(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4426797号
(P4426797)
(45)発行日　平成22年3月3日(2010.3.3)　(24)登録日　平成21年12月18日(2009.12.18)

(51)Int.Cl.　F I
G06F 9/44 (2006.01)　G06F 9/06 620K

請求項の数 8 (全 37 頁)

(21)出願番号　特願2003-305288(P2003-305288)
(22)出願日　平成15年8月28日(2003.8.28)
(65)公開番号　特開2004-103014(P2004-103014A)
(43)公開日　平成16年4月2日(2004.4.2)
審査請求日　平成15年8月28日(2003.8.28)
審判番号　不服2006-26740(P2006-26740/J1)
審判請求日　平成18年11月27日(2006.11.27)
(31)優先権主張番号　10/241397
(32)優先日　平成14年9月11日(2002.9.11)
(33)優先権主張国　米国(US)

(73)特許権者　390009531
インターナショナル・ビジネス・マシーンズ・コーポレーション
ＩＮＴＥＲＮＡＴＩＯＮＡＬ　ＢＵＳＩＮＥＳＳ　ＭＡＳＣＨＩＮＥＳ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国１０５０４　ニューヨーク州　アーモンク　ニュー　オーチャード　ロード
(74)代理人　100086243
弁理士　坂口　博
(74)代理人　100091568
弁理士　市位　嘉宏
(74)代理人　100108501
弁理士　上野　剛史

最終頁に続く

(54)【発明の名称】依存性に基づく影響シミュレーションおよび脆弱性分析のための方法および装置

(57)【特許請求の範囲】
【請求項１】
コンピューティング環境に関連付けられた少なくとも１つの対象コンポーネントのシミュレートされたコンポーネント故障によって潜在的に影響を受けるエンドユーザのリスト、コンポーネントのリストおよびホストのリストのうち少なくとも１つを明らかにするための装置であって、
少なくとも１つのプロセッサであって、前記装置は、前記少なくとも１つのプロセッサを用いて、（ｉ）前記少なくとも１つの対象コンポーネントに依存する１つ以上のコンポーネントを識別すること、ここで、当該コンポーネントそれぞれは、機能モデル、構造モデル、及び動作モデルによって記述されるサービスライフサイクルのいずれかのモデルに属しており、及び当該サービスライフサイクルの間に利用可能な依存性情報が各モデル内に格納されており、並びに前記機能モデルはコンポーネント間の機能アプリケーション依存性についてのモデルであり、前記構造モデルは前記機能モデルにおいて規定されたサービスを実現するソフトウェアコンポーネントであり、及び前記動作モデルは前記ソフトウェアコンポーネントがインスタンス化され、前記ソフトウェアコンポーネント間の実行時の結合が確立することによって生成されたモデルであり、（ｉｉ）前記１つ以上の識別されたコンポーネントの各々に関連付けられたシミュレートしたサービス使用不能の潜在的な影響を検索すること、ここで該検索は、特定のサービスに直接依存するエンティティ、特定のサービスに依存するサービス集合全体、又はエンドユーザサービスからなる最高サービス層のいずれかの検索である、および（ｉｉｉ）前記検索された潜在的な影響に基づ

いて前記１つ以上の識別されたコンポーネントの少なくとも一部に関する情報をユーザのためにディスプレイ上に提示する、前記少なくとも１つのプロセッサと、

前記少なくとも１つのプロセッサに結合されたメモリであって、前記装置は、前記メモリ内に、前記識別および提示動作に関連付けられた結果の少なくとも一部を格納する、前記メモリと

を含む、装置。

【請求項２】

前記コンピューティング環境は分散型コンピューティング環境を含む、請求項１に記載の装置。

【請求項３】

前記コンピューティング環境は自律型コンピューティング環境を含む、請求項１に記載の装置。

【請求項４】

前記ユーザに提示される前記情報は、前記検索された潜在的な影響に基づいて、前記１つ以上の識別されたコンポーネントの少なくとも一部に関連付けられた脆弱性の表現を含む、請求項１に記載の装置。

【請求項５】

前記表現は、前記１つ以上の識別されたコンポーネントの各々に関連付けられた故障の各可能性を示す脆弱性マップを含み、該脆弱性マップは前記故障によって影響を受ける可能性があるサービス、ユーザまたはホストシステムを視覚的に表す、請求項４に記載の装置。

【請求項６】

前記ユーザに提示される前記情報は、前記コンポーネント故障によって潜在的に影響を受けるエンドユーザのリスト、前記コンポーネント故障によって潜在的に影響を受けるコンポーネントのリスト、および前記コンポーネント故障によって潜在的に影響を受けるホストのリストのうち少なくとも１つを含む、請求項１に記載の装置。

【請求項７】

前記少なくとも１つの対象コンポーネントは、前記コンピューティング環境において用いられる実際のコンポーネントである、請求項１に記載の装置。

【請求項８】

前記少なくとも１つの対象コンポーネントは、前記コンピューティング環境において用いることも可能であるコンポーネントを表す仮想コンポーネントである、請求項１に記載の装置。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、分散型コンピューティングシステムに関し、更に特定すれば、かかる分散型コンピューティングシステムの様々なコンポーネント間の依存性に基づいてシミュレートしたサービス使用不能の潜在的な影響を分析し判定するための方法および装置に関する。

【背景技術】

【０００２】

分散型システムのコンポーネント間の依存性の識別および追跡は、統合化された故障管理のためにますます重要になってきている。アプリケーション、サービス、およびそれらのコンポーネントは、様々なサポートサービスに頼っており、これらのサービスはサービス提供業者に外部委託される場合がある。更に、新たに生まれているウエブを用いた（ワールドワイドウエブを用いた）ビジネスアーキテクチャによって、ウエブを用いたｅ－ビジネス（電子ビジネス）アプリケーションを実行時に構成することが可能となっている。

【０００３】

「実行時」という言葉は、一般に、１本のソフトウエアがコンピュータのハードドライブ上の記憶装置に休止して単に存在しているのとは対照的に、１本のソフトウエアが実行

されており、コンピュータシステムのメモリ内でアクティブである時間期間を指すことは理解されよう。このため、実行時にｅ－ビジネスアプリケーションを構成可能であるということは、システム／アプリケーションをダウンさせて再起動する必要なく、更に、アプリケーションを再コンパイルする必要なく、そうする機能を有するということである。従来、コンピュータプログラムのライフサイクルは、プログラムコードを書き、次いでコンパイルし（機械コードに翻訳し）、次いで実行する、というものであった。このため、上述の機能によって、数本のソフトウエアを組み合わせて１つの新しいアプリケーションを「実行中に」形成すること、すなわちアプリケーションをダウン／コンパイル／再起動する必要なく形成することが可能となる。

【０００４】

しかしながら、結果として、あるサービスにおいて生じた故障が、カスタマに提供されている他のサービスに影響を与える。すなわち、サービスは、他のサービスに対する依存性を有する。依存性は、単一のシステム上の異なるサービスのコンポーネント間に存在し、更に、多数のシステムおよびドメインにまたがる１つのサービスのクライアントおよびサーバコンポーネント間にも存在する。ここで、他のサービスに依存するサービスを依存者と呼び、他のサービスに依存されるサービスを先行者と呼ぶ。

【０００５】

多くの場合、１つのサービスが双方の役割を果たすことを注記しておくことは重要である（例えば、名前サービスは、多くのアプリケーションおよびサービスに必要とされるが、それ自体、オペレーティングシステムおよびネットワークプロトコルおよびインフラストラクチャ等、他のサービスの適切な機能に依存する）。更に、依存性の関係は推移的なものである。すなわち、所与のコンポーネントの依存者は、コンポーネント自体に加えて、コンポーネントの先行者（複数の先行者）を必要とする。

【０００６】

依存性は、エンドユーザサービス、システムサービス、アプリケーション、およびそれらの論理的および物理的コンポーネント等、分散型システムの様々なコンポーネント間に存在する。しかしながら、サービス依存性は、今日のシステムにおいて明確にされず、このため、問題の確定、切り離し、および解決が、特に難しくなる。

【０００７】

ソフトウエア開発（米国特許第４，７５１，６３５号および米国特許第５，９６０，１９６号等）、保守（米国特許第５，４９３，６８２号等）、およびソフトウエアパッケージング（米国特許第５，８３５，７７７号等）の分野における既存の技術は、プログラムパッケージの微小部分を形成する個々のソフトウエア要素およびモジュールを扱っており、ソフトウエアを構築してそれをソフトウエアプロダクトに抱き合わせで販売するために、プログラムソースコードの可用性を必要とする。ソースコードは、ソフトウエア開発者に利用可能であるが、サービスユーザには利用できない。本発明は、主に、すでにパッケージ化されているソフトウエアプロダクトに着目する。

【０００８】

電気通信学会規格１３８７．２（「Ｐｏｒｔａｂｌｅ Ｏｐｅｒａｔｉｎｇ Ｓｙｓｔｅｍ Ｉｎｔｅｒｆａｃｅ（ＰＯＳＩＸ）ｓｙｓｔｅｍ ａｄｍｉｎｉｓｔｒａｔｉｏｎ，ｐａｒｔ２：Ｓｏｆｔｗａｒｅ Ａｄｍｉｎｉｓｔｒａｔｉｏｎ」、ＩＥＥＥ、１９９５年）は、ソフトウエアの分配／展開／設置について述べている。ＩＥＥＥ規格が規定する機構は、新しいソフトウエアコンポーネント（インストールされる予定である）が既存のソフトウエアインストールと競合しないことを保証する。ＩＥＥＥ規格は、３種類の関係を識別する。すなわち、前提条件、前条件、同時条件であり、これらによって、かかるコンパチビリティのチェックが容易になる。これは、新しいソフトウエアをインストールする必要がある全てのシステムについて、個別に行われる。ＩＥＥＥ規格では、他のシステム上に存在するソフトウエアのインベントリは考慮されない。更に、ＩＥＥＥ規格は、インスタンス化したアプリケーションおよびサービスを扱っておらず、従って、実行時のコンポーネント間の依存性を求める手段については何も記述していない。

【０００９】
Ｏｐｅｎ Ｇｒｏｕｐ（Ｓｙｓｔｅｍｓ Ｍａｎａｇｅｍｅｎｔ：Ｄｉｓｔｒｉｂｕｔｅｄ Ｓｏｆｔｗａｒｅ Ａｄｍｉｎｉｓｔｒａｔｉｏｎ，ＣＡＥ Ｓｐｅｃｉｆｉｃａｔｉｏｎ Ｃ７０１，Ｔｈｅ Ｏｐｅｎ Ｇｒｏｕｐ，１９９８年1月）は、ＩＥＥＥ１３８７．２を拡張し、特定のシステム上のソフトウエアインストールツールによって呼び出されるいくつかのコマンド（ｓｗｉｎｓｔａｌｌ、ｓｗｌｉｓｔ、ｓｗｍｏｄｉｆｙ等）を規定している。また、Ｏｐｅｎ Ｇｒｏｕｐは、ソフトウエア定義ファイルフォーマットを規定して、上述のコマンドが必要とする情報が、そのコマンドが呼び出されるシステムから利用可能であることを確実とする。ＩＥＥＥ１３８７．２の欠点（すなわち、単一の切り離されたシステムに限定され、実行時のソフトウエア依存性を求める手段が無い）は、Ｏｐｅｎ Ｇｒｏｕｐの仕様にも当てはまる。

【００１０】
Ｃｕｒｒｅｎｔ Ｏｐｅｒａｔｉｎｇ Ｓｙｓｔｅｍ Ｉｎｖｅｎｔｏｒｙの実施（ＩＢＭのＡＩＸ Ｏｂｊｅｃｔ Ｄａｔａ Ｍａｎａｇｅｒ（ＯＤＭ）、ＬｉｎｕｘのＲｅｄ Ｈａｔ Ｐａｃｋａｇｅ Ｍａｎａｇｅｒ（ＰＲＭ）、またはＭｉｃｒｏｓｏｆｔのＷｉｎｄｏｗｓ（登録商標） Ｒｅｇｉｓｔｒｙ等）は、ＯｐｅｎＧｒｏｕｐの仕様およびＩＥＥＥ１３８７．２規格に従うか、または自社開発フォーマットのソフトウエアインベントリを記述する。このため、上述の制約は、かかるＣｕｒｒｅｎｔ Ｏｐｅｒｅｔａｉｎｇ Ｓｙｓｔｅｍ Ｉｎｖｅｎｔｏｒｙの実施にも当てはまる。

【００１１】
全プログラムパッケージの電子ソフトウエア分配（米国特許第６，００９，５２５号および米国特許第５，７２１，８２４号等）またはアップデート／修正／修理／パッチ（米国特許第５，９９９，７４０号、米国特許第５，８０５，８９１号、および米国特許第５，９５３，５３３号等）のための技法は、定義上、（一度に１つまたは多くの）物理的なソフトウエアパッケージの分配／展開／インストールに制限されており、アプリケーションの実行時段階を考慮しない。更に、それらは、一度に１つのシステムを扱っており、複数のシステムにまたがったアプリケーションおよびサービスの局面を考慮しない。

【００１２】
また、既存のソフトウエア／ハードウエア構成（米国特許第５，８６７，７１４号等）における競合を求めるための技法も、単一のシステムに限定されており、実行時の局面を考慮しない。

【００１３】
既存の研究（米国特許第５，９１７，８３１号等）は、多くの場合、イベントの相関関係の範囲内のものであり（例えば、Ｇｒｕｓｃｈｋｅ等の「Ｉｎｔｅｇａｔｅｄ Ｅｖｅｎｔ Ｍａｎａｇｅｍｅｎｔ：Ｅｖｅｎｔ Ｃｏｒｒｅｌａｔｉｏｎ Ｕｓｉｎｇ Ｄｅｐｅｎｄｅｎｃｙ Ｇｒａｐｈｓ」、ＤＳＯＭ ‘９８、１９９８年、およびＫａｔｋｅｒ等、「Ｆａｕｌｔ Ｉｓｏｌａｔｉｏｎ ａｎｄ Ｅｖｅｎｔ Ｃｏｒｒｅｌａｔｉｏｎ ｆｏｒ Ｉｎｔｇｒａｔｅｄ Ｆａｕｌｔ Ｍａｎａｇｅｍｅｎｔ」、ＩＭ ‘９７、１９９７年）、自社開発のフォーマットにおけるサービス依存性の識別および記述に焦点を当てているが、故障管理プロセスの異なるエンティティ間でどのように依存性情報が実際に交換される可能性があるかについては不明確なままである。外部委託されたアプリケーションの故障管理プロセスに関与する異なる関係者が、依存性追跡のために同一のツールセットを用いる可能性は低いので、依存性情報を規定し交換するためのオープンフォーマットを規定することが根本的に重要である。

【００１４】
また、故障管理プロセスが関与する分散型システムのコンポーネントに関連した異質性のため、既存の技法に制約があることを考えると、シミュレートしたシステム故障（例えばサービスの使用不能）の潜在的な影響を明らかにすることは極めて難しい。

【特許文献１】米国特許第４，７５１，６３５号
【特許文献２】米国特許第５，９６０，１９６号

【特許文献３】米国特許第５，４９３，６８２号
【特許文献４】米国特許第５，８３５，７７７号
【特許文献５】米国特許第６，００９，５２５号
【特許文献６】米国特許第５，７２１，８２４号
【特許文献７】米国特許第５，９９９，７４０号
【特許文献８】米国特許第５，８０５，８９１号
【特許文献９】米国特許第５，９５３，５３３号
【特許文献１０】米国特許第５，８６７，７１４号
【特許文献１１】米国特許第５，９１７，８３１号
【非特許文献１】電気通信学会規格１３８７．２（「Ｐｏｒｔａｂｌｅ Ｏｐｅｒａｔｉｎｇ Ｓｙｓｔｅｍ Ｉｎｔｅｒｆａｃｅ（ＰＯＳＩＸ）ｓｙｓｔｅｍ ａｄｍｉｎｉｓｔｒａｔｉｏｎ，ｐａｒｔ２：Ｓｏｆｔｗａｒｅ Ａｄｍｉｎｉｓｔｒａｔｉｏｎ」、ＩＥＥＥ、１９９５年）
【非特許文献２】Ｏｐｅｎ Ｇｒｏｕｐ（Ｓｙｓｔｅｍｓ Ｍａｎａｇｅｍｅｎｔ：Ｄｉｓｔｒｉｂｕｔｅｄ Ｓｏｆｔｗａｒｅ Ａｄｍｉｎｉｓｔｒａｔｉｏｎ，ＣＡＥ Ｓｐｅｃｉｆｉｃａｔｉｏｎ Ｃ７０１，Ｔｈｅ Ｏｐｅｎ Ｇｒｏｕｐ，１９９８年1月
【非特許文献３】Ｇｒｕｓｃｈｋｅ等、「Ｉｎｔｅｇａｔｅｄ Ｅｖｅｎｔ Ｍａｎａｇｅｍｅｎｔ：Ｅｖｅｎｔ Ｃｏｒｒｅｌａｔｉｏｎ Ｕｓｉｎｇ Ｄｅｐｅｎｄｅｎｃｙ Ｇｒａｐｈｓ」、ＤＳＯＭ ‘９８、１９９８年
【非特許文献４】Ｋａｔｋｅｒ等、「Ｆａｕｌｔ Ｉｓｏｌａｔｉｏｎ ａｎｄ Ｅｖｅｎｔ Ｃｏｒｒｅｌａｔｉｏｎ ｆｏｒ Ｉｎｔｇｒａｔｅｄ Ｆａｕｌｔ Ｍａｎａｇｅｍｅｎｔ」、ＩＭ ‘９７、１９９７年
【発明の開示】
【発明が解決しようとする課題】
【００１５】
要約すると、既存の技術において、ソフトウエアプロダクト間の関係を決定することに関するいくつかの技法が記述され実施されている。これらの既存の技法は、以下の欠点のうち１つ以上を有する。すなわち、
（ａ）ソフトウエアプロダクトのインストールおよび展開の段階のみに対処する。すなわち、設計および実行時の局面を取り入れようとしない。
（ｂ）多数のシステムにまたがる終端間のアプリケーションおよびサービスを扱わない。すなわち、単一の切り離されたシステムに常駐するソフトウエアの特徴を扱う。
（ｃ）ソフトウエアインベントリ情報は、自社開発のフォーマットにおいて記述されるので、様々な異種システム間でこの情報を共有することが極めて難しくなる。
（ｄ）シミュレートされたサービス使用不能の潜在的な影響を効果的に識別しない。
【課題を解決するための手段】
【００１６】
本発明は、コンピューティング環境に従ってシミュレートされたコンポーネント故障の潜在的な影響を識別し、かかるシミュレートされたコンポーネント故障に対する前記コンピューティング環境の脆弱性を明らかにするための技法を提供する。一例として、本発明の技法は、分散型コンピューティング環境に適用可能である。また、コンピューティング環境は、自律型コンピューティング環境とすることも可能である。
【００１７】
例えば、本発明の一態様において、コンピューティング環境に関連した少なくとも１つの対象コンポーネントのシミュレートされた状況の潜在的な影響を明らかにするためのコンピュータに基づいた技法は、以下のステップ／動作を具備する。最初に、少なくとも１つの対象コンポーネントに依存する１つ以上のコンポーネントを識別し、１つ以上の識別されたコンポーネントの各々に関連付けられた潜在的な状況ステータスを判定する。次いで、該判定された状況ステータスに基づいて１つ以上の識別されたコンポーネントの少なくとも一部に関する情報をユーザに提示する。

【００１８】
ユーザに提示される情報は、判定された状況ステータスに基づいて１つ以上の識別されたコンポーネントの少なくとも一部に関連した脆弱性の表現を含む場合がある。一例として、この表現は、１つ以上の識別されたコンポーネントの各々に関連した故障の各可能性を示す脆弱性マップを含む場合がある。更に、ユーザに提示される情報は、潜在的に影響を受けるエンドユーザのリスト、潜在的に影響を受けるコンポーネントのリスト、および／または潜在的に影響を受けるホストのリストを含む場合がある。更に、ユーザに提示される情報は、少なくとも１つの対象コンポーネントのシミュレートされた状況の潜在的な影響の１つの判定と、少なくとも１つの対象コンポーネントのシミュレートされた状況の潜在的な影響の１つの他の判定との比較を含む場合がある。

【００１９】
また、対象コンポーネントは、コンピューティング環境における既存のコンポーネント（例えばその環境において用いられる実際のコンポーネント）とすることができ、または、対象コンポーネントは、コンピューティング環境における既存のコンポーネントではない（例えば、その環境において用いることも可能であるコンポーネントを表す仮想コンポーネント）場合もある。

【００２０】
更に、１つ以上のコンポーネントの識別は、コンピューティング環境のコンポーネントの少なくとも一部に関連した１つ以上の関係の存在を表すモデルの少なくとも一部を横断することを含み、コンピューティング環境の少なくとも１つのコンポーネントに関連した全ライフサイクル（例えば、展開、インストール、および実行時を含む）を考慮に入れることができる。このモデルは、機能カテゴリ分類、構造カテゴリ分類、および動作カテゴリ分類から成る形態とすることができる。

【００２１】
一例として、コンポーネントは、サービス、アプリケーション、ミドルウエア、ハードウエア、デバイスドライバ、オペレーティングシステム、またはコンピューティング環境に関連したシステムとすることができる。しかしながら、「コンポーネント」という言葉は、これらの例に限定されるわけではない。

【００２２】
本発明のこれらおよび他の目的、特徴、および利点は、添付図面に関連付けて読むことができる以下の例示的な実施形態の詳細な説明から、明らかになろう。

【発明を実施するための最良の形態】

【００２３】
本発明について、例示的な分散型コンピューティング環境という文脈の中で、以下に説明する。しかしながら、本発明は、かかる特定のコンピューティング環境に限定されないことは理解されよう。逆に、本発明は、より一般的に、問題を判定し、切り離し、解決するタスクを著しく容易にするために、依存性を管理する（例えば計算する、問い合わせる等）ことが望ましいコンピューティング環境であれば、どんなものにも適用可能である。

【００２４】
本明細書中で用いる場合、考察の文脈に応じて、「システム」という言葉を用いて、コンピュータシステム、ソフトウエアシステム、および／またはそれらの何らかの組み合わせを指すことができる。また、「システム」という言葉を用いて、アプリケーションおよび／またはサービスを指すことができる。このため、「多数のシステム」という言葉は、いくつかのシステムの集合を指す。また、「コンポーネント」という言葉は、システム自体、または、システムの１つ以上の部分を指すことができる。

【００２５】
上述のように、今日のシステムにおいてサービス依存性は明白になっておらず、このため、問題の判定、切り離し、および解決のタスクが特に難しくなっている。この問題を解決するには、異なるシステムおよびドメインにまたがるサービスおよびアプリケーション間の依存性を判定および計算する必要がある。すなわち、「グローバルな」サービス依存

性モデルを確立し、結果として得られる有向グラフをシステムの管理者が上から下までおよび逆の順序で自由に見られるようにすることが必要である。かかる機構に対する必要性は、以下の２つの状況によって最も良く例示される。

【００２６】

第１の状況は、通常インターネットまたはアプリケーションサービスプロバイダ（ＩＳＰ／ＡＳＰ）によって提供される、外部委託されたサービスの管理を扱う。外部委託されたサービスは、層化されたサービス階層となり、例えば、ＡＳＰのサービスはＩＳＰが提供するＩＰ接続性（インターネットプロトコル接続性）に依存し、一方、ＩＳＰは、電気通信事業者のワイドエリアネットワークに頼っている。全ての層において、サービスは、サービスアクセスポイント（ＳＡＰ）を介してアクセスされる。ＳＡＰは、異なる組織ドメイン間の境界を定めるものであり、サービスレベルアグリーメント（ＳＬＡ）を規定し観察する場所である。通常、これは、全ての層において、プロバイダが提示する特定のパラメータ集合を監視することによって行われる。上層サービスにおいて使用不能または性能劣化が起こった場合、サービス階層を上から下まで通り抜けて、問題の根本的な原因の識別をする必要がある。

【００２７】

第２の状況は、「実行中に」行うことができない、従ってサービスおよびそれらのカスタマに影響を与える、定期的な保守タスクを扱う。例えば、ｅ－メールサーバをそれらのオペレーティングシステムの新しいレリースによって更新する、ネットワークデバイスを新しいファームウエアバージョンによって交換または更新する等である。あらゆる場合において、ネットワークおよびサーバ管理者は、保守によって、いくつの、更に具体的には、どのサービスおよびユーザが影響を受けるかを前もって判定することが重要である。このタスクを影響分析と呼ぶ。

【００２８】

影響分析の概念を採用することは、更に、影響シミュレーションおよび脆弱性分析の概念につながる。コンピュータシステム上で実行している特定のサービス（または特定のサービスインスタンス）を選択すると、影響シミュレーションによって、サービスの使用不能が起こると他のサービス、ユーザ、およびホストシステムのどれが影響を受ける可能性があるかを判定することができる。これによって、管理者は、既存の欠点を識別し、それを修理し、適用した修理が成功したか否かを検証するため、システム使用不能をシミュレートし、それらの環境の脆弱性を先回りして分析することができる。影響分析器に体系的にサービスデータを供給すると、分散型システム環境の「脆弱性マップ」を設定することができる。

【００２９】

上述のタスクは、更に、以下のファクタによって悪化する。

【００３０】

依存性モデルは、観察された問題の根本原因であり得るものを識別するための直接的な手段を提供する。システムの依存性グラフが明らかになれば、障害が起こったサービスからその先行者（同一のホストまたは異なるシステム上にコロケートしている）へとグラフを自由に見ることによって、どのエンティティが故障した可能性があるかが明らかとなる。グラフをそのルートへと（すなわち上方向に）横切ることによって、サービスの依存者、すなわち、このサービスが使用不能となった場合に故障する恐れのあるコンポーネントがわかる。以下の問題に対処する必要がある。

【００３１】

（ａ）規模：多くの関与するシステム間の依存性の数を計算することができるが、これは極めて多い場合がある。技術上の観点から見ると、完全な、インスタンス化された依存性モデルを単一の場所に格納することは、多くの場合、望ましくない（時には不可能である）。プラットフォームデータベースにおいてインスタンス化したネットワークマップを維持する等、ネットワーク管理プラットフォームにおいて用いられる従来の機構は、関与する依存性の変化する数およびダイナミクスのため、依存性に適用することはできない。

【００３２】
これらの２つの事実によって、アプリケーション、サービス、およびミドルウエア依存性モデルの展開のための「ネットワーク－管理－スタイル」の手法に従うことができなくなる。一例として、通常のサービス外部委託者のデータセンターは、極めて大量（数千）のウエブアプリケーションおよびデータベースサーバのホストとして働く。これは、例えばウエブアプリケーションおよびデータベースサーバの、非常に多くのプログラムインスタンスが同時に実行していることを意味する。依存性モデルを構築可能であるシステムは、管理プロセスに関与する複数のシステム間に依存性の格納および計算を分散させることによって、適切な拡張性を可能とする機構を提供しなければならない。
【００３３】
（ｂ）ダイナミクス：ホストの下のアプリケーション（ウエブアプリケーションサーバ内で実行する）は、寿命が極めて短く、多くの場合、わずか数秒である。要求を受けると、ウエブアプリケーションのビジネス論理（通常、１つ以上のＪａｖａ（登録商標）　Ｓｅｒｖｌｅｔとして実施される）は、アプリケーションサーバのＳｅｒｖｌｅｔエンジンによってインスタンス化され、そのタスクを実行し、次いで、Ｓｅｒｖｌｅｔエンジンによって除去される。この結果、これらの動的エンティティ間の依存性を計算するためのシステムは、データの正確さとこのデータを検索するために発生した作業負荷との間のトレードオフに対処しなければならない。
【００３４】
（ｃ）異質性：異質性は、３つの異なる特徴で生じる。第１に、カスタマに提供されるサービスには、大きな相違がある。第２に、サービスをカスタマに提供することに関与する様々な提供業者が存在し得る。最後に、サービスを実施する製品は、様々なベンダーから提供される可能性がある。依存性を計算するシステムは、特定のオペレーティングシステム、ネットワークプロトコル、ソフトウエアプロダクト、およびカスタマに提供されるサービスから独立した言語を提供しなければならない。
【００３５】
（ｄ）手作業による依存性データの保守：サービス依存性モデルの獲得は、単一のホストシステムに限定しても、今日のシステムが適切な管理手段を提供していないので、それ自体が難題である。「手段」という言葉は、管理アプリケーションによってアクセス可能であるように、明確に定義された（時には標準化さえされている）インタフェースを介して、（管理された）リソースの管理特徴および機能を提示するプログラムコードを指すことは理解されよう。更に、管理されたリソースから利用可能であっても、依存性データは、今日の管理システムによって活用されない。代わりに、依存性情報は、特定の管理コンポーネントに手作業で入力されなければならないだけでなく、自社開発フォーマットで入力されなければならない。従って、依存性情報は、完全でなく、（誤りを起こしがちな手作業処理のため）時代遅れになっており、時には一貫性がない場合さえある。なぜなら、異なるオペレータがルールを別個に入力し、自動化した方法で一貫性についてルールベースをチェックする方法がないからである。
【００３６】
（ｅ）依存性の分類：依存性の概念は、極めて粗いものであり、有用にするためには改良する必要がある。この例は、依存性の強さ（コンポーネントの先行者が故障した場合にそのコンポーネントが影響を受ける可能性および程度を示す）、重大さ（企業の目標および方針に関してこの依存性がどのくらい重要であるか）、形式化の程度（すなわち依存性を取得するのがどのくらい難しいか）、および他の多くのことである。依存性に属性を付加して、より適切にそれらを限定することができるようにする必要があり、従って、依存性表現にこれらの属性を反映させる必要がある。
【００３７】
（ｆ）問題判定機構：全システムに格納されたローカルな依存性グラフを均一な依存性モデルに組み合わせるための更に別の機能が望まれる。更に、これらの機能は、管理アプリケーションが依存性モデルに対して問い合わせを発することができるＡＰＩ（アプリケ

ーションプログラミングインタフェース）を提供しなければならない。これらの問い合わせは、特定のサービスが直接依存するエンティティを検索すること、または、副先行者を含むノード集合全体を再帰的に判定することを可能とする。管理アプリケーションによって受信されたノードのリストによって、特定の問題判定ルーチンを実行して、これらのサービスが動作中であるか否かを調べることができる。

【００３８】

先の考察によって、サービスライフサイクルの異なる段階間でマッピングを行うことが重要であることが示される。

【００３９】

（ａ）（抽象的な）サービスのカスタマへの提供。例えば、「ウエブホスティング」、「管理された記憶装置」、「ＩＰ接続性」、「管理されたデータベース」等である。

（ｂ）サービスの実施。すなわち、例えば「ＩＢＭ　Ｕｎｉｖｅｒｓａｌ　Ｄａｔａｂａｓｅバージョン７．１」、「ＷｅｂＳｐｈｅｒｅ　Ａｐｐｌｉｃａｔｉｏｎ　Ｓｅｒｖｅｒバージョン３．２」のようなサービスを提供するためのプロダクト（複数のプロダクト）の使用。

（ｃ）実施のインスタンス（複数のインスタンス）の実行。すなわち、プロセスまたはタスク、例えば「ｄｂ２　ｄａｅｍｏｎ」、「ｎｆｓ　ｄａｅｍｏｎ」。

【００４０】

全ての段階において利用可能な情報を個別に得るタスクは実現可能であるが、３つの段階を均一な依存性モデルに組み合わせることは、難題であり、以前の研究では行われていない。更に、人による介入および依存性データの保守の必要性をなくしつつ、基礎的な環境の規模、ダイナミクス、および異質性の要求に対処する、効率的に計算可能な依存性モデルを確立する必要がある。

【００４１】

図面に関連付けて以下に例示するが、本発明は、これらおよび他の必要性に応える。すなわち、本発明は、管理アプリケーションの代わりに分散型システムのコンポーネント間の実行時依存性（「依存性モデル」）を計算する機構を有する。本発明は、個々のコンピュータシステムの構成情報を検索するための機構を提供するか、またはかかるデータを機械可読フォーマットで提供する、コンピュータシステムから依存性情報を検索するための一般的かつ一貫した手法を提供する。

【００４２】

上述のシステムの１つの利点は、個々のアプリケーション／サービスに手段を備える必要なく、これらのコンピュータシステムから大量のアプリケーション／サービス管理情報を取得可能なことである。しかしながら、かかるアプリケーション／サービスの手段が利用可能である場合、これを本発明によって用いることができる。

【００４３】

本発明によって説明されるシステムの実行は、特定の（管理）アプリケーション（影響分析器、根本原因分析器等）、ネットワーク管理プラットフォーム（ＩＢＭ／Ｔｉｖｏｌｉ　ＮｅｔＶｉｅｗ、ＨＰ　ＯｐｅｎＶｉｅｗまたはＡｐｒｉｓｍａ　Ｓｐｅｃｔｒｕｍ）、または従来のネットワーク管理システムおよびプラットフォームに基づく管理アプリケーションのいずれかによって作動させることができる。

【００４４】

本発明は、とりわけ、以下のための機構を提供する。

（ａ）加入サービスの性能劣化および使用不能を観察する。

（ｂ）依存性モデルの異なる層を上から下まで通り抜けることによって、問題の根本原因を突き止める（なぜなら、様々なサービスが他のサービス提供業者に外部委託されており、この（再帰的な）依存性モデルの横断はドメインの境界を横切るからである）。

（ｃ）依存性モデルを下から上まで自由に見ることによって、サービスの使用不能の影響を分析する。

（ｄ）サービスまたはホストシステムの使用不能が、他のサービス、ユーザ、および他

のホストシステムに及ぼす潜在的な影響をシミュレートする。

【００４５】

本発明は、アプリケーションまたはサービスのライフサイクルの間に利用可能な依存性情報を組み合わせる（すなわち、アプリケーション／サービスの設計から展開、インストール、および実行時段階まで）。この情報は、以下のモデル内に維持される。

【００４６】

（ａ）機能モデル：好適な実施形態では、機能モデルは、異なる一般サービス（データベースサービス、名前サービス、ウエブアプリケーションサービス、接続性サービス等）間の依存性を規定する。機能モデルは、特定のサービス内のクライアント／サーバ関係を記述しない。更に、機能モデルは、サービスを実施するためにどの具体的な製品を選択したかも、それらの実際の構成も、考慮しない。機能モデルは、他のモデル（以下で説明する）を結び付ける主要な制約を確立する。すなわち、別のモデルは、具体的なシステムインフラストラクチャに関して機能モデルにおいて規定される依存性を改良することができるが、サービスカテゴリ間に新たな依存性を導入してはならない。このモデルは、極めてコンパクトであり、一般的であり、好ましくは管理システム上に格納される。

【００４７】

（ｂ）構造モデル：好適な実施形態では、構造モデルは、機能モデルにおいて規定されたサービスを実現するソフトウエアコンポーネントの詳細な記述を含む。構造モデルは、インストール／展開段階の間に捕捉された詳細事項を提供し、具体的なシステムのソフトウエアインベントリを考慮することによって機能モデルを補足する。構造モデルは、特定のシステム上でどのサービスをインストールおよび構成するか、ならびに、全てのサービスについて、システムがクライアントまたはサーバのどちらの役割で動作するかに関する情報を提供する。システムおよびサービスの数が潜在的に多いことによって、遠隔位置からこれらの依存性を追跡することは難しくなる。このため、管理されたリソースの近くで、または管理されたリソースにおいて、このモデルを格納することが望ましい。

【００４８】

（ｃ）動作モデル：好適な実施形態では、ソフトウエアパッケージがインスタンス化され、サービスとアプリケーションとの間の結合が確立されると、依存性の動作モデルが生成される。このモデルおよび多数の関与するシステムの高度なダイナミクスのために、完全なモデルをインスタンス化し格納することができる程度が制限される。かかるモデルを規定し格納することは現実的でなく、むしろ、モデルは動的かつ段階的に計算しなければならない。従って、動作モデルは、「オンデマンド」で計算され、機能モデルおよび構造モデルに頼っている。

【００４９】

予想されるように、大規模な分散型システムでは、依存性の量は極めて多く、それらのダイナミクスは極めて高度である。本発明の機構は、分散型システムに対するそれらの（リソースおよび帯域幅の使用という点での）影響を、できる限り小さく抑え、ユーザまでの性能に影響する可能性がある多くの構成オプションを与える。この例として、更新した依存性モデルを検索するための時間間隔、依存性を追跡しなければならないシステムの範囲、依存性モデルの深さ（直ちに影響を受けるサービスのみ、所与のサービスの推移的な終了、またはサービス階層全体）が挙げられる。

【００５０】

本発明は、好ましくは、以下の依存性情報の特徴を利用する。

【００５１】

（ａ）異なるサービス間の依存性を層化する。更に、それらの依存性グラフは有向であり無周期である。後者の表現は、ＤＮＳ（Ｄｏｍａｉｎ　Ｎａｍｅ　Ｓｙｓｔｅｍ）、ＮＦＳ（Ｎｅｔｗｏｒｋ　Ｆｉｌｅ　Ｓｙｓｔｅｍ）、ＤＦＳ（Ｄｉｓｔｒｉｂｕｔｅｄ　Ｆｉｌｅ　Ｓｙｓｔｅｍ）、ＮＩＳ（Ｎｅｔｗｏｒｋ　Ｉｎｆｏｒｍａｔｉｏｎ　Ｓｙｓｔｅｍ）等のＩＰに基づくネットワークサービスによる経験を反映しているが、いくつかのシステムにおいて相互依存性が生じる場合もあり得る。かかる相互依存性の異常な例は

、ＤＮＳサーバが、ファイルシステムを搭載し、遠隔システムからＮＦＳを介してそのＤＮＳ構成を格納するというものである。かかる構成は技術的には実現可能であるが、システム設計の欠陥をもたらす。なぜなら、これによって、立ち上げが非決定論的である恐れがあり、従って回避しなければならない不安定なシステムとなるからである。周期的な依存性を発見する依存性チェックアプリケーションは、管理者に警告を発しなければならない。

【００５２】

（ｂ）全ての依存性は、カスタマ／提供業者ドメインの境界において可視的であり、ＳＬＡによって明白になる。その結果、観察可能な依存性の数は有限である。

【００５３】

（ｃ）依存性モデルによって、依存性連鎖をトップダウンで横断することが可能となる。

【００５４】

（ｄ）異なるシステム間（システム間）の依存性は、同一サービスのクライアントおよびサーバの部分間の依存性として知覚される。サービスＡのクライアントが、異なるサービスＢを提供するサーバに要求を出すことは不可能である。

【００５５】

本発明の１つの目的は、具体的なサービス／アプリケーション手段から最大の程度の独立性を達成するため、主にいくつかの周知の／明確に規定された場所（例えばシステムレポジトリ）から情報を検索することである。これを実現するため、本発明は、最小かつ十分な量の共通して利用可能な依存性情報を規定する。

【００５６】

本発明は、依存性モデルを永続的に格納するための機能を具備するか、または、これを、本発明を用いる管理アプリケーションまたは別のサービスの自由裁量に委ねる。

【００５７】

本発明は、依存性モデルの変化を検出し判定するための履歴の概念を有することができる。この場合、本発明は、依存性モデル内の変化について以前に登録されたソフトウエアコンポーネントに通知するための発行／加入インタフェースを提供する。本発明の別の可能な使用法は、依存性モデルの変化の検出を、管理アプリケーション（または変化管理サービス）の自由裁量に委ねて、依存性モデルの変化が生じたか否かを判定するために本発明に定期的に呼を発することである。

【００５８】

更に、以下に例示するように、本発明は、影響シミュレーションおよび脆弱性分析を実行するための技法を提供する。

【００５９】

本発明は、様々なサービス、ユーザ、およびホストシステムを、それらの依存性と共に表すグラフィカルユーザインタフェースを備えている。グラフィカルユーザインタフェースは、管理者によって特定のサービスが選択されると、サービスの使用不能によって影響を受ける可能性があるサービス／ユーザ／ホストシステムを視覚的に表すマップを表示する。かかる「脆弱性マップ」は、分散型環境内の既存の欠点および単一点の故障を正確に指摘するのに役立つ。

【００６０】

また、本発明は、「合成」（すなわち仮想）サービスインスタンス／ユーザ／ホストシステムを分散型環境内に導入して、新たに導入したサービス／ホストシステムが既存の脆弱性を排除するのに役立つか否かを評価するための機構を備えている。

【００６１】

また、本発明は、あるサービスの使用不能が、分散型環境の他のサービス、ユーザ、およびホストシステムに対して及ぼす影響をシミュレートする。サービス使用不能の影響を判定することは、サービス階層を上から下まで通過して、問題を生じる可能性がある依存者サービスを識別することを示す。なぜなら、この問題は、対象のサービスから依存者に

伝搬する恐れがあるからである。使用不能が観察されたサービスからその依存者に向かってのこの通過を行う際には、特定のサービスに直接依存するエンティティ、特定のサービスに依存するサービス集合全体、またはエンドユーザサービスから成る最高サービス層のいずれかを検索する。影響シミュレータによって検索された依存者ノードのリストによって、管理者は、「ｗｈａｔ－ｉｆ」分析を実行して、サービス使用不能に対する分散型環境の脆弱性を正確に指摘することができる。影響シミュレータは、問題の根本原因であると見なされるサービスに対する依存性関係を有しないノードを除去することを注記しておくことは重要である。

【００６２】

本発明によって、管理者は、コンピュータシステム上で実行している特定のサービスまたは特定のサービスインスタンスを選択し、サービスの使用不能が起こると他のサービス、ユーザ、およびホストシステムのどれが影響を受ける可能性があるかを判定することができる。これによって、管理者は、既存の欠点を識別し、それを修理し、適用した修理が成功したか否かを検証するため、システム使用不能をシミュレートし、それらの環境の脆弱性を先回りして分析することができる。影響分析器に体系的にサービスデータを供給すると、管理者は、分散型環境の脆弱性マップを設定することができる。

【００６３】

分散型環境の脆弱性マップを設定して本発明を用いる例は、以下の通りである。管理者は、分散型環境のサービスおよびそれらの依存性を表すグラフィカルユーザインタフェースを用い、ホストシステム上の特定のサービスを選択し、使用不能によって影響を受けるサービスについて問い合わせることによって影響シミュレータを実行する。次いで、影響シミュレータは、結果のリストを計算する。このリストは、（最終的に確率または重大さの程度に依存して）影響を受けるサービスを表すアイコンの色を変えることによって、グラフィカルユーザインタフェース上に表示されると好ましい。かかる脆弱性マップによって、実際の環境のコンポーネントと接続する必要なく、管理者は、実際の（「現実の」）データに基づいて、分散型環境における「弱い」サービスを正確に指摘することができる。

【００６４】

更に、本発明の方法は、１つ以上の対象コンポーネントについて実質的に同時に（例えば、一度に１つのコンポーネント、または多数のコンポーネントを並列に）実行することができる。本発明の方法の結果の少なくとも一部を永続的に格納することができるが、かかる結果は永続的に格納しない場合もある。

【００６５】

更に、本発明の方法に関連した結果の履歴を維持することができる。このため、本発明は、影響シミュレータの実行結果を、以前の影響シミュレーション結果に対して比較することができるように格納するための機構を備える。これによって、影響シミュレータの１度の実行と影響シミュレータの１度以上の他の実行との間で実施されるように、分散型システムから欠点および単一点の故障を排除するための動作が成功したか否かを評価し検証することができる。

【００６６】

本発明に従って成される上述の達成および本発明に関連した一般的な特徴を想定し、詳細な説明の残り部分では、図１ないし１５の状況においてかかる達成および特徴を実施するための技法を例示的に説明する。

【００６７】

最初に図１を参照すると、ブロック図が、本発明の機構と相互作用して情報を生成可能なクライアント－サーバアプリケーションアーキテクチャの形態で電子商取引システムの一例を示す。図１のアーキテクチャについて、以下で説明して、かかるアーキテクチャが本発明の技法がない場合にトランザクションをどのように扱うことができるかを示す。

【００６８】

図示するように、クライアントシステム１０５を用いて、例えばキーボードを介して要

求を開始する。しかしながら、要求は、マウスクリック、音声コマンド、バーコード読み取り等のいずれかの従来の手段によって開始することも可能である。クライアントシステム１０５の例は、パーソナルコンピュータ、キオスク、データ入力端末、スキャナ、電話、ポケットベル、手持ち式または着用式デバイス、無線デバイス、パーソナルデジタルアシスタント、ネットワークによって動作可能となる時計等である。

【００６９】

要求が定式化され、ネットワーク１１０を介してウエブアプリケーションサーバ１２０に送信された場合、１つ以上のネットワークアクセス１１５デバイスを通過することによって、要求がローカルに実行される。ネットワーク１１０および通信プロトコルの例は、ローカルエリアネットワーク（ＬＡＮ）を介したＴＣＰ／ＩＰ（Ｔｒａｎｓｍｉｓｓｉｏｎ Ｃｏｎｔｒｏｌ Ｐｒｏｔｏｃｏｌ／Ｉｎｔｅｒｎｅｔ Ｐｒｏｔｏｃｏｌ）転送によるソケットを用いた通信であり、これは、ルータ等のネットワークアクセス１１５デバイスによって接続され、多くの切り替え位置を含むワイドエリアネットワーク（ＷＡＮ）に切り替わって、サービス提供業者および最終的にウエブアプリケーションサーバ１２０へのバーチャル回路を生成する。ウエブアプリケーションサーバ１２０の例は、高性能パーソナルコンピュータ、ＲＩＳＣベースのＰｏｗｅｒＰＣ、ＵＮＩＸ（登録商標）ベースのワークステーション、ミニコンピュータ、またはメインフレームコンピュータであり、これらは、クライアントからの要求を受けて、その要求を適切なバックエンドデータベースサーバに適宜分配するソフトウエアを実行する。

【００７０】

例示の目的のため、ここで、インターネットを用いて品物を購入するためにウエブブラウザ（クライアントシステム１０５上で動作している）内で開始する電子商取引トランザクションについて説明する。本発明の技法は、いかなる形態のトランザクションでも機能することができることは理解されよう。ウエブアプリケーションサーバの例は、ＷＥＢＳＰＨＥＲＥの商標でＩＢＭ社から入手可能なもの、ＷＥＢＬＯＧＩＣの商標でＢＥＡ Ｓｙｓｔｅｍｓ社から入手可能なもの、またはＬＯＴＵＳ ＤＯＭＩＮＯ ＳＥＲＶＥＲの商標でＬｏｔｕｓ社から入手可能なものが含まれるが、これらに限定されるわけではない。

【００７１】

例示のトランザクションにおいて、ウエブアプリケーションサーバ１２０のビジネスロジックは、入来する要求を処理し、クライアントシステム１０５の認証および／または識別を行う。いったん、ウエブアプリケーションサーバ１２０が実施するビジネスロジックが、クライアントがこの購入を続行可能であると判定すると、これは、ネットワーク１２３を介してデータベースサーバ１２５に対して別の要求を伝達して、インベントリを減らす。データベースサーバ１２５は、この要求を処理し、そのデータベース１３０にアクセスし、ウエブアプリケーションサーバ１２０への応答を準備する。データベースサーバの例は、ＳＱＬ／ＳＥＲＶＥＲまたはＴＲＡＮＳＡＣＴＩＯＮ ＳＥＲＶＥＲの商標でＭｉｃｒｏｓｏｆｔ社によって販売されているもの、またはＤＢ２ ＵＮＩＶＥＲＳＡＬ ＤＡＴＡＢＡＳＥ ＳＥＲＶＥＲの商標でＩＢＭ社から販売されているものを含むが、これらに限定されるわけではない。

【００７２】

ウエブアプリケーションサーバ１２０は、データベースサーバ１２５から応答を受け取り、これを、ネットワーク１１０を介してクライアントシステム１０５に戻す。次いで、クライアントシステム１０５は、この応答を処理して、これを表示するためにフォーマットし、トランザクションイニシエータが調べるためにこの応答を提示する。

【００７３】

管理者１００は、サービス提供業者の場所に位置する、ビジネストランザクションを処理する様々なソフトウエアおよびハードウエアコンポーネントを観察し、それらが適正に機能しているか否かを判定する。データベース１３０において、テーブルスペースが壊れたり、データベース実行時システムが故障したりする等の使用不能１３５が起こった場合

、管理者１００のタスクは、使用不能の原因を突き止め、問題を正し、システム全体が再び適正に機能しているか否か確認することである。本発明は、あらゆる形態の使用不能または性能劣化に対する動作を意図することは理解されよう。

【００７４】

管理者１００は、ソフトウエアおよびハードウエアコンポーネントと、直接にまたは管理システムを介して対話する。この管理システムは、明確に規定された管理インタフェースにおいてソフトウエアおよびハードウエアコンポーネントが提示する管理情報（ステータスおよび健康データ等）を処理する。いずれの場合であっても、ハードウエアおよびソフトウエアコンポーネントは、管理者によって、別個のリソースとして認知され、特定のビジネス目的に供するシステム全体の一部として認知されるものではないことを注記しておくことは重要である。

【００７５】

特に、１つのコンポーネントにおいて生じているエラーは、見つけることができない場合がある。なぜなら、管理者は、連続的に監視していないので、それらに気づくことがないからである。更に、本発明の技法を用いなければ、様々なコンポーネント間の相互依存性に関する明確な情報は、管理者に直接的に利用可能ではない。このため、連続的に監視されていないコンポーネント内のエラーは、その故障が被監視コンポーネントに伝搬するまで、見つからないままである可能性がある。

【００７６】

上述のデータベース使用不能１３５の場合、管理者は、ウエブアプリケーションサーバ１２０が適正に機能していない場合に使用不能について最終的に気づくのみである場合がある（例えば、ウエブアプリケーションサーバは、データベースサーバ１２５に対する接続を再試行し続け、クライアントシステム１０５が送信した要求を完了することができないので、ウエブアプリケーションサーバに対する負荷が激増する）。このため、管理者１００は、最初に、ウエブアプリケーションサーバ１２０について調査し、次いで、ネットワーク１２３の接続性の問題があるか否か判定し、最終的に、データベースサーバ１２５が、データベース１３０における内部エラーに起因し得る問題に直面しているか否かを検証する。

【００７７】

上述のクライアントサーバアプリケーションアーキテクチャは、ＩＢＭ社が「自律的」コンピューティング環境と呼ぶ新生のコンピューティング環境の先駆けと考えることができる。ＩＢＭ Ｒｅｓｅｒｃｈ、２００１年１０月の、Ｐ．Ｈｏｒｎによる、「Ａｕｔｏｎｏｍｉｃ Ｃｏｍｐｕｔｉｎｇ：ＩＢＭ’ｓ Ｐｅｒｓｐｅｃｔｉｖｅ ｏｎ ｔｈｅ Ｓｔａｔｅ ｏｆ Ｉｎｆｏｒｍａｔｉｏｎ Ｔｅｃｈｎｏｌｏｇｙ」は、自律的コンピューティングを、人による介入を最小にした自己管理コンピューティングシステムに対する包括的かつ全体論的な取り組み方法として規定する。この開示は、引用により本願にも含まれるものとする。この語は、自覚も介入もせずに重要な機能を制御する身体の自律神経系に由来する。更に具体的には、自律的コンピューティングの目的の１つは、管理者１００が通常実行するタスクの一部または全てを自動化することである。そうするための動機は、以下の通りである。

【００７８】

コンピューティングが進化すると、重複する接続、依存性、および相互作用するアプリケーションによって、いかなる人間が行い得るよりも速い管理的な意思決定および応答が必要となる。故障の根本原因を正確に指摘することは更に難しくなり、一方で、システム効率を向上させる方法を見出すことは、どんな人が解決を期待するよりも多くの変数を有する問題を発生させる。自律的コンピューティング環境の異なるシステム間の依存性を識別し追跡する問題は、以下のように特徴付けることができる。１つのシステムは多くのレベルで存在し得るので、自律的システムは、それ自身を制御するため、そのコンポーネント、現在のステータス、最終的な容量、および他のシステムとの接続全てについて、詳細な知識を必要とする。本発明を自律的コンピューティング環境において実行可能であるこ

とは、当業者には認められよう。

【００７９】

ここで図２を参照すると、ブロック図は、本発明の一実施形態による、依存性管理を行うためのシステムを示す。更に具体的には、図２は、上述の問題に対処する依存性管理システムを示す。このシステムは、４つの層（アプリケーション層２００、サービス層２０５、ミドルウエア層２１０、およびリソース層２１５）、ならびに、管理者グラフィカルユーザインタフェース２８５を備え、このインタフェース２８５によって管理者１００はシステムと対話する。

【００８０】

最下層は、リソース層２１５である。リソース層２１５は、被管理リソース２２０、リソース依存性レポジトリ２２５、およびレポジトリエージェント２３０を備える。被管理リソース２２０の例は、物理的および論理的ハードウエアコンポーネント（前者の例は、ハードディスク、ランダムアクセスメモリ、中央演算処理装置、ネットワークアダプタ、チャネルコントローラ等であり、後者の例は、ディスクパーティション、ファイルシステム等である）、ならびにソフトウエアコンポーネント（オペレーティングシステム、プリントスプーラーまたは名前サービスのようなシステムサービス、およびエンドユーザアプリケーション等）を含むが、これらに限定されるわけではない。

【００８１】

リソース依存性レポジトリ２２５は、全ての被管理リソース２２０のハードウエアおよびソフトウエアコンポーネントのインベントリおよび依存性情報（すなわち被管理リソース２２０内のコンポーネント間の依存性）をリソースごとに含む。リソース依存性レポジトリ２２５は、個々の被管理リソース２２０の全てにコロケートするか、または、１つの場所に集中化して置くことができる。リソース依存性レポジトリ２２５は、レポジトリエージェント２３０を介して、問い合わせ、更新、変更することができる。レポジトリエージェント２３０は、リソース依存性レポジトリ２２５の情報をシステムの他のコンポーネントに利用可能とする。

【００８２】

ミドルウエア層２１０は、プロトコルおよびオブジェクト要求ブローカ等の管理通信インフラストラクチャ２３５を備え、これによって、システムの異なるコンポーネントが（管理）情報を交換する。

【００８３】

サービス層２０５は、政策、イベント、およびディレクトリ等の様々な一般管理サービス２５０を備え、これらは、様々な管理アプリケーションによって使用可能である。特に重要なサービスは、依存性サービス２４５であり、これは、被管理リソース２２０およびレポジトリエージェント２３０の双方から情報を検索し、この情報を処理して、リソース環境全体の終端間依存性モデルを確立する。このモデル（またはその部分）は、依存性サービス２４５の必要に応じて（例えば、より迅速な検索のためのキャッシュすること）終端間依存性レポジトリ２４０に格納される。依存性サービス２４５は、上述のシステムにおいて、終端間依存性レポジトリ２４０と直接対話する唯一のコンポーネントであることを注記しておく。

【００８４】

上述の依存性モデルおよびその部分は、先に引用し、同時に出願された、「Ｍｅｔｈｏｄｓ　Ａｎｄ　Ａｐｐａｒａｔｕｓ　Ｆｏｒ　Ｍａｎａｇｉｎｇ　Ｄｅｐｅｎｄｅｎｃｉｅｓ　ｉｎ　Ｄｉｓｔｒｉｂｕｔｅｄ　Ｓｙｓｔｅｍｓ」と題する弁理士整理番号第ＹＯＲ９２００２００９７ＵＳ１が付された米国特許出願において開示された技法に従って発生することができる。この出願の一部の例示的な詳細事項を、以下に述べる。しかしながら、他のモデル発生技法も使用可能である。

【００８５】

アプリケーション層２００は、一般管理サービス２５０および／または依存性サービス２４５を用いる様々な管理アプリケーションを備える。かかる管理アプリケーションの例

は、故障マネージャ２６０、トポロジ発生器２６５、影響分析器２７０、影響シミュレータ２７５、および根本原因分析器２８０を含むが、これらに限定されるわけではない。

【００８６】

根本原因分析器２８０は、使用不能によって影響を受けるコンポーネントからその先行者に向けて、（依存性サービス２４５が提供する）依存性モデルを横断することに基づいて、使用不能の根本原因（すなわち、その使用不能を最初に引き起こしたコンポーネント）を決定する。根本原因分析器は、先に引用し、同時に出願された、「Ｍｅｔｈｏｄｓ Ａｎｄ Ａｐｐａｒａｔｕｓ Ｆｏｒ Ｒｏｏｔ Ｃａｕｓｅ Ｉｄｅｎｔｉｆｉｃａｔｉｏｎ ａｎｄ Ｐｒｏｂｌｅｍ Ｄｅｔｅｒｍｉｎａｔｉｏｎ ｉｎ Ｄｉｓｔｒｉｂｕｔｅｄ Ｓｙｓｔｅｍｓ」と題する弁理士整理番号第ＹＯＲ９２００２００９６ＵＳ１が付された米国特許出願において開示された技法を採用することができる。しかしながら、他の根本原因分析技法も使用可能である。

【００８７】

影響分析器２７０は、使用不能を経験しているコンポーネントからその依存者に向けて、（依存性サービス２４５が提供する）依存性モデルを横断することに基づいて、使用不能の影響（すなわち、使用不能によって影響を受ける可能性のあるコンポーネント）を決定する。影響分析器は、先に引用し、同時に出願された、「Ｍｅｔｈｏｄｓ Ａｎｄ Ａｐｐａｒａｔｕｓ Ｆｏｒ Ｉｍｐａｃｔ Ａｎａｌｙｓｉｓ ａｎｄ Ｐｒｏｂｌｅｍ Ｄｅｔｅｒｍｉｎａｔｉｏｎ 」と題する弁理士整理番号第ＳＯＭ９２００２０００４ＵＳ１が付された米国特許出願において開示された技法を採用することができる。

【００８８】

以下で更に詳細に述べる影響シミュレータ２７５によって、管理者１００は、分散型環境全体に対して特定のコンポーネントの使用不能が与える影響をシミュレートすることによって、「ｗｈａｔ－ｉｆ」分析を実行することができる。これによって、適切なフェイルオーバ解決策を提供することができる。

【００８９】

故障マネージャ２６０は、根本原因分析器２８０または影響分析器２７０のいずれかによって故障の候補として識別されたコンポーネントに対し、適切な「健全さのチェック」すなわちテストを実行する。すなわち、故障マネージャは、根本原因分析器２８０または影響分析器２７０の指示によって、かかるテストを実行し（すなわち、これらのモジュールに対するインタフェースとして機能し）、それに結果を報告することができる。しかしながら、根本原因分析器２８０または影響分析器２７０は、故障マネージャとは独立してそれら自身のテストを実行することも可能である。

【００９０】

故障マネージャは、好ましくは、特定用途向けまたは特定リソース向けのツールの集合から成り、これによって、テストされているコンポーネントが適性に機能しているか否かを判定可能であることは理解されよう。このため、関連ツールによってコンポーネントをテストした後、故障マネージャは、そのコンポーネントが「機能している」か、または「機能していない」かを示すメッセージを戻すことができる。これらのツールは、自動化されているもの、および／または手作業によるものとすることができる。自動化の一例として、いわゆる「ピング」プログラムが、ネットワーク接続性をチェックする。対象の遠隔システムがピングに応答した場合、これはオンラインであり、そのネットワークプロトコルスタック（および全ての基礎にあるハードウエア、例えばネットワークアダプタ、ケーブル、中間ネットワークコンポーネント等）は機能している。遠隔システムが応答しない場合、少なくとも何かが不調であることがわかり、別のツール（複数のツール）（の集合）を用いて問題を決定することができる。このように、故障マネージャはピングプログラムを用いることができ、更に、分散型コンピューティング環境のコンポーネントをテストするために必要な他のツールを、いくつでも、またどんな種類のものでも、使用可能である（例えば、心拍検出、ステータス指示等）。

【００９１】

トポロジ発生器２６５は、ウエブアプリケーション、データベースインスタンス、およびトランザクション等、大量の極めて動的なコンポーネントから成る分散型システムの全体的なトポロジ（の副集合）を確立する。トポロジ発生器２６５を用いる一例は、特定のクライアントシステム１０５の要求を満たすのに関与する分散型システムのコンポーネントを表示することである。依存性モデル（またはその部分）を、トポロジ発生器２６５の必要（例えば、より迅速な検索のためにキャッシュすること）に従って、トポロジデータベース２５５に格納する。トポロジ発生器２６５は、上述のシステムにおいて、トポロジデータベース２５５と直接対話する唯一のコンポーネントであることを注記しておく。トポロジ発生器は、先に引用し、同時に出願された、「Ｍｅｔｈｏｄｓ Ａｎｄ Ａｐｐａｒａｔｕｓ Ｆｏｒ Ｔｏｐｏｌｏｇｙ Ｄｉｓｃｏｖｅｒｙ ａｎｄ Ｒｅｐｒｅｓｅｎｔａｔｉｏｎ ｏｆ Ｄｉｓｔｒｉｂｕｔｅｄ Ａｐｐｌｉｃａｔｉｏｎｓ ａｎｄ Ｓｅｒｖｉｃｅｓ」と題する弁理士整理番号第ＳＯＭ９２００２０００３ＵＳ１が付された米国特許出願において開示された技法を採用することができる。しかしながら、他のトポロジ発生技法も使用可能である。

【００９２】

ここで図３を参照すると、図面に示され、ここで詳細に説明する、依存性管理を行うためのシステムの様々な機能コンポーネント／モジュールを実施するのに適切なコンピュータシステムの一般化ハードウエアアーキテクチャを例示するブロック図が示されている。依存性管理システムの個々のコンポーネント、すなわち、グラフィカルユーザインタフェース２８５、アプリケーション層２００、サービス層２０５、およびミドルウエア層２１０（図２）に関連したコンポーネントは、図３に示すようなアーキテクチャを有する１つ以上のコンピュータシステム上で実施可能であることは理解されよう。図２に示す他のコンポーネント、例えばリソース層２１５に関連したコンポーネントも、同様のコンピュータシステム上で実施可能である。

【００９３】

図示のように、コンピュータシステムは、プロセッサ２９０、メモリ２９２、およびＩ／Ｏデバイス２９４に従って実施することができる。「プロセッサ」という言葉は、ここで用いる場合、例えば、ＣＰＵ（中央演算処理装置）および／または他の処理回路を含むもの等、あらゆる処理デバイスを含むことが意図されることは認められよう。「メモリ」という言葉は、ここで用いる場合、例えば、ＲＡＭ、ＲＯＭ、固定メモリデバイス（例えばハードドライブ）、着脱可能メモリデバイス（例えばディスケット）、フラッシュメモリ等、プロセッサまたはＣＰＵに関連したメモリを含むことを意図する。更に、「入力／出力デバイス」または「Ｉ／Ｏデバイス」という言葉は、ここで用いる場合、例えば、処理ユニットにデータを入力するための１つ以上の入力デバイス（例えばキーボード）、および／または、処理ユニットに関連した結果を提示するための１つ以上の出力デバイス（例えばＣＲＴディスプレイおよび／またはプリンタ）を含むことを意図する。

【００９４】

また、「プロセッサ」という言葉は２つ以上の処理デバイスを指す場合があること、更に、１つの処理デバイスに関連した様々な要素を他の処理デバイスによって共有可能であることも理解されよう。

【００９５】

従って、ここで述べるような本発明の方法論を実行するための命令またはコードを含むソフトウエアコンポーネントは、関連する１つ以上のメモリデバイス（例えばＲＯＭ、固定または着脱可能メモリ）に格納することができ、利用する準備が整った場合、部分的にまたは全体的に（例えばＲＡＭに）ロードし、ＣＰＵによって実行することができる。

【００９６】

ここで図４を参照すると、ブロック図が、本発明の一実施形態によるサービスの機能依存性モデルを示す。更に具体的には、図４は、図１に示したもののような電子商取引システムにおける様々なコンポーネント間の機能アプリケーション依存性の図を示す。この機能依存性モデルは、分散型システムの機能コンポーネントおよびそれらの依存性の双方を

表す。このため、このモデルは、ビジネスの観点から見ると微小と考えられる一般的なサービス間の依存性を規定する。これは、機能モデルが、ビジネスサービス内で発生する依存性とは関連しないことを意味する。かかる分解は、サービスを実施するために用いられている特定の製品の範囲において意味を成し、以下で、図５を参照して更に詳細に説明する。

【００９７】

コンポーネント間の依存性は、矢印として示す。１つの矢印は、常に依存者から先行者を指し示す。機能コンポーネントは、（副）サービスであって、カスタマに終端間サービスを提供するためにサービス提供業者によって展開する必要がある。後者はサービスレベルアグリーメントにおいて規定されている。機能モデルは、終端間サービスの設計に焦点を当て、終端間サービスの技術的実現に関する詳細事項を抽象化する。この詳細事項とは、サービス提供のために用いられている製品、それらの位置（ローカルまたは遠隔システム）、提供業者のドメイン（すなわち、提供業者自身がそのサービスの一部をユーザに透過的な別のサービス提供業者に外部委託するか否か）等である。

【００９８】

図示のように、ｅ－ビジネスアプリケーション３００サービスは、ビジネスロジックのホストとなるためにウエブアプリケーションサービス３０５に依存する。適正に機能するために、ウエブアプリケーションサービス３０５は、２つの更に別のサービスを必要とする。電子商取引ウエブサイトの静的コンテンツはウエブサービス３１０によって提供され、一方、バックエンドデータベースサービス３３０は、カスタマに提供されているｅ－ビジネスアプリケーション３００の動的コンテンツ（製品記述、ユーザおよび製造業者のデータ、ショッピングカート、ユーザのプロファイルおよび好み、支払い情報等）を格納する。ウエブサービス３１０は、それ自体、２つのサービス、すなわち、ホスト名をＩＰアドレスにマッピングするための名前サービス３１５、および、ネットワーク接続性のためのＩＰサービス３２０に依存する。

【００９９】

依存性関係は推移的であること、すなわち、所与のコンポーネントの依存者は、コンポーネント自体に加えて、コンポーネントの先行者（複数の先行者）も必要とすることを思い出されよう。このため、ＩＰサービス３２０およびデータベースサービス３３０に加えて、全ての図示したサービスは、オペレーティングシステム（ＯＳ）３２５のサービスの存在を必要とする。簡潔さのため、ハードウエアコンポーネントに対するＯＳ３２５の依存性関係は図示しないが、それらは機能モデル内に存在している。

【０１００】

ここで図５を参照すると、ブロック図は、本発明の一実施形態によるサービスの構造依存性モデルを示す。更に具体的には、図５は、図１に示したもののような電子商取引システムにおける様々なコンポーネント間の構造アプリケーション依存性の図を示す。

【０１０１】

この構造依存性モデルは、以下の方法で機能的モデル（図４）を拡張する。構造依存性モデルは、ビジネスサービスの実施を処理し、具体的な製品およびそれらの論理的（モジュール、コンポーネント）および物理的（ファイル、共有ライブラリ）アーキテクチャに焦点を当てる。構造依存性モデルは、ソフトウエアコンポーネントの詳細な説明すなわちシステムインベントリを捕捉する。システムインベントリは、通常、様々なシステムレポジトリ、または、例えば被管理リソース２２０の構成ファイルのような、明確に規定された場所に記録されている。

【０１０２】

構造モデルは単一のシステムのコンポーネントを処理するが、これは、他のシステムがホストとなるサービスおよびアプリケーションに対する参照を維持することができることを注記しておく。なぜなら、システム上に位置する構成ファイルがこの情報を含むことができるからである。システムレポジトリの例は、ＩＢＭ ＡＩＸ Ｏｂｊｅｃｔ Ｄａｔａ Ｍａｎａｇｅｒ（ＯＤＭ）、Ｌｉｎｕｘ Ｒｅｄ Ｈａｔ Ｐａｃｋａｇｅ Ｍａｎ

ａｇｅｒ（ＲＰＭ）、またはＭｉｃｒｏｓｏｆｔ Ｗｉｎｄｏｗｓ Ｒｅｇｉｓｔｒｙを含むが、これらに限定されるわけではない。ソフトウエアコンポーネントに関する情報は、通常、ソフトウエアパッケージのインストールおよび展開の間に補足される。加えて、構造モデルは、矢印として示す様々なシステムコンポーネント間の依存性を含む。明確さのため、図５において、ビジネスサービスの名前は引用符を用いずに示すが、構造モデルの要素の名前は引用符を用いて示す。

【０１０３】

十分に承認されたドメイン名ｗｓｌａｂ８．ｗａｔｓｏｎ．ｉｂｍ．ｃｏｍ４００を有するシステムは、以下のコンポーネントのホストとして働く。ｅ－ビジネスアプリケーション（機能モデルにおいて規定されたビジネスサービス）。これは、ｓｔｏｒｅｆｒｏｎｔｓｅｒｖｌｅｔｓ４１０として実施され、後者はアプリケーションのビジネスロジックを内包する。ウエブアプリケーションサービスは、ＩＢＭ ＷｅｂＳｐｈｅｒｅバージョン３．５ ４１５によって実現され、ウエウサービスはＩＢＭ ＨＴＴＰ Ｓｅｒｖｅｒバージョン１．３．６ ４２０によって実現される。ＩＰサービスは、デフォルトのＩＰプロトコルスタック４３０によって実施され、オペレーティングシステム（ＯＳ）は、Ｗｉｎ（ｄｏｗｓ）ＮＴバージョン４４２５である。

【０１０４】

十分に承認されたドメイン名ｒｓｌａｂ２．ｗａｔｓｏｎ．ｉｂｍ．ｃｏｍ４０５を有するシステムは、以下のコンポーネントのホストとして働く。（ＩＢＭ）ＤＢ２ Ｕｎｉｖｅｒｓａｌ Ｄａｔａｂａｓｅ（ＵＤＢ）バージョン５．２ ４３５によって実施されるデータベースサービス、および、ここでは（ＩＢＭ）Ａｄｖａｎｃｅｄ Ｉｎｔｅｒａｃｔｉｖｅ Ｅｘｃｅｃｕｔｉｖｅ（ＡＩＸ）バージョン４．３．３ ４４０であるオペレーティングシステム。

【０１０５】

ここで図６を参照すると、ブロック図は、本発明の一実施形態による、機能、構造、および動作依存性モデルによって記述されるサービスライフサイクルを示す。更に具体的には、図６は、上述した機能モデル５００と構造モデル５１０との間の関係を示し、第３の依存性モデルすなわち動作モデル５２０を導入する。これらの３つのモデルによって、本発明は、それらの全ライフサイクルの間、すなわち、インストール段階から設置および展開段階、更に動作または実行時段階まで、サービスを追跡することができる。

【０１０６】

先に説明したように、機能モデル５００は、ビジネスサービスの設計に関し、従って、ビジネスシステムの設計時に捕捉される。いったん、機能モデル５００が記述するシステムが、インストールまたは展開されると（ステップ５０５）、構造モデル５１０が確立される。構造モデル５１０の様々なコンポーネントがインスタンス化され（ステップ５１５）、それらの間の実行時の結合が確立されると、動作モデル５２０が生成される。動作モデルは、実行時に、先に説明したモデルの特徴を提示する。ここで、上述の概念を示すいくつかの状況を説明する。

【０１０７】

ウエブアプリケーションサーバ３０５は、ＩＢＭ ＷｅｂＳｐｈｅｒｅ４１５によって実施される。後者の１つ以上のインスタンスを、ｗｅｂｓｐｈｅｒｅ－ｄａｅｍｏｎ５４５と呼ぶ。ここで、ウエブ（またはWWW）サービス３１０は、２つの製品、すなわちＡｐａｃｈｅ１．３．４ ５２５およびＬｏｔｕｓ Ｄｏｍｉｎｏ５３０によって実施される。これらの製品の実行中のインスタンスは、ｈｔｔｐ ｄａｅｍｏｎｓ「ｈｔｔｐｄ」５５０として識別することができる。データベースサービス３３０は、２つの製品、すなわち、Ｏｒａｃｌｅ ｖ７ ５３５およびＤＢ２ ＵＤＢ４３５によって実施される。しかしながら、動作モデル５２０ではサーバプロセスは見えないので、Ｏｒａｃｌｅ ｖ７ ５３５のインスタンスはアクティブではない。これに対して、ＤＢ２ ＵＤＢ４３５の４つのインスタンスは実行中である。このことは、動作モデル５２０における４つのＤＢ２ ｄａｅｍｏｎ「ｄｂ２ｄ」５５５の存在からわかる。名前サーバ３１５は、ＢＩＮＤ

バージョン５．６　５４０によって実施される。ＢＩＮＤの実行中のインスタンスは、動作モデル５２０において、「ｎａｍｅｄ」５６０として観察することができる。

【０１０８】

依存性は、機能モデルから構造モデルおよび動作モデルへと伝搬することを注記しておく。これが必要である理由は、実行中のアプリケーションインスタンスからは、適正に機能するために、他のアプリケーションインスタンス（複数のインスタンス）のどれが必要であるかを判定することができないからである。

【０１０９】

一部のアプリケーションインスタンスは短命であるので、動作モデル５２０は高度に動的であり、潜在的に極めて大きい。機能および構造依存性モデルとは対照的に、動作モデル５２０は、レポジトリにもデータベースにも格納されないが、オンデマンドで、必要な程度に、計算される。

【０１１０】

ここで図７を参照すると、ブロック図は、本発明の一実施形態による、機能、構造、および動作依存性モデル間の関係を示す。更に具体的には、図７は、３つの依存性モデルに用いるデータテンプレートの詳細および、これらのモデルを結び付けるための手段を一例として示す。この例は、ライフサイクル中に名前サービスを記述するためのテンプレートおよびその関連する値を詳述する。

【０１１１】

機能モデル５００に用いられる機能テンプレート６０５は、「ｈｏｓｔＮａｍｅ」（サービスのホストとなるコンピュータシステムの一意の名前）、「ｓｅｒｖｉｃｅＮａｍｅ」（サービスの名前）、および「ｃｏｍｐｏｎｅｎｔＴｙｐｅ」「このサービスが担う役割、すなわちクライアントまたはサーバ」を含む。この情報によって、分散型の環境内で、サービスを一意に識別することができる。しかしながら、本発明の精神から逸脱することなく、記述データを含む更に別のフィールドを追加することも可能である（サービスの目的の記述、このサービスに加入しているカスタマ等）。最後に、「Ａｎｔｅｃｅｄｅｎｔ」フィールドは、適正に機能するためにこのサービスが必要とするサービス（複数のサービス）を含む。

【０１１２】

構造モデル５１０に用いられる構造テンプレート６１０は、機能テンプレート６０５のフィールドを全て含み、これによって、機能モデル５００から構造モデル５１０への移動、またはその逆の移動を行うために、機能テンプレート６０５を構造テンプレート６１０にリンクさせることができる。更に、構造テンプレート６１０は、「ｃｏｍｐｏｎｅｎｔＮａｍｅ」（プロダクトコンポーネントの名前）、「ｉｄｅｎｔｉｆｉｅｒ」（コンポーネントを識別するための全体に渡って一意の名前）、「ｖｅｒｓｉｏｎ」、「ｒｅｌｅａｓｅ」、および「ｍｏｄｉｆｉｃａｔｉｏｎ」（例えば保守またはパッチ／修理レベル）の数、「ｉｎｓｔａｌｌＳｔａｔｅ」（コンポーネントのインストールがうまく完了したか否かを示す）、および「ｐｒｏｃｅｓｓＮａｍｅ」（このプロダクトコンポーネントを実行時に識別するプロセス（複数のプロセス）の名前）を含む。更に、「Ａｎｔｅｃｅｄｅｎｔ」フィールドは、このコンポーネントが動作可能となるために必要なコンポーネント（複数のコンポーネント）を列挙する。

【０１１３】

動作モデル５２０に用いられる動作テンプレート６１５は、フィールド「ｈｏｓｔＮａｍｅ」（サービスのホストとなるコンピュータシステムの一意の名前）、「ｐｒｏｃｅｓｓＮａｍｅ」（プロダクトコンポーネントを実行時に識別するプロセス（複数のプロセス）の名前）を含む。これらの２つのフィールドは、構造モデル５１０から動作モデル５２０への移動、またはその逆の移動を行うために、構造テンプレート６１０を動作テンプレート６１５にリンクする。更に、動作テンプレート６１５は、フィールド「ｏｐｅｒＳｔａｔｅ」（プロセスの動作的状態、すなわち実行中、割り込みを受けた、ゾンビ等）、「ｐｏｒｔＮｕｍｂｅｒ」（アプリケーションをプロセスに接続することができるＴＣＰ／

ＵＤＰポートの数）、および「ｉｎｓｔａｎｃｅＩＤ」（コンピュータシステムの範囲内で様々なアプリケーションインスタンスを識別する）を含む。

【０１１４】

３つの依存性モデルは、異なる場所で格納および計算されて、最大の効率を達成する。機能モデル５００は、管理システム６２０、すなわち、管理者１００が分散型環境と対話する制御の中心点に集められて格納される。この選択の理由のいくつかは以下の通りである。図４および６の記載に見られるように、機能モデル５００は、可能なビジネスサービスの量が限られているので、きわめてコンパクトである。更に、機能モデルは、過度に頻繁な変更を受けない。機能モデルは、ビジネスサービスをカスタマに供給した時に規定され、このサービス提供期間が終了するまで不変のままである。管理者１００は機能モデル５００のセットアップおよび更新を担うので、これを管理システム６２０の近くに保持することは当然の選択である。

【０１１５】

図５および６の記載に見られるように、構造モデル５１０は、対照的に、ソフトウエアコンポーネントの詳細な記述すなわちシステムインベントリを捕捉する。システムインベントリは、通常、様々なシステムレポジトリ、または、例えば被管理リソース２２０の構成ファイルのような明確に規定された場所に記録されている。この結果、これはサイズが大きく（システムレポジトリの内容は、数百キロバイトから数メガバイトの間になる傾向がある）、更に、頻繁に変更される。従って、システムの構造モデル５１０を被管理リソース２２０に保持すること自体が、モデルを更新するための通信オーバーヘッドを排除し、更に、全ての被管理リソース（２２０）の構造モデル５１０が１つの場所に集中化して格納されている場合に生じる恐れのある大量の記憶の必要も無くなる。

【０１１６】

動作モデル５２０は、図６において、極めて動的であり、非常に大きいものとして記述した。なぜなら、これは、分散型環境のコンピュータシステム上に存在する全てのアプリケーションの潜在的に多数のインスタンスおよびそれらの間の依存性関係を扱うからである。インターネット／アプリケーション／記憶サービス提供業者および外部委託者の現在のデータセンタが、数千のコンピュータシステムから成り、各々が１００近くのアプリケーションおよびシステムサービスのホストとして働くと想定すれば、現在インスタンス化されているアプリケーションおよびそれらの依存性の全てを１つの動作モデルが含むことは、非現実的である場合がある。このため、現実的な手法は、オンデマンドで動作モデルの関連部分を計算することである（ステップ６２５）。これが依存性サービス２４５の目的である。

【０１１７】

ここで図８を参照すると、ブロック図は、本発明の一実施形態による、終端間依存性の計算に関与するコンポーネントを示す。更に具体的には、図８は、終端間依存性を問い合わせて計算するために用いられる様々なコンポーネント間のデータフローを示す。被管理リソース２２０は、それらのシステムインベントリ、構成ファイル、およびそれらの様々な依存性のＸＭＬ（Ｅｘｔｅｎｓｉｂｌｅ　Ｍａｒｋｕｐ　Ｌａｎｇｕａｇｅ）記述を提供可能であると仮定する。しかしながら、本発明に従って、いかなる記述フォーマットも使用可能であることを注記しておく。この情報をどのように取得するかについての詳細は、以下の通りである。

【０１１８】

１つの直接的な方法は、システムおよびそのアプリケーションおよびサービス内に適切な手段を提供することである。この情報は、フラットＸＭＬファイル７４０において記述され、ウエブサーバ７２５を介してシステムの他のコンポーネントに利用可能である。

【０１１９】

あるいは、依存性サービス２４５は、適切なサービス依存性情報を発生するために、システムレポジトリ７４５に格納された情報を利用する。この情報は、ウエブサーバ７３０を介してシステムの他のコンポーネントに利用可能である。

【０１２０】
第３に、被管理リソース２２０は、ＣＩＭプロバイダ７５０と呼ばれる手段エージェントによって、それらの情報を提示する。これは、Ｄｉｓｔｒｉｂｕｔｅｄ Ｍａｎａｇｅｍｅｎｔ Ｔａｓｋ Ｆｏｒｃｅ（ＤＭＩＴＦ）によって提案されたように、ＣＩＭ Ｏｂｊｅｃｔ Ｍａｎａｇｅｒ（ＣＩＭＯＭ）７３５と対話する。次いで、ＣＩＭＯＭは、必要な情報を、対象のコンポーネントに提示する。

【０１２１】
図８の中央に、サービスサーバ２０５の一部である様々な管理サービスが示されている。これらは、名前サービス７００、取引業者サービス７１０、イベントサービス７１５、および依存性サービス２４５である。依存性サービス２４５は、その管理システム、または通信プロトコル（例えばＪａｖａ Ｒｅｍｏｔｅ Ｍｅｔｈｏｄ Ｉｎｖｏｃａｔｉｏｎ（ＲＭＩ））を用いてアプリケーション層２００に位置するいずれかの管理アプリケーションを介して管理者１００の問い合わせによってトリガされ、それらを処理し、結果を影響分析器２７０に送り返す。影響分析器２７０は、次いで、更に処理するために、その結果を影響シミュレータ２７５に転送する。最後に、その結果は、管理者１００に送出される。

【０１２２】
依存性サービス２４５の主なタスクは、以下の通りである。

【０１２３】
（ａ）管理システムまたはアプリケーション層２００に位置するいずれかの管理アプリケーションとの対話。管理システムは、依存性サービス２４５のアプリケーションプログラミングインタフェース（ＡＰＩ）に問い合わせを発する。

【０１２４】
（ｂ）「ドリルダウン」方法を提示し、サービスの識別子を受け取ると、以下を戻す。
（ｉ）その直接の先行者の記述、すなわちサービスを表すノードの下の第１のレベル、または
（ｉｉ）サービスを表すノードの下のサブグラフ全体、
（ｉｉｉ）依存性グラフの任意のサブセット（所与のノードの下のレベルｍからｎ）。

【０１２５】
（ｃ）サービスの依存性を対象として、同じ機能によって「ドリルアップ」方法を提供する。

【０１２６】
（ｄ）非管理オブジェクトのクラスおよび特性のための情報を収集しフィルタに通すための追加の方法がある。

【０１２７】
（ｅ）ｈｔｔｐ（ＨｙｐｅｒＴｅｘｔ Ｔｒａｎｓｐｏｒｔ Ｐｒｏｔｏｃｏｌ）上で問い合わせを発し、それに（管理者１００が規定したような）フィルタリングルールを適用することによって、被管理リソース２２０から依存性情報を取得する。

【０１２８】
（ｆ）情報をデータ構造に組み合わせて、これをＸＭＬドキュメントとして管理システムに送り返す。

【０１２９】
上述のように、完全に分散型の性質を有するため、本発明は、全ての関与するシステムに対する負荷をできる限り小さく抑えることを目指している。本発明は、被管理リソース２２０から管理システムを分離させ、時間のかかるフィルタおよび連結動作を依存性サービス２４５に内包し、これを様々なシステム上で再現することができる。従って、問い合わせ動作のための最高レベルの並列処理を達成することができる。なぜなら、依存性サービス２４５のインスタンスの選択は、管理システムによって柔軟に実行可能だからである。

【０１３０】
別の重要な利点は、（極めて大きく高度に動的な）動作モデル５２０は、特定の場所に格納されるのではなく、オンデマンドで段階的に計算されるということである。構造モデル５１０の異なる部分は、被管理リソース２２０に格納される。従って、管理システムは、常に最も新しい情報を受信するが、なお、綿密なキャッシュ政策に従ってそれを自由に格納することができる。
【０１３１】
ここで図９を参照すると、ブロック図は、本発明の一実施形態による影響シミュレータのコンポーネントを例示する。図示のように、影響シミュレーションプロセス全体のフローコーディネータとして機能する影響相関器８３０は、管理者１００からサービス問題報告８４０を受信する。サービス問題報告８４０は、その使用不能（または劣化）の影響をシミュレートするサービスの名前およびホストの名前（および、図１５において具体的に示すように、最終的に他のパラメータ）を含む。影響相関器８３０は、（弁理士整理番号第ＳＯＭ９２００２０００４ＵＳ１が付された先に引用した米国特許出願において記載されたような）影響分析器２７０と対話し、次いで、依存性サービス２４５と対話して、対象のサービスに依存する高レベルサービスのリストを取得する。高レベルサービスの例は、電子メールサービス、ウエブサービス等である。
【０１３２】
依存性サービス２４５のタスクは、電子商取引環境が異なる被管理ドメイン８００に及ぶとしても、対象のサービスの依存性を見出すことである。多数のドメインを扱うために、依存性サービス２４５の様々な（カスケード接続された）インスタンスが共に機能することができる。図９において、電子商取引環境は、点線の矩形によって示される。通常、かかる環境は１つ以上の被管理ドメインを含み、結局、その各々が、それ自身の依存性データベース８１０および依存性サービス２４５を有する。
【０１３３】
依存性サービス２４５は、依存者の名前および識別子を影響分析器２７０に戻し、次いで影響分析器２７０は、その結果を影響相関器８３０に転送し、更に処理が行われる。この処理には、故障の可能性を計算すること、または、直接依存者のみ報告するか、または故障の可能性に関するある閾値をこえた依存者のみ報告するか等、要求８４０において特定されたパラメータに従って得られた結果をフィルタリングすることが含まれる場合がある。
【０１３４】
要求８４０が、サービスの使用不能によって影響を受ける可能性があるユーザについて要求する場合（図１５におけるＡＰＩの記載を参照のこと）、影響相関器８３０は、どのユーザがいずれかの依存者サービスに加入しているのかを検証する。これは、全ての依存者サービスについて、加入したユーザについてユーザ加入マネージャ８２０に問い合わせることによって行われる。ユーザ加入マネージャ８２０は、この計算のため、ユーザ／サービスデータベース８３５に頼る。ユーザ／サービスデータベース８３５は、登録ユーザと、彼らが加入しているサービスとの間のマッピングを含む。
【０１３５】
要求８４０が、提供するサービスのうち１つの使用不能を生じた可能性のあるホストシステムの名前を要求する場合（図１５におけるＡＰＩの記載を参照のこと）、影響相関器８３０は、直接的な方法でホストの名前を確定する。なぜなら、依存性サービス２４５および影響分析器２７０は、定義上サービスおよびホストの名前の双方を含むユニフォームリソースロケータ（ＵＲＩ）の形態で、サービスの名前を戻すからである。このため、影響相関器８３０は、ホストの名前（これは、全ＵＲＩのプレフィックスである）を抽出し、この特定のホストの名前が、重複を排除するための結果のリストにすでに現れているか否かを検証する。使用不能によって影響を受ける可能性がある依存者サービスを、あり得る最小のサービスおよびリソース集合に絞り込んだ後、影響相関器８３０は、この情報８５０を管理者に戻す。

【０１３６】
ここで図１０を参照すると、フロー図は、本発明の一実施形態による、依存性サービスを呼び出し、動作モデルに対して影響シミュレーションおよび脆弱性分析を実行するためのステップを示す。更に具体的には、図１０は、依存性サービス（例えば依存性サービス２４５）を呼び出し、その結果を収集し、それらに影響シミュレーションおよび脆弱性分析を適用する方法を示す。この方法は、管理者１００または管理アプリケーションのいずれかによって開始する。
【０１３７】
この方法は、ブロック９００において開始し、以下のように進行する。最初に、通常は機能モデルから、ビジネスサービスを選択する（ステップ９０５）。なぜなら、管理者は、分散型システムが提供するビジネスサービスに関心があるからである。ビジネスサービスを選択すると、構造モデルに問い合わせて、ビジネスサービスの提供に関与するホストの選択を提供する。これは、分散型システムの全ホスト上に存在する構造モデルの位置を特定することか、または、（効率の目的のため）管理システムに格納された（周期的に更新される）サービス／ホストルックアップ表に問い合わせることのいずれかによって行うことができる。このルックアップ表は、分散型システムに存在するサービスとホストとの間のマッピングを含む。次いで、管理者は、自由裁量で、ホストを選択する（ステップ９１０）。
【０１３８】
更に、管理者は、問い合わせを作成する（ステップ９１５）。問い合わせパラメータの例は、横断の方向（サービス依存者に向かって、またはその先行者に向かって）、横断の深さ（例えば直接の先行者／依存者のみ、可能な先行者／依存者の全て、すなわち動作モデルの完全な推移的な終了、動作モデルのｍ番目の層とｎ番目の層との間のみ等）、属性の存在またはそれらの値のいずれかに関連するフィルタリング基準を含むが、これらに限定されるわけではない。
【０１３９】
サービスの選択（ステップ９０５）、ホストの選択（ステップ９１０）、および問い合わせを作成するためのオプションのステップの順序をここに定めるという事実によって、（既存の技術の「ホスト中心の」手法に対して）本発明の「サービス中心の」手法が強調される。しかしながら、本発明の精神および範囲から逸脱することなく、ステップ（ステップ９０５、９１０、および９１５）の順序を変更可能であることは、関連する分野の技術を有する者には認められよう。
【０１４０】
かかる変更の例は、ユーザに、（例えばグラフィカルユーザインタフェースによって）任意の順序で選択プロセスの３つのステップを実行する選択肢を与えること、最初にホストを選択して次いで構造モデルに問い合わせることによってそのホスト上に存在するサービスを検索し、これによって選択のためのサービスの可能な候補を限定することである。
【０１４１】
サービスおよびホストを選択し、問い合わせを作成した後、これらのパラメータによって、依存性サービスを呼び出す（ステップ９２０）。呼び出しのモードは、同期（すなわち依存性サービスによって結果が戻されるまで呼び出し側を阻止する）、または非同期（呼び出し側は計算の間に追加のタスクを実行可能である）のいずれかとすることが可能であることを注記しておく。
【０１４２】
依存性サービスは、動作モデルの適切な部分を計算し、呼び出しのモードに応じて、結果を呼び出し側に返送するか、または呼び出し側に、結果が利用可能であることを通知する。次いで、呼び出し側は、結果を収集し、それらに影響シミュレーションおよび脆弱性分析を適用する（ステップ９２５）。この方法は、ブロック９３０において終了する。
【０１４３】
ここで図１１を参照すると、フロー図が、本発明の一実施形態に従って機能依存性モデ

ルを生成および更新するための管理者のタスクを示す。これが必要であるのは、新しい（ビジネス）サービスを展開および提供する場合、または、既存のモデルに変更を行う場合、または、既存の（ビジネス）サービスを提供品から外す場合である。

【０１４４】

この方法は、ブロック１０００において開始し、以下のように進行する。管理者または管理アプリケーションは、新しいビジネスサービスを追加するべきか、または既存のサービスを消去するかを評価する（ステップ１００５）。これが必要でない場合、方法は直接ブロック１０２５に進む。その他の場合、ステップ１０１０において、サービスおよびその記述を、図７に記載した機能モデルのテンプレート６０５に入力（またはテンプレート６０５から除去）する。

【０１４５】

次いで、ステップ１０１５において、サービス依存性すなわちその先行者に関する関係性を、機能モデルのテンプレート６０５に追加（またはテンプレート６０５から除去）する必要がある。消去の場合、サービス依存者からの依存性は、除去されるサービスの先行者を指すように調整する必要があることを注記しておく。これは、先行者の依存性内に結果として起こる重複する記述についてのチェックを必要とする場合がある。最後に、管理システムのレポジトリに、更新した機能モデルを格納する（ステップ１０２０）。この方法は、ブロック１０２５において終了する。

【０１４６】

ここで図１２を参照すると、フロー図は、本発明の一実施形態による、影響シミュレーションおよび脆弱性分析に基づいて分散型環境の脆弱性を排除するための管理上の手順を示す。この方法は、ブロック１１００において開始し、以下のように進行する。

【０１４７】

管理者１００は、特定のホストシステム上の特定のサービス（例えば電子メールサービス）を選択し、入力パラメータの形態で追加の処理命令を提供する（ステップ１１０５）。管理者が指定する基準に従って、依存者サービスについてシステムに問い合わせることで、影響シミュレーションが開始する（ステップ１１１０）。これを実行するには、影響分析器２７０および、影響シミュレータ２７５から依存性サービス２４５を呼び出し、結果を検索する。

【０１４８】

問い合わせの目的に従って、異なる手順を適用する。管理者は、対象のサービスの使用不能によって影響を受ける可能性のあるサービスに関心を持っている（ブロック１１１５）か、エンドユーザに関心を持っている（ブロック１１２０）か、または様々な依存者サービスのホストとして機能するホストシステムに関心を持っている（ブロック１１２５）可能性がある。第１の場合、影響シミュレータは、この特定のサービスの使用不能によって実際に影響を受けるサービスのリストを返送する。依存性の重みに従って使用不能の確率を計算する等、応答について更に別の処理が必要である場合がある（ステップ１１２７）。

【０１４９】

第２の場合、影響シミュレータは、この特定のサービスの使用不能によって実際に影響を受けるユーザのリストを返送する。影響シミュレータはサービスのリストを受信するので、ユーザ／サービスデータベースを検索することで、これらのサービスを、依存者サービスに加入しているユーザにマッピングする。

【０１５０】

第３の場合、影響シミュレータは、この特定のサービスの使用不能によって実際に影響を受けるサービスを提供するホストシステムのリストを返送する。依存性サービス２４５および影響分析器２７０は双方とも、定義上サービスおよびホストの名前の双方を含むＵＲＩを戻すので、影響シミュレータは、直接的にホストの名前を確定する。このため、影響シミュレータ（２７５）は、ホストの名前（これは、全ＵＲＩのプレフィックスである）を抽出する必要があり（ステップ１１３５）、この特定のホストの名前が、重複を排除

するための結果のリストにすでに現れているか否かを結果から検証する（ステップ１１４０）。

【０１５１】

これらのステップを実行した後、管理者が、サービス、ユーザ、またはホストシステムのどれに関心を持っているか否かに応じて、結果を管理者に返送する。管理者は、分散型環境のコンポーネントのマップを、それらの依存性と共に表すグラフィカルユーザインタフェースを用いることができる。この場合、影響シミュレータの結果を用いて、例えば対応するアイコンを特定の色で表示することによって、影響を受けるサービス（またはユーザまたはホストシステム）を強調する（ステップ１１４５）。あるいは（または、グラフ表現に加えて）、管理者は、結果のリストを、以前の影響シミュレーションの結果と比較して、以前の実行の脆弱性がまだ存在するか否かを判定することを選択することも可能である（ステップ１１５０）。

【０１５２】

排除する必要がある脆弱性が存在する場合、管理者は、以降のステップにおいて、補正のための動作（バックアップｅ－メールサーバまたは適切なフェイルオーバ解決策の展開等）を行う。これらの補正動作が成功したか否かを検証するため、管理者は、影響シミュレータを再び実行し（ステップ１１０５に戻る）、その結果を、（ステップ１１４５において）以前の影響シミュレーションの結果と比較する。

【０１５３】

第２の実行によって戻された結果のリストが著しく小さい（または空である）場合、管理者は、その分散型環境から脆弱性の源を除去することに成功している。この方法は、ブロック１１６０において終了する。

【０１５４】

管理者によって、合成または仮想のサービス、ユーザ、およびホストシステム（すなわち実際の分散型環境における「現実の」コンポーネントを有しないコンポーネント）を追加して、実際の「現実の」コンポーネントを展開する必要なく、計画の目的のための堅固な基礎が得られるようにすることで、本発明を拡張することができることを注記しておく。本発明の精神および範囲から逸脱することなく、かかる変更および拡張を実行可能であることは、関連する分野の技術を有する者には認められよう。

【０１５５】

ここで図１３を参照すると、フロー図は、本発明の一実施形態による、動作モデルに対する影響シミュレーションおよび脆弱性分析の実行を示す。この方法はブロック１２００において開始し、以下のように進行する。動作依存性モデルに対するシミュレーションおよび分析を実行するシステムは、常時、システムが実行されるホストの特定のポートにおいて要求がないかと監視している。これは、ブロック１２０５をそれ自身に接続するループによって示される。これは、いつでもアプリケーションによって呼び出すことができる、サービスを実施するサーバプロセス（「ｄａｅｍｏｎｓ」）について標準的な挙動である。

【０１５６】

要求を受けると、システムは、要求から入力パラメータを抽出する（ステップ１２１０）。図１０の記載において述べたように、入力パラメータの例は、対象のサービスおよびホストの名前、横断の方向、横断の深さ、属性の存在またはそれらの値のいずれかに関連したフィルタリング基準を含むが、これらに限定されるわけではない。次いで、これらの入力パラメータを用いて、動作モデルの計算を呼び出す。これは、ステップ１２１５において呼び出される。

【０１５７】

更に、計算の結果、すなわち動作モデルを収集する。次いで、動作モデルに対し、影響シミュレーションおよび脆弱性分析を実行する（ステップ１２２０）。呼び出しの時に規定された呼び出しのモードに従って、影響分析の結果を呼び出しアプリケーションに戻す（ステップ１２２５）。このステップの後、システムが実行しているホストの割り当てら

れたリソースはいずれも解放される（ステップ１２３０）。ホストリソースの例は、メモリ、ディスクスペース、またはＣＰＵレジスタを含むが、これらに限定されるわけではない。最後に、システムはその最初の段階に戻り、続けて入来する要求がないかと監視する（ステップ１２０５）。

【０１５８】

ここで図１４を参照すると、フロー図は、本発明の一実施形態による、サービスの依存者に対する影響シミュレーションおよび脆弱性分析の実行を示す。この方法は、ブロック１３００において開始し、次のように進行する。

【０１５９】

最初に、目的とするサービスおよびホストの名前を、様々な他のユーザ規定パラメータと共に取得する（ステップ１３０５）。これらのパラメータは、呼び出し管理アプリケーションによって提供され、管理者から直接、または管理コンソールに到着するイベントメッセージから取得される。更に、依存者のステータスに従って検索基準を指定する。それらは、問題を生じた（「不良の」）サービスまたは適切に実行しているサービスをシステムが戻すべきかを示す。通常、管理アプリケーションが関心を持つのは、前者すなわち不良のサービスである。更に、検索経路を指定する。管理アプリケーションが関心を持つ可能性があるのは、サービスの直接依存者（検索経路長＝１）、サービスに直接的に（または間接的に）依存する全サービス集合（検索経路＝再帰的）、または依存者の最高レベル、すなわち対象のサービスに依存する最高レベル（エンドユーザ）サービス（検索経路＝最大）のいずれかである。

【０１６０】

次いで、パラメータ「サービスの名前」、「ホストの名前」、「依存者のステータス」、「検索経路」に従って、依存性サービスによって動作モデルの計算を実行する（ステップ１３１０）。次に、ステップ１３１５において、結果、すなわち依存者サービス要素のリスト（「候補者リスト」）を依存性サービスから取得する。

【０１６１】

候補者サービス要素のリストが空になるまで、以下のステップを実行する（１３２０）。

【０１６２】

候補者リストの第１のサービス要素を選択し（ステップ１３２５）、更に計算を行う必要があるか否かを判定する（ステップ１３３０）。更に別の計算は、故障可能性の計算、依存者サービスに加入したユーザの参照、または戻されたＵＲＩからホストの名前を取得することを含む場合がある。この手順のステップは、図９の状況において先に詳細に説明した。更に計算を行う必要がある場合、これを（最初の要求に従って）影響シミュレータによって実行する（ステップ１３３５）。最後に、サービスの名前（および、要求に従った更に別の情報）を結果リストに追加する（ステップ１３３７）。

【０１６３】

しかしながら、更に計算を行う必要がないとわかった場合は、サービスの名前を直接結果リストに追加する（ステップ１３４０）。最後に、入力したサービス要素リストから、対象のサービス要素を除去し（ステップ１３４５）、この方法はステップ１３２０に戻る。サービス要素のリストが空である場合、この方法は直接ステップ１３５０に進み、ここで、実行中のサービス要素のリストを呼び出し側に返すか、または不良のサービス要素のリストを返す。結果内容は、ステップ１３０５において呼び出し側が求めた詳細事項に依存する。この方法は、ブロック１３５５において終了する。

【０１６４】

図１５に、ステータスチェック手順のステップを詳細に記載する。対象のサービス要素が適正に機能している場合（ステータス＝「ＯＫ」）、これを、実行中のサービス要素リストすなわち「ＯＫ」リストに追加する（ステップ１３３５）。しかしながら、サービス要素が問題を生じていることがわかった場合、これを「不良」リストに追加する（ステップ１３４０）。最後に、対象のサービス要素を、候補者サービス要素リストから除去し（

ステップ１３４５）、この方法はステップ１３２０に戻る。

【０１６５】

候補者サービス要素リストが空である場合、この方法は直接ステップ１３５０に進み、実行中のサービス要素リストまたは不良のサービス要素リストのどちらかを呼び出し側に戻す。結果内容は、ステップ１３０５において呼び出し側が実行中または不良のサービスのどちらを求めていたかに依存する。この方法は、ブロック１３５５において終了する。

【０１６６】

ここで図１５を参照すると、本発明の一実施形態による影響シミュレーションアプリケーションプログラミングインタフェース（ＡＰＩ）の例が示されている。このテーブルは、所与のサービスおよびホストの名前について適切な動作モデルの受信を発生、送出、および要求することができるベースＡＰＩを含む。ＡＰＩは、１つ以上のパラメータ（図示せず）を用いて、ＡＰＩが用いる（機能記述において指定される）特性を識別可能であることは、当業者には認められよう。

【０１６７】

具体的には、「ｇｅｔＩｍｐａｃｔｅｄＤｉｒｅｃｔＤｅｐｅｎｄｅｎｔｓ（ｐａｒａｍｅｔｅｒｓ）」ＡＰＩは、特定のホスト上に配置されたサービスの使用不能によって影響を受ける可能性がある直接の依存者（対象のサービスよりも１つ上のレベルに位置するサービス）を検索する。

【０１６８】

「ｇｅｔＩｍｐａｃｔｅｄＤｅｐｅｎｄｅｎｔｓＲｅｃｕｒｓｉｖｅ（ｐａｒａｍｅｔｅｒｓ）」ＡＰＩは、再帰的「ドリルアップ」を実行する、すなわち、このＡＰＩは、特定のホスト上に配置された所与のサービスの依存者を全て検索する。従って、このＡＰＩは、使用不能によって影響を受ける可能性がある、依存性階層内の所与のサービスよりも「上に」ある全てのサービスを戻す。

【０１６９】

「ｇｅｔＩｍｐａｃｔｅｄＨｉｇｈｅｓｔＤｅｐｅｎｄｅｎｔｓ（ｐａｒａｍｅｔｅｒｓ）」ＡＰＩは、使用不能によって影響を受ける可能性のある、特定のホスト上に配置されたサービスの最高の依存者を検索する。このＡＰＩによって、依存者を全く有しないサービスが生じる（すなわち、それらは、所与のサービスの依存性階層の最高層に位置する）。このＡＰＩは、ユーザまたは管理アプリケーションが、「中間」サービスを扱う必要なく、いずれかの「エンドユーザ」サービス（例えば電子メールサービスまたはウエブサービス）
が使用不能によって影響を受ける可能性があるか否かを迅速に判定したい場合に、特に有用である。

【０１７０】

「ｇｅｔＩｍｐａｃｔｅｄＵｓｅｒｓ（ｐａｒａｍｅｔｅｒｓ）」ＡＰＩは、特定のホスト上に配置されたサービスサービスの依存者に加入したユーザの識別子を全て検索する。従って、これらのユーザは、使用不能によって影響を受ける可能性がある。

【０１７１】

「ｇｅｔＩｍｐａｃｔｅｄＨｏｓｔｓ（ｐａｒａｍｅｔｅｒｓ）」ＡＰＩは、特定のホスト上に配置されたサービスの依存者のホストとなるシステムのホストの名前を全て検索する。従って、これらのホストは、１つ以上の提供するサービスの使用不能を経験する可能性がある。

【０１７２】

「ｇｅｔＩｍｐａｃｔｅｄＴｏｐＮＤｅｐｅｎｄｅｎｔｓ（ｐａｒａｍｅｔｅｒｓ）」ＡＰＩは、使用不能によって最も影響を受ける依存者を（依存性の重みに従って）検索する。ｎは、検索されるアイテムの最大数を指定する。このＡＰＩは、全ての依存性に重みが添付されて存在すると仮定し、これが、その先行者が使用不能を経験した場合にこのサービスが故障する可能性を示す。更に、管理者は、戻された結果の最大数を示すことによって、結果をカスタマイズすることができる。

【０１７３】
例えば、管理者は、特定のホスト上に配置された特定のサービスが使用不能に直面した場合に少なくとも７５％の可能性で故障すると考えられる依存者サービスの上位１０個を要求することができる。次いで、ＡＰＩは、故障の可能性とは逆の順序で、最大１０のタプルを有するリスト（サービス識別子およびその故障可能性を含む）を戻す。この可能性は、７５％以上である（例えば、［サービスＡ、９８％］、［サービスＢ、８２％］、［サービスＣ、７９％］、［サービスＤ、７５％］）。依存者の位置（直接、間接、または最高）は、このＡＰＩについて役割を果たしていないことを注記しておく。
【０１７４】
「ｇｅｔＩｍｐａｃｔｅｄＴｏｐＮＨｉｇｈｅｓｔＤｅｐｅｎｄｅｎｔｓ（ｐａｒａｍｅｔｅｒｓ）」ＡＰＩは、使用不能によって最も影響を受ける最高の依存者を（依存性の重みに従って）検索する。ｎは、検索されるアイテムの最大数を指定する。このＡＰＩは、全ての依存性に重みが添付されて存在すると仮定し、これが、その先行者が使用不能を経験した場合にこのサービスが故障する可能性を示す。更に、管理者は、戻された結果の最大数を示すことによって、結果をカスタマイズすることができる。
【０１７５】
例えば、管理者は、特定のホスト上に配置された特定のサービスが使用不能に直面した場合に少なくとも７５％の可能性で故障すると考えられる依存者サービスの上位１０個を要求することができる。このＡＰＩについて、依存者の位置が役割を果たすことを注記しておく。なぜなら、ＡＰＩは、動作依存性モデルにおけるサービスの最高層にのみ焦点を当てている、すなわち、このＡＰＩは、全く依存者を有しないサービスを生じるからである（すなわち、それらは、所与のサービスの依存性階層の最高層に位置する）。このＡＰＩは、ユーザまたは管理アプリケーションが、どの「エンドユーザ」サービス（例えば電子メールサービスまたはウエブサービス）が使用不能によって最も影響を受ける可能性があるか否かを迅速に判定したい場合に、特に有用である。
【０１７６】
本発明の例示的な実施形態について、添付図面を参照して説明したが、本発明は、それらの正確な実施形態に限定されるわけではなく、本発明の範囲および精神から逸脱することなく、当業者によって、様々な他の変更および修正を行い得ることは理解されよう。
【０１７７】
まとめとして、本発明の構成に関して以下の事項を開示する。
【０１７８】
（１）コンピューティング環境に関連した少なくとも１つの対象コンポーネントのシミュレートした状況の潜在的な影響を明らかにするためのコンピュータに基づいた方法であって、
前記少なくとも１つの対象コンポーネントに依存する前記コンピューティング環境における１つ以上のコンポーネントを識別し、前記１つ以上の識別されたコンポーネントの各々に関連した潜在的な状況ステータスを求めるステップと、
対応する状況ステータスに基づいて前記１つ以上の識別されたコンポーネントの少なくとも一部に関して情報をユーザに提示するステップと、
を具備することを特徴とする、方法。
（２）前記コンピューティング環境は分散型コンピューティング環境から成ることを特徴とする、（１）の方法。
（３）前記コンピューティング環境は自律型コンピューティング環境から成ることを特徴とする、（１）の方法。
（４）前記ユーザに提示される前記情報は、前記対応する状況ステータスに基づいて前記１つ以上の識別されたコンポーネントの少なくとも一部に関連した脆弱性の表現を含むことを特徴とする、（１）の方法。
（５）前記表現は、前記１つ以上の識別されたコンポーネントの各々に関連した故障の各可能性を示す脆弱性マップを含むことを特徴とする、（４）の方法。

（６）前記ユーザに提示される前記情報は、潜在的に影響を受けるエンドユーザのリスト、潜在的に影響を受けるコンポーネントのリスト、および潜在的に影響を受けるホストのリストのうち少なくとも１つを含むことを特徴とする、（１）の方法。

（７）前記ユーザに提示される前記情報は、前記少なくとも１つの対象コンポーネントのシミュレートされた状況の潜在的な影響の１度の判定の、前記少なくとも１つの対象コンポーネントのシミュレートされた状況の潜在的な影響の少なくとも１度の他の判定に対する比較を含み、前記ユーザは、前記潜在的な影響を少なくとも最小限に抑えるために、前記１度の判定と前記少なくとも１度の他の判定との間で行われる１つ以上のステップが、実質的に有効であるか否かを検証することができるようになっていることを特徴とする、（１）の方法。

（８）前記少なくとも１つの対象コンポーネントは、前記コンピューティング環境における既存のコンポーネントであることを特徴とする、（１）の方法。

（９）前記少なくとも１つの対象コンポーネントは、前記コンピューティング環境における既存のコンポーネントではないことを特徴とする、（１）の方法。

（１０）前記識別するステップは、更に、前記コンピューティング環境のコンポーネントの少なくとも一部に関連した１つ以上の関係の存在を表すモデルの少なくとも一部を横断することを含み、前記コンピューティング環境の少なくとも１つのコンポーネントに関連した全ライフサイクルを考慮に入れることができることを特徴とする、（１）の方法。

（１１）前記モデルは、機能カテゴリ分類、構造カテゴリ分類、および動作カテゴリ分類から成る形態であることを特徴とする、（１０）の方法。

（１２）前記識別するステップは、更に、少なくとも１つの入力基準に基づいて前記モデルの前記動作カテゴリ分類を計算し、前記動作カテゴリ分類を横断して、前記少なくとも１つの対象コンポーネントに依存する前記コンピューティング環境における前記１つ以上のコンポーネントを識別することを含むことを特徴とする、（１１）の方法。

（１３）前記少なくとも１つの入力基準は、前記少なくとも１つの対象コンポーネントの名前、前記少なくとも１つの対象コンポーネントに関連したホスト、前記横断の方向、前記横断の深さ、前記横断の経路、および少なくとも１つのフィルタリング基準のうち少なくとも１つを含むことを特徴とする、（１２）の方法。

（１４）前記少なくとも１つの対象コンポーネントの前記状況は、故障、非故障、および劣化のうち１つであることを特徴とする、（１）の方法。

（１５）コンポーネントは、サービス、アプリケーション、ミドルウエア、ハードウエア、デバイスドライバ、オペレーティングシステム、および前記コンピューティング環境に関連したシステムのうち１つであることを特徴とする、（１）の方法。

（１６）コンピューティング環境に関連した少なくとも１つの対象コンポーネントのシミュレートした状況の潜在的な影響を明らかにするための装置であって、

少なくとも１つのプロセッサであって、（ｉ）前記少なくとも１つの対象コンポーネントに依存する前記コンピューティング環境における１つ以上のコンポーネントを識別し、前記１つ以上の識別されたコンポーネントの各々に関連した潜在的な状況ステータスを求める、および（ｉｉ）対応する状況ステータスに基づいて前記１つ以上の識別されたコンポーネントの少なくとも一部に関して情報をユーザに提示させる、ように動作するプロセッサと、

前記少なくとも１つのプロセッサに結合され、前記識別および提示動作に関連した結果の少なくとも一部を格納するように動作するメモリと、

を具備することを特徴とする、装置。

（１７）前記コンピューティング環境は分散型コンピューティング環境から成ることを特徴とする、（１６）の装置。

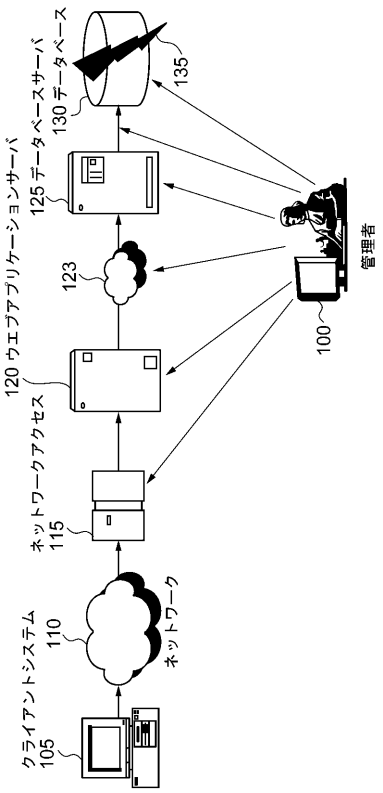
（１８）前記コンピューティング環境は自律型コンピューティング環境から成ることを特徴とする、（１６）の装置。

（１９）前記ユーザに提示される前記情報は、前記対応する状況ステータスに基づいて前記１つ以上の識別されたコンポーネントの少なくとも一部に関連した脆弱性の表現を含

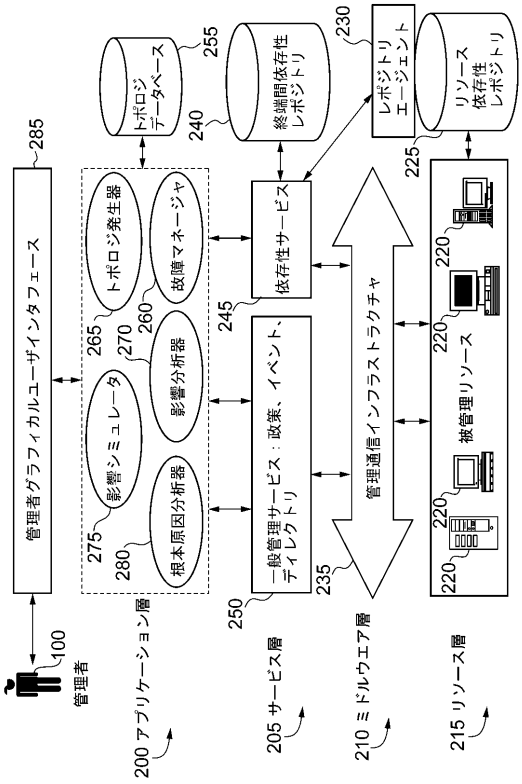
むことを特徴とする、（１６）の装置。
（２０）前記表現は、前記１つ以上の識別されたコンポーネントの各々に関連した故障の各可能性を示す脆弱性マップを含むことを特徴とする、（１９）の装置。
（２１）前記ユーザに提示される前記情報は、潜在的に影響を受けるエンドユーザのリスト、潜在的に影響を受けるコンポーネントのリスト、および潜在的に影響を受けるホストのリストのうち少なくとも１つを含むことを特徴とする、（１６）の装置。
（２２）前記ユーザに提示される前記情報は、前記少なくとも１つの対象コンポーネントのシミュレートされた状況の潜在的な影響の１度の判定の、前記少なくとも１つの対象コンポーネントのシミュレートされた状況の潜在的な影響の少なくとも１度の他の判定に対する比較を含み、前記ユーザは、前記潜在的な影響を少なくとも最小限に抑えるために、前記１度の判定と前記少なくとも１度の他の判定との間で行われる１つ以上のステップが、実質的に有効であるか否かを検証することができるようになっていることを特徴とする、（１６）の装置。
（２３）前記少なくとも１つの対象コンポーネントは、前記コンピューティング環境における既存のコンポーネントであることを特徴とする、（１６）の装置。
（２４）前記少なくとも１つの対象コンポーネントは、前記コンピューティング環境における既存のコンポーネントではないことを特徴とする、（１６）の装置。
（２５）前記識別動作は、更に、前記コンピューティング環境のコンポーネントの少なくとも一部に関連した１つ以上の関係の存在を表すモデルの少なくとも一部を横断することを含み、前記コンピューティング環境の少なくとも１つのコンポーネントに関連した全ライフサイクルを考慮に入れることができることを特徴とする、（１６）の装置。
（２６）前記モデルは、機能カテゴリ分類、構造カテゴリ分類、および動作カテゴリ分類から成る形態であることを特徴とする、（２５）の装置。
（２７）前記識別動作は、更に、少なくとも１つの入力基準に基づいて前記モデルの前記動作カテゴリ分類を計算し、前記動作カテゴリ分類を横断して、前記少なくとも１つの対象コンポーネントに依存する前記コンピューティング環境における前記１つ以上のコンポーネントを識別することを含むことを特徴とする、（２６）の装置。
（２８）前記少なくとも１つの入力基準は、前記少なくとも１つの対象コンポーネントの名前、前記少なくとも１つの対象コンポーネントに関連したホスト、前記横断の方向、前記横断の深さ、前記横断の経路、および少なくとも１つのフィルタリング基準のうち少なくとも１つを含むことを特徴とする、（２７）の装置。
（２９）前記少なくとも１つの対象コンポーネントの前記状況は、故障、非故障、および劣化のうち１つであることを特徴とする、（１６）の装置。
（３０）コンピューティング環境に関連した少なくとも１つの対象コンポーネントのシミュレートした状況の潜在的な影響を明らかにするための製造の物品であって、１つ以上のプログラムを含む機械読み取り可能媒体を具備し、前記プログラムは、実行された場合、
前記少なくとも１つの対象コンポーネントに依存する前記コンピューティング環境における１つ以上のコンポーネントを識別し、前記１つ以上の識別されたコンポーネントの各々に関連した潜在的な状況ステータスを求めるステップと、
対応する状況ステータスに基づいて前記１つ以上の識別されたコンポーネントの少なくとも一部に関して情報をユーザに提示するステップと、
を実施することを特徴とする、物品。
【図面の簡単な説明】
【０１７９】
【図１】本発明の機構と相互作用して情報を生成することができるクライアントーサーバアプリケーションアーキテクチャの一例を示すブロック図である。
【図２】本発明の一実施形態による、依存性管理を行うためのシステムを例示するブロック図である。
【図３】本発明の一実施形態による、依存性管理を提供するためのシステムを実施するの

に適切なコンピュータシステムの一般化ハードウエアアーキテクチャを例示するブロック図である。
【図４】本発明の一実施形態によるサービスの機能依存性モデルを例示するブロック図である。
【図５】本発明の一実施形態によるサービスの構造依存性モデルを例示するブロック図である。
【図６】本発明の一実施形態による、機能、構造、および動作依存性モデルによって記述されるサービスライフサイクルを例示するブロック図である。
【図７】本発明の一実施形態による、機能、構造、および動作依存性モデル間の関係を例示するブロック図である。
【図８】本発明の一実施形態による、サービスの使用不能の潜在的な影響を分析し計算する際に関与するコンポーネントを例示するブロック図である。
【図９】本発明の一実施形態による、影響シミュレータおよび脆弱性分析器のコンポーネントを例示するブロック図である。
【図１０】本発明の一実施形態による、依存性サービスを呼び出すと共に動作モデルに対して影響シミュレーションおよび脆弱性分析を実行するためのステップを例示するフロー図である。
【図１１】本発明の一実施形態による、機能依存性モデルを生成し更新するための管理者のタスクを例示するフロー図である。
【図１２】本発明の一実施形態による、影響シミュレーションおよび脆弱性分析に基づいて分散型コンピューティング環境の脆弱性を排除するための管理上の手順を示すフロー図である。
【図１３】本発明の一実施形態による、動作モデルに対する影響シミュレーションおよび脆弱性分析の実行を示すフロー図である。
【図１４】本発明の一実施形態による、サービスの依存者に対する影響シミュレーションおよび脆弱性分析の実行を示すフロー図である。
【図１５】本発明の一実施形態による、影響シミュレーションおよび脆弱性分析のアプリケーションプログラミングインタフェースの例を示す。

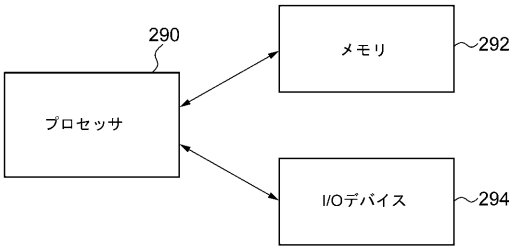
【図１】


135
130 データベースサーバ
125 データベースサーバ
123
管理者
100
120 ウエブアプリケーションサーバ
ネットワークアクセス
115
ネットワーク
110
クライアントシステム
105

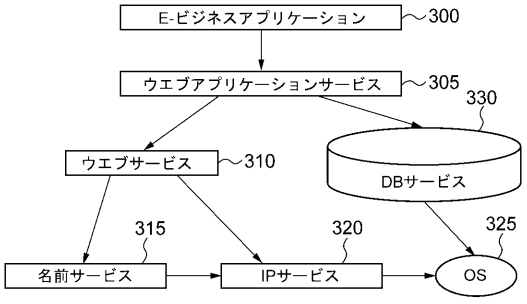
【図２】


トポロジデータベース
255
終端間依存性レポジトリ
240
レポジトリエージェント
230
リソース依存性レポジトリ
225
285
トポロジ発生器
265
故障マネージャ
260
影響分析器
270
影響シミュレータ
275
根本原因分析器
280
依存性サービス
245
一般管理サービス：政策、イベント、ディレクトリ
250
管理通信インフラストラクチャ
235
被管理リソース
220
管理者グラフィカルユーザインタフェース
100
管理者
200 アプリケーション層
205 サービス層
210 ミドルウエア層
215 リソース層

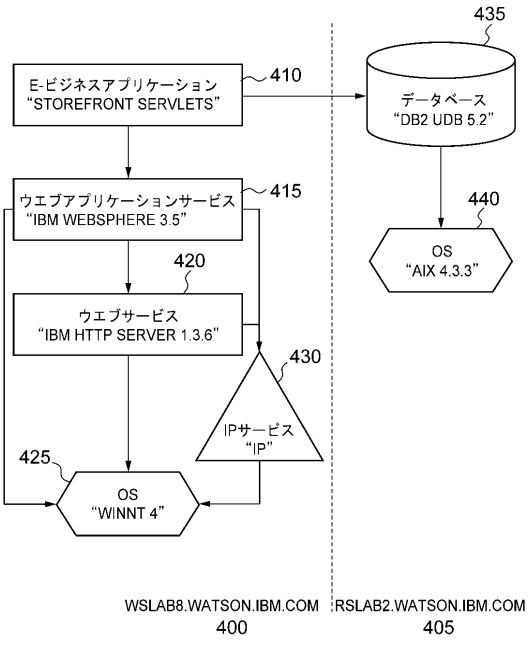
【図３】


290
プロセッサ
メモリ
292
I/Oデバイス
294

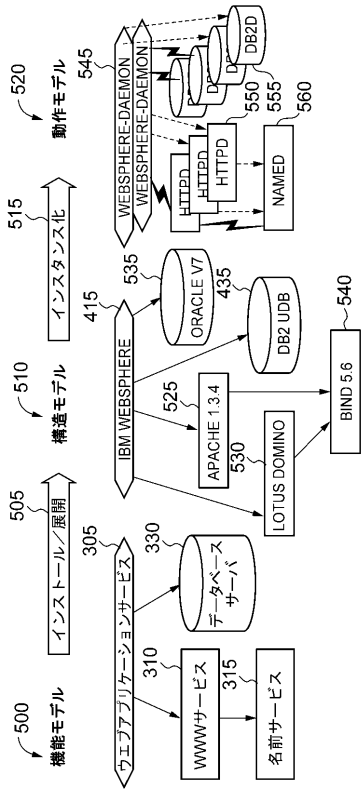
【図４】


E-ビジネスアプリケーション
300
ウエブアプリケーションサービス
305
330
ウエブサービス
310
DBサービス
315
320
325
名前サービス
IPサービス
OS

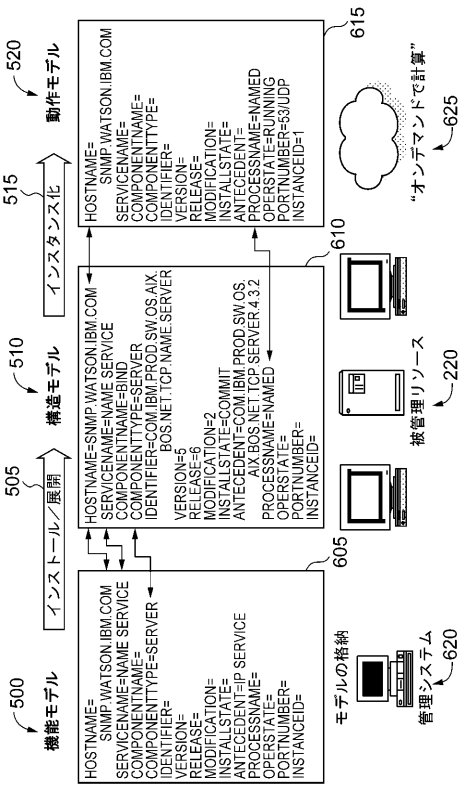
【図５】


435
E-ビジネスアプリケーション
“STOREFRONT SERVLETS”
410
データベース
“DB2 UDB 5.2”
ウエブアプリケーションサービス
“IBM WEBSPHERE 3.5”
415
440
OS
“AIX 4.3.3”
420
ウエブサービス
“IBM HTTP SERVER 1.3.6”
430
IPサービス
“IP”
425
OS
“WINNT 4”
WSLAB8.WATSON.IBM.COM
400
RSLAB2.WATSON.IBM.COM
405

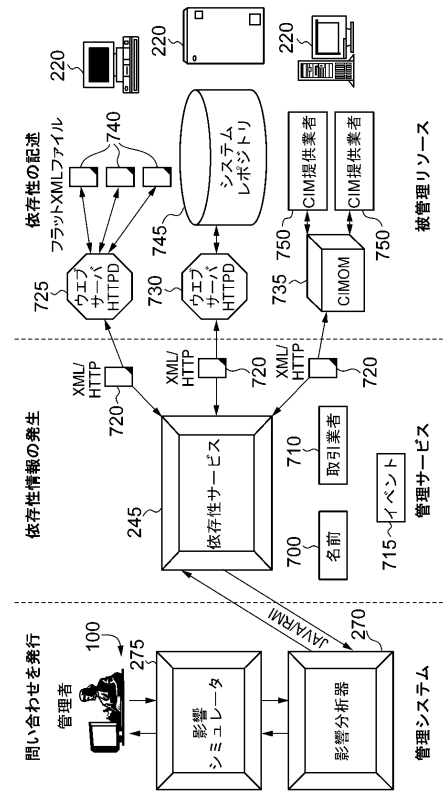
【図６】


500
機能モデル
505
インストール／展開
510
構造モデル
515
インスタンス化
520
動作モデル
ウェブアプリケーションサービス
305
310
WWWサービス
315
名前サービス
330
データベースサーバ
IBM WEBSPHERE
415
APACHE 1.3.4
525
530
LOTUS DOMINO
ORACLE V7
535
435
DB2 UDB
BIND 5.6
540
WEBSPHERE-DAEMON
545
HTTPD
550
DB2D
555
NAMED
560

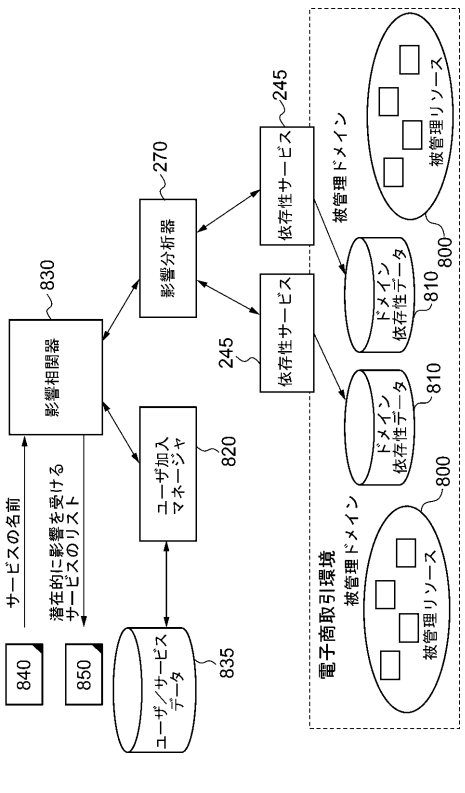
【図７】


500
機能モデル
505
インストール／展開
510
構造モデル
515
インスタンス化
520
動作モデル
HOSTNAME=
SNMP.WATSON.IBM.COM
SERVICENAME=NAME SERVICE
COMPONENTNAME=
COMPONENTTYPE=SERVER
IDENTIFIER=
VERSION=
RELEASE=
MODIFICATION=
INSTALLSTATE=
ANTECEDENT=IP SERVICE
PROCESSNAME=
OPERSTATE=
PORTNUMBER=
INSTANCEID=
HOSTNAME=SNMP.WATSON.IBM.COM
SERVICENAME=NAME SERVICE
COMPONENTNAME=BIND
COMPONENTTYPE=SERVER
IDENTIFIER=COM.IBM.PROD.SW.OS.AIX.
BOS.NET.TCP.NAME.SERVER
VERSION=5
RELEASE=6
MODIFICATION=2
INSTALLSTATE=COMMIT
ANTECEDENT=COM.IBM.PROD.SW.OS.
AIX.BOS.NET.TCP.SERVER.4.3.2
PROCESSNAME=NAMED
OPERSTATE=
PORTNUMBER=
INSTANCEID=
HOSTNAME=
SNMP.WATSON.IBM.COM
SERVICENAME=
COMPONENTNAME=
COMPONENTTYPE=
IDENTIFIER=
VERSION=
RELEASE=
MODIFICATION=
INSTALLSTATE=
ANTECEDENT=
PROCESSNAME=NAMED
OPERSTATE=RUNNING
PORTNUMBER=53/UDP
INSTANCEID=1
605
610
615
モデルの格納
管理システム
620
被管理リソース
220
“オンデマンドで計算”
625

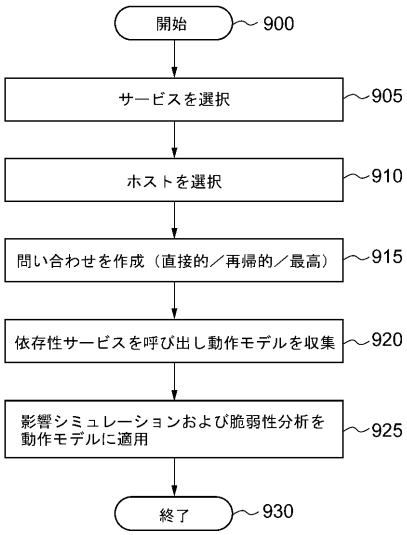
【図８】


問い合わせを発行
管理者
100
275
影響シミュレータ
JAVA/RMI
270
影響分析器
管理システム
依存性情報の発生
XML/HTTP
720
依存性サービス
245
700
名前
710
取引業者
715
イベント
管理サービス
依存性の記述
フラットXMLファイル
740
ウエブサーバHTTPD
725
730
745
システムレポジトリ
735
CIMOM
750
CIM提供業者
被管理リソース
220

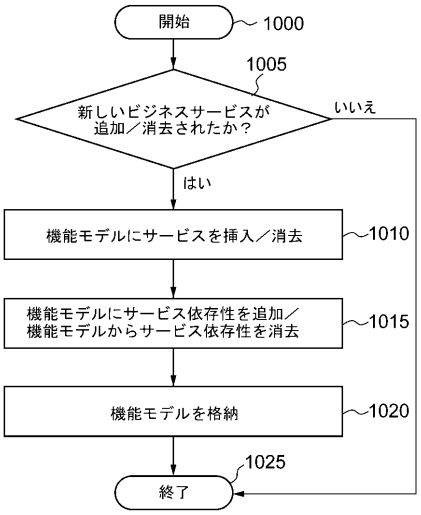
【図９】


サービスの名前
840
潜在的に影響を受けるサービスのリスト
850
影響相関器
830
ユーザ／サービスデータ
835
ユーザ加入マネージャ
820
影響分析器
270
依存性サービス
245
ドメイン依存性データ
810
電子商取引環境
被管理ドメイン
800
被管理リソース

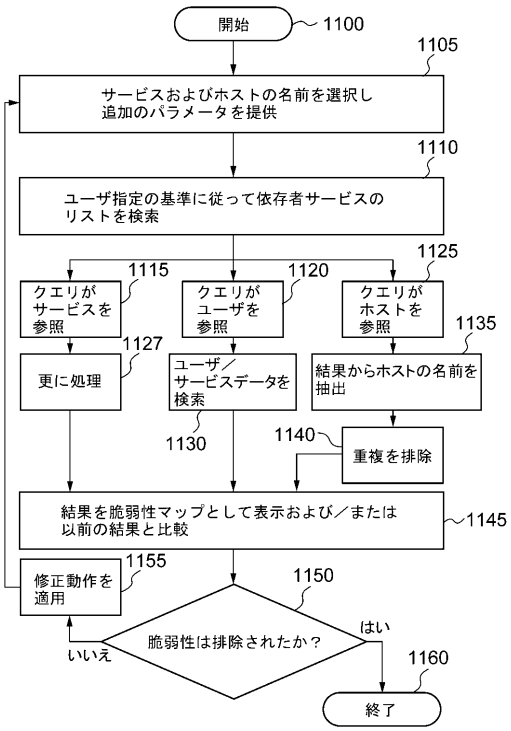
【図１０】



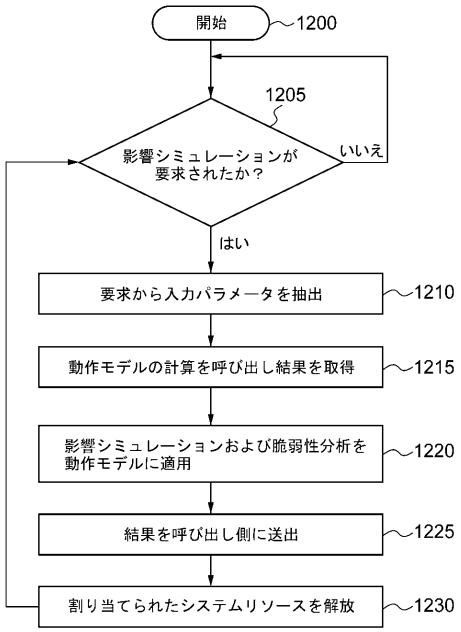
【図１１】



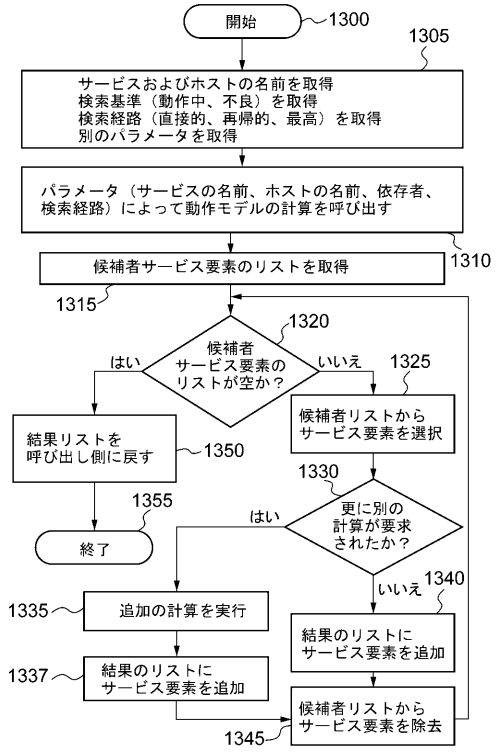
【図１２】



【図１３】



【図１４】



【図１５】

APIS	機能記述
GETIMPACTDIRECTDEPENDENTS( )	特定のホスト上に配置されたサービスの直接先行者を検索
GETIMPACTHIGHESTDEPENDENTS( )	特定のホスト上に配置されたサービスの最高依存者を検索
GETIMPACTDEPENDENTSRECURSIVE( )	再帰的ドリルアップ；特定のホスト上に配置されたサービスの依存者を全て検索
GETIMPACTEDUSERS( )	特定のホスト上に配置されたサービスの依存者に加入したユーザIDを全て検索
GETIMPACTEDHOSTS( )	特定のシステム上に配置されたサービスの依存者のホストとなるホストの名前を全て検索
GETIMPACTEDTOPNDEPENDENTS( )	最も影響を受ける依存者を（依存性の重みに従って）検索；Nは検索されるアイテムの最大数を指定する
GETIMPACTEDTOPNHIGHESTDEPENDENTS( )	最も影響を受ける最高依存者を（依存性の重みに従って）検索；Nは検索されるアイテムの最大数を指定する

フロントページの続き

(72)発明者　アレクサンダー・ケラー
アメリカ合衆国１００１１　ニューヨーク州ニューヨーク　ウエスト・トゥエンティセカンド・ストリート　３０２　アパートメント　４８

合議体
審判長　山崎　達也
審判官　宮司　卓佳
審判官　冨吉　伸弥

(56)参考文献　丸山勝久，”ネットワークに潜在する依存関係のグラフ表現とその応用”，　第５６回（平成１０年前期）全国大会講演論文集（３），社団法人情報処理学会，１９９８年3月１９日，ｐｐ．６６３－６６４
梅原系，”ａｄｍｉｎｉｓｔｒａｔｏｒ管理用ツールを使いこなす”，Ｗｉｎｄｏｗｓ２０００ｍａｇａｚｉｎｅ，株式会社アスキー，２０００年１２月３１日，第3号，ｐｐ．１２４－１３１

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F　9/44
H04L 12/24