数のマニフェストを永続的に保存する動作であって、特定の実行可能なエンティティの複数の表現が存在する場合、別個のマニフェストが前記特定の実行可能なエンティティの各表現に関連付けられ、前記マニフェストは、
　　前記複数の自己記述ソフトウェアアーチファクト、および
　　関連する実行可能なエンティティ
　のメタデータ宣言型記述を含む、動作と、
　各マニフェストは、静的マニフェストの形態または動的マニフェストの形態のいずれかであって、
　　マニフェストが静的マニフェストの形態の場合、自己記述ソフトウェアアーチファクトに関連する前記静的マニフェストを格納する動作と、
　　マニフェストが動的マニフェストの形態の場合、前記動的マニフェストが、ランタイムに構築されて複数のランタイムシステム要素を結び付ける動的メタデータを含むように、関連する実行可能なエンティティのランタイム中に前記動的マニフェストを使用する動作と
　を含む処理を実行するプロセッサ実行可能命令を有することを特徴とする１つまたは複数のコンピュータ記録媒体。
【請求項６】
　前記処理は、前記マニフェストを検査して、前記複数の自己記述ソフトウェアアーチファクトに関する情報を検出する動作をさらに含むことを特徴とする請求項５に記載の１つまたは複数のコンピュータ記録媒体。
【請求項７】
　プロセッサによって実行されると、
　コンピューティングシステム上に永続的に保存された複数の自己記述ソフトウェアアーチファクトと前記複数の自己記述ソフトウェアアーチファクトに関連するマニフェストと

を検査して、前記自己記述ソフトウェアアーチファクトと前記マニフェストに関する情報を収集する動作であって、

各ソフトウェアアーチファクトは、実行可能なエンティティのコンテンツおよび構成を表すオフライン表現を含み、各実行可能なエンティティは、プロセス、アプリケーション、またはオペレーションシステムのコンポーネントのうちの１つであり、

前記マニフェストは、前記複数の自己記述ソフトウェアアーチファクトおよび関連する実行可能なエンティティのメタデータ宣言型記述を含み、

特定の実行可能なエンティティの複数の表現が存在する場合、別個のマニフェストが、前記特定の実行可能なエンティティの各表現に関連付けられ、

各マニフェストは、静的マニフェストの形態または動的マニフェストの形態のいずれかで存在し、

前記静的マニフェストの形態のマニフェストは、ソフトウェアアーチファクトに関連して格納され、

前記動的マニフェストの形態のマニフェストは、ランタイムに構築されて複数のランタイムシステム要素を結び付ける動的メタデータを含むように、関連する実行可能なエンティティのそれぞれのランタイム中に使用される、収集する動作と、

前記複数の自己記述ソフトウェアアーチファクトに対して検証を実行する動作と、

前記検証の結果を報告する動作と

を含む処理を実行するプロセッサ実行可能命令を有することを特徴とする１つまたは複数のコンピュータ記録媒体。

【請求項８】

前記処理は、アプリケーションをコンピュータにインストールする前に、該アプリケーション、および該アプリケーションの構成コンポーネントと依存関係のすべてが、前記コンピュータからアクセス可能であることを検証する動作をさらに含むことを特徴とする請求項７に記載の１つまたは複数のコンピュータ記録媒体。

【請求項９】

前記処理は、新たなアプリケーションのインストールが、既存のアプリケーションと競合しないことを検証する動作をさらに含むことを特徴とする請求項７に記載の１つまたは複数のコンピュータ記録媒体。

【請求項１０】

前記処理は、アプリケーションを、他のアプリケーションとの依存関係を切ることなく削除することができることを検証する動作をさらに含むことを特徴とする請求項７に記載の１つまたは複数のコンピュータ記録媒体。

【請求項１１】

プロセッサによって実行されると、

ソフトウェアベースのコンピュータ上にインストールされたソフトウェアコンポーネントのコンテンツおよび構成を表すオフラインシステムイメージのコピーを獲得する動作であって、前記インストールされたソフトウェアコンポーネントは、前記システムイメージ上で自己記述ソフトウェアアーチファクトとして表わされ、各ソフトウェアアーチファクトは、実行可能なエンティティのコンテンツおよび構成を表すオフライン表現を含み、各実行可能なエンティティは、プロセス、アプリケーション、オペレーティングシステムのコンポーネントのうちの１つであり、各自己記述ソフトウェアアーチファクトは、前記マニフェストのソフトウェアアーチファクトのメタデータ宣言型記述および関連する実行可能なエンティティのメタデータ宣言型記述を含む、関連する永続的に保存されたマニフェストを有し、特定の実行可能なエンティティの複数の表現が存在する場合は、別個のマニフェストが前記特定の実行可能なエンティティの各表現に関連付けられ、各マニフェストは、静的マニフェストの形態または動的マニフェストの形態のいずれかで存在する、獲得する動作と、

第１のマニフェストが前記静的マニフェストの形態で存在する場合、前記第１のマニフェストをソフトウェアアーチファクトに関連して格納する動作と、

第２のマニフェストが前記動的マニフェストの形態で存在する場合、前記第２のマニフェストが、ランタイムに構築されて複数のランタイムシステム要素を結び付ける動的メタデータを含むように、関連する実行可能なエンティティのそれぞれのランタイム中に前記第２のマニフェストを使用する動作と、

前記オフラインシステムイメージの前記自己記述ソフトウェアアーチファクトを分析する動作と、

前記分析する動作の結果を報告する動作と

を含む処理を実行するプロセッサ実行可能命令を有することを特徴とする１つまたは複数のコンピュータ記録媒体。

【請求項１２】

前記処理は、前記分析する動作に基づき、前記オフラインシステムイメージの中の前記ソフトウェアコンポーネントを識別する動作をさらに含むことを特徴とする請求項１１に記載の１つまたは複数のコンピュータ記録媒体。

【請求項１３】

前記処理は、

前記分析する動作に基づき、前記オフラインシステムイメージの中の前記ソフトウェアコンポーネントを識別する動作と、

前記オフラインシステムイメージの中の前記識別されたソフトウェアコンポーネントが、不具合を含むかどうかを判定する動作と

をさらに含むことを特徴とする請求項１１に記載の１つまたは複数のコンピュータ記録媒体。

【請求項１４】

前記コンピューティングシステムの前記ソフトウェアコンポーネントは、オペレーティングシステム要素およびアプリケーションを含むことを特徴とする請求項１１に記載の１つまたは複数のコンピュータ記録媒体。

【請求項１５】

前記オフラインシステムイメージは、前記自己記述ソフトウェアアーチファクトの宣言型記述を有する永続的に保存されたシステムマニフェストを含むことを特徴とする請求項１１に記載の１つまたは複数のコンピュータ記録媒体。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、一般に、１つまたは複数のコンピュータシステムのソフトウェアコンポーネントを管理するための技術に関する。

【背景技術】

【０００２】

従来のソフトウェアベースのコンピュータのコンポーネントを信頼できる形でインストールすること、構成すること、保存すること、および削除することは、ずっと以前から、困難な問題であった。そのようなコンポーネントの例には、以下が含まれる。すなわち、オペレーティングシステム（ＯＳ）、アプリケーションプログラム、デバイスドライバ、アプリケーションプログラミングインタフェース、およびその他のシステムプログラムである。従来のソフトウェアベースのコンピュータの例には、多くの既存のＯＳの１つを実行する、通常の汎用パーソナルコンピュータ（ＰＣ）が含まれる。

【０００３】

ソフトウェアベースのコンピュータは、通常、コンピュータの２次コンピュータ記憶システム（例えば、「ハードディスク」）の永続的なコンテンツ、および構成によって実現される。従来のコンピュータを実現するコンテンツおよび構成は、単に、時間とともに、集中化された監督および調整なしに蓄積したビットの集合に過ぎない。通常、それらの蓄積したビットは、コンピュータの寿命全体にわたる、一連の個々の場当たり的なイベントの結果である。変更の例には、例えば、プログラムのインストール、レジストリキーにお

ける構成設定の変更、ファイルの削除、またはソフトウェアパッチのインストールが含まれる。

【０００４】

ソフトウェアベースのコンピュータは、起動されると、コンピュータがハードディスク上で見出す物ものであれば何であれ、単に実行する。コンピュータのディスク上のコンテンツの正しさは、最終的には、コンピュータの寿命全体にわたる、それらの場当たり的なイベントのそれぞれの正しさに依存するので、コンピュータのコンテンツおよび構成は、容易に破損する、損傷する、ゆがむ、陳腐化する、または単に正しくなくなる可能性がある。

【０００５】

コンピュータのコンテンツおよび構成の正しさは、ウイルス、ワーム、スパイウェアなどによる、悪意のある攻撃が関与する、他の外部で開始された場当たり的なイベントによってさらに脅かされる。ソフトウェアベースのコンピュータのユーザが気付かないうちに、それらの悪意のある攻撃は、コンピュータのコンテンツおよび構成を、たいてい、ユーザの所望とは矛盾する形で、コンピュータのコンテンツおよび構成を変更する。

【０００６】

破損した、損傷した、ゆがんだ、および／または攻撃されたコンピュータのコンテンツおよび構成を検出し、訂正するために、様々な製品およびサービス（例えば、いわゆる「ウイルス対策」ユーティリティおよび「ディスククリーンアップ」ユーティリティ）が、入手可能である。明らかに善意からであるが、それらのプログラムおよびサービスは、さらに別の場当たり的なイベントを、コンピュータのディスク上のビットの結果としての蓄積に導入することにより、問題を単にいっそうひどくする可能性がある。

【０００７】

従来のソフトウェアベースのコンピュータは、本質的に脆弱である。１つの理由は、コンピュータの蓄積したビットの集合が、せいぜい、個々の事例に基づく（ａｎｅｃｄｏｔａｌ）不完全な記述を有するからである。それらの不完全な記述は、単に、同一の一連の場当たり的なイベントの結果に過ぎず、ディスク上のビット、またはそれらのビットをもたらした一連のイベントを体系的に記述しない。また、それらの記述には、システム構成が、どのようであるべきかの、完全な、または部分的な指定、あるいは指定に照らして状態を確認する方法も揃えられていない（ｕｎｍａｔｃｈｅｄ）。

【０００８】

以下の事実が、従来のソフトウェアベースのコンピュータの欠点を示す。すなわち、任意のオフライン「システムイメージ」を所与として、一般に、そのシステムイメージが、動作可能なＯＳ、または特定の動作可能なアプリケーションを含むと決定的に判定することができない。システムイメージは、従来のソフトウェアベースのコンピュータのハードディスク上に通常に保存された（ｐｅｒｓｉｓｔｅｄ）コンテンツおよび構成情報のビットごとに対応する（ｂｉｔ－ｆｏｒ－ｂｉｔ）コピーである。それらのコンテンツおよび構成が、前述したとおり、コンピュータを実現する。

【０００９】

システムイメージを所与として、そのイメージに特定のファイルが存在するかどうかを確認することができる。この確認は、特定のＯＳ、または特定のアプリケーションに関して、いずれの特定のファイルがインストールされるかについての経験的な知識を利用して、行われることが可能である。しかし、そのような経験的な証拠は、必要なコンポーネント（特定のＯＳ、または特定のアプリケーションの）のすべてがインストールされているかどうかを明かさない。そのような経験的な証拠は、競合するコンポーネントがインストールされているかどうかを明かさない。さらに、そのような経験的な証拠は、コンポーネント（特定のＯＳ、または特定のアプリケーションの）のすべてが、動作可能なコンピュータをもたらすように正しく構成されているかどうかを明かさない。そのような確認は、必要であるが、不十分である。

【００１０】

特定のＯＳまたはアプリケーションが機能するのに必要な特定のファイルのすべての存在を経験的に特定した場合でも、その事実は、イメージ上の特定のＯＳまたはアプリケーションが正しく機能することを知るのに十分ではない。この場合も、それらの検査は、必要であるが、十分ではない。

【００１１】

実際、唯一の有効な従来の頼みの綱は、イメージのオフライン検査を放棄し、オンライン検査を実施することである。システムイメージを使用してコンピュータを起動し、結果を観察することができる。この従来のアプローチは、しばしば、実際的でなく、不十分であり、危険である。明らかに、このアプローチは、スケーラブル（ｓｃａｌａｂｌｅ）ではない。

【００１２】

オンライン検査の従来のアプローチを使用しても、いずれの特定のアプリケーションおよび／またはＯＳコンポーネントがインストールされているか、もしくは、現在、実行されているかさえ、識別することは、しばしば、間違いなく困難である。しばしば、確認しているのは、特定の名前を有するアプリケーションまたはコンポーネントが、コンピュータ上に存在することだけである。この確認は、ソフトウェア開発者が、誤解させるような名前の使用を避けていることに頼る。そのような誤解させるような名前は、不注意で、または意図的に生じる可能性がある。

【００１３】

例えば、市販される既存のＯＳ（Ｗｉｎｄｏｗｓ（登録商標）ＸＰまたはＬｉｎｕｘのような）は、プロセスを開始するのに使用されるファイルの名前によって、実行中のプロセスのリストを示すことが可能であるが、それぞれの実行中のプロセスの名前は、プロセスの真の正体についての「ヒント」に過ぎない。例えば、当たり障りのない名前が付いたプロセスが、ウイルスによって乗っ取られ、別の潜在的に悪意のあるタスクに変えられている可能性がある。代替として、適切に名前の付けられたプロセスが、例えば、管理者が、一見、無関係のように見えるアプリケーションをインストールすることにより、壊れている可能性がある。

【００１４】

従来のソフトウェアベースのコンピュータでは、低レベルのソフトウェア抽象化と高レベルのソフトウェア抽象化の間に、記述的な構造上のリンクは、全く存在しない。これは、低レベルのソフトウェア抽象化と高レベルのソフトウェア抽象化を記述的に、必然的にリンクする、従来のソフトウェアベースのコンピュータの構造またはアーキテクチャに本来備わったものは何も存在しないことを意味する。

【００１５】

低レベルのソフトウェア抽象化には、例えば、ディスク上の特定のファイル群（例えば、ロードモジュール群）、およびコンピュータ上で実行される特定のプロセス群が含まれる。高レベルのソフトウェア抽象化には、例えば、アプリケーションプログラム群（例えば、Ｍｉｃｒｏｓｏｆｔ（登録商標）Ｐｕｂｌｉｓｈｅｒ（登録商標）デスクトップパブリッシング製品）、およびアプリケーションのファミリ（例えば、オフィス生産性製品のＭｉｃｒｏｓｏｆｔ（登録商標）Ｏｆｆｉｃｅ（登録商標）スイート）が含まれる。

【００１６】

例えば、アプリケーションプログラムの概念は、ユーザ中心モデルの一部である。ユーザが特定の作業（例えば、ワードプロセッシング、スプレッドシート分析、およびデータベース管理）を達するのを助けるアプリケーションプログラム（またはグループのプログラム）が、ユーザに見える場合に、従来のソフトウェアベースのコンピュータには、単に、１つまたは複数のアクティブなプロセスが見える。アクティブなプロセス（およびそれらのプロセスのロードモデルソース）を、ユーザに見える（通常、グラフィカルユーザインタフェース（ＧＵＩ）プロセスを介して）アプリケーションプログラムの表現に記述的に、必然的にリンクする、従来のソフトウェアベースのコンピュータのアーキテクチャに本来備わったものは何も存在しない。

【００１７】
いくつかの文献に上述のような従来の技術に関連した技術内容が開示されている（例えば、特許文献１、２参照）。
【００１８】
【特許文献１】米国特許第６，２９２，９４１号明細書
【特許文献２】米国特許第６，４８７，７２３号明細書
【発明の開示】
【発明が解決しようとする課題】
【００１９】
従来のシステムには上述したような種々の問題があり、さらなる改善が望まれている。
【００２０】
本発明は、このような状況に鑑みてなされたもので、その目的とするところは、自己記述アーチファクトおよびアプリケーション抽象化を提供することにある。
【課題を解決するための手段】
【００２１】
本明細書で説明するのは、ソフトウェアベースのコンピュータの１つまたは複数のコンピュータ記憶媒体上に保存された複数の自己記述ソフトウェアアーチファクト（artifact）を使用する少なくとも１つのインプリメンテーションである。このインプリメンテーションでは、各アーチファクトは、コンピューティングシステムのソフトウェアコンポーネント（例えば、ロードモジュール、プロセス、アプリケーション、およびオペレーティングシステムコンポーネント）の少なくとも一部分を表し、各アーチファクトは、関連するアーチファクトのメタデータ宣言型記述（ｍｅｔａｄａｔａ　ｄｅｃｌａｒａｔｉｖｅ　ｄｅｓｃｒｉｐｔｉｏｎ）を含む、少なくとも１つの関連する「マニフェスト」によって記述される。
【００２２】
同一の符号は、すべての図面で、同様の要素および特徴を指すのに使用される。
【発明を実施するための最良の形態】
【００２３】
以下、図面を参照して本発明を適用できる実施形態を詳細に説明する。
【００２４】
以下の説明は、自己記述ソフトウェアアーチファクトを使用するソフトウェアベースのコンピュータのためのコンピューティング技術を実施する技術を提示する。それらの技術の例示的なインプリメンテーションは、「例示的な自己記述アーチファクトアーキテクチャ」と呼ぶことができる。
【００２５】
例示的な自己記述アーチファクトアーキテクチャは、コンピュータサイエンスの分野に新鮮で、活性化をもたらすアプローチを提供する。ソフトウェアベースのコンピュータの寿命の間における一連の場当たり的なイベントからもたらされたビットの蓄積に過ぎないのではなく、この新たなアーキテクチャを利用するコンピュータのコンテンツおよび構成は、自己記述ソフトウェアアーチファクトの整理され、安定性があり、信頼できる、堅牢で、決定論的に構築可能な集合である。
【００２６】
新たなアーキテクチャを記述する前に、用語を簡単に説明するのが適切である。本明細書で使用する以下の用語を、ここで簡単に定義する。しかし、読者は、完全な説明の文脈で各用語の完全な意味を理解し、認識するように、完全なテキストを読むよう勧められる。
・ソフトウェアアーチファクト（または、単に「アーチファクト」）は、実行可能なエンティティ（例えば、プロセス、アプリケーション、オペレーティングシステムのコンポーネント）のオフライン表現（ｍａｎｉｆｅｓｔａｔｉｏｎ）であり、例えば、ロードモジュールおよび構成ファイルが含まれる。

・マニフェストは、実行可能なエンティティのメタデータ宣言型記述である。マニフェストは、実行可能なエンティティの各表現に関連することが可能である。マニフェストは、静的であっても、動的であってもよい。
・プロトタイプは、実行可能なエンティティの実行可能な（ｅｘｅｃｕｔａｂｌｅ）（または「実行可能な（ｒｕｎａｂｌｅ）」）表現であるが、エンティティのプロトタイプは、実行状態にはない。
・抽象化は、実行可能なエンティティが実行状態にある（「エンティティが実行されている」）場合の、実行可能なエンティティの表現である。
・コンポーネントは、実行可能なエンティティの表現の一部分、一部、または一構成要素であり、例えば、アプリケーションは、プロセスコンポーネントを含み、プロセスは、実行可能な命令をコンポーネントとして含む。

【００２７】

例示的な自己記述アーチファクトアーキテクチャ

図１は、例示的な自己記述アーチファクトアーキテクチャ１００の１つの図を示す。この図では、アーキテクチャ１００は、メモリ１１０（例えば、揮発性メモリ、不揮発性メモリ、リムーバブルなメモリ、リムーバブルでないメモリなどの）を備えて構成された、ソフトウェアベースのコンピュータ１０２上で実施される。コンピュータ１０２は、メモリ１１０内でアクティブなオペレーティングシステム（ＯＳ）１１２を有する。

【００２８】

コンピュータ１０２は、少なくとも１つのコンピュータ記憶装置１２０（例えば、「ハードディスク」）へのアクセスを有する。コンピュータ記憶装置１２０は、コンピュータ１０２を実現するコンテンツおよび構成を含む。コンテンツには、オペレーティングシステム（ＯＳ）、ＯＳ要素、すべてのインストール済みのアプリケーション、および他のすべての関連するコンポーネント（例えば、デバイスドライバ、インストールファイル、データ、ロードモジュールなど）を含む（例として、限定としてではなく）、様々なソフトウェアコンポーネントが含まれる。構成には、各ソフトウェアコンポーネントの指定されたプロパティ、およびコンポーネント間の定義された相互関係が含まれる。

【００２９】

この説明では、「システム」についての言及は、記憶装置１２０のコンテンツおよび構成によって実現された、ソフトウェアベースのコンピュータ１０２を表す。システムの保存されたオフライン（すなわち、実行されていない）コピーを、本明細書では、「システムイメージ」と呼ぶことができる。

【００３０】

図１は、例えば、記憶装置１２０上に格納された３つのアーチファクト（１３０、１４０、および１５０）を示す。本明細書で、「ソフトウェアアーチファクト」、または単に「アーチファクト」は、コンピュータ記憶装置１２０上に格納された個々のソフトウェアアイテムの集合である。それらのアイテムの一部は、ファイルシステム、データベース、構成レジストリなどを含め、様々なシステムストアの中に格納されてもよい。それらのアーチファクトは、システムを実現するコンテンツおよび構成を表す。コンピュータの記憶装置は、多数のアーチファクトを有することが可能である。コンピュータのシステムイメージは、多数のアーチファクトを含む。

【００３１】

従来のソフトウェアベースのコンピュータとは異なり、コンピュータ１０２のアーチファクトは、単に、コンピュータの寿命の間における一連の場当たり的なイベントからもたらされるビットの蓄積ではない。それどころか、コンピュータ１０２のアーチファクトのそれぞれは、少なくとも１つのマニフェストに関連する。例えば、システムアーチファクト１３０は、アーチファクト１３０とともに関連するマニフェスト１３２を格納しているか、または記憶装置１２０上の何らかの導き出せる（ｄｅｒｉｖａｂｌｅ）ロケーション、または既知のロケーションに格納している。アーチファクト１４０は、関連するマニフェスト１４２を、アーチファクト１５０は、関連するマニフェスト１５２をそれぞれ有す

る。
【００３２】
それらのアーチファクトは、アーチファクトのそれぞれが（関連するメタデータマニフェストを介して）、自己を記述するため、「自己記述アーチファクト」と呼ばれる。宣言型（例えば、実行されるべきアクションのリスト）であるのではなく、自己記述メタデータ記述は、アーチファクトの所望される状態の宣言型記述である。
【００３３】
各記述は、アーチファクトに整合性があり、正しいために必要な状態の完全な規範的レコードである。類比により（ｂｙ　ａｎａｌｏｇｙ）、指示セットが手続き型である一方で、正確なアドレスは、宣言型であり、新たな計算を許す、例えば、そのアドレスを使用して、異なる開始点に関する新たな指示セットを特定することができるという意味で、より強力である。宣言型記述には、アーチファクトの相互依存関係、およびシステムの他のコンポーネントとの相互関係の全部のレコードが含まれる。
【００３４】
それらのメタデータ記述は、低レベルのソフトウェア抽象化と高レベルのソフトウェア抽象化を実質的に橋渡しする。低レベルのソフトウェア抽象化には、例えば、記憶装置上の特定のアーチファクト（例えば、ロードモジュール）、およびコンピュータ上で実行される特定のプロセスが含まれる。高レベルのソフトウェア抽象化には、例えば、実行中のアプリケーションプログラム、およびアプリケーションのファミリが含まれる。また、高レベルのソフトウェア抽象化には、実行中のオペレーティングシステム（ＯＳ１１２のような）およびそのようなＯＳの諸要素も含まれることが可能である。
【００３５】
図１に示すとおり、コンピュータ１０２は、メモリ１１０内に、コンピュータ１０２上のアーチファクトの自己記述プロパティを利用する３つの監督機能および管理機能のコンポーネントを有する。それらの機能コンポーネントには、自己記述アーチファクトマネージャ１６０、実行ゲートキーパ１６２、およびシステムベリファイア（ｖｅｒｉｆｉｅｒ）１６４が含まれる。
【００３６】
それらの機能コンポーネントのそれぞれを、図１では別々に示しているが、コンポーネントの機能を結合して、より少ないコンポーネントにしても、追加のコンポーネントに拡張してもよい。さらに、それらの機能コンポーネントは、コンピュータのＯＳ１１２の一部であっても、コンピュータ１０２のＯＳではないコンポーネントの一部であってもよい。
【００３７】
自己記述アーチファクトマネージャ１６０は、記憶装置１２０上の自己記述アーチファクトを管理する。その管理の一環として、マネージャは、アーチファクトの保存（ｐｅｒｓｉｓｔｅｎｃｅ）および構造化を容易にすることができる。マネージャは、各アーチファクトとそのアーチファクトのマニフェストの間の関連付けの保持を確実にすることができる。マネージャは、アーチファクトの、そのアーチファクトのマニフェストの中の記述に対する整合性を確実にすることができる。さらに、マネージャは、システムの構成の変更に応答して、自己記述アーチファクトを更新することができる。
【００３８】
自己記述アーチファクトマネージャ１６０は、記憶装置１２０からのアーチファクトの読み込みに関する最適化を支援することができる。自己記述アーチファクトマネージャは、コンピュータのアプリケーション群およびＯＳの全体的動作のより広い見通しを有することができる。その見通しに基づき、自己記述アーチファクトマネージャ１６０は、いずれのアーチファクト（例えば、ロードモジュール）が、特定のアプリケーションためのプロセスにおいて組み合わせられるかを特定することができる。次に、マネージャは、アーチファクトを組み合わせて、より少ない、おそらく最適化された数のアーチファクトにすることができる。同様に、システムマニフェスト（システム全体の宣言型記述を含む）を

使用して、いずれのアプリケーションが間もなく呼び出されるかを特定して、自己記述アーチファクトマネージャは、一部のアプリケーションの起動を、それらのアプリケーションが実際に呼び出される前に、促すことができる。

【００３９】

実行ゲートウェイ１６２が、アプリケーション（および、場合により、他の実行可能なコンポーネント）の呼び出しを許可する。アプリケーションの呼び出しが、保留中である際、実行ゲートキーパ１６２は、アプリケーションの関連する自己記述アーチファクトを検査する。自己記述アーチファクトの中の宣言型記述は、関連するアプリケーションに要求される明示的な静的状態または動的状態を指定することが可能である。典型的な明示的条件の例が、アプリケーションの実行が成功するためにシステム上に存在しなければならない、必要なコンポーネントのリストである。典型的な明示的条件の別の例が、アプリケーション自体が起動される前に、起動されていなければならない（または、指定された現在の状態にある）アプリケーション群およびシステムコンポーネント群のリストである。

【００４０】

ゲートキーパは、現在の条件を検査し、それらの条件が、関連するアーチファクトの宣言型記述の中で指定された諸要件を満たす場合、アプリケーションの呼び出しを許す。諸要件を満たさない場合、ゲートキーパは、アプリケーションの呼び出しを許可しない。

【００４１】

実行ゲートキーパ１６２の支配で、いずれのコードも、そのコードが、関連するマニフェストの中で記述されていない限り、コンピュータ１０２上で実行されない。一実施形態では、関連するマニフェストの中で記述され、信頼されるソフトウェア発行者によって署名されたコードだけが、コンピュータ１０２上にインストールされ、実行されることが可能である。

【００４２】

さらに、実行ゲートキーパ１６２は、外部変更（例えば、悪意のないデータ破損、または悪意のある攻撃を介する）に対する検査として、システムの整合性の監査を実行することができる。監査は、システムの自己記述アーチファクトのマニフェストに基づく。

【００４３】

システムベリファイア１６４が、自己記述アーチファクトに対して１つまたは複数の検証を実行する。ベリファイア１６４は、そうするようにという手動の要求に応答して、アクション（例えば、新たなソフトウェアのインストール）に応答して、新たな情報が利用可能になったことに応答して、かつ／またはそうするようにスケジュールされて、検証を実行することができる。さらに、システムベリファイア１６４によって行われる検証は、少なくともある程度、自己記述アーチファクトのマニフェストの検査から収集された情報に基づく。

【００４４】

システムベリファイア１６４は、以下の検証の１つまたは複数を実行する。すなわち、

・コンピュータ１０２にインストール済みのソフトウェアのすべての依存関係が満たされていること、

・コンピュータ１０２のオペレーティングシステムが、コンピュータ１０２のハードウェア構成上で実行されるのに必要なすべてのデバイスドライバを含むこと、

・コンピュータ１０２におけるコードおよび構成設定が、偶然に、または悪意で変更されていないこと、

・アプリケーションが、コンピュータ１０２に正しくインストールされていること、

・既知の障害のある、または悪意のあるプログラムが、コンピュータ１０２にインストールされていないこと、

・インストール前に、アプリケーション、ならびにそのアプリケーションの構成コンポーネントおよび依存関係のすべてが、コンピュータ１０２上に存在すること、

・アプリケーションのコンポーネントをシステムに読み込む前に、そのアプリケーションが、コンピュータ１０２にインストール可能であること、

・新たなアプリケーションまたはシステムコンポーネントのインストールが、既存のアプリケーション群またはコンポーネント群と競合しないこと、
・アプリケーションまたはオペレーティングシステムコンポーネントを、他のアプリケーション群またはコンポーネント群からの依存関係を切ることなしに、削除することができること、
・アプリケーションまたはオペレーティングシステムが、事前定義されたローカルポリシーを遵守すること、
・アプリケーション起動のための必要な事前条件が、アプリケーションが起動される前に満たされること、
・事前条件が満たされないアプリケーションは、実行されることが許されないこと、および
・実行中、実行のための必要条件が、もはや有効でない場合、アプリケーションは、もはや実行されることが許されないことである。

【００４５】

図１に示すとおり、別のコンピュータ１７０が、独自のメモリ１７２（例えば、揮発性メモリ、不揮発性メモリ、リムーバブルなメモリ、リムーバブルでないメモリなど）を備えて構成される。このメモリは、内部にシステムインスペクタ（ｉｎｓｐｅｃｔｏｒ）１８０を有する。

【００４６】

大きい矢印（ｌａｒｇｅ－ｈｅａｄｅｄ　ａｒｒｏｗ）で表すとおり、システムインスペクタ１８０は、入力として、コンピュータ１０２の「システムイメージ」を受け取る。つまり、インスペクタ１８０は、コンピュータ１０２のシステムを実現するコンテンツおよび構成のコピーを受け取る。このコピーは、コンピュータ１０２から直接に受け取られても、「システムイメージ」の別個のコピーとして間接的に受け取られてもよい。

【００４７】

システムインスペクタ１８０は、オフライン「システムイメージ」の分析を実行して、コンピュータ１０２が、特定の機能コンポーネント（ＯＳまたはアプリケーション群などの）を含むと決定的に検証する。より詳細には、インスペクタは、自己記述アーチファクトを検査して、必要なコンポーネント（自己記述アーチファクトのマニフェストによって、そのようなコンポーネントとして記述され、参照される）のすべてが、見つかり、適切に識別されるかどうかを調べる。インスペクタは、その検査の結果を報告する。

【００４８】

アーチファクトに関するマニフェストの中に含まれる情報は、アーチファクトのライブラリ、コンポーネント、および依存関係のすべてを記述することにより、コンパイラ、または他の最適化ツールによって、インストール時、プログラム読み込み時、またはユーザが選択した別の時点で、アーチファクトの中のコードの最適化を円滑にするのに使用されることが可能である。この記述により、コンパイラまたはツールは、アーチファクトがアーチファクトの中のコードを実行する環境についての、より正確な想定を行うことができるようになる。

【００４９】

アーチファクトに関するマニフェストの中に含まれる情報は、アーチファクトのライブラリ、コンポーネント、および依存関係のすべてを記述することにより、エラー検出ツールによって、インストール時、プログラム読み込み時、またはユーザが選択した別の時点で、アーチファクトの中のコードの正しさを確実にすることを円滑にするのに使用されることが可能である。この記述により、ツールが、アーチファクト、ならびにアーチファクトの中のコードが実行される環境についての、より正確な想定を行うことができるようになる。

【００５０】

マニフェスト

マニフェストは、コンピュータ１０２のアーチファクトを記述するメタデータを含む。

また、メタデータは、外部依存関係および外部インタフェースを含め、アーチファクトに関係する構成情報も記述する。また、マニフェストは、ソフトウェアコンポーネント間の接続関係も記述する。

【００５１】

例えば、発行者によって提供される「ＰｒｏｇｒａｍＡ」と呼ばれるアプリケーションに関するマニフェストは、バイナリロードモジュール（ＥＸＥ、ＤＬＬなど）のリスト、コンポーネントおよび提供元の真正性を証明する証明書、構成設定の名前と、それらの設定の既定値のリスト、プログラムのロードモジュールによって要求される外部バイナリロードモジュールのリスト、プログラムによってアクセスされる外部の設定および名前のリスト、ならびに、ＰｒｏｇｒａｍＡが、「．ＺＺＺ」という拡張子を有するファイルに関する既定のエディタであるのを望むことをオペレーティングシステムに知らせるのに要求される情報のような、ＰｒｏｇｒａｍＡによって公開される名前および設定のリストを含む。

【００５２】

少なくとも１つの実施形態では、自己記述アーチファクトのマニフェストは、以下を可能にするのに十分な情報を提供する宣言型記述を含む。すなわち、

・プログラムインストールソフトウェアが、プログラム内に含まれるいずれの場当たり的なコードも実行することなしに、プログラムコンポーネントのインストールまたはアンインストールを行うこと、

・プログラムをインストールする個人またはエージェントが、プログラムコンポーネントを他のプログラム群またはリソース群にバインドすること、

・プログラムをインストールする個人またはエージェントが、あらゆる構成可能な既定の設定を無効にすること、

・プログラムをインストールする個人またはエージェントが、プログラムを、システム全体に関するマニフェストの一部にすること、

・特定のプログラムのインストールまたはアンインストールが正しく行われたことを、インスペクタソフトウェア（例えば、システムインスペクタ１８０）が、プログラム独自のコンテキストと、システム全体のコンテキストの両方において検証すること、

・特定のプログラムが、現在、実行可能であるかどうかを、検証ソフトウェア（例えば、システムベリファイア１６４）が判定すること、

・特定のプログラムがインストールされ、実行されることを可能にするのに必要なすべての依存関係が、システム上で満たされているかどうかを、プレインストール（ｐｒｅ－ｉｎｓｔａｌｌａｔｉｏｎ）ソフトウェアが判定すること、

・特定のプログラムをインストールする個人またはエージェントが、プログラム、またはシステム全体の将来の動作の他の諸態様を特定すること、

・コンパイラまたは最適化ツールが、アーチファクトの中のコード、ならびにそのコードが実行される環境の正確な記述を得ること、および

・エラー検出ツールが、アーチファクトの中のコード、ならびにそのコードが実行される環境の正確な記述を得ることである。

【００５３】

マニフェストは、プログラムまたはシステムに関する、すべての入手可能なメタデータを含む必要はないが、さらなるメタデータを信頼できる形で探し出すことを可能にするのに十分な情報を提供する必要がある。一実施形態では、例えば、あるプログラムに関するバイナリロードモジュール（ＥＸＥ、ＤＬＬなど）が、特定のロードモジュールに関連するメタデータを参照するメタデータを含む。この実施形態では、マニフェストは、ロードモジュール内部の、そのさらなるメタデータの存在をシステムに知らせる。

【００５４】

一実施形態では、マニフェストは、マニフェストの各サブコンポーネントのタイプを明らかにする。サブコンポーネントタイプは、サブコンポーネントのコンテンツを解釈し、コンポーネントからさらなるメタデータを抽出し、コンポーネントに関するさらなるメタ

データを導出することをどのように行うかを知っている、ヘルパ（ｈｅｌｐｅｒ）ソフトウェアを明らかにする。

【００５５】

例えば、一実施形態では、マニフェストの中で記述されたロードモジュールは、それらのモジュールが、通信要件の遵守のような、あるソフトウェアプロパティに従うかどうかを、検証ツールが判定することを可能にする、抽象命令セットで表現される。ロードモジュールに関するマニフェストは、各ロードモジュールのために使用される正確な抽象命令セットを明らかにして、いずれのヘルパソフトウェアを読み込んで、通信要件などの特定のシステムプロパティを検証するかを、システムベリファイア１６４が特定することができるようにする。

【００５６】

ある実施形態のさらに別の態様では、システムの諸部分の間の互換性を判定するのに使用される情報が、コンポーネントと独立に提供されるとともに、コンポーネントと一緒にも提供される。その情報は、多くのソースから届くことが可能であり、ローカル管理者またはローカルエージェントは、部分的な情報、または矛盾する情報のあいまいさを除くために、規則を定義すること、または規則に従うことができる。

【００５７】

ある実施形態のさらに別の態様では、システムの諸部分の間の互換性を判定するのに使用される情報は、新たな情報が、適切な場所で入手可能になるにつれ、または古い情報が取り消されるにつれ、時間とともに変化する。

【００５８】

マニフェストは、組み合わせて、任意の複雑さのソフトウェアシステムを記述するグラフにすることができる。マニフェストは、依存関係として外部マニフェストを参照することができる。また、マニフェストは、埋め込まれたマニフェストを含むことも可能である。一実施形態では、アプリケーションに関するマニフェストは、アプリケーションのサブコンポーネントに関するマニフェストを含むか、またはそのようなマニフェストを使用する。

【００５９】

アプリケーションの発行者によって行われたパッケージ決定に依存して、サブコンポーネントは、マニフェストの中に埋め込まれることも、外部エンティティとして参照されることも可能である。一実施形態では、外部依存関係は、依存関係の名前およびバージョン番号、または他の明確にする情報を含む。別の実施形態では、外部依存関係には、署名入りのダイジェスト（ｓｉｇｎｅｄ　ｄｉｇｅｓｔ）を介して名前が付けられる。別の実施形態では、この情報は、更新されること、取り消されること、および明確にされること（すなわち、あいまいさが除かれること）が可能である。

【００６０】

一実施形態では、アプリケーションに関するマニフェストは、パッケージされ、関連するアプリケーションとともに提供される。別の実施形態では、アプリケーションに関するマニフェストは、パッケージされ、関連するアプリケーションとは別個に提供される。これにより、おそらく、アプリケーションの複数のコンポーネントが、別々に、関連するマニフェストの提供の後に、提供されることが可能である。

【００６１】

一実施形態では、外部マニフェストは、外部情報のソースとして参照されることも可能である。それらの外部マニフェストには、個々に名前が付けられても、グループのメンバとして名前が付けられてもよい。

【００６２】

２つの形態のマニフェスト、すなわち、静的マニフェストおよび動的マニフェストが存在する。静的マニフェストは、ソフトウェアアーチファクトに関連して格納される。動的マニフェストは、そのマニフェストに関連する実行可能なコンポーネントのランタイム中に使用される。動的マニフェストは、静的メタデータ（やはり、ランタイムに入手可能な

）、ならびに、ランタイムに構築されて、プロセスやオペレーティングシステムオブジェクトのようなランタイムシステム要素を結び付ける、さらなる動的メタデータを含む。

【００６３】

この態様は、低レベルのインプリメンテーションの概念から、より高いレベルの概念にまで橋渡しすることをさらに可能にする。ソフトウェアアーチファクトの自己記述の特徴は、静的システム上だけでなく、実行中のアクティブなシステム上でも役立つ。例えば、実行中のプロセスシステムの「適格さ（ｗｅｌｌ－ｆｏｒｍｅｄｎｅｓｓ）」および／または整合性は、システムイメージの検証と同様の形で検証することができる。

【００６４】

一実施形態では、「ueber」マニフェストは、コンピュータ１０２上で、直接に、または間接的に利用可能なすべてのソフトウェアを、そのようなソフトウェアがインストールされているか否かに関わらず、記述する。

【００６５】

例示的な自己記述アーチファクトの管理およびゲートキーピングの方法上のインプリメンテーション

図２は、自己記述アーチファクトマネージャ１６０および／または実行ゲートキーパ１６２によって実行される方法２００を示す。この方法上のインプリメンテーションは、ソフトウェアで実行されても、ハードウェアで実行されても、ソフトウェアとハードウェアの組み合わせで実行されてもよい。理解を容易にするため、方法は、図２に独立のブロックとして表す別々のステップとして示すが、それらの別々に示したステップは、ステップの実行の際に必ずしも順序に依存していると解釈してはならない。さらに、説明のため、方法２００は、図２を参照して説明する。

【００６６】

図２の２１０で、自己記述アーチファクトマネージャ１６０が、マニフェストを関連するアーチファクトと一緒に保存することを円滑にする。図１に示したとおり、マニフェスト１３２は、システムアーチファクト１３０に関連して保存されるか、または記憶装置１２０上の何らかの導き出すことができるロケーション、または既知のロケーションに格納される。同様に、マニフェスト１４２および１５２は、アプリケーションアーチファクト１４０および１５０に関連して格納される。

【００６７】

２１２で、自己記述アーチファクトマネージャ１６０は、システムのコンテンツおよび／または構成の変化に応じて、システムの自己記述アーチファクトを更新する。そのような変化は、例えば、新たなコンテンツのインストール、手動の構成変更、およびオペレーティングシステムによって実行された自動的構成変更の結果であることが可能である。更新は、適用される前に、システムマニフェストの集合のコンテキストにおいてチェックされて、適用された場合、更新が、存立可能なシステムをもたらすことが確実にされることが可能である。

【００６８】

２１４で、自己記述アーチファクトマネージャ１６０は、実行のためにアーチファクトの使用を最適化する。マネージャは、いずれのロードモジュールが、アプリケーションに関するプロセスの中で組み合わせられるかを特定することができる。次に、マネージャは、ロードモジュールを組み合わせて、一緒に最適化されている、より少ない数のロードモジュールにすることができる。同様に、システムマニフェストを使用して、いずれのアプリケーションが間もなく呼び出されるかを特定して、自己記述アーチファクトマネージャは、一部のアプリケーションの起動を、それらのアプリケーションが実際に呼び出される前に、促すことができる。

【００６９】

２１６で、実行ゲートキーパ１６２は、関連する自己記述アーチファクトを検査して、関連するアプリケーション（または他のプログラム）の実行を許すかどうかを、現在の条件、および関連するマニフェストの宣言型記述に基づき、判定する。そのような判定が行

われると、ゲートキーパは、関連するアプリケーション（または他のプログラム）の実行を制限する、または阻止することができる。

【００７０】

例えば、コンピュータのローカルポリシーにより、いずれのアプリケーションを呼び出すことができ、いずれのアプリケーションを呼び出すことができないか、ならびにアプリケーションを呼び出すことができる形が、正確に記述される可能性がある。記述される場合、ゲートキーパは、指定された形での呼び出しだけを許す。

【００７１】

２１８で、システムベリファイア１６４が、外部変更に対して、システムの整合性を監査する。監査は、システムの自己記述アーチファクトのマニフェストに基づく。例えば、ロードモジュールのマニフェストは、１つまたは複数の関連するロードモジュールのコンテンツの署名入りダイジェストを含むことが可能である。ゲートキーパは、すべてのロードモジュールのコンテンツを定期的に確認して、それらのコンテンツが、それらのコンテンツの指定されたダイジェストと合致するかどうかを調べることができる。

【００７２】

例示的なシステム検証の方法上のインプリメンテーション

図３は、システムベリファイア１６４によって実行される方法３００を示す。この方法上のインプリメンテーションは、ソフトウェアで実行されても、ハードウェアで実行されても、ソフトウェアとハードウェアの組み合わせで実行されてもよい。理解を容易にするため、方法は、図３に独立のブロックとして表す別々のステップとして示すが、それらの別々に示したステップは、ステップの実行の際に必ずしも順序に依存していると解釈してはならない。さらに、説明のため、方法３００は、図３を参照して説明する。

【００７３】

図３の３１０で、システムベリファイア１６４は、トリガイベントに応答する。トリガイベントの例には（例として、限定としてではなく）、手動検証要求を受け取ること、別のプログラム（例えば、インストールソフトウェア）によるアクションの実行、およびスケジュール時刻イベントが含まれる。そのトリガイベントは、識別可能であり、特定のタイプの所望される検証に関連することが可能である。

【００７４】

３１２で、システムベリファイア１６４が、自己記述アーチファクトのマニフェストを検査して、それらのマニフェストから情報を収集する。

【００７５】

３１４で、ベリファイアは、コンピュータ１０２のオンラインのアクティブなシステムの検証を実行する。より詳細には、この検証は、自己記述アーチファクトの検証である。可能なこととして、それらの検証は、オフラインの「システムイメージ」に対しても実行されることが可能である。

【００７６】

ベリファイアによって実行される検証は、完全に機能する条件におけるシステムの安定性、整合性、および堅牢性を促進するように設計される。以下は、システムベリファイア１６４によって実行されることが可能な、例示的な検証のリストである（リストは、例として、限定としてではなく、提供されている）。すなわち、

・コンピュータ１０２にインストール済みのソフトウェアのすべての依存関係が満たされていることを検証すること、

・コンピュータ１０２のオペレーティングシステムが、コンピュータ１０２のハードウェア構成上で実行されるのに必要なすべてのデバイスドライバを含むことを検証すること、

・コンピュータ１０２におけるコードが、偶然に、または悪意で変更されていないことを検証すること、

・アプリケーションが、コンピュータ１０２に正しくインストールされていることを検証すること、

・既知の障害のある、または悪意のあるプログラムが、コンピュータ１０２にインストー

ルされていないことを検証すること、
・インストール前に、アプリケーション、ならびにそのアプリケーションの構成コンポーネントおよび依存関係のすべてが、コンピュータ１０２上に存在することを検証すること、
・アプリケーションのコンポーネントをシステムに読み込む前に、そのアプリケーションが、コンピュータ１０２にインストール可能であることを検証すること、
・新たなアプリケーションまたはシステムコンポーネントのインストールが、既存のアプリケーション群またはコンポーネント群と競合しないことを検証すること、
・アプリケーションまたはオペレーティングシステムコンポーネントを、他のアプリケーション群またはコンポーネント群からの依存関係を切ることなしに、削除することができることを検証すること、および
・アプリケーションまたはオペレーティングシステムが、事前定義されたローカルポリシーを遵守することを検証することである。

【００７７】

３１６で、ベリファイアは、検証の結果を何であれベリファイアを呼び出したもの（例えば、ＯＳ１１２および／またはユーザ）に報告する。

【００７８】

システム検査

この新たなアーキテクチャは、従来のソフトウェアベースのコンピュータの欠点の多くを克服する。例えば、新たなアーキテクチャを使用するソフトウェアベースのコンピュータの任意のオフライン「システムイメージ」を所与として、実際、そのイメージが、動作可能なＯＳ、または特定の動作可能なアプリケーションを含むと決定的に判定することができる。これは、従来のアーキテクチャを使用するソフトウェアベースのコンピュータでは、行うことができない。

【００７９】

各アーチファクトのマニフェストは、ソフトウェアベースのコンピュータのコンテンツおよび構成の永続的（ｐｅｒｓｉｓｔｅｎｔ）「システムイメージ」内に格納される（または、そのような「システムイメージ」とともに取り出すことができる）。アーチファクトは、関連するメタデータ（アーチファクトのマニフェストの）が、イメージがオフラインである場合に、検査されることが可能である（例えば、システムインスペクタ１８０によって）ような形で格納される。さらに、イメージの他のコンテンツも、検査されることが可能である。

【００８０】

以上を行う際に、システムインスペクタ１８０は、システムのコンテンツ、および将来の動作について明確な言明（ｓｔｒｏｎｇ　ｓｔａｔｅｍｅｎｔ）を行う。これは、システムイメージのメタデータ、および他の部分が、分散ストアにわたって分散しており、それらが、システムが起動され、実行される際にだけ一緒になるようになっている場合でも、可能である。自己記述アーチファクトの静的マニフェスト、ならびに関連するプロトタイプおよび抽象化の提案される、または予期される動的マニフェストを検査することにより、システムインスペクタ１８０は、ソフトウェアベースのコンピュータ１０２のいくつかのプロパティを検証することができる。

【００８１】

例えば、インスペクタは、構成検証（ｃｏｍｐｏｓｉｔｉｏｎａｌ　ｖｅｒｉｆｉｃａｔｉｏｎ）をサポートするプロパティのクラスを検証することができる。つまり、インスペクタは、ソフトウェアベースのコンピュータ１０２のコンポーネントの必要な要素のすべてが、保存されたシステムイメージ上に存在するかどうか、ならびにそれらの要素が、正しく構成されていることを確認することができる。

【００８２】

プロパティは、そのプロパティに関して、コンポーネントを個々に検証することができる場合、構成上、検証可能であり、構成されると、システムは、コンポーネントのすべて

を再検証することなしに、同一のプロパティを保持するものと理解されることが可能である。例えば、プログラミングシステムでは、型の安全性は、個々のロードモジュールが、型安全であると検証されることが可能である場合、構成上、検証可能であると考えられ、正当に組み合わせられると、ロードモジュールは、型の安全性を保持する。そのケースでは、システムインスペクタ１８０は、新たなロードモジュールが追加されるたびに、システム全体にわたる複雑な検証を要することなしに、各ロードが、型安全であることを検証し、次に、ロードモジュールが正当な形で組み合わせられていることを検証することができる。

【００８３】

例示的なシステム検査の方法上のインプリメンテーション

図４は、システムインスペクタ１８０によって実行される方法４００を示す。この方法上のインプリメンテーションは、ソフトウェアで実行されても、ハードウェアで実行されても、ソフトウェアとハードウェアの組み合わせで実行されてもよい。理解を容易にするため、方法は、図４に独立のブロックとして表す別々のステップとして示すが、それらの別々に示したステップは、ステップの実行の際に必ずしも順序に依存していると解釈してはならない。さらに、説明のため、方法４００は、図４を参照して説明する。

【００８４】

図４の４１０で、システムインスペクタ１８０が、コンピュータ１０２のような、ソフトウェアベースのコンピュータのオフライン「システムイメージ」のコピーを獲得する。このアクションは、図１に、大きい矢印で示す。

【００８５】

４１２で、システムインスペクタ１８０は、オフラインシステムイメージの分析を実行して、コンピュータ１０２が、特定の機能コンポーネント（ＯＳまたはアプリケーション群などの）を含むと決定的に検証する。より詳細には、インスペクタは、自己記述アーチファクトを検査して、必要なコンポーネント（自己記述アーチファクトのマニフェストによって、そのようなコンポーネントとして記述され、参照される）のすべてが、見つかり、適切に識別されるかどうかを調べる。

【００８６】

４１４で、インスペクタは、その分析の結果を報告する。

【００８７】

抽象化

オペレーティングシステムは、計算を枠に入れ、プログラマが、専門の分野により完全に集中することにより、ソフトウェアをより容易に作成することができるようにする抽象化を提供する。抽象化は、１つまたは複数のコンポーネントのモデルを表し、このモデルは、それらのコンポーネントを、他のすべての種類のコンポーネントから区別する、それらのコンポーネントの基本的な特性を表し、明確に定義された概念上の境界を提供する。

【００８８】

既存のオペレーティングシステム抽象化の例には、ストレージを制御し、管理するファイルシステム抽象化、Ｉ／Ｏデバイス群を制御するＩ／Ｏ抽象化、グラフィカルユーザインタフェース（ＧＵＩ）抽象化、計算を保持するプロセス抽象化、およびプロセス間の通信を可能にするプロセス間通信（ＩＰＣ）抽象化が含まれる。

【００８９】

以上の基本的な抽象化がなければ、プログラマは、一般的なタスクを実行するために、独自の場当たり的な方法を考案することを余儀なくされる。決まって、そのような多様な場当たり的な方法は、プログラマの生産性を低下させ、広範にわたる作業の繰り返しをもたらし、システムエラーを増加させる。

【００９０】

少なくとも１つのインプリメンテーションでは、例示的な自己記述アーチファクトアーキテクチャは、以下を含む、新たなオペレーティングシステム抽象化を作成する。すなわち、

・オペレーティングシステムおよびプログラム群を含む、「実行可能な」（例えば、コンピュータ１０２上で実行可能な）ソフトウェアシステムを表すシステムプロトタイプ、
・オペレーティングシステムおよびプログラム群を含む、アクティブな、または「実行中の」システムを表すシステム抽象化、
・実行可能なアプリケーションプログラムを表すアプリケーションプロトタイプ、
・アクティブな、または「実行中の」プログラムのインスタンスを表すアプリケーション抽象化、および
・実行可能なプロセスを表すプロセスプロトタイプである。

【００９１】

図５は、この新たなアーキテクチャを実施することができるソフトウェアベースのコンピュータの例示的な構造５００を示す。この構造５００は、従来の抽象化およびプロトタイプのコンテキストにおいて、この新たなアーキテクチャによって導入される新たな抽象化およびプロトタイプを含む。図５の上から下まで、この例示的な構造５００内の４つの層は、以下のとおりである。すなわち、

【００９２】

１．以下を含むシステム層５１０、すなわち、
・システム抽象化５１２（オペレーティングシステムコンポーネント（例えば、スケジューラ、ＩＰＣマネージャ、Ｉ／Ｏマネージャ、セキュリティマネージャ、ガベージコレクション（ｇａｒｂａｇｅ　ｃｏｌｌｅｃｔｉｏｎ）、およびメモリマネージャなど）、および他の任意のシステムレベルのコンポーネント）、ならびに、包含により、すべてのアプリケーション、
・少なくとも１つのシステムに関連するシステムプロトタイプ５１４、
・ＯＳおよびアプリケーション群を含む、少なくとも１つのシステムに関連するシステムマニフェスト５１６、

【００９３】

２．以下を含むアプリケーション層５２０、すなわち、
・１つまたは複数のアプリケーション抽象化５２２、
・アプリケーション抽象化の少なくとも１つに関連するアプリケーションプロトタイプ５２４、
・少なくとも１つのアプリケーションベースのアーチファクトに関連するアプリケーションマニフェスト５２６、

【００９４】

３．以下を含むプロセス層５３０、すなわち、
・１つまたは複数のプロセス抽象化５３２、
・プロセス抽象化の少なくとも１つに関連するプロセスプロトタイプ５３４、
・少なくとも１つのプロセスベースのアーチファクトに関連するプロセスマニフェスト５３６、および

【００９５】

４．少なくとも１つのロードソースアーチファクト（例えば、ロードモジュール）に関連するロードソースマニフェスト５４６を含む、ロードソース層５４０である。

【００９６】

完全にするため、システムモデルは、実行中のソフトウェアシステム全体を表すシステム抽象化も含むことが可能である。実際には、オペレーティングシステム５１２自体、通常、システム抽象化として作用する。

【００９７】

マニフェストは、ソフトウェアコンポーネント（例えば、プロセス要素、アプリケーション要素、またはＯＳ要素）の宣言型記述である。マニフェストとソフトウェアコンポーネントが、互いに関連付けられて保存されている場合、コンポーネントは、自己記述アーチファクトである。

【００９８】

マニフェストは、プロトタイプを作成するのに使用される。マニフェスト、および記述されるアーチファクトをシステムにインストールする動作により、プロトタイプが作成される。プロトタイプは、ソフトウェアコンポーネントの「実行可能な（ｒｕｎａｂｌｅ）」、または「実行可能な（ｅｘｅｃｕｔａｂｌｅ）」表現である。プロトタイプは、ソフトウェアコンポーネントのインスタンス、または抽象化を作成するのに使用される。インスタンスまたは抽象化は、プロトタイプの「実行中の（ｒｕｎｎｉｎｇ）」または「実行中の「ｅｘｅｃｕｔｉｎｇ」」表現である。

【００９９】

例示的な構造５００では、右から左への矢印は、「によって記述される」と読まれる。したがって、例えば、最上層５１０は、次のように読むことができる。すなわち、ＯＳ抽象化５１２が、システムプロトタイプ５１４「によって記述され」、プロトタイプ５１４は、「システムマニフェスト」５１６「によって記述される」。

【０１００】

この構造では、左から右への矢印は、「作成するのに使用される」と読まれる。したがって、例えば、第２の層５２０は、次のように読むことができる。すなわち、アプリケーションマニフェスト５２６が、アプリケーションプロトタイプ５２４を「作成するのに使用され」、プロトタイプ５２４は、アプリケーション抽象化５１６を「作成するのに使用される」。

【０１０１】

同様に、各層のメンバは、他の層におけるメンバ間の定義された関係を有する。順々に高い層の各メンバは、より低い層の同様のメンバを参照するか、または含む。

【０１０２】

例示的な構造５００では、上から下への矢印は、「含む」または「監督する（ｓｕｐｅｒｖｉｓｅ）」と読まれる。したがって、例えば、システムマニフェスト５１６は、アプリケーションマニフェスト５２６を「含み」、マニフェスト５２６は、プロセスマニフェスト５３６を「含み」、マニフェスト５３６は、ロードソースマニフェスト５４６を「含む」。

【０１０３】

この構造では、下から上への矢印は、「の中に含まれる」または「によって監督される」と読まれる。したがって、例えば、プロセスプロトタイプ５３４は、アプリケーションプロトタイプ５２４「によって監督され」、プロトタイプ５２４は、システムプロトタイプ５１４によって「監督される」。

【０１０４】

代替のインプリメンテーションでは、例示的な自己記述アーチファクトアーキテクチャは、異なる数の層、異なる層構成、および／または異なる抽象化を有する。

【０１０５】

システムマニフェスト５１６は、システム全体（すなわち、アーチファクトのすべて）を記述する。マニフェスト５１６は、各オペレーティングシステムコンポーネントおよび各アプリケーションに関するマニフェストをポイントする最上レベルのマニフェストである。範囲に依存して、個々のオペレーティングシステムコンポーネントは、アプリケーションマニフェスト、プロセスマニフェスト、またはロードソースマニフェストで記述される。

【０１０６】

アプリケーションマニフェスト５２６は、プロセスマニフェスト５３６を含む。このため、アプリケーションマニフェストは、アプリケーション（アプリケーション抽象化５２２によって表される）が実行された際に作成されるプロセスを記述する、もしくは指定する。また、アプリケーションマニフェストは、アプリケーションによって公開される、または要求されるプロセス間通信インタフェースを明らかにし、アプリケーション内のプロセスのプロセス間通信インタフェース間のバインドを記述することもできる。

【０１０７】

プロセスマニフェスト５２６は、プロセス（プロセス抽象化５３２によって表される）の中に含まれるロードモジュールを記述する、または指定するロードモジュールマニフェストを含む。プロセスマニフェストは、各プロセスによって公開される、または要求されるプロセス間通信インタフェースを明らかにし、ロードモジュール上のコードとデータインタフェース群の間のバインドを記述することができる。

【０１０８】

ロードソースマニフェスト５３６は、ロードモジュールの実行可能なコードを含む、保存されたバイナリファイルを記述し、もしくは指定し、そのロードモジュールによって要求される、さらなるロードモジュールを明らかにする。ロードソースマニフェストは、ロードモジュールによって公開される、または要求されるコード、およびデータインタフェース群を明らかにする。

【０１０９】

ある実施形態は、実行中のプロセスに関するマニフェスト、およびプロセスプロトタイプに関するマニフェスト、実行中のアプリケーションに関するマニフェスト、およびアプリケーションプロトタイプに関するマニフェスト、実行中のオペレーティングシステムコンポーネントに関するマニフェスト、およびそれらのコンポーネントのプロトタイプに関するマニフェスト、ハードウェアデバイスに関するマニフェスト、およびシステム全体に関する１つまたは複数のマニフェストを含め、いくつかのタイプのマニフェストをサポートすることができる。そのような実施形態では、相異なるマニフェストが、同一の構造上の要素を共有し、再使用することが可能である。

【０１１０】

アプリケーション抽象化

単にユーザ中心モデルの一部であるのではなく、アプリケーションプログラムの概念は、実際に、以上の例示的な自己記述アーチファクトアーキテクチャ１００の一部である。詳細には、ＯＳ１１２（またはＯＳ１１２の諸部分）が、「アプリケーション抽象化」の概念を本来的に認識する。関連するマニフェストによって記述されるとおり、アプリケーション抽象化は、特定の低レベルの抽象化（アクティブなプロセス、およびそれらのプロセスのロードソースなどの）、および特定の高レベルの抽象化（オペレーティングシステムなどの）に記述的、かつ必然的にリンクされる。

【０１１１】

ユーザが、直接にか、または間接的に、プログラムを実行すると、ＯＳ１１２が、アプリケーションプロトタイプ（アプリケーションプロトタイプ５２４のような）から、アプリケーション抽象化（アプリケーション抽象化５２２のような）のインスタンスを作成する。アプリケーションのインスタンスを作成することは、プロセスプロトタイプによって記述されるプロセスの新たなインスタンスを作成することを含む。

【０１１２】

アプリケーションの静的記述は、１つまたは複数のアプリケーションマニフェスト（アプリケーションマニフェスト５２６のような）として実現される。ＯＳ１１２は、プロセスをアプリケーションにリンクし、プロセスをプロセスプロトタイプにリンクし、アプリケーションをアプリケーションプロトタイプにリンクする「動的」メタデータを保持する。また、ＯＳ１１２は、プロセスプロトタイプおよびアプリケーションプロトタイプをそれぞれのマニフェストにリンクする、さらなる動的メタデータも保持する。

【０１１３】

アプリケーション抽象化は、動的オブジェクトとして実現される。他のソフトウェアコンポーネント（ＯＳの一部などの）は、動的アプリケーション抽象化オブジェクトと通信して、いずれのアプリケーションが実行されているかを特定し、いずれのプロセスが、アプリケーションに属するかを特定し、やはり入手可能な、マニフェストなどの、他のメタデータを取得することができる。例えば、プロセスのＩＤを所与として、プログラムは、そのプロセスが属するアプリケーションのＩＤをＯＳに求めることができ、アプリケーションのＩＤを所与として、プログラムは、そのアプリケーションのアプリケーションプロ

トタイプのＩＤをオペレーティングシステムに求めることができるといった具合である。
【０１１４】
例示的なアプリケーション抽象化管理の方法上のインプリメンテーション
図６は、アプリケーション抽象化を作成し、管理する目的で、ＯＳ１１２によって実行される方法６００、または方法６００の諸部分を示す。この方法上のインプリメンテーションは、ソフトウェアで実行されても、ハードウェアで実行されても、ソフトウェアとハードウェアの組み合わせで実行されてもよい。理解を容易にするため、方法は、図６に独立のブロックとして表す別々のステップとして示すが、それらの別々に示したステップは、ステップの実行の際に必ずしも順序に依存していると解釈してはならない。さらに、説明のため、方法６００は、図６を参照して説明する。
【０１１５】
図６の６１０で、ＯＳ１１２は、トリガイベントを認識する。トリガイベントの例には（例として、限定としてではなく）、ユーザから手動プログラム呼び出し要求を受け取ること、別のプログラムから呼び出し要求を受け取ること、およびプログラム呼び出しのためのスケジュールされた時刻のイベントが含まれる。そのトリガイベントは、通常、呼び出されるべきアプリケーションを明らかにする。
【０１１６】
６１２で、ＯＳ１１２は、アプリケーションプロトタイプによって記述される、明らかにされたアプリケーションのインスタンスを作成する。このインスタンスが、「アプリケーション抽象化」と呼ばれる。図５の例示的な構造５００で示すとおり、アプリケーション抽象化を作成することは、プロセスプロトタイプによって記述される、関連するプロセスの新たなインスタンスを作成することを含む。
【０１１７】
６１４で、ＯＳは、関連するプロセスを、明らかにされたアプリケーション抽象化にリンクし、プロセスをプロセスプロトタイプにリンクし、アプリケーション群をアプリケーションプロトタイプ群にリンクする「動的」メタデータを保持する。
【０１１８】
６１６で、ＯＳは、他のソフトウェアコンポーネントによるアプリケーション抽象化の識別、およびアプリケーション抽象化との通信を円滑にする。
【０１１９】
例示的なコンピューティングシステムおよびコンピューティング環境
図７は、本明細書で説明する、例示的な自己記述アーチファクトアーキテクチャを実施することができる（完全にか、または部分的に）、適切なコンピューティング環境７００の実施例を示す。コンピューティング環境７００は、本明細書で説明するコンピュータアーキテクチャおよびネットワークアーキテクチャにおいて利用することができる。
【０１２０】
典型的なコンピューティング環境７００は、コンピューティング環境の一実施例に過ぎず、コンピュータアーキテクチャおよびネットワークアーキテクチャの用法または機能の範囲について、全く限定を示唆することは意図していない。また、コンピューティング環境７００が、例示的なコンピューティング環境７００に示したコンポーネントのいずれの１つ、または組み合わせに関連する依存関係または要件も有すると解釈してはならない。
【０１２１】
例示的な自己記述アーチファクトアーキテクチャは、他の多数の汎用または専用のコンピューティングシステム環境またはコンピューティングシステム構成で実施することができる。使用に適する可能性がある、周知のコンピューティングシステム、コンピューティング環境、および／またはコンピューティング構成の例には、パーソナルコンピュータ、サーバコンピュータ、シンクライアント、シッククライアント、ハンドヘルドデバイスまたはラップトップデバイス、マルチプロセッサシステム、マイクロプロセッサベースのシステム、セットトップボックス、パーソナルデジタルアシスタント（ＰＤＡ）、機器、専用エレクトロニクス（例えば、ＤＶＤプレーヤ）、プログラマブル家庭用電化製品、ネッ

トワークＰＣ、ミニコンピュータ、メインフレームコンピュータ、以上のシステムまたはデバイスのいずれかを含む分散コンピューティング環境などが含まれるが、以上には限定されない。

【０１２２】

例示的な自己記述アーチファクトアーキテクチャは、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、プロセッサ実行可能命令の一般的な状況で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、特定のタスクを実行する、または特定の抽象データ型を実装する、ルーチン、プログラム、オブジェクト、コンポーネント、データ構造などが含まれる。また、例示的な自己記述アーチファクトアーキテクチャは、通信ネットワークを介してリンクされたリモート処理装置によってタスクが実行される、分散コンピューティング環境で実施することもできる。分散コンピューティング環境では、プログラムモジュールは、メモリ記憶装置を含む、ローカルコンピュータ記憶媒体とリモートコンピュータ記憶媒体の両方の中に配置することができる。

【０１２３】

コンピューティング環境７００は、コンピュータ７０２の形態で汎用コンピューティングデバイスを含む。コンピュータ７０２のコンポーネントには、１つまたは複数のプロセッサ７０４、システムメモリ７０６、ならびにプロセッサ７０４からシステムメモリ７０６までを含む様々なシステムコンポーネントを結合するシステムバス７０８が含まれることが可能であるが、以上には限定されない。

【０１２４】

システムバス７０８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、アクセラレーテッドグラフィックスポート（ａｃｃｅｌｅｒａｔｅｄ　ｇｒａｐｈｉｃｓ　ｐｏｒｔ）、およびプロセッサバスまたはローカルバスを含め、いくつかのタイプのバス構造のいずれかの１つまたは複数を表す。例として、そのようなアーキテクチャには、ＣａｒｄＢｕｓ、パーソナルコンピュータメモリカードインタナショナルアソシエーション（ＰＣＭＣＩＡ）、アクセラレーテッドグラフィックスポート（Ａｃｃｅｌｅｒａｔｅｄ　Ｇｒａｐｈｉｃｓ　Ｐｏｒｔ）（ＡＧＰ）、スモールコンピュータシステムインタフェース（ＳＣＳＩ）、ユニバーサルシリアルバス（ＵＳＢ）、ＩＥＥＥ１３９４、ビデオエレクトロニクススタンダーズアソシエーション（Ｖｉｄｅｏ　Ｅｌｅｃｔｒｏｎｉｃｓ　Ｓｔａｎｄａｒｄｓ　Ａｓｓｏｃｉａｔｉｏｎ）（ＶＥＳＡ）ローカルバス、およびメザニン（Ｍｅｚｚａｎｉｎｅ）バスとしても知られる、ペリフェラルコンポーネントインタコネクツ（Ｐｅｒｉｐｈｅｒａｌ　Ｃｏｍｐｏｎｅｎｔ　Ｉｎｔｅｒｃｏｎｎｅｃｔｓ）（ＰＣＩ）バスが含まれることが可能である。

【０１２５】

コンピュータ７０２は、通常、様々なプロセッサ可読媒体を含む。そのような媒体は、コンピュータ７０２がアクセスすることができる任意の利用可能な媒体であることが可能であり、揮発性媒体と不揮発性媒体、リムーバブルな媒体とリムーバブルでない媒体がともに含まれる。

【０１２６】

システムメモリ７０６は、ランダムアクセスメモリ（ＲＡＭ）７１０のような、揮発性メモリ、および／または読み取り専用メモリ（ＲＯＭ）７１２のような不揮発性メモリの形態のプロセッサ可読媒体を含む。起動中などに、コンピュータ７０２内部の要素間で情報を転送するのを助ける基本ルーチンを含む基本入出力システム（ＢＩＯＳ）７１４が、ＲＯＭ７１２の中に格納される。ＲＡＭ７１０は、通常、プロセッサ７０４が即時にアクセスすることができ、かつ／または、現在、処理しているデータおよび／またはプログラムモジュールを含む。

【０１２７】

また、コンピュータ７０２は、他のリムーバブルな／リムーバブルでない、揮発性／不揮発性のコンピュータ記憶媒体も含むことが可能である。例として、図７は、リムーバブルでない不揮発性の磁気媒体（図示せず）に対して読み取りおよび書き込みを行うための

ハードディスクドライブ７１６、リムーバブルな不揮発性の磁気ディスク７２０（例えば、「フロッピー（登録商標）ディスク」）に対して読み取りおよび書き込みを行うための磁気ディスクドライブ７１８、およびＣＤ－ＲＯＭ、ＤＶＤ－ＲＯＭ、または他の光媒体などの、リムーバブルな不揮発性の光ディスク７２４に対して読み取りおよび／または書き込みを行うための光ディスクドライブ７２２を示す。ハードディスクドライブ７１６、磁気ディスクドライブ７１８、および光ディスクドライブ７２２はそれぞれ、１つまたは複数のデータ媒体インタフェース７２５でシステムバス７０８に接続される。代替として、ハードディスクドライブ７１６、磁気ディスクドライブ７１８、および光ディスクドライブ７２２は、１つまたは複数のインタフェース（図示せず）でシステムバス７０８に接続してもよい。

【０１２８】

ディスクドライブ群、および関連するプロセッサ可読媒体により、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、およびその他のデータの不揮発性ストレージが、コンピュータ７０２に提供される。実施例は、ハードディスク７１６、リムーバブルな磁気ディスク７２０、およびリムーバブルな光ディスク７２４を示しているが、磁気カセットまたは他の磁気記憶装置、フラッシュメモリカード、ＣＤ－ＲＯＭ、デジタルバーサタイルディスク（ＤＶＤ）または他の光ストレージ、ランダムアクセスメモリ（ＲＡＭ）、読み取り専用メモリ（ＲＯＭ）、電気的に消去可能なプログラマブルな読み取り専用メモリ（ＥＥＰＲＯＭ）などの、コンピュータがアクセスできるデータを格納することが可能な、他のタイプのプロセッサ可読媒体を利用して、例示的なコンピューティングシステムおよびコンピューティング環境を実施することも可能であることを認識されたい。

【０１２９】

例として、オペレーティングシステム７２６、１つまたは複数のアプリケーションプログラム７２８、その他のプログラムモジュール群７３０、およびプログラムデータ７３２を含め、任意の数のプログラムモジュールを、ハードディスク７１６、磁気ディスク７２０、光ディスク７２４、ＲＯＭ７１２、および／またはＲＡＭ７１０に格納することができる。

【０１３０】

ユーザは、キーボード７３４やポインティングデバイス７３６（例えば、「マウス」）などの入力デバイス群を介して、コマンドおよび情報をコンピュータ７０２に入力することができる。他の入力デバイス群７３８（特に図示せず）には、マイク、ジョイスティック、ゲームパッド、サテライトディッシュ、シリアルポート、スキャナ、および／または類似物が含まれることが可能である。以上、およびその他の入力デバイスは、システムバス７０８に結合された入出力インタフェース群７４０を介してプロセッサ７０４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、他のインタフェースおよびバス構造で接続してもよい。

【０１３１】

また、モニタ７４２、または他のタイプのディスプレイデバイスも、ビデオアダプタ７４４のようなインタフェースを介して、システムバス７０８に接続されることが可能である。モニタ７４２に加え、他の周辺出力デバイス群には、入出力インタフェース群７４０を介してコンピュータ７０２に接続することができる、スピーカ（図示せず）やプリンタ７４６などのコンポーネントが含まれることが可能である。

【０１３２】

コンピュータ７０２は、リモートコンピューティングデバイス７４８などの、１つまたは複数のリモートコンピュータに対する論理接続を使用する、ネットワーク化された環境において動作することもできる。例として、リモートコンピューティングデバイス７４８は、パーソナルコンピュータ、ポータブルコンピュータ、サーバ、ルータ、ネットワークコンピュータ、ピアデバイス、または他の一般的なネットワークノードなどであることが可能である。リモートコンピューティングデバイス７４８は、コンピュータ７０２に関して、本明細書で説明した諸要素および諸特徴の多く、またはすべてを含むことが可能なポ

ータブルコンピュータとして図示する。

【０１３３】

コンピュータ７０２とリモートコンピュータ７４８の間の論理接続は、ローカルエリアネットワーク（ＬＡＮ）７５０、および一般的なワイドエリアネットワーク（ＷＡＮ）７５２として図示する。そ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網、イントラネット、およびインターネットで一般的である。そのようなネットワーキング環境は、有線であることも、無線であることも可能である。

【０１３４】

ＬＡＮネットワーキング環境で実施される場合、コンピュータ７０２は、ネットワークインタフェースまたはネットワークアダプタ７５４を介して、ローカルネットワーク７５０に接続される。ＷＡＮネットワーキング環境で実施される場合、コンピュータ７０２は、通常、ワイドネットワーク７５２を介して通信を確立するためのモデム７５６、またはその他の手段を含む。コンピュータ７０２の内部にあることも、外部にあることも可能なモデム７５６は、入出力インタフェース群７４０、または他の適切な機構を介してシステムバス７０８に接続することができる。図示したネットワーク接続は、例示的であり、コンピュータ７０２とコンピュータ７４８の間で通信リンクを確立する他の手段を使用してもよいことを認識されたい。

【０１３５】

コンピューティング環境７００で示したような、ネットワーク化された環境では、コンピュータ７０２に関連して示したプログラムモジュール群、またはプログラムモジュール群の諸部分は、リモートメモリ記憶装置の中に格納することができる。例として、リモートアプリケーションプログラム群７５８が、リモートコンピュータ７４８のメモリデバイス上に存在する。例示のため、アプリケーションプログラム群、ならびにオペレーティングシステムなどの、他の実行可能なプログラムコンポーネント群を本明細書では、別々のブロックとして示すが、そのようなプログラム群およびコンポーネント群は、様々な時点で、コンピューティングデバイス７０２の異なるストレージコンポーネントの中に存在し、コンピュータのデータプロセッサによって実行されることを認識されたい。

【０１３６】

プロセッサ実行可能命令

例示的な自己記述アーチファクトアーキテクチャのインプリメンテーションは、１つまたは複数のコンピュータまたは他のデバイスによって実行される、プログラムモジュールなどの、プロセッサ実行可能命令の一般的な状況で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、特定のタスクを実行する、または特定の抽象データ型を実装する、ルーチン、プログラム、オブジェクト、コンポーネント、データ構造などが含まれる。通常、プログラムモジュール群の機能は、様々な実施形態において、所望に応じて組み合わせても、分散させてもよい。

【０１３７】

例示的な動作環境

図７は、例示的な自己記述アーチファクトアーキテクチャを実施することができる適切な動作環境７００の実施例を示す。具体的には、本明細書で説明する例示的な自己記述アーチファクトアーキテクチャは、図７の任意のプログラムモジュール群７２８～７３０および／またはオペレーティングシステム７２６またはその一部によって実施することができる（全体として、または部分的に）。

【０１３８】

この動作環境は、適切な動作環境の実施例に過ぎず、本明細書で説明する例示的な自己記述アーチファクトアーキテクチャの機能の範囲または用法について、全く限定を示唆することは意図していない。使用するのに適した、他の周知のコンピューティングシステム、コンピューティング環境、および／またはコンピューティング構成には、パーソナルコンピュータ（ＰＣ）、サーバコンピュータ、ハンドヘルドデバイスまたはラップトップデバイス、マルチプロセッサシステム、マイクロプロセッサベースのシステム、プログラマ

ブル家庭用電化製品、無線電話および無線機器、汎用機器および専用機器、特定用途向け集積回路（ＡＳＩＣ）、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、以上のシステムまたはデバイスのいずれかを含む分散コンピューティング環境などが含まれるが、以上には限定されない。

【０１３９】

プロセッサ可読媒体

例示的な自己記述アーチファクトアーキテクチャのインプリメンテーションは、何らかの形態のプロセッサ可読媒体上に格納すること、またはそのような媒体を介して伝送することができる。プロセッサ可読媒体は、コンピュータがアクセスすることができる任意の利用可能な媒体であることが可能である。例として、プロセッサ可読媒体は、「コンピュータ記憶媒体」および「通信媒体」を含むことが可能であるが、以上には限定されない。

【０１４０】

「コンピュータ記憶媒体」には、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、またはその他のデータなどの情報を格納するために任意の方法または技術で実装された、揮発性媒体および不揮発性媒体、リムーバブルな媒体およびリムーバブルでない媒体が含まれる。コンピュータ記憶媒体には、ＲＡＭ、ＲＯＭ、ＥＥＰＲＯＭ、フラッシュメモリまたは他のメモリ技術、ＣＤ－ＲＯＭ、デジタルバーサタイルディスク（ＤＶＤ）または他の光ストレージ、磁気カセット、磁気テープ、磁気ディスクストレージまたは他の磁気記憶装置、あるいは所望の情報を格納するのに使用することができ、コンピュータがアクセスすることができる他の任意の媒体が含まれるが、以上には限定されない。

【０１４１】

「通信媒体」は、通常、搬送波などの変調されたデータ信号、または他のトランスポート機構で、プロセッサ可読命令、データ構造、プログラムモジュール、またはその他のデータを実現する。また、通信媒体には、あらゆる情報配信媒体も含まれる。

【０１４２】

結論

本明細書で説明した諸技術は、プログラムモジュール、汎用コンピューティングシステムおよび専用コンピューティングシステム、ネットワークサーバおよびネットワーク機器、専用エレクトロニクスおよび専用ハードウェアを含む（ただし、以上には限定されない）、多くの形で、１つまたは複数のコンピュータネットワークの一部として実施することができる。本明細書で説明した諸技術は、例えば、図７に示したコンピュータシステム上で実施することができる。より詳細には、それらの技術は、例えば、図７に示したコンピュータシステム上のオペレーティングシステムによって実施されることが可能である。

【０１４３】

以上に説明した１つまたは複数のインプリメンテーションを、構造上の特徴および／または方法上のステップに固有の言い回しで説明してきたが、他のインプリメンテーションは、説明した特定の特徴またはステップなしに実施してもよいことを理解されたい。むしろ、特定の特徴およびステップは、１つまたは複数のインプリメンテーションの好ましい形態として開示している。

【図面の簡単な説明】

【０１４４】

【図１】本明細書で説明するインプリメンテーションに関する例示的な動作シナリオを示す図である。

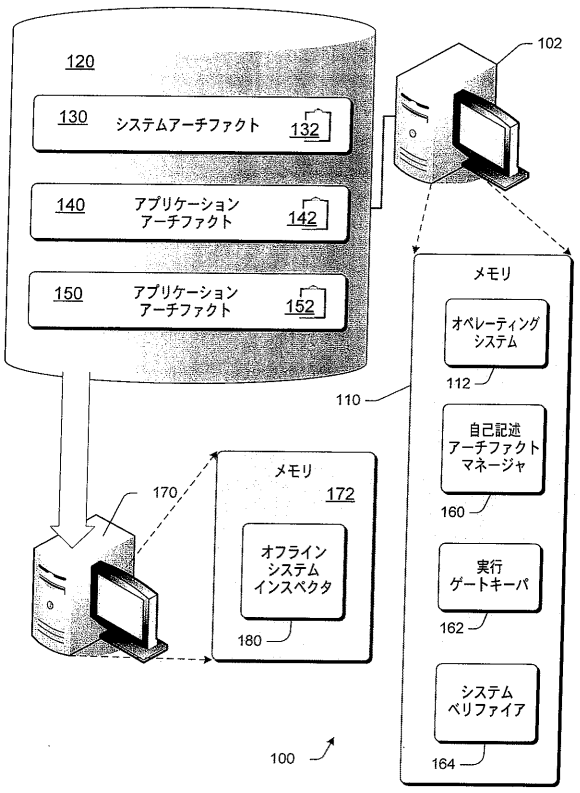
【図２】保存された（ｐｅｒｓｉｓｔｅｄ）自己記述アーチファクトの管理のため、ならびに少なくとも部分的に自己記述アーチファクトから成るソフトウェアコンポーネントの実行に関するゲートキーピング（ｇａｔｅｋｅｅｐｉｎｇ）を実行するための、本明細書で説明する、１つまたは複数の方法上のインプリメンテーションを示す流れ図である。

【図３】保存された自己記述アーチファクトを検証する本明細書で説明する方法上のインプリメンテーションを示す流れ図である。

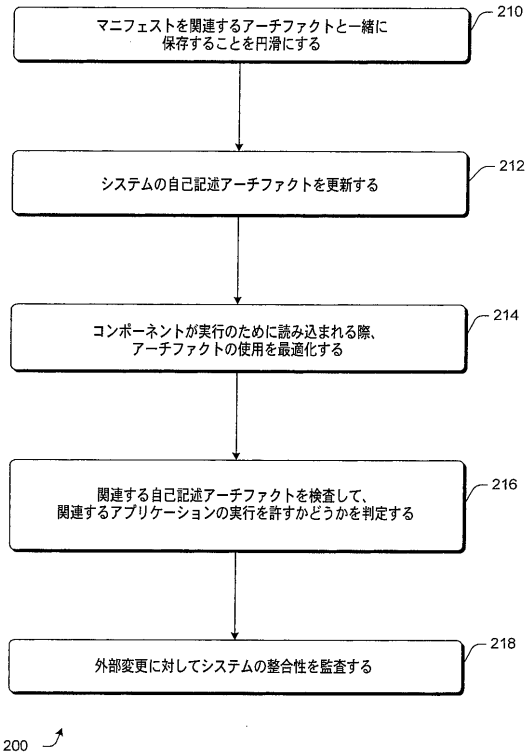
【図４】少なくとも部分的に、保存された自己記述アーチファクトから成るオフライン「

システムイメージ」を検査する本明細書で説明する方法上のインプリメンテーションを示す流れ図である。
【図５】ソフトウェアコンポーネント（例えば、ロードモジュール、プロセス、アプリケーション、およびオペレーティングシステムコンポーネント）の間の、本明細書で説明するインプリメンテーションによる例示的な相互関係構造を示す図である。
【図６】アプリケーション抽象化を作成し、管理する、本明細書で説明する方法上のインプリメンテーションを示す流れ図である。
【図７】本明細書で説明する少なくとも１つの実施形態を実施することができる（完全に、または部分的に）コンピュータ動作環境の実施例を示す図である。
【符号の説明】
【０１４５】
１１０、１７２　メモリ
１１２　オペレーティングシステム
１３０　システムアーチファクト
１４０　アプリケーションアーチファクト
１５０　アプリケーションアーチファクト
１６０　自己記述アーチファクトマネージャ
１６２　実行ゲートキーパ
１６４　システムベリファイア
１８０　オフラインシステムインスペクタ

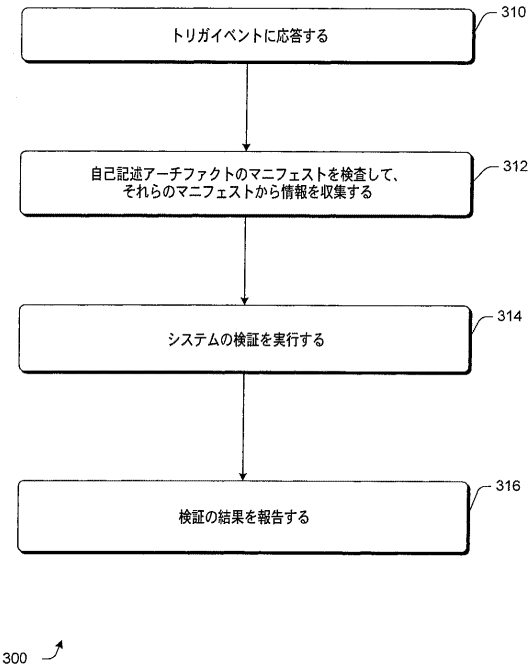
【図１】



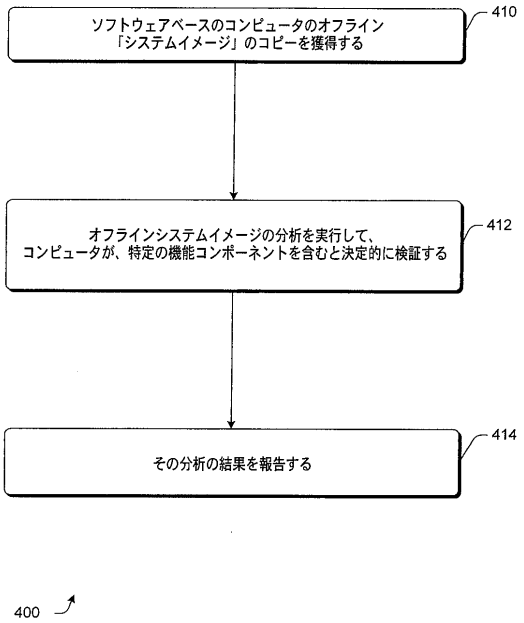
【図２】



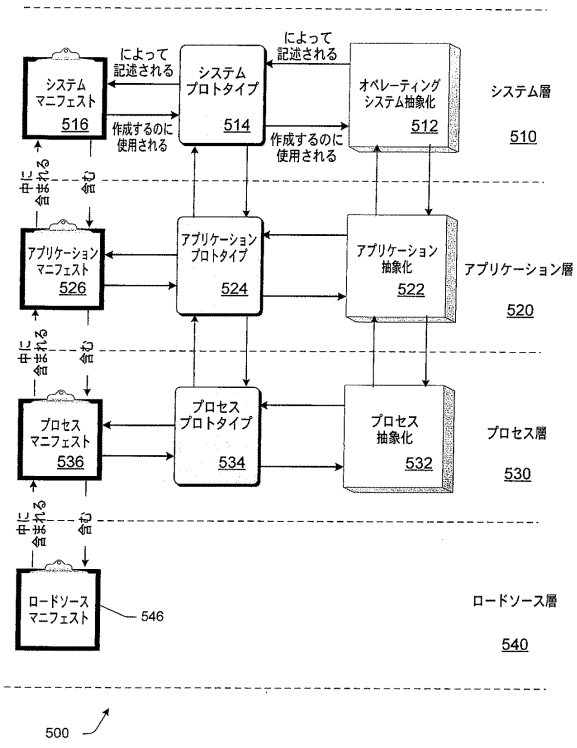
【図３】


トリガイベントに応答する
310
自己記述アーチファクトのマニフェストを検査して、
それらのマニフェストから情報を収集する
312
システムの検証を実行する
314
検証の結果を報告する
316
300

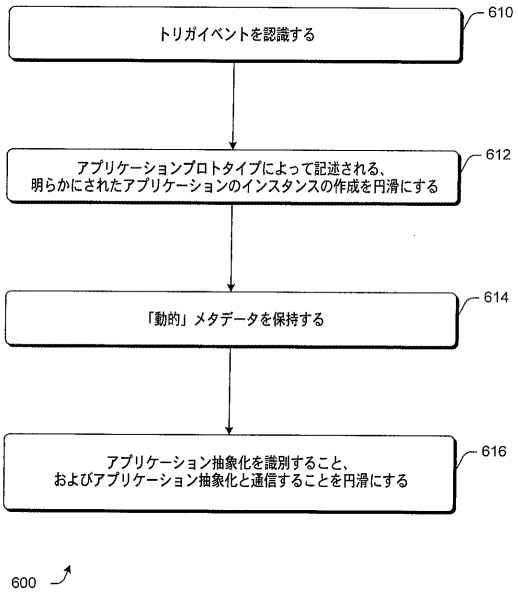
【図４】


ソフトウェアベースのコンピュータのオフライン
「システムイメージ」のコピーを獲得する
410
オフラインシステムイメージの分析を実行して、
コンピュータが、特定の機能コンポーネントを含むと決定的に検証する
412
その分析の結果を報告する
414
400

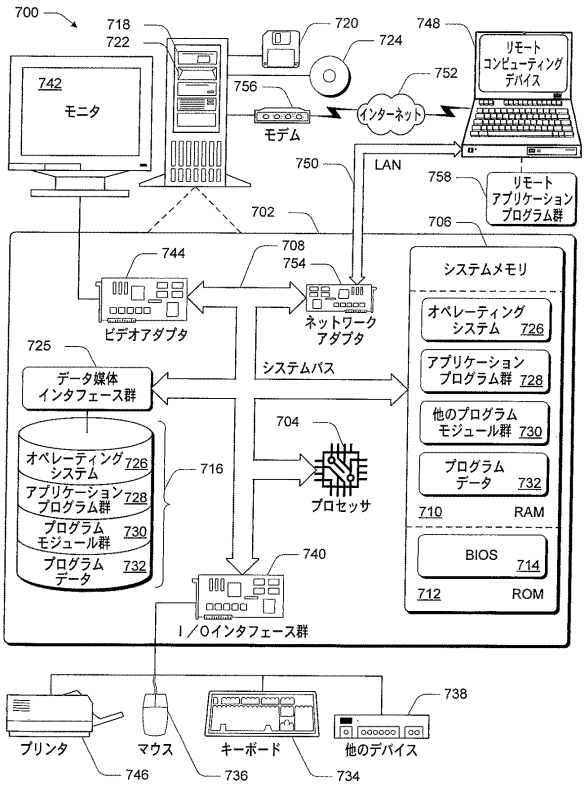
【図５】


システム
マニフェスト
516
によって
記述される
作成するのに
使用される
システム
プロトタイプ
514
によって
記述される
作成するのに
使用される
オペレーティング
システム抽象化
512
システム層
510
アプリケーション
マニフェスト
526
アプリケーション
プロトタイプ
524
アプリケーション
抽象化
522
アプリケーション層
520
プロセス
マニフェスト
536
プロセス
プロトタイプ
534
プロセス
抽象化
532
プロセス層
530
ロードソース
マニフェスト
546
ロードソース層
540
500

【図６】


トリガイベントを認識する
610
アプリケーションプロトタイプによって記述される、
明らかにされたアプリケーションのインスタンスの作成を円滑にする
612
「動的」メタデータを保持する
614
アプリケーション抽象化を識別すること、
およびアプリケーション抽象化と通信することを円滑にする
616
600

【図７】


700
718
722
720
724
748
リモート
コンピューティング
デバイス
742
モニタ
756
モデム
752
インターネット
750
LAN
758
リモート
アプリケーション
プログラム群
702
706
システムメモリ
744
708
754
ビデオアダプタ
ネットワーク
アダプタ
オペレーティング
システム 726
アプリケーション
プログラム群 728
他のプログラム
モジュール群 730
プログラム
データ 732
710 RAM
BIOS 714
712 ROM
725
データ媒体
インタフェース群
システムバス
704
プロセッサ
716
オペレーティング
システム 726
アプリケーション
プログラム群 728
プログラム
モジュール群 730
プログラム
データ 732
740
I／Oインタフェース群
738
プリンタ
マウス
キーボード
他のデバイス
746
736
734

フロントページの続き

(72)発明者 ガレン シー. ハント
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
(72)発明者 ジェームズ アール. ラルス
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション内
(72)発明者 ジョン ディー. デトレビル
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 マニュエル ファーンドリッチ
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 スティーブン ピー. レビ
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション内
(72)発明者 トーマス ローダー
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
(72)発明者 ヴォルフガング グリースカンプ
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 後藤 彰

(56)参考文献 特開２００３－２３３５２１（ＪＰ，Ａ）
特表２００４－５１３４１２（ＪＰ，Ａ）
特表２００２－５０６２４７（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ ２１／２２