(19)日本国特許庁(JP)　(12)**公表特許公報(A)**　(11)特許出願公表番号
特表2009-516239
(P2009-516239A)
(43)公表日　**平成21年4月16日(2009.4.16)**

(51)Int.Cl.		FI			テーマコード（参考）
G06F 11/28	***(2006.01)***	G06F	11/28	310E	5B042
G06F 11/34	***(2006.01)***	G06F	11/34	L	

審査請求 未請求　予備審査請求 未請求　（全 89 頁）

(21)出願番号	特願2008-535693 (P2008-535693)
(86)(22)出願日	平成18年10月10日 (2006.10.10)
(85)翻訳文提出日	平成20年6月9日 (2008.6.9)
(86)国際出願番号	PCT/US2006/039998
(87)国際公開番号	WO2007/044875
(87)国際公開日	平成19年4月19日 (2007.4.19)
(31)優先権主張番号	11/248,981
(32)優先日	平成17年10月11日 (2005.10.11)
(33)優先権主張国	米国 (US)

(71)出願人　508108660
ケーエヌオーエー　ソフトウエア，インク
．
アメリカ合衆国，ニューヨーク州　１００
０３，ニューヨーク，ユニオン　スクエア
　ウエスト　５
(74)代理人　100080089
弁理士　牛木　護
(74)代理人　100137800
弁理士　吉田　正義
(74)代理人　100148253
弁理士　今枝　弘充
(74)代理人　100148079
弁理士　梅村　裕明

最終頁に続く

(54)【発明の名称】コンピュータアプリケーションの追跡及びモニタリングを行う汎用のマルチインスタンスメソッド及びＧＵＩ検出システム

(57)【要約】
【課題】コンピュータの対象アプリケーションのプレセンテーション層から発生するイベントモニタリングシステムとその方法を提供する。
【解決手段】対象アプリケーションのソースコードの再コンパイルとは独立し、対象アプリケーション内のレベルで実行されるスクリプトを提供するステップを含んでいる。スクリプトは、対象アプリケーションにおけるオブジェクトのランタイムのインスタンス化を読み取る構造を、オブジェクトのインスタンス化にリアルタイムで割り当てる。これらの割り当てられた構造は、対象アプリケーションの構造の相似物を作成するために適用される。この構造は、所定のオブジェクト構造を、検出されたオブジェクトの動作環境スペクトラムを捕捉するように適合させる検出されたオブジェクトのインスタンス化と共に、使用される。サーバ機器とクライアントのローカル機器のうちの少なくとも一つに発生する状態機械を、本システムは処理し、状態機械のイベントを動作環境スペクトラムと相互に関連付け、相互に関連付けられた状態機械のイベントを基にして、ユーザの体験を推論する。

10

【特許請求の範囲】
【請求項１】
　構造を有する対象アプリケーションにおいて、対象アプリケーションのソースコードの再コンパイルとは独立に、前記対象アプリケーション内の所定のレベルで動作可能なスクリプトを提供する提供ステップと、
　前記対象アプリケーションの、メソッドの呼び出しと、メソッドの戻しと、GUIオブジェクトとを有する、オブジェクトのランタイム・インスタンス化を読み取る読み取りステップと、
　前記オブジェクトのインスタンス化に対して、リアルタイムで割り当て構造を割り当てる割り当てステップと、
　前記対象アプリケーションの構造を反映した構造を作成するために、前記割り当て構造を適合させる適合ステップと、
　所定のオブジェクトの構造に一致する１以上のオブジェクトのインスタンス化を検出する検出ステップと、
　検出された前記オブジェクトの動作環境スペクトラムの少なくとも一部を捕捉する捕捉ステップと、
を備えることを特徴とするプレゼンテーション層と、前記対象アプリケーションのソフトウェア技法と、から導出されるイベントモニタリング方法。
【請求項２】
　検出された前記オブジェクトの捕捉された前記動作環境スペクトラムを、処理して論理イベントにするステップと、
　個別なコンテキストが前記対象アプリケーションの従前の状態である場合において、前記識別可能なコンテキスト内のマルチインスタンス化された構造を反映するために、前記論理イベントを再構築するステップと、
　前記再構築されたマルチインスタンス化された構造を相互に関連付けて、前記対象アプリケーションの前記従前の状態を分析するステップと、を
備えることを特徴とする請求項１に記載のイベントモニタリング方法。
【請求項３】
　前記捕捉された動作環境スペクトラムが、検出された前記オブジェクトに関する分類された情報を有し、前記情報により、前記対象アプリケーションの動作環境に関する知見が提供される、ことを特徴とする請求項１に記載のイベントモニタリング方法。
【請求項４】
　前記読み取りステップが、少なくとも１つの予め定義されたモニタリング基準に基づいて、オブジェクトのインスタンス化を選択するステップを、さらに有することを特徴とする請求項１に記載のイベントモニタリング方法。
【請求項５】
　前記提供ステップは、クライアントコンピュータ上に、前記スクリプトを提供することを特徴とする請求項１に記載のイベントモニタリング方法。
【請求項６】
　夫々が前記対象アプリケーションの各インスタンス化と関連している複数のスクリプト間で、前記対象アプリケーション構造を反映した構造を共有するステップを更に備え、
　前記反映した構造は、収集された追跡データに対してGUIコンテキストを提供することを特徴とする請求項１に記載のイベントモニタリング方法。
【請求項７】
　関連するオブジェクトの分類を検出することができるスクリプト表現を使用して、前記オブジェクトの汎用インスタンス化を検出するステップと、
　コンテキスト内における前記対象アプリケーションのイベントの発生を判定するために、前記汎用インスタンス化のコンテンツ及びプロパティを処理するステップと、を
さらに備えることを特徴とする請求項１に記載のイベントモニタリング方法。
【請求項８】

10
20
30
40
50

追跡メッセージ・レベルと、予め定義された動作及び仕様と、を使用することにより、ユーザコンテキストを構築するステップを、さらに備えることを特徴とする請求項７に記載のイベントモニタリング方法。

【請求項９】

前記対象アプリケーションの一部を表し、少なくとも前記捕捉された動作環境スペクトラムからのコンテキスト情報を含む階層組織エンティティを宣言するステップと、

前記階層組織エンティティによりサポートされる、前記捕捉された動作環境スペクトラムを、分析するステップと、を

さらに備えることを特徴とする請求項１に記載のイベントモニタリング方法。

【請求項１０】 10

前記対象アプリケーションの実行中に、複数のオブジェクトのインスタンス化を区別するために、コンテキスト・レベル値を評価するステップを備えることを特徴とする請求項９に記載のイベントモニタリング方法。

【請求項１１】

前記対象アプリケーションの複数のインスタンス化が存在し、前記読み取りステップは、各前記対象アプリケーションのインスタンス化を読み取り、前記検出ステップは、１以上の前記対象アプリケーションのインスタンス化に関するオブジェクトのインスタンス化を検出する、ことを特徴とする請求項２に記載のイベントモニタリング方法。

【請求項１２】

前記捕捉ステップは、任意の対象アプリケーションのインスタンス化から検出されたオブジェクトのコンテンツを捕捉する、ことを特徴とする請求11に記載のイベントモニタリング方法。 20

【請求項１３】

前記クライアントコンピュータ上の前記スクリプトが、前記対象アプリケーションに対してローカルであることを特徴とする請求４に記載のイベントモニタリング方法。

【請求項１４】

対象アプリケーションとのインタラクションの内、ユーザと対象アプリケーションから発生するイベントとの少なくとも一つを感知する感知ステップと、

前記感知ステップの結果に応じて、前記スクリプトの実行に基いて選択的にイベントを発生させる発生ステップと、 30

１以上の前記発生させたイベントをインタープリタへ渡す譲渡ステップと、

選択的に前記発生させたイベントを分析する分析ステップと、

機能的マッピングによる分析を、外部ソースと相互に関連付ける関連付けステップと、を

備えることを特徴とする請求項１に記載のイベントモニタリング方法。

【請求項１５】

前記分析ステップと前記関連付けステップの結果を含む、レポート、警報、相関関係、応答の内の少なくとも何れか一つを発生するステップを備えることを特徴とする請求項14に記載のイベントモニタリング方法。

【請求項１６】 40

前記読み取りステップ、前記割り当てステップ、前記適合ステップ、前記検出ステップ、及び前記捕捉ステップが、所定の間隔で繰り返され、前記対象アプリケーションの最新アプリケーション構造を反映した構造を作成することを特徴とする請求項１に記載のイベントモニタリング方法。

【請求項１７】

前記読み取りステップ、前記割り当てステップ、前記適合ステップ、前記検出ステップ、及び前記捕捉ステップが、検出されたイベントにより呼び出され、前記対象アプリケーション構造の最新アプリケーション構造を反映した構造を作成することを特徴とする、請求項１に記載のイベントモニタリング方法。

【請求項１８】 50

前記対象アプリケーションにおいて、ウィンドウの単一プロセスと前記スクリプトとを対応付けるステップと、
スレッドIDと対応付けられたメッセージを検出するステップと、
前記スクリプトにおける動作状態のフック・ルーチンにより、前記メッセージをインターセプトするステップと、
前記メッセージのコンテンツを評価するステップと、
前記対象アプリケーションに関する情報を獲得するために、前記メッセージからイベントとプロパティを導出するステップと、を
備えることを特徴とする請求項１に記載のイベントモニタリング方法。
【請求項１９】
アプリケーション・ウィンドウのオブジェクトをモデル化可能で、所定のモニタリング基準に基づいて認識されるオブジェクトをモニタリング可能な、独立したモニタリングプログラムを提供する提供ステップと、
前記アプリケーション・ウィンドウに対する、動作環境スペクトラムの指示を導出する導出ステップと、
サーバ機器とクライアント／ローカル機器との内の少なくとも何れか１つにおいて発生する状態機械のイベントを処理する処理ステップと、
前記状態機械のイベントと、導出された前記動作環境スペクトラムの指示とを、相互に関連付ける関連付けステップと、
前記関連付けステップの結果に基づいて、ユーザ経験の兆候を推定する推定ステップと、
システムユーザに、推定した前記兆候を報告する報告ステップと、を
備えることを特徴とするメソッドイベントの分析方法。
【請求項２０】
前記推定ステップが、前記サーバ側により受信された指示に基づくものであることを特徴とする請求項19に記載のメソッドイベントの分析方法。
【請求項２１】
前記関連付けステップが、前記動作環境スペクトラムの指示による共同作用である前記状態機械イベントから論理イベント・データを収集することを備えることを特徴とする請求項20に記載のメソッドイベントの分析方法。
【請求項２２】
前記独立したモニタリングプログラムは、
バイト命令コードパターンを識別する識別ステップと、
アプリケーション処理のイベント及びプロパティと、アプリケーション・ウィンドウ、或いは、ウィンドウのGUIオブジェクトへインターフェイスを与える前記識別されたバイト命令コードパターンの内の、少なくとも１つのインターフェイス関数を検出するステップと、
前記検出されたインターフェイス関数のアドレスを判定するステップと、
前記検出されたインターフェイス関数に対する呼び出しの追跡において、呼び出しインターセプト・ルーチンを使用することによって、前記検出されたインターフェイス関数とリンクするステップと、
を実行可能とする命令を備えることを特徴とする請求項19に記載のメソッドイベントの分析方法。
【請求項２３】
前記リンクするステップが、戻りの追跡において、戻りインターセプト関数を使用することを特徴とする、請求項22に記載のメソッドイベントの分析方法。
【請求項２４】
前記独立したモニタリングプログラムは、
汎用関数のインターセプトが、前記検出されたインターフェイス関数のパラメータ及びフォーマットの事前情報を必要としない場合において、

10

20

30

40

50

特定の関数のインターセプトを処理する方法とは異なる方法で、汎用関数のインターセプトを行うステップを実行する命令を、備えることを特徴とする請求項22に記載のメソッドイベントの分析方法。
【請求項２５】
前記独立したモニタリングプログラムは、
特定関数のインターセプトが、前記検されたインターフェイス関数の公開されたパラメータ及びフォーマットの指示を有し、前記特定の関数のインターセプトが、スクリプトにおける前記検出されたインターフェイス関数パラメータを直接利用可能な場合において、
前記特定関数のインターセプトを処理する方法とは異なる方法で、汎用関数のインターセプトを処理するステップを実行可能な命令を、備えることを特徴とする請求項22に記載のメソッドイベントの分析方法。
【請求項２６】
ホストアプリケーションにおけるスタックは変更されず、インスタンス情報レジスタが、メソッド呼び出しコンテキストを保存するために、呼び出されている関数のインスタンス情報に対するポインタを有する場合において、
前記インスタンス情報レジスタを使用して、メソッド呼び出しに前記インスタンス情報を渡すステップを、備えることを特徴とする請求項22に記載のメソッドイベントの分析方法。
【請求項２７】
コンピュータ上の所定の構造を有する対象アプリケーションが有するプレセンテーション層におけるイベントモニタリングシステムは、
通信ネットワーク上で相互に接続されているサーバ及びクライアントコンピュータと、
前記プレゼンテーション層より下位のレベルで実行可能なスクリプトを有する前記サーバ上で動作するモニタリングプログラムと、を備え、
前記モニタリングプログラムは、
前記対象アプリケーションのオブジェクトのランタイム・インスタンス化を読み取るステップと、
前記オブジェクトのインスタンス化に対して、リアルタイムで構造を割り当てるステップと、
前記対象アプリケーションの構造を反映した構造を作成させるために、前記割り当てられた構造を適合させるステップと、
所定のオブジェクト構造に一致する１以上のオブジェクトのインスタンス化を検出するステップと、
前記検出されたオブジェクトの少なくともコンテンツを捕捉するステップと、
を備えることを特徴とするイベント・モニタリングシステム。
【請求項２８】
前記スクリプトが、前記対象アプリケーションに対して、ローカルコンピュータ上に存在していることを特徴とする請求項27に記載のイベント・モニタリングシステム。
【請求項２９】
前記ローカルコンピュータであるクライアントコンピュータが、パーソナルコンピュータ、ワークステーション、携帯端末、及びコンピュータの動作命令を実行可能なプラットフォームの内の１つであることを特徴とする請求項28に記載のイベント・モニタリングシステム。
【請求項３０】
ホストアプリケーション内のメソッド関数を表すバイト命令コードパターンを識別するステップと、
前記メソッド関数の内の少なくとも１つのアドレスを決定するステップと、
前記アドレスを使用して、前記メソッド関数を、呼び出しインターセプト・ルーチンと、情報構造インスタンスとにリンクするステップと、
スタック操作により、前記メソッド関数への呼び出しからの戻りをモニタリングするス

10

20

30

40

50

テップと、
を備えることを特徴とするホストアプリケーション機能のモニタリング方法。
【請求項３１】
前記リンクするステップが、ジャンプルーチン命令を使用することを特徴とする請求項30に記載のホストアプリケーション機能のモニタリング方法。
【発明の詳細な説明】
【技術分野】
【０００１】
著作権表示
この特許明細書の開示の一部は、著作権の保護の対象となるものが含まれる。著作権所有者は、米国特許商標局が保管・記録するとおりに、如何なる者によってなされる特許明細書の複製または特許開示を行うことについて差し支えはないが、それ以外は如何なるものも著作権を留保する。
【０００２】
本発明は、ユーザのコンピュータアプリケーションとのインタラクションのモニタリングシステム及び方法に関し、さらに詳しくは、セッション中のユーザとアプリケーションとのインタラクションを、アプリケーションのパフォーマンス及び動作も一緒に、モニタリングし、分析するシステム及び方法に関する。
【背景技術】
【０００３】
今日の企業では、エンタープライズアプリケーションは、複雑化したグローバル経済において、ビジネスプロセスを実施する機能を提供するものであり、そのため、アプリケーションは、多種多様なビジネスの要求に応えるために、非常に複雑になってきている。アプリケーションを使いこなすスキルを習得する努力に加え、成長するインフラに応じて変化するアプリケーションにおいて、アプリケーション上におけるユーザパフォーマンスとユーザに対するアプリケーションパフォーマンスの測定との両者に関して、システマティックな測定を行うことは、非常に難しいものである。
【０００４】
一般的に、ユーザモニタリング製品は、パフォーマンスモニタリング捕捉、HTTPモニタリング、HTTP捕捉、ホストアプリケーション・ランタイム・ソースコード挿入、ホストアプリケーション・ディベロップメント・ソースコード挿入、またはコンパイルのようにいくつかの分類に分けられる。以下で取り上げる製品は、それぞれの分類の代表的な製品である。
【０００５】
エンドユーザのコンピュータ上におけるパフォーマンスをモニタリング及び捕捉する製品である。EdgeSightの製品名でマサチューセッツ州ウエストフォードのReflectent Software が取り扱う製品を含むビューは、リアルタイム及び履歴的な両方における特定のアプリケーションの実行及び利用についての知見を得ることが可能なエンドユーザの視点から、パフォーマンスとアプリケーション及びＩＴサービスの可用性とを定量化する。クラッシュ、エラー、及びハングアップが発生した場合、原因究明に必要な情報及びデータは、直ちに問題の根本原因にピンポイントで利用可能となる。しかし、それは、ホストアプリケーションのGUIプレゼンテーション層からモニタリングデータを生成せず、トランザクション間における如何なる動作もモニタしてない。GUIレベルにおけるホストアプリケーションのマルチインスタンスをサポートすることができず、関数呼び出しのモニタリングメソッドを実行する機能も有していない。また、分散型のローカル及びリモート状態機械に関する効率的で高速な処理をサポートすることもない。
【０００６】
従来の製品であっても、クライアントコンピュータにおけるHTTPのトラフィックをモニタリング可能である。例えば、カリフォルニア州マウンテンビューのMercury Corporationが販売する「End User Monitoring」は、エンドユーザの視点から、リアルタイムにウェ

ブサイト及びアプリケーションの可用性を能動的にモニタリングする。また、アプリケーションに対して、エンドユーザのビジネスプロセスを能動的にエミュレートする。しかし、それは、ホストアプリケーションのGUIプレゼンテーション層からモニタリングデータを生成せず、トランザクション間における如何なる動作もモニタリングしてない。それは、汎用のGUIオブジェクトのサブセット及び分類化されたセットを検出することができず、さらに、モニタリングメソッドを実行する機能も有していない。それは、GUIレベルにおけるホストアプリケーションのマルチインスタンスをサポートすることもできない。また、分散型のローカル及びリモート状態機械に関する効率的で高速な処理をサポートすることもない。

【０００７】

ETEWatchの登録商標でCandle Corp (IBM/Tivoli)が販売する製品は、アプリケーションが画面情報をロードするまたは動作を実行する時の、トランザクションレベルにおける端末間の応答時間とユーザが経験する待ち時間とを計測する。しかし、それは、ホストアプリケーションのGUIプレゼンテーション層からモニタリングデータを生成せず、トランザクション間における如何なる動作もモニタリングしない。また、それは、汎用GUIオブジェクトのサブセット、分類化されたセットを検出することができない。それは、GUIレベルにおけるホストアプリケーションのマルチインスタンスをサポートすることができず、関数呼び出しのモニタリングメソッドを実行する機能も有していない。また、分散型のローカル及びリモート状態機械に関する効率的で高速な処理をサポートすることもない。

【０００８】

Vantage 9.7の登録商標でミシガン州デトロイトのCOMpuwareが販売する製品は、アプリケーションのパフォーマンスデータ及びイベントを、アプリケーション、トランザクション、職務、及び場所などのビジネスにおいて重要な要素に相互に関連付ける。また、Additional viewsは、拡張トラブルシューティング機能により、エンドユーザ応答時間の評価指標を統合する。これにより、IT部門が、パフォーマンス問題を解消しアプリケーション及びインフラを能動的に管理できる。しかし、それは、ホストアプリケーションのGUIプレゼンテーション層からモニタリングデータを生成せず、トランザクション間における如何なる動作もモニタリングしない。また、それは、汎用GUIオブジェクトのサブセット、分類化されたセットを検出することができない。GUIレベルにおけるホストアプリケーションのマルチインスタンスをサポートすることができず、関数呼び出しのモニタリングメソッドを実行する機能も有していない。分散型のローカル及びリモート状態機械に関する効率的で高速な処理をサポートすることもない。

【０００９】

サーバのHTTPのトラフィックを捕捉する製品もいくつか販売されている。RealiTeaの製品名でカリフォルニア州サンフランシスコのTeaLeaf Technology, Inc.が販売する製品は、リアルタイムで全ての顧客が行う動作及び閲覧しているものを捕捉することで、直ちに問題を検出、分析して対応できる。これにより、細分化されたビジネスプロセスの垣根を越えて、思惑から外れる各顧客のユニークな行動を有効活用する。しかし、それは、ホストアプリケーションのGUIプレゼンテーション層からモニタリングデータを生成せず、トランザクション間における如何なる動作もモニタリングしない。GUIレベルにおけるホストアプリケーションのマルチインスタンスをサポートすることができず、透過的に実装されず、アプリケーションソースコードの変更、またはコンパイルプロセスからのいくらかの出力を必要とする。分散型のローカル及びリモート状態機械に関する効率的で高速な処理をサポートすることはない。

【００１０】

実行時にホストアプリケーションにコード挿入を行う製品もいくつか販売されている。i^3の登録商標でVeritasが販売する製品は、SAP, Siebelや他の企業のアプリケーションに対して能動的にモニタリング、分析、チューニングを行う。アプリケーションのインフラ（ウェブサーバ、アプリケーションサーバ、データベース、ストレージ）の各サポート階層からのパフォーマンス評価指標を、捕捉、計測して相互に関連付けることにより、アプ

リケーションパフォーマンスの全体像を提供する。この製品はGUIイベントを見てはいるものの、ウェブアプリケーションに限定され、一般GUIオブジェクトセットを検出するようなコンセプトはない。また、アプリケーションのサイトにおいて要求されるソースコードの挿入は、非常に煩わしいプロセスであり、透過的な実装プロセスでもない。また、Veritasの製品は、OSレベルから如何なるものも直接検出することができず、コンポーネントメソッド（関数）またはWin32のアプリケーションのインターフェイスから直接に、イベントの検出またはプロパティの取得を行うことができない。さらに、Veritasの製品では、例えば、ウェブ及びWin32などの複数の異なるタイプのアプリケーションにおいて、同時測定を行うことができない。また、それは、リアルタイムな警告の実行に対するローカル状態機械のコンセプトがなく、ページを跨っての持続性が欠如している。また、分散型のローカル及びリモート状態機械に関する効率的で高速な処理をサポートすることもない。

【００１１】

開発時のソースコードのコンパイルにおいて生成されるファイルを使用する製品もいくつか販売されている。AppSight Black Boxの製品名でニューヨーク州ニューヨークのIdentify Softwareが販売する製品は、サーバ及び／またはクライアント上に分散し、動的でユーザ定義の記録プロファイルに基づき、複数の同期したレベルにおいてアプリケーションの実行を記録する。Black Boxでは、ソースコードまたは実行可能ファイルの変更をする必要はない。クライアント側において、ユーザの動作及びスクリーン上でのイベントがビデオのように捕捉されている。この製品は、トランザクション応答時間の計測を行わない。さらに、この製品は、モニタリングメソッドの形成を行う一方で、開発段階のソースコードのコンパイル中に生成されるファイルに依存しており、透過的な実装プロセスを厳密にはサポートしない。本明細書で記載された本発明は、コンパイルされたソースコードを必要とせずモニタリングメソッドを実現し、さらに、関数の復帰も同様にモニタリングする。しかし、それは、GUIオブジェクトモデルからGUIオブジェクトレベルにおいて、ホストアプリケーションのGUIプレゼンテーション層からモニタリングデータを生成せず、汎用GUIオブジェクトのサブセット、分類化されたセットを検出することができない。GUIレベルにおけるホストアプリケーションのマルチインスタンスをサポートすることができず、透過的に実装されず、アプリケーションのソースコードに対する変更、またはコンパイルプロセスからの出力を必要とする。また、分散型のローカル及びリモート状態機械に関する効率的で高速な処理をサポートすることもない。

【００１２】

米国特許第６１０８７００号明細書「アプリケーションの両端末間応答時間の測定及び分解」は、全ての階層に包括的に跨る両端末間計測に関し、多種多様なデータ型を統合するフレームワークに関するものである。また、米国特許第６０７８９５６号明細書「ワールドワイドウェブにおけるエンドユーザ応答時間のモニタリング」は、第１HTTP要求に関連付けられた応答時間を計算し、その結果をサーバに送信して後日使用するためにログを取っておくHTTPレベルでのモニタリング要求を開示している。また、米国特許第５９９１７０５号明細書「キューの開始及び停止を用いたコンピュータプログラムに関する両端末間応答時間計測」及びその継続出願である米国特許第６２０２０３６号明細書のそれぞれは、両端末間においてコンピュータにより実行されるトランザクションの応答時間を計測するシステムに関する。さらに、米国特許第６１８９０４７号明細書「プラグ脱着可能なイベントキューによりイベントキュー動作をモニタリングする装置及び方法」は、アプリケーションにより使用され、ユーザのGUIインタラクションにより生じるシステムメッセージキューを反映するようにカスタマイズされたメッセージキューの使用を開示している。上記した各米国特許は、その全体を参照により本明細書に組み込まれる。

【００１３】

特に、先行技術特許である米国特許第５９９１７０５号明細書、米国特許第６１０８７００号明細書、及び米国特許第６０７８９５６号明細書は、ホストアプリケーションのGUIプレゼンテーション層からモニタリングデータを生成可能な方法またはシステムに関し

て開示がなされていない。先行技術文献の技術はどれも、分散型（ローカル及びリモート）状態機械における高速なイベント処理を搭載しておらず、単純なキューを用いない如何なるモデリングまたはツリー構造のGUIオブジェクトセットを実装も、GUIマルチインスタンスの取り扱いも搭載していない。先行技術は、汎用GUIオブジェクトのサブセット、分類化されたセットの使用を開示しておらず、さらに、モニタリングメソッドを実行する機能を提供していない。上記の先行技術は、その他全ての非GUIデータのタイプに関するGUIコンテキストも提供しない。

【００１４】

本明細書は、出願人の取得特許（米国特許第６３４０９７７号明細書「動作及びホストアプリケーションモデルを用いたソフトウェアアプリケーションにおける動的サポートに関するシステム及び方法」）と係属中の継続出願（２００１年１１月２０日出願、米国特許出願公開第０９／９８９７１６号明細書）との類似分野を対象にしている。また、出願人は、商業的な実施製品として、User Performance Measurement 3.0 (Knoa Operation Response Time Monitor, Knoa User & Application Errors Monitor, Knoa Application Usage Monitor)、Knoa Business Process Measurement 3.0 (Knoa Business Process Monitor)、及びKnoa COMpliance Measurement 2.0 (Knoa user COMpliance Monitor)の製品名で販売している。本明細書は、出願人の現在の製品に対する問題の解決及び改良に関する。

【００１５】

ホストアプリケーションのマルチインスタンスの検出の自動化とGUI構造のサブセットのマルチインスタンスの検出とを可能にする、GUI検出及びモニタリングシステムが先行技術では欠けている。さらに、ホストアプリケーションまたはホストアプリケーションのサブセットのマルチインスタンス内のGUIオブジェクトの分類の汎用検出が可能なシステムが先行技術では欠けている。本発明は、これらの及びその他の必要とするものを提供する。

【００１６】

本発明は、第１実施形態で、コンピュータにおいて対象とされるアプリケーションのプレゼンテーション層から派生するイベントをモニタリングする方法を、提供する。この方法は、対象のアプリケーションのソースコードの再コンパイルの独立性と、対象のアプリケーションのレベル内において実行するスクリプトとを提供する提供ステップを備える。スクリプトは、対象のアプリケーションのオブジェクトのランタイムインスタンス化を読み取る読み取りステップ、そして、構造を、リアルタイムで、オブジェクトのインスタンス化に割り当てる。これらの割り当てられた構造は、対象のアプリケーション構造のアイデアを生成するために適応される。ここで、対象のアプリケーション構造は、検出されるオブジェクトの動作環境スペクトラムの一部を捕捉するために、所定のオブジェクト構造に適合する検出されたオブジェクトインスタンス化に沿って使用される。

【００１７】

本発明の別の実施形態では、メソッドは、少なくとも1つのサーバ機械及びクライアント/ローカル機械において生じる状態機械を処理し、その状態機械イベントを、動作環境の動作環境スペクトラムと関連付け、そして、関連付けられた状態機械イベントに基づくユーザ経験を減少させる。

【００１８】

本発明のさらに別の実施形態では、コンピュータにおいて対象とされるアプリケーションのプレゼンテーション層において、イベントをモニタリングするために設定されるシステムは、サーバと、通信ネットワークに接続されるクライアントコンピュータとで構成される。ここで、通信ネットワークにおいて、サーバの動作命令は、プレゼンテーション層以下のレベルで実行動作が可能なスクリプトを有するモニタリングプログラムを備える。モニタリングプログラムは、オブジェクトのランタイムインスタンス化のスキャンを行う命令を有する。その命令は、メソッド呼び出し、メソッド戻り、及び対象のアプリケーションのGUIオブジェクト、構造をリアルタイムでオブジェクトインスタンス化に割り当て

ること、対象のアプリケーション構造のアイデアを生成するため割り当てられた構造を適用すること、そして、所定のオブジェクト構造に適合する１つ以上のオブジェクトインスタンス化を検出することとを備える。

【００１９】

これら及びその他の実施形態、特徴、及び利点は、添付の図及び図示された実施例の説明によりさらに理解される。

【００２０】

本明細書で、本発明の文脈内及び各用語が使用されている特定の文脈において使用されている用語は、一般的に、業界内で用いられる通常の意味を有する。いくつかの用語は、本発明のデバイス及び方法とその製作及び使用方法とについて説明する追加的なアドバイスを実施者に行うために、以下においてまたは明細書の他の箇所において論じている。また、ここで説明した方法以外で、同じ用語を説明することは可能である。

【００２１】

従って、ここで論じられる１つ以上の用語に関して、他の言語及び同意語が、使用されることが可能であり、ここで用語が詳細に述べられるか、または論じられるかであるなしにかかわらず、如何なる特別な意味にも、置き換えられるべきではない。いくつかの用語の同意語は記述されている。いくつかの同意語の詳説は、その他の同意語の使用を排除している訳ではない。ここで論じる如何なる用語の使用例をも含み、本明細書の如何なる例の使用も実例としてだけの意味であり、本発明の意義及び範囲、または如何なる例示した用語について制限するものではない。同様に、本発明は、好ましい実施形態により制限されることはない。

【００２２】

「エージェント」は、１つ以上の実行中のシングルエンジンのDLLのインスタンスに関連付けられたプライマリコードユニットである。

【００２３】

「コンポーネント」は、実行されるプログラミングコード及びデータからなる汎用ＯＳパッケージを意味する。コンポーネントプログラミングコードは、アプリケーションまたはサービスの機能を実行して、複雑なホストアプリケーションに関するコードの再利用及びモジュールのパッケージ化を容易にするメソッドまたは呼び出し関数を備える。コンポーネントは、外部アプリケーションに関するインターフェイスの再利用及び配布を促進する企業により作成された多種多様な技術を実装する。オブジェクト指向のクラスまたはプログラミングコードを有するコンポーネントは、マイクロソフト（ワシントン州レドモンド）のコンポーネントオブジェクトモデル（COM）、プログラムコンポーネントオブジェクトCOMの開発及びサポートためのフレームワーク、または他のコンポーネント技術などのコンポーネント技術に実装され得る。

【００２４】

「動作環境スペクトラム」は、対象のアプリケーションに提供される不変条件及び時変条件を意味し、オブジェクト、GUI、メッセージ、メソッド呼び出し及びメソッド戻りに明示されるが、これに制限されるものではない。

【００２５】

「イベントモデル」は、ホストアプリケーションにおいて発生するいくつかの現実のプロセスを、イベントの検出及び収集が反映及び象徴するイベントの組を意味する。

【００２６】

「エクスポート関数」は、エクスポート関数名がプログラムモジュールヘッダに記録されたDLLコンポーネントにおいて公開されるメソッドインターフェイスである。

【００２７】

「フレームワーク」は、GUIオブジェクト及びメソッド呼び出しのモデリング状態及び検出とイベントの復帰に関する動作の基本構造を意味する。

【００２８】

「汎用」は、対象物のセットを検出する動作を意味する。このセットは、クラスまたは

類似のオブジェクトの分類として参照され、ここでは、GUIオブジェクト、メソッド、または他のイベントのセットを説明するために用いられる。「汎用」の反対で、通常、単一のGUIオブジェクト、メソッド、または他のイベントを参照する特定が用いられる。

【００２９】

「gMgエンジン」は、モニタリング対象内において選択的にモニタリングを実行する論理的及び条件的な記述を含むバイナリスクリプトファイルを処理するスクリプト・インタープリタの意味である。

【００３０】

「gMg GUIオブジェクトツリー」は、ホストアプリケーションのユーザインターフェイスからなるライブGUIオブジェクトをモデル化及び表現するために、本発明において使用される階層構造を意味する。

【００３１】

「gMg GUIコンテキスト」は、gMgシステム内において、アプリケーションに関するユーザ経験をより良く反映するための分析を目的とするGUI（またはユーザ）コンテキストを提供するGUIデータのストリーム内における全データ型のインターリービングを意味する。

【００３２】

「gMgソリューション」は、ホストアプリケーションのモニタリングを目的として実行される条件、論理、及び選択的対象を記述するコンパイルされたスクリプトのバイナリパッケージングを意味する。

【００３３】

「GUI」は、ホストアプリケーションにより制御されるグラフィカルユーザインターフェイスの略であり、ユーザとホストアプリケーションとの間の主要なインターフェイスである。「GUIオブジェクト」は、ホストアプリケーションのコードを介して、コンピュータの入力デバイスに対して表示または反応、及びコンピュータの入力デバイスにより制御されるユーザインターフェイスを備える、通常はRAM内にプログラムされたGUI構造の一部であるオブジェクトの型である。

【００３４】

「フック」は、フィルタリングを行うまたはホストアプリケーションが実行時に生成するメッセージに基づいて動作を変更する外部システムにより、さらに実行されるホストアプリケーションが生成するメッセージ、またはOSのメッセージの捕捉を行うセンサの一部を意味する。

【００３５】

「ホストアプリケーション」は、モニタリングの対象とされるアプリケーションを意味する。

【００３６】

「IE」は、インターネットのブラウザであるマイクロソフトのInternet Explorerを略したものである。

【００３７】

「インスタンス」は、プログラムの命令コードが抽出セットまでさかのぼることができるプログラムのいくつかの機能において、単一の抽出セットが機能を実装するために、１つ以上の命令ユニットをメモリにインスタンス化することを意味する。抽出命令の結果、メモリ上においてインスタンス化され実行中のプログラムコードは、インスタンスとして参照される。

【００３８】

「論理イベント」は、例えば、クライアントまたはサーバなどの物理的に異なる場所における状態機械プロセッシングの連続段階など、１つ以上のレベルで処理可能な対象の状態を表す状態機械の処理イベントを意味する。

【００３９】

「メソッド」は、プログラム関数において、関数と同意語的に使用され、逆もまた同様

である。
【００４０】
「メソッドイベント」は、コンポーネントのメソッドまたは関数（例えばプログラム関数またはメソッド）をモニタリングすることにより生成されるイベントを意味する。
【００４１】
「マルチインスタンス」は、指定されたユニットの２つ以上のインスタンスがメモリ上に同時に常駐していることを意味する。
【００４２】
「オブジェクト変数」は、GUIオブジェクトツリーに常駐している既存のGUIオブジェクトメンバに付随する本発明の特定オブジェクトを意味する。
10
【００４３】
「OS」は、コンピュータオペレーションシステムの略である。
【００４４】
「センサ」は、ハブリックまたはプライベートのインターフェイスを使用するアプリケーションのコンポーネントからリアルタイムデータを取り出す特定のコードユニットである。
【００４５】
「特定」は、１つの命令が単一のGUIオブジェクトなどの１つの論理条件を検出することを意味する。
【００４６】
20
「状態機械」は、永続性記憶装置に接続され、入力及び出力のイベントを有し、入力イベントの条件処理により離散的で特有の状態を決定する処理装置を意味する。
【００４７】
「サンク」は、モニタリングメソッドにおいて、コードの実行またはインスタンス情報の保存に使用される動的割り当て構造を意味する。サンク構造は、捕捉の関数呼び出しまたは関数戻りの様々なシナリオに適用される。
【００４８】
「仮想関数」は、サブクラスによりオーバーライドされたときに、ベースクラスにより呼び出された関数を意味する。
【００４９】
30
「VTable」は、クラスに属し、メソッドまたは関数へのメソッドポインタを有するCOMクラス内のテーブルである仮想テーブルを意味する。
【特許文献１】特開２００７－２６２７６９号公報
【発明の開示】
【発明が解決しようとする課題】
【００５０】
前提として、ホストアプリケーションの複数のインスタンスにおける特定のイベントまたは一般的に構造化されたイベントの適宜な追跡及びモニタリングを提供するシステム及び方法の実施形態を示す。追跡及びモニタリングは、例えば、GUI、関数レベル又はメソッドレベルである、アプリケーションのプレゼンテーション層から派生される。イベントが発生し、その結果生じるパターンの検出・解析により、知見の提供及び計測値の統計を行う。ユーザモニタリングは、応答時間の計測、エラーの検出、及びアプリケーションイベントの収集を含むが、これに限定されるものではない。ユーザモニタリングは、ユーザのコンテキスト及びアプリケーション・インタラクションを認識できることから得られる情報を提供する。
40
【００５１】
企業環境においてアプリケーションがより複雑化しているため、実際のユーザ経験を把握したり計測したりすることが、益々困難となっている。例えば、読み込みや応答時間の遅延、不可解なエラーメッセージ、部分的に表示されるウェブページから、不完全なデータベースのテーブル及びレコードなどに至る、ユーザが経験するすべて範囲に関して問題
50

が生じ得る。実際は、会社に収益をもたらしているのがビジネスユーザであるにもかかわらず、目下、バックエンドのインフラに関して、ＩＴ部門が全面的に注目を集めている。

【００５２】

ユーザ経験の定量的な計測を行うために、ユーザ又はサーバ環境において利用可能な多くの情報源が潜在的に存在する。集中データソースとのインタラクション及びトランザクションを実行する多くのコンポーネントが存在するため、各コンポーネントはユーザ経験を構成するイベントの長いチェーンにおけるリンクとなる。このようなコンポーネントは、ネットワーク層、ハードウェア層（CPU、メモリ、仮想メモリ）、GUIプレゼンテーション層、アプリケーションサーバ等に存在し得る。

【００５３】

ユーザは、アプリケーションのGUI層とインタラクトしながら、重要な業務プロセスを行っているため、アプリケーションの任意のインフラ層内におけるすべてのアクション及び活動に関するGUIレベルでのコンテキスト及び認識の問題は、理想的には、いくつかのユーザコンテキスト内におけるユーザアクションにさかのぼられるべきである。GUI層からのイベントは、まさにこのようなコンテキストを提供する。システムはこのレベルにおいて、ユーザが見るものを追跡しモニタリングする。GUIは、階層構造の中で組織化され、多くのGUIオブジェクトのサブセットを有しており、そのサブセットは、ユーザアクション及びアプリケーションの組織化の論理に応じて、実行時に複製することができる。これにより、どのようにGUIオブジェクト構造が生成され、実行時にメモリに保持されるかを制御する。システムは、他の状況では同一であるように見える構造を区別する識別支援メカニズムを提供する。特に、インフラの任意の階層から発生したエラーメッセージに、ユーザが遭遇する場合において、アプリケーションの改善策を促進するために後で解析可能なように、エラーインスタンスが捕捉される。

【００５４】

今日のインターネットの世界では、アプリケーションのマルチインスタンス内のGUIオブジェクトを正確に検出するという課題を、すべて同時に生じかねない同一構造を有するウィンドウにおけるマルチインスタンスで、アプリケーションが動作することは非常に一般的である。特に、内部の構造的には同一であるが、ユーザに対して異なって見えるインターフェイスの識別を区別する課題が存在する。最近のアプリケーションは、同一の内部オブジェクト構造を持つが、文字列がラベル付けされたウィンドウをユーザインターフェイス上に多数生成する可能性がある。また、このマルチインスタンス化は、手続きの共有を最大にするように追求する最近のプログラミング技法の後遺症でもあり、すなわち実行時間の効率性及びメンテナンス性の両方をより高めるためにコンポーネントを再利用することである。

【００５５】

透過性の問題に関し、モニタリングシステムは、対象となるホストアプリケーションに侵入せずに透過され、かつモニタリングコンポーネントはあたかも存在しないかのように動作することが必要である。ほとんどの場合、企業アプリケーションの配置は、インターフェイスを利用可能にするためにソースコードへのアクセスを提供するのではなく、モニタリングプログラムのような外部アプリケーションにカスタマイズされたイベントを公開する。アプリケーションの変更なく選択的モニタリングを行うために、モニタリングシステムによりアプリケーションをどのように拡張するかが問題となる。この透過性を得るために、モニタリングシステムは、入受信イベントを傍受及びフィルタリングを行い、それらを読み込んで処理を行う。また、システムは、ホストアプリケーションに対して継続的に、どちらのイベントを送るのか任意にフィルタリングすることも可能である。

【００５６】

利用可能な計測ソースの問題に関して、企業ホストアプリケーションにおける業務プロセスの追跡は、利用可能な計測ソースにより実際のユーザパフォーマンスモニタリングを実行することに関して、非常に困難となり得る。GUIレベルにおいて、複雑なプロセスに関する見識は、標準形式で容易に利用可能な規模の容量よりさらに大きな容量から、より

多くのデータを必要とする。多くのAPIが、企業アプリケーションメーカーにより公開される一方で、大部分は隠され、そして大抵は、いくつかの重要なイベントまたはプロパティが、このような非公開のコンポーネントから取り出されることのみ可能である。これらの非公開のコンポーネントへのアクセスにより、精密性とアクセスの容易さが向上する。

【００５７】

モニタリング可能範囲の問題では、今日の企業アプリケーションの複雑さと規模において、モニタリング可能範囲に対してモニタリングを行うことは、困難な作業である。何百又は何千にも達するモジュールと、データから導出される条件に基づき、動的にユーザインターフェイスを生成可能な大規模なデータベースの置換及び組み合わせの特性により可能となる何千もの操作とがある。基本的な利用をモニタリングするためにですら、システムは、リアルタイムで効率良く、何百もの状態又は条件を検出し、データを捕捉しなければならない。さらに、ソリューションモニタリング実行システムは、企業の要請へ十分迅速に対応しなければならず、またフィードバックとモニタリング可能範囲とを適切に提供しなければならない。

【００５８】

集約されたリアルタイム検出の問題に関して、多数のユーザに関するイベント検出は複雑であるが、限られたリソース環境における効率的なプログラミングを行うことは、取り組むべき課題である。大抵の場合、企業アプリケーションは、データ通信量、通信速度、及び透過的なモニタリングの制限を鑑みて、選択的業務プロセスや操作の処理及び検出が、非常に取り組むべき課題であると理解され得る、多くのイベントを生成することが可能である。

【００５９】

したがって、上記の問題に取り組む本発明は、多くの機構を提供する。

【課題を解決するための手段】

【００６０】

GUIレベルでのコンテキストの問題及び識別に対処するため、汎用マルチインスタンス・メソッド及びGUI検出（gMg）システム内に、GUIイベントのストリームであるユーザコンテキストがあり、このGUIイベントは、環境内で利用可能な他のどのような型のデータともインターリーブされ、他の外部ソース、特にサーバ又は環境インフラ、からのデータと関連することができる。GUI層で検出されるイベントは、ユーザ経験及びアプリケーションのプレゼンテーション層とのインタラクションの直接的なマッピングを与える。このシステムは、GUIコンテキストにおけるすべてを統合的見解として、異なるセンサ、すなわち、メソッドセンサの実施形態により、多種多様のデータソースに対応する能力を有する。特殊な透過性のGUIセンサの使用により、実際の実行時におけるユーザ経験にイベントへの直接対応を提供するため、他のすべてのデータ形式が、GUIイベントストリーム内においてインターリーブ可能である。

【００６１】

gMgシステムでは、アプリケーションのマルチインスタンスに関する問題に対処するため、動的モデルが、リアルタイムにホストアプリケーションの構造と同調され、類似または同一のGUI構造の複数のインスタンスが共通である場合において、複合的なホストアプリケーションのグラフィカルインターフェイスに対するサポートを提供する。動的なGUIオブジェクトツリーは、ホストアプリケーションのGUIオブジェクトをモデルとしている。動的な変数は、GUIオブジェクトのマルチインスタンスをサポートする、GUIオブジェクトツリー内のGUIオブジェクトを選択するために設けることが可能である。

【００６２】

企業アプリケーションの複雑さ及びそれらの配置により、企業アプリケーションに対するどのような変更も非常に遅くなり、仮に可能だとしても、煩雑となる。gMgシステムは、企業アプリケーションのソースコードを変更することなく、企業アプリケーション及びその環境からイベント及びプロパティを読み出すことにより、アプリケーションの透過性に役立つ様々なセンサの実施形態を備える。従って、これらのセンサは、ホストアプリケ

ーションのどのコンポーネントも再コンパイルすることなく、アプリケーションの機能を拡張できる。これらのセンサは、企業アプリケーションのモデルの実装に必要なデータを捕捉するために、最小限かつ侵入しないで介在する。これらのモデルから、解析が行えるように、正確なイベントが生成可能である。

【００６３】

機能性をモニタリングするgMgシステムのメソッドは、利用可能な計測ソースの範囲を大きく拡大する。ホストアプリケーションの環境内において、２つのインターフェイスの種類、すなわち、パブリック及びプライベートの２つが存在する。パブリック・インターフェイスは、外部システムへ組み込むために、公開及びドキュメント化される。アプリケーション・プログラミング・インターフェイス（API）は、プログラミング言語が有する取り決めにより、コンポーネントの機能性を外部システムの外部関数と統合化を行う。プライベート・インターフェイスは、公にはドキュメント化されないが、仮にそれらの動作が発見された場合、変数及び時には重要なイベントまたはプロパティを提供でき、それらすべてがGUIコンテキスト内において構成され得る、計測用の付加的なコンテキストを構築する。以下は、パブリック及びプライベートなコンポーネントの両方を透過的に（例えば、ホストアプリケーションのソースコードを変更することなく）調べる機能と、それらのイベントを単一のプログラム開発環境内において他のタイプのものと統合する能力とを有することにより、計測処理に対してより豊富な多様性を有するセンサについて記載したものである。

【００６４】

GUI層のモニタリング可能範囲に関して、従来技術は、アプリケーションのGUIの広域の選択的な計測と同様、すべての特定のGUIオブジェクトに対応する記述子を手動で実装する必要がある。従って、アプリケーションの効果的なモニタリングを、十分な範囲において効率的に達成するには、適度に少ない操作で、広範な範囲においてモニタリングを行う、より効果的な機構が必要とされる。gMgシステムは、計測に関するソリューションの実装の労力を最適化し、大きく軽減することで、メソッドまたはGUI構造をgMgシステムの汎用の検出により、これを実現する。

【００６５】

GUI層及び他の構造において、可視的なプレゼンテーション層を実行及び描画するために、関連するオブジェクトのセットを有することは、アプリケーションの内部機構の特質である。この内在する機構を利用することで、イベントは、関連するランタイムGUIオブジェクト構造の検出から生成され、そのイベントは、区分されたイベントを発生させ、GUIオブジェクトのセットをモニタリングするために一般的な操作として実行される。このシステムのアプローチの大きな利点は、モニタリングソリューションの開発者が、開発中に、出来る限りの出力を記述する必要はなく、その代わりに、現在から将来にわたりすべての類似構造を検出する、構造化された演算子式を解析することのみ必要であることである。その後、一般的に収集されたデータは、システムの他のソースに組み込まれ、ユーザとホストアプリケーションとのインタラクションの多次元的な視点を提供する。

【００６６】

集約的なリアルタイム検出に関し、検出された第１レベルのイベントを処理する、分散型ローカル及びリモート状態機械の別の実施形態が、gMgシステムにおいて示されている。ソースにおいて、GUIコンテキスト・データのストリーム内で、状態機械は、ローカライズされたイベントの処理、より高次の論理状態の生成、その後の収集サーバへ追跡メッセージを発信することができる。論理状態を受け取るとすぐに、他のリモート状態機械は、他のユーザコンピューターから受信した他の類似した論理イベント型と一緒に、この論理イベント型をさらに処理することができる。この分散処理形態は、大規模ユーザベースに対し、非常に高速な検出速度を提供する。

【００６７】

ｇＭ^{2}ｇ（ここではgMgと称される）は、ローカル（クライアント）又はリモート（サーバ）GUIモデルを用いて、マルチインスタンス化及び並列ホスト・ソフトウェア・アプリ

10

20

30

40

50

ケーションまたは並列ホストアプリケーションの一部をモデル化するための、自動化された特定及び汎用のGUI及びメソッド追跡のシステムである。gMg機能は、対象アプリケーションのソースコードの再コンパイルを要求することなく、透過的に適用される。

【００６８】

ローカルクライアント又はリモートサーバに常駐しているgMgモデルは、ホストアプリケーションの実際の構造を反映する、GUI及びメソッドオブジェクトのフレームワークであり、収集データを再構成し、それを出力解析に解釈するシステム機構である。論理GUIオブジェクトは、実際のGUIオブジェクトのセットを検出するために、イベント及びGUIオブジェクトのプロパティをフィルタリングする、コードコマンドにより追跡される。メソッドオブジェクトは、ホスト・アプリケーション・メソッドのコンポーネントに対する呼び出し及び戻りに応じて、解釈エンジンにより生成されたイベントを検出するコードコマンドで追跡される。その後、GUI、他の任意のイベント、及び利用可能なそれらのプロパティを、コンポーネントから取り出すことが可能である。所定のgMgモデルを使用する動的システムとして、gMg処理は、ホストアプリケーションにおける実在のGUI及びメソッドオブジェクトが存在するマルチインスタンスにリアルタイムで構造を割り当てて適合させる。これらの構造は論理イベントを得るために処理され、ホストアプリケーションとのすべての関連性またはプロパティを事前に取得する必要なしに、個別のコンテキスト内におけるマルチインスタンス構造を反映するために再構築される。

【００６９】

ホストアプリケーションからリアルタイムで取り出されたデータを、オブジェクト、プロセス、及びコードモジュールツリーへ動的に整理することで、gMgシステムは、複雑で多重な構造を、これらに対応する複数のプロパティセット（例えば、テキスト）により処理し、精度の高い解析のためにこれらの構造を相互に関連付ける。報告、警告、又は応答の解析は、様々な形態のクライアント及び／又はサーバ上で実行される接続された状態機械により、マルチインスタンス操作を通して得られる。

【００７０】

単一のホストアプリケーションに関連する複数のインスタンス内での発生を検出するため、GUIレベルからコンポーネントメソッド呼び出しに至るまでの、アプリケーションや、その動作環境内の様々なポイントで現れるイベントカテゴリーを検出する異なるタイプのセンサに対し、異なる拡張可能なコードメンバが与えられる。システムは、特別に又は一般的に、センサから送られたイベントをフィルタリングして解釈する専用スクリプトを解析する、マルチインスタンス・インタープリタ・コンポーネントを備える。イベント及びプロパティの条件に応じて、インタープリタは、追跡メッセージを生成する。また、解釈されたイベントは、ローカルの状態機械のコンポーネント内のストリームにおいて、追跡メッセージを生成する前に任意に解析される。そして、通信コンポーネントは、解釈またはローカルの状態機械の追跡メッセージのいずれかをパッケージして、受信した追跡メッセージをさらに解析するリモート状態機械のコンポーネントに、それらを送信する。また、ディスプレイコンソールへのさらに適宜な送信及び表示のために、リモートの状態機械は、直接メッセージを別の又は他の状態機械へ送ることができる。

【００７１】

レポーティング・コンポーネントは、集約された視点、トレンド、グループ化及びカテゴリーに関する解析を実行できる。解析は、様々な職務のユーザが利用可能なように、グラフ、表、テキストなど様々な形式で表示されることができる。このような職務には、技術インフラ・サポート・グループ、ライン部門管理、ユーザサポート、教育及びその他の多くのことが考えられる。

【００７２】

ここに記載されたシステム及びメソッドは、多くの領域に応用可能である。例えば、実稼働環境に配置されるとき、継続的なモニタリングの必要性がある、ユーザパフォーマンスに関する広範囲のアプリケーションにおいて利用できる。インフラサポートに関して、システムは、応答時間の計測に利用でき、さらにユーザが遭遇するすべてのエラーメッセ

ージを検出することができる。トレーニングに関して、ユーザが困難を感じるのはアプリケーションのどの部分かを、使用情報は検出できる。コンプライアンスに関して、モニタリングプロセスは、ユーザが承認したポリシーの実践から逸脱している箇所を検出して、管理部門（管理者）に警告を発することができる。アプリケーションのセキュリティに関して、１つ以上のアプリケーションにおいて、疑わしいユーザの行為を検出した場合、セキュリティ管理者に警告を発することができる。リソースプランニングに関して、使用情報は、リソースがどのように適用され、ピーク時間のプロファイルをどのように示すかを表す。プログラミング開発に関して、フィードバック、正確な応答時間及びエラーメッセージにより、修正及び新版のより迅速な配信を行うことができるようになる。異なるデータ型にインターリーブし、その異なるデータ型をユーザのGUIコンテキスト内における単一セットとして提示するトランザクションの間において、GUIイベントは両方のトランザクションを計測する。ヘルプデスク・サポートに関して、使用、エラー及び応答時間に関する情報により、問題に関するヘルプチケットのプロファイルが自動生成されて提供されるため、トラブルシューティング処理が促進される。

【発明を実施するための最良の形態】

【００７３】

説明は、以下のように行う。ハードウェア及びGUI環境の簡単な概要を示し、その後、モニタリング対象の異なる型についての議論からgMgモニタリングシステムの説明を始め、続いて配置されるコンポーネントの概要を説明する。その後、イベントモデルを説明し、続いてGUIデータ構造（例えば、gMg GUIオブジェクトツリー）の探索について説明する。そして、マルチプロセス及びマルチスレッド環境におけるgMgエンジンとスクリプトとのマルチインスタンス化の実装方法についての説明を行う。続いて、汎用または特定GUIオブジェクト検出について説明し、その後、gMgメソッドシグネチャとCOMオブジェクト内の仮想関数とを用いる、メソッド呼び出し及び戻りのモニタリングについて説明する。最後に、迅速なリアルタイム検出及びアクションを実行するための、階層的な分散型状態機械を用いる解析について検討する。

【００７４】

図１は、汎用GUIメソッドモニタリングシステム及びメソッドの、実行及び操作可能なクライアントコンピュータ１００及びサーバ１０５の配置の実施形態を示す。gMgシステム及びメソッドは、クライアントコンピュータ１００により実行される、アプリケーション・プログラム又はソフトウェアであり得る。クライアントコンピュータは、パーソナルコンピュータ、ワークステーション、又は他のコンピュータ・プラットフォーム（例えば、パーソナル・デスクトップ・アシスタント（ＰＤＡ）又は、パーソナル電子デバイス（ＰＥＤ）で、ＰＥＤについては、携帯電話、ＭＰ３プレイヤ、さらにビデオデジタルカメラ、及びパーソナル・ビデオレコーダ等を備えることが可能）であってもよい。サーバ１０５は、クライアントコンピュータとは別のコンピュータであり、自身の有するサーバソフトウェアにより区別され、多くのクライアントコンピュータからデータを収集し、かつクライアントのタスク処理を補う。ネットワーク接続機器１１０は、例えばサーバなどの外部コンピュータとの接続を提供する。標準接続プロトコルは、リモートコンピュータ間の、いかなるフォーマットのデータをも送受信する。クライアントコンピュータ１００は、内部システムバスにより互いに接続されたサブシステムを備えることができる。システムＲＡＭ１３０内にある命令は、CPU１５０により実行される。

【００７５】

シリアルポート（図示されず）上のモデムなどの外部周辺機器への接続を行うポート１４０を通して、多くの外部デバイスが、入力／出力（Ｉ／Ｏ）制御装置１１５に接続される。２次元及び３次元のグラフィック及びビデオの表示、又は任意の形式の動画も、モニタ１２５上にグラフィックバッファの内容を表示するビデオアダプター１２０によって処理される。音響処理は、サウンドハードウェア１３５によって実行される。ポインティングデバイス１５５及びキーボード１４５は、表示される視覚情報とユーザとがインタラクションすることで生成されるユーザ入力等を、入力データに変換する。固定記憶装置１６

０は、コンピュータの電源がオフの状態の間データを保持し、必要な場合は、実行時においてもアクセス可能である。

【００７６】

図２は、典型的なアプリケーションGUIである、ウェブ・インターネット・アプリケーションを本実施例において示す。実際にモニタリングされている２種類のGUIオブジェクトである、Win32及びWeb IEコントロールがある。メインウィンドウ２０５は、すべてのサブセクションGUI構造を備える。それらのサブセクションGUI構造は、Webページで構成され、多くのWebページ（すなわち、html要素）を備えることのできるブラウザのページ内容と同様に、コンテナ・アプリケーション（すなわちブラウザ）も有する。コンテナ・アプリケーションであるブラウザは、Win32要素から構成される（例えば、メインウィンドウ２０５、ウィンドウタイトル、メインウィンドウ・コントロール２１５、メニューアイテム２２５、URLインターフェイス・アドレス表示２２０、そしてナビゲーション及び操作アイコン２３０）。さらに、メインウィンドウの要素は、状態インジケータ２３５及びセキュリティ状態インジケータ２４０を備える。メインWebページ自身は、サーバにより配信されるホストアプリケーションのhtmlページを表示し、ユーザインタラクションと情報を読み出すための様々な機能的インターフェイスとで構成される。

10

【００７７】

図２は、ビュー及びフィルタリングの異なる操作を実行する様々な要素を利用した、単純なカレンダー機能を表す。ここで、ビュー及びフィルタリングの異なる操作は、以下の機能を有する。すなわち、アプリケーションのメイン構成セクションの様々なタブコントロール及びアイコン２４５、２５０、プロジェクトビュー２５５、ユーザフィルタリング、プロジェクトフィルタリング及びカテゴリフィルタリング２６５、追加２６０及び検索２７５操作部、日付表示部２７０、グラフィカルカレンダー表示部２８０、そしてプロジェクト状態アイコン２８５である。この実施例は、GUIがどのように２種類のGUIアプリケーション処理と組み合わせされて表示されるかを示す。つまり、gMg Win32及びWeb IEセンサに２種類の対象を提供することである。アプリケーション・パフォーマンスまたは業務プロセスのより精度の高い解析を行うために、ユーザ経験に直接接続する目的で、統計データから、応答時間、コンピュータ・パフォーマンス・カウンタ、メソッドイベントまでの他の多くの種類のカウンタが、GUIコンテキスト内に設置可能であることに対するGUIイベントの重要なストリームを提供するのは、ユーザインターフェイスである。単純なカレンダー機能は、ホストアプリケーションの例として提供される。本発明は、いかなる特定の種類又は特性のホストアプリケーションに制限されるものではない。

20

30

【００７８】

モニタリング対象は、ホストアプリケーション内での任意のエンティティ又は組織的なリアルタイムなプロセスであり、そのホストアプリケーションのプロパティ、イベント、又はアクションは、gMgシステムによる解釈及び解析のために、検出を通じて論理イベントに変換される。ＯＳ又はアプリケーションのいずれかにおいて、プライベート・メソッド又はパブリックAPIに分けられる対象の動作環境内で、潜在的なモニタリング対象が、広い範囲で存在する。各モニタリング対象は、自身の値を有する、または、任意の関連及び注目する領域に関する動作プロファイルの作成に使用可能である、検出可能なイベントを有することが可能である。

40

【００７９】

図３に示すように、ユーザ３０２は、ホストアプリケーションGUI３０４とインタラクトし、ホストアプリケーションGUI３０４は、そのアプリケーションサーバにより送信されるアプリケーション・コンポーネント３０６を順々に制御する。GUI及びコンポーネント層は、すべて順々に、必要なところで、すべての機能を得るためにＯＳのコンポーネント３１０～３２６とインタラクトする。ＯＳには多くの特性があるが、ここでは、カーネル３１２、ネットワーキング３１４、３２２、３２４、CPU、メモリ、プロセス及びスレッドのようなコア操作３２６、ファイル３２０、及び多岐にわたるＯＳコンポーネント３１０、３１８に関連するいくつかの重要な機能を示す。ネットワーク通信、グラフィック

50

表示等の基本的な機能を実行するため、ホストアプリケーション・コンポーネント３０６は、任意またはすべてのこれらＯＳのサブシステムとインタラクトを行う。

【００８０】

動作環境によって異なる標準のサブシステムは、ＯＳメーカーにより、パブリック・アプリケーション・プログラミング・インターフェイス（API）３０８及び３２８～３３６を介して、利用可能になっている。これらのサブシステムは、３５６でgMgシステムに組み込まれ、gMgセンサ３３８に存在する。アプリケーションAPI３０８は、内部情報の幅広いカテゴリー領域からイベント及びプロパティを提供できる。その内部情報は、顧客又はサービスによって、カスタマイズ目的で、アプリケーションの機能を拡張するために、通常、アプリケーション・パートナー用に公開される。ＯＳのAPI３１８は、ＯＳのプロセス、スレッド、メモリ及びメモリ割り当て等から、ＯＳのすべての特性に関する機能を提供する。パフォーマンスモニタリング（”PerfMon”）は、マイクロソフトウィンドウズ(登録商標)で配布される、特別に構築されたAPI３１６である。マイクロソフトウィンドウは、ハードウェア、又はサービスの多くの特性、例えば、CPU利用、仮想メモリ等に関するカウンタ及びデータを提供する。ブラウザ制御API３２８は、ブラウザ・コンテナ・アプリケーションのドキュメント・オブジェクトモ・デル（DOM）に関するプロパティ及びイベントを提供し、ウェブGUIモニタリングアプリケーションをサポートするために使用される。ブラウザ制御に対応するものは、Win32クライアントアプリケーションのGUIをモニタリングするWin32 GUI APIメッセージインターフェイス３３０である。クライアントアプリケーションとされる、Java（登録商標）（Sun Microsystems, Inc. カリフォルニア州サンタバーバラから利用可能）、Visual Basic及び.NET（どちらもMicrosoft Corporation ワシントン州レドモンドから利用可能）等に関するさらに多くのGUI API３３２がある。gMgシステムは、データの収集及び解析処理用の異なるモニタデータソースを作成するために、上記インターフェイスの種類すべてに形成される。

【００８１】

gMgシステムは、アダプタを介して、様々な利用可能なモニタリング対象インターフェイス（３０８、３２８～３３６）にリンクされる様々な種類のセンサ３３８により、データソースの生成及び統合が行われる。これらのセンサは配置され、gMg抽象化層３４８に入力された後そのgMg抽象化層から出力されるイベントである、gMgイベント３５０を検出する。これらの出力イベントは、収集及び解析サーバ３５４に入力される追跡イベントのさらに別の層を生成するリアルタイム条件解析のために、gMgインタープリタ３５２に入力される。解析は、警告から、相関関係、直接のアクションまでの範囲の出力を生成する。gMgシステムの一部として、特別なgMgメソッドセンサ（すなわち、gMgメソッドモニタリング３４０、３４２、３４４、３４６）がある。そのメソッドセンサは、ホストアプリケーションからＯＳまでの範囲のコンポーネント内において対象となるメソッド呼び出しの一般的なモニタリングを行う。メソッドセンサは、適切な条件及びシナリオのもとで、プライベート及びパブリックなコンポーネントからイベント及びプロパティを取り出すことができる。広い範囲で拡張及び適用可能なセンサが与えられることで、gMgシステムは、すべてのユーザGUIコンテキスト内で、ホストアプリケーション及び動作環境のプロセスの包括的なモニタリングを提供するために、多数のソースからのデータを配列可能である。操作中に、GUIインターフェイスを介する直接的なユーザインタラクションのないサーバ環境において、パブリック及びプライベートの両方を含む他のすべてのモニタリングインターフェイスは、gMgモニタデータソースを生成するために有効な状態のままである。サーバ内で（ユーザ３０２なしで）、GUIランタイム層３０３、３０４、及び３２７は、存在しないかもしれないが、しかし、GUIのサーバ管理ツールが存在すれば、GUIのサーバ管理ツールはモニタリングされる。

【００８２】

図４は、典型的な企業環境に配置されたgMgコンポーネントの説明図であり、ユーザ４０２は、ひとつ以上のホストアプリケーション４０４、４０６、４０８に、又はホストアプリケーションサーバ４１０により、配信される同様のアプリケーションのひとつ以上の

インスタンスに、アクセス可能である。

【００８３】

gMgシステム開発ツール４１２は、ユーザのコンピュータ使用環境に配置するために用意されるgMgソリューション４２６を作成及びパッケージすることができる。開発ツールは、GUI及びメソッド検査ツール、コンパイラ、デバッガ、サーバ解析状態機械ツール、管理及び配置パッケージング・ユーティリティ、及び統合開発環境を備える。gMg実行ファイルは、単一のエージェントgMg.Exe４１４及びひとつ以上のEngin.DLL４１６、４１８、４１９、４２０を備える。エージェントgMg.Exe、ひとつ以上のEngine.DLL、及びgMgソリューションの収集は、gMgシステム４３６のインスタンスを備える。図４は、ホストアプリケーション４０６及びEngine.DLL４１９をモニタリングする複数のエンジンを示す。共有メモリ機構４４２は、複数のエンジンにわたりリソースの調整及び共有を行う役割を有する。gMgシステムは、必要であれば、ホストアプリケーション内において、複数のプロセスを動的に追跡し、センサ４３８、４３９、４４０は、収集データ用ユーザコンテキストを作成するために、ホストアプリケーション内において、メソッド呼び出し又はGUIイベントを検出する。システムコンテキスト（例えば、メソッド呼び出し、メソッド戻り及びメッセージ）を有するユーザコンテキストは、ホストアプリケーション及びシステムの環境の残像であるスナップショットを作成するために捕捉される。

【００８４】

エージェントgMg.Exe及びエンジンは、デスクトップ・コンピュータ、サーバ（GUI又は非GUIモードの両方）であれ、又はターミナルサーバ（ラスタライズ化された表示データを、ユーザに配信するターミナルサーバ上で、クライアントコンポーネントを実行するサーバ）であれ、目的の環境に配置される。開発段階で使用されるオプションのデバッグDLL４２２、４２４は、リアルタイム・デバッギングに関して、ブレイクポイント、変数表示、トレース、及び他の一般的なデバッグツールの機能を提供する開発ツールと連動して使用される。小型で、簡単かつ動的にアップデートされるgMgソリューション４２６，４２８、４３０、４３２、及び４３４は、個々の又は重複しているモニタリング対象を定義及び記述して、追跡動作を実行するコンパイルされたスクリプトで構成される。これらのソリューションは、異なるEngine.DLLのインスタンスによって実行され、必要であれば分離又は重複ですら可能なモニタリング対象を追跡できる。GUI又はメソッドイベントが検出され次第、追跡メッセージが作成され、通信モジュール４４４によりサーバ及び追跡データベース４４６に送信される。収集サーバは、利用可能な要件及び計算リソースに応じて、リアルタイム、ほぼリアルタイム、又は後で、解析を実行する。処理及び解析モジュール４４８の出力は、警告４５０、外部データソースとの相互関係（例えば、バックエンドのモニタリング統計４５２）、又はレポート４５４を備える。レポート４５４は、真のGUIユーザのコンテキスト内におけるアプリケーションのプロセスとインタラクションに関する、グループ又は詳細な個別の動作を明らかにする集中処理されたデータを持つ多種多様な観点からのものとなり得る。企業において、技術サポートから、トレーニング、インフラサポート等までの様々な職務４５８は、改善及び是正措置４５６の実行、又はエンドユーザのサポートを行うために、レポート情報又は警告へアクセス可能である。

【００８５】

図５に示すように、ユーザ５０３は、ホスト・アプリケーション・サイト５０６、それに関連するコンポーネント５１２、及びホストアプリケーションのプレゼンテーションGUI５０９の１つ以上のインスタンス５４５を有する、デスクトップ又はターミナルサービスにおける１つ以上の並列ホストアプリケーションとインタラクトする。ホストアプリケーションの要素をプレゼンテーション層内に置くことができ、それにより、ユーザが動作を呼び出して、アプリケーションからの応答を受け取ることができる。ホストアプリケーションは、１つ以上のインスタンスを並列で実行してもよく、又はホストアプリケーションのサブセットである、GUIのセットのマルチインスタンスを並列で動作することができる。

【００８６】

10

20

30

40

50

GUIは、メモリ内に階層的に組織化され、実行時に、ユーザアクション、及びGUIオブジェクト構造がどのように生成されるかを制御するアプリケーションの組織化論理に応じて、複製される多くのGUIオブジェクトのサブセットを有する。プレゼンテーション層の他に、アプリケーションも、多くのコンポーネント又は機能（例えば、サーバとの通信、計算、動作環境プロパティの取得、コンピュータのＯＳとのインタラクション等）を実行するDLL５１２を備える。コンポーネント・ライブラリは、機能を実行するメソッドを備え、メッソドイベントのgMgの検出は、プロパティ、入力、出力、イベントへのアクセス、又はホストアプリケーションのサポートにおけるコンポーネントの機能に関連したプロセスのその他の情報へのアクセスを提供する。メソッドイベントは、GUI、又は幾つかの他の内在するホストアプリケーション機能を提供するライブラリ・コンポーネントから検出される。

【００８７】

gMgシステムは、解析をサポートするために、GUIオブジェクトプロパティ５１８及びオブジェクト状態情報を利用する。１組の対のネーム値は、他のオブジェクトとのそれらの関連と同様に、常にオブジェクト状態情報を反映する。プロパティの値は、ポーリングを用いて、常に、プログラムで読み出される。オブジェクトは、スクリプトを介してセンサにより定義及び制御され、GUIオブジェクトプロパティと対応付けられたカスタムプロパティも、有することもできる。付加的な状態情報は、状態と先行イベントの組み合わせから得ることができる。ホストアプリケーションのプロセスフローに対するgMg検出エンジン５４８は、一組のリアルタイムイベント５１５を発生することができ、そのリアルタイムイベント５１５は、ユーザ、アプリケーション、ＯＳイベント、GUIオブジェクト及びメソッドプロパティの特定値から派生するイベントである。それらのイベントは、ある時刻において、又はアプリケーションの現在の状態（すなわち動作環境スペクトラム）において、満たされる条件を反映する。これらのイベントは、幾つかのカテゴリーの中にある。すなわち、コンピュータマウス又はキーボードなどのような入力装置で発生するユーザイベント、ユーザアクションにより間接的に派生するイベント、アプリケーションにより自立的に生成されるアプリケーションイベント（例えば、最新ニュースに伴うウィンドウの更新）、ファイル操作、登録アクション、又はネットワークなどにより生じるＯＳイベント、そして、特定のコンポーネントの関数をモニタリングするためにセットアップされるメソッド呼び出し及びリターンイベントである。付加的なイベントは、詳細を示すか、又は、さらにイベントを限定するプロパティを有することができる。この低レベルのイベントの入力から、論理イベント５２１は、状態機械を介して、状態を決定するため、条件付き論理処理を用いて任意に導かれることができる。論理的及び未処理イベントの両方が、通信モジュール５２４に到達する時、それらをリモート収集サーバ５５７に送信する準備が行われる。このモジュールは、送信前に、バッファリング及び圧縮、又はセキュリティを制御する。バッファリングは、また、収集、ネットワーク、割り込みのイベントにおいて、または非ネットワーク環境において、データを一時的に保存する機構を備える。

【００８８】

クライアント５４８とサーバ５５７間で、情報交換が行われる。クライアント側の通信モジュールは、論理イベント値か、選択されたGUIオブジェクトのプロパティかの、どちらか一方を配信し、またサーバからの制御命令も受信する。収集・解析の段階においては、処理可能なまたはデータベースに収納可能な追跡メッセージの形式で、収集サーバーインスタンス５５７へデータを送信する。イベントストリームは、解析パラメータにより動的に決定される、分離されたユーザ論理コンテキストに分解される。

【００８９】

GUIオブジェクトのプロパティとしてデータが到着するか否かに応じて、より低レベルのイベント、または、論理レベルのイベントにおいて、そのデータがどのルート５５４を通過したかが決定される。より低レベルのマルチ・イベントとして到着した場合は、さらに状態機械の処理により、論理イベント５３０とコンテキストが生成される。しかしながら、処理済の状態機械の論理イベント５２１として、そのデータが到着した場合は、状態

機械５２７の処理により、新しいコンテキストの生成が可能である。出力として、汎用検知または特定検知のどちらを通過するかにより、汎用イベントセットまたは特定イベントセット同様に、GUIオブジェクトのセットも形成される。さらに、解析により、所定の計測と状態特定パターン５３６が終了し、処理済のデータのセットが生成される。この処理済のデータは、集合、トレンド、及びグラフィック表示用レポート処理を行うため、さらに、gMgレポートコンソールでの使用のために、解析及びレポート段階５３９へ送られる。

【００９０】

広範囲のユーザ向けに、このコンソールは、変更制御、プログラム修復、サポート、デバッグ、最適化作業等々の目的のために使用される。また、誤りまたは応答時間遅延などの、より高い優先度を要求する閾値条件を有する表示と通信とを備えるシステムのアクション・コンポーネント５４２へ、処理済のイベントは、直接送信可能である。アクションは、サポート・チケットの発行、自動修復、トレーニング、或いは、最終的にユーザに影響を与えるその他の定義されたアクションなどの機能を提供する外部システムとの通信を行うことが可能である。

10

【００９１】

図６を参照すると、gMgシステムは、いくつかの主要な種類のイベントソースを使用する。汎用イベント６１０内には、API、フック関数、及びコールバック経由で読み出されるウィンドウズ３２ＯＳのイベントが、存在する。ユーザ、アプリケーション、ＯＳ間の、インタラクション処理の様々な態様の相互作用処理への、アクセスを可能とする、多くの種類のメッセージ、プロパティ、関数に対するアクセスが、これらのイベントにより可能となる。イベントの別のソースは、ブラウザ（例えば、インターネット・エクスプローラー・イベント）からでもよく、コンテナ・アプリケーションが、リスナー及び利用可能なCOMオブジェクトにより読み出される、外部処理用のドキュメント・オブジェクト・モデル（DOM）などのデータモデル及びデータ型に関するイベントとプロパティとを公開する方法を、ブラウザは、裏付けしている。例えばIEなどのウェブブラウザは、複雑にマルチインスタンス化されたウィンドウ構造を示す最近のインターネット・アプリケーションGUIの動的な特性の、良い例である。gMgシステム内で、COMリスナーが有効化され、コールバック関数(IEEvents.cpp)を使用して、IEイベントを検出する。クラスがIEにより登録され、COMインターフェイスを経由して与えられるIEオブジェクトプロパティを読み出すことでイベントが発生する場合はいつでも、そのクラスにおけるメソッドが呼び出される。イベントの種類の別の例は、コール・リタン・メソッド・イベントであり、このイベントは、gMgセンサモニタリングコンポーネントにより作成される広範囲のセンサイベントを提供するものである。

20

30

【００９２】

これらのイベントは総て、解釈されるgMgスクリプト６２５へ送られる。インタープリタに到達する前に、これらのイベントはロック機構６２０を使用して、シリアライズされる。ロック作用は、さらに、エンジン及び／またはスクリプト６２５、６３０、６３５の異なるN個のインスタンスに分けられる。一旦、スクリプトが分析されすでにシリアライズされた処理済のイベントであれば、関数は、ホストアプリケーションＧＵＩ６０５に対して断続的に行うポール操作（アプリケーション・スキャン６４０を経由して）を実行する。同時に、タイマーコールバック手続き６１５を経由して到着する、ＯＳのタイマーイベントが存在する。この手続きは、設定・調整可能な設定速度で、ホストアプリケーションＧＵＩ６４０のスキャン・スケジュールを決める。この処理が発生する場合、分岐記録構造６５５、６６０、６６５ から構成されるgMg GUIオブジェクトツリー６６０は、そのGUIのアプリケーション状態のシステムの現在の状態のスナップショットに、更新されるか、削除されるか、或いは、コピーされる。gMgスクリプトは、また、gMgのWhenObjectコマンドに対しての優先度は低い状態で、処理される他のタイマー６４５を設定する。WhenObject処理６５０は、ホストアプリケーションとＯＳにより、全くイベントが生成されることのない６４０における同期を要求するGUIオブジェクトに関するgMg生成・破棄イベン

40

50

トを作成する。GUIオブジェクトの特定のセットにポーリングするために、WhenObjectは、gMgのシミュレートタイマーを与える。

【００９３】

別のモニタリング処理のgMgセットは、アプリケーション環境からＯＳにわたり、広範囲のソースカテゴリを提供する。追跡メッセージとそれ以後の解析作業に関する詳細を抽出して記録するために、プロパティ読み出しを行う読み出し専用操作に関するインタラクション速度は、制御可能である。

【００９４】

以下に、GUI、マルチインスタンス化されたプロセス、及び起動処理中に立ち上げられる、ランタイム・スクリプト処理のサポート機構に関して説明する。gMgの特徴は、GUIオブジェクトツリー、マルチスクリプトのロード及び組織化、ホスト・アプリケーション・マルチインスタンス処理（例えば、オブジェクト変数、とWhenObject指令）機能を備えたスクリプトに続き、コンテキスト、フック、そして最後に、GUIオブジェクトセットの汎用検出の提供を備える。

【００９５】

gMgシステムは、配置に関して、多くの異なる方法で構成される。クライアント／サーバ構成に、ローカル・エージェントとそのエンジン及びソリューションとをインストールし、ローカルコンピュータ上で実行することで、イベントを検出できる。ウェブ環境においては、gMgソリューションコードを、サーバにインストールして、gMgエージェント及びエンジンがすでにインストールされている、ユーザのローカルコンピュータへ動的に送られる。これらの配置のうちの一つは有益であるが、この双方は、共通構造に基づいてイベントを検出し、必要に応じて、特定のプロパティも取得可能である。gMg配置構成は、上記には限定されず、特に、gMgシステムが動作可能である限り、特に他の構成も本発明の範囲内に入るものである。

【００９６】

一旦、gMgシステムが配置されると、起動開始時に、コマンドラインは、ロードスクリプト、停止等のオプションについて分析され、メッセージキューとの通信状態を確立する。スクリプトまたはスクリプトのディレクトリを有することが可能な.iniファイル内の実行オプションが、調べられ、メイン分析コマンドラインのルーチンに送られる。.iniファイル内のスクリプトのリストは、スタートアップ時に実行でき、マルチプロセスへ挿入可能である。ChangehookTidL関数への呼び出しにより、gMg.exeが、自身のウィンドウを経由して、自身の中にエンジンを挿入し、あたかも他のアプリケーションをモニタリングしているかのごとく、自身が有するインタープリタ（gMgエンジン）を実行させる。また、別のアプリケーションに挿入されたものと同一のAgent.exeファイルに関連づけられた、別のgMgエンジンに対するgMgプロトコルと共に、ウィンドウに関連づけられたgMgは、エンジン間通信としての機能も実行できる。

【００９７】

ＯＳ内のgMg処理を登録する特定のウィンドウズ・メッセージを構築・登録するために、ファイルパスが使用される。gMg.exeモジュールのフルファイルパス名が、使用され、さらに、同時に実行される、他の総てのインスタンスと、コピーと、エージェント及びエンジンのバージョンとを識別するために、ウィンドウズ・メッセージ用の個有識別子になり、この結果、動的で個有なインスタンスが実行可能である。インスタンス間の識別能力により、多数のディレクトリにインストールされたgMg.exeが、正確な同時処理を目的とし、自身のランタイム構造を構築するために分離可能である。３つ以上のホストアプリケーションが実行されており、並列で２つ以上のgMgエンジン・インスタンスが実行中で、双方のエンジンが同一の要求（例えば、エクスポートリスト）をする場合、ファイルが開かないことや、一方のファイルが他のファイルに上書きされるなどの、プライマリ・ディレクトリにおけるファイル衝突が起きる場合がある。この問題を解決するためには、ファイル名がインスタンスに付けられることが好ましい。このことが、動的なマルチインスタンス・サポートの基本の一部となる。この条件下では、同一名の２つのファイルが、同じ

位置に存在することはあり得ない。

【００９８】

　起動時では、フックを使用したgMgエンジンモニタリングセンサをセットアップすることが、目的であり、アプリケーションがウィンドウ・メッセージを受け取る前に、フックにより、このウィンドウズ・メッセージのインターセプトを実行する必要がある。フックは、スレッドレベルにあり、スレッドレベルでは、１スレッド当たりに１つのフックが存在する。ウィンドウ・メッセージ用のスレッドをモニタリングすることにより、gMgのDLLが、１プロセス毎に挿入される１つのgMgのDLL（CALLWNDPROCを経由してGETMESSAGEがSetWindowsHookExをフックする）が存在する、関連したプロセス空間に挿入される。gMgスクリプトは、必要に応じて、他のプロセスをフックして、このDLLをこのプロセス空間に挿入する。ハンドルを与えられて、モニタリングされているオブジェクトに属するスレッドオブジェクトが読み出され、スレッドオブジェクトは、スレッドをフッキング、或いは、アンフッキングするための、インスタンス（すなはち、スレッドid）を見つけるために、スレッドオブジェクトを導出する。このスレッド・フッキングをさらに制御するため、ChangeHookTid LockedとChangeHookTid Unlocked関数のバージョンが複数存在する。Lockedバージョンは、ロックの設定が必要な場合に、使用される。ロックが、すでに設定されている場合は、Unlockedバージョンの関数が、呼び出される。共有メモリが初期化された後、起動コードもloadDBGDLLを呼び出し、指定ディレクトリに（オプションとして）デバッグのDLLが存在すれば、それを同一のプロセス空間にロードする。通常は、１つのアプリケーションには、多数のスレッドが有るので、スクリプト・インタープリタを使用する場合は、gMg gMglockは、ロックされたプロセスを実行し、総てのイベントを、強制的に一本のスレッドにする。アプリケーションにおける２つのスレッドが、gMgが関係するイベントを発生しようとする場合、或いは、異なったソースからのＮ個のイベントがある場合は、エンジンは、イベントのシリアライゼーションを強制する。シリアライゼーションとは、順々に処理されたイベントのことである。インタープリタのオペコードである実行コードは、先ず、gMglockによりロックされ、次に、各イベントを処理するために、DogMgEvent()に対する呼び出しが実施される。

【００９９】

　gMg.exeにより呼び出されるLoadProgram関数は、スレッドid、スクリプト・ファイル名とオプション（ロード/アンロード）を渡される。gMgシステムは、ワイド文字を解析して、ファイル名をユニコードに変換する。特定のスレッドidが、アスタリスク「＊」付きのChangeHook()を呼び出す場合、総てのフックが取り外される操作を意味する。このシステムは、エンジンのマルチインスタンスをサポートするので、複数のスクリプトの実行が可能であるが、２つ以上のスクリプトを呼び出す時は、このシステムは、スクリプトの起動可能数を制御する（アスタリスクのワイルドカード付きの起動スクリプトは、許可されない）。スクリプトにワイルドカードが付いていない場合、フッキングの設定が、開始され、フッキングに渡されるスレッドをフックする。次に、Changescript関数は、ファイル名をフッキングに渡すスレッドに、メッセージと、以下の可能なオプションである、スクリプトのロード／アンロード、直接のプログラムのロード、実行中の全プログラムのリストの検査、アンロード・オプションを有する１つのスクリプトのロード、或いは、スクリプトのリロードのいずれかと一緒にスレッドidを、送信する。

【０１００】

　一旦、スクリプトが実行すると、Do_gMgEvent()関数は、イベントにオブジェクトを渡し、すでにロックが掛かっている場合には、この関数は、処理中のイベント数を計算する。最初のイベントがこのスクリプトにより処理される前に、図７と図８のGUIオブジェクトツリーは、起動時に構築されなければならない。好適には、全スクリプトが処理されるが、しかし、アンロードフラッグが設定されている場合は、スクリプト終了追跡メッセージの送信などの、スクリプトのアンロード操作が行われる。しかしながら、スクリプトの初期化が未だ行われていない場合は、起動追跡メッセージ（ヌルイベントがゼロのオブジェクト）が、このスクリプトに最初に送信され、存在する総てのオブジェクトに対するse

nd Existイベントと共に、そのスクリプトを初期化する。この動作により、イベントに対するリスニングが開始される。スクリプトの削除、初期化、或いは、イベントへの渡しの何れか１つに代わり、総てのスクリプトに対して、この起動ループは、順に開始する。

【０１０１】

センサとエンジンの同期化のメカニズムは、総てのgMgメイン実行可能モジュールのウィンドウのハンドルを有している共有メモリの集中ブロックである。gMgコードは、実行可能モジュールの終了を確認するか、或いは、その他の実行中のモジュールがある場合と、実行可能なウィンドウがもはや存在しない場合とには、フラッグがクリアされる。状態チェックが実施され、その他のフラッグが設定されている場合は、その状態と、他のgMg.exeがバックグラウンドで実行中であるか否かについてとが、オプションとしてメッセージ・ボックスに表示される。実行可能モジュールが他に検出された場合は、WM_COPYメッセージを使用して、gMgは、データブロックとして、総てのコマンドラインを他の全エンジンのインスタンスに渡し、存続する。しかしながら、他のインスタンスが実行されていない場合は、処理作業は進行し、メインウィンドウを作成する。決して表示されることのないgMg.exeのメインウィンドウは、gMg.exeがアプリケーションとして自身のモニタリングが可能となるメッセージを送るために、使用される。共有メモリに保持されているハンドルは、gMg.exe自分自身のインスタンスを示すように設定される。

【０１０２】

終了段階では、gMgマルチインスタンス処理は、まだ必要である。シャットダウン時において、DLLProcessDETACHへの呼び出し実行中は、他の動作が未だ非同期の状態で実行中であるので、このgMgマルチインスタンス処理におけるフラッグにより、現在実行中のタスクがそれ以外の動作を実行しないように抑制される。ウェブアプリケーション中のCleanIEthread()を呼び出す関数Cleanthread()は、切断と、任意のコールバックまたは解放とを、呼び出すか、或いは、IE全体に残されているCOMオブジェクトを、一掃する。DLLProcessDETACH（と、これに対応するDLLProcessATTACH）は、複数のインスタンス化と、動的なホストアプリケーション環境と、をサポートするように、再帰的に用いられる。

【０１０３】

ホストアプリケーションのプロセス毎に、少なくとも１つのgMgスクリプトが存在する。また、複数のgMgスクリプトが、１つのプロセスをモニタリングできる。スクリプトは、プロセスの総てのスレッドを扱えるように設計されているが、ウェブアプリケーションにおいて普通に見られるような１つのスレッドに１つのウィンドウで構築されているアプリケーションで見られる任意のスレッド数を、１つのスクリプトでモニタリング可能である。このスクリプトは迅速に処理され、良好なパフォーマンスのために、属性読み出しは、大量にキャッシュされる。プロパティをフェッチする、或いは、同一のスレッド内で安全に属性を取得するために、負荷がかかる場合はいつでも、キャッシングが強制される。

【０１０４】

クロススレッドのアクセス中、或いは、マーシャリング動作が不明のCOMオブジェクトから、スレッドがアクセスされる場合、特にIE環境で実行中である場合は、問題の発生が見込まれる。しかしながら、COMオブジェクトのクロススレッドのアクセスに関する問題は（殊に、IE環境において）重要ではなくなる。その理由は、gMgイベントは、特定のスレッドにより発生し、そのスレッドに関係するオブジェクトへのみをアドレスし、異なるスレッドに常駐しているオブジェクトに反応することは稀であるためである。異なるスレッドからのオブジェクトを処理する場合、gMg処理は、他のスレッドのオブジェクトをアドレスする前に、現在のスレッド処理を終了させせねばならない。２つのスレッドが、共通の変数を共有し修正する場合などの、不確実性を避けるイベント処理を連続化することにより、衝突が避けられる。本技術分野で知られているように、gMgにおける処理は、共通の並列問題の原理により、導かれる。

【０１０５】

シリアライゼーションの原理は、COMまたはnon-COM（コンポーネント）アプリケーションなどの双方のコンポーネント環境に適用される。COM環境内で、インターフェイス・メ

ソッドは、多くのスレッドに呼び出される。スレッド化モデル（例えば、セーフスレッド、単一スレッドなど）には関係なく、スレッドのモニタリングは、gMglockによりシリアライズされる。IEイベントの処理は、スレッド毎に行われるが、gMgスクリプトは、決定論的である必要があり、従って、イベント毎にシリアライズされる。スレッド毎の処理は、関連するイベントにとって重要であるが、その関連性は、gMgスクリプト・コンテントにより決定される。gMgシステムは、イベント処理に対しては、単一スレッドを使用し、これを任意のイベントに拡張する。呼び出しために渡されるパラメータは総て、イベントのプロパティを読み出すために、後ほど使用されるポインタを有している。スクリプトがイベントのプロパティを要求する場合は、スクリプトは、イベントパラメータに対するグローバルポインタの内の１つを使用して、イベントの詳細を読み出す。このメカニズムは、読み出し作業を集中化し、過剰なスタックのプッシュ／ポップ操作の原因となる過剰な位置に亘って、読み出しが分布することを防止する。イベント及びスクリプトのシリアライゼーションは、効率的でなくてはならない。例えば、１つのアプリケーションGUIが、単一のスレッドにより処理される場合は、gMg検出が容易に処理されるが、しかし、１つのアプリケーションが、それぞれ自分自身のスレッドを有する多くのウィンドウを備えている場合は、スレッドの衝突が起こり得る。他のスレッドBが、グローバルgMglock上で、強制的に待機させられている状態で、１つのスレッドAに１つのイベントを処理してもらうことによって、この衝突の可能性は避けることが、好ましい。メッセージは、スレッドAにより処理を許可されるので、スレッドBは、gMglockが解放されるのを待つのである。

【０１０６】

gMgシステムは、ロック機構を使用して、発生したイベントのシリアライゼーションを構築する。ロック・マクロであるBeginLockは、パラメータのインターロック交換を行い、BeginLockが０を返す場合、すなわち以前の値が０の場合は、ロックが実行される。スレッドidが設定され、呼び出し元の位置は、ロックが使用された場所に、ロックを行っているスレッドidを転送することにより決定される。BeginLockは、そのスレッドを記録することで、別のマクロgTidは、gMglockに格納されているスレッドidを使用する。gMgが設定されたことが一旦確認されると、そのロックに格納されているスレッドidを、フェッチする必要があるために、gTidは、非常に高速である。

【０１０７】

デバッグのデッドロックに対して、beginLockDは、再帰的ロックが行われる場合は、特定のアクションを処理することができる。単にタイムアウトの実行または不注意で失敗するのとは対照的に、再帰を実行する異なるアクションで、あるレベルの再帰呼び出しが許可されるある箇所において、BeginLockDは、再帰的なロックのための特別なアクションも処理することができる。BeginLockDは、インターロックの交換を行い、スレッドidを格納する。gMgシステムのロック機構は、クリティカルセクションであるが、ロックが共有メモリに存在する場合には、プロセス全体に亘って動作し、衝突が起こらない場合は、非常に高速で動作する。衝突が起こる場合に、スレッドidが、現在のスレッドidであるのかどうかを検討するためにテストを行うQuicklockを実行するための呼び出しを、gMg処理は強制的に行う。このスレッドidが、現在のスレッドidである場合は、統計上の理由で、再帰呼び出し数のカウントが行われ続け、－１を返す。そうでなければ、再帰呼び出しは、フラグが立てられ、衝突カウントが増加する。制御ループが次に実行され、制御ループでは、ループは1ミリ秒間スリープし、次に10ミリ秒のタイムアウトで、ウェイクアップする。ロックがグローバルgMglock上で行われたかどうかの、チェックが行われ、実行中の命令のループカウント（すなわち、スクリプト・カウント）が存在する。多数のパラメータがフェッチされ、総てのオブジェクトの総てのプロパティがフェッチされる場合において、gMgスクリプトが大きければ、フェッチのタイムアウトは、デフォルト設定のタイムアウト時間から延長される。gMglockとスクリプトコードに対する待ちの実行中である場合、Quicklockは、衝突が存在する場合にのみ、呼び出される。時折、再帰的ロック（関数BegLockD）が起こり、自己診断のために、メッセージが出される。Quicklockは、ロックアウトにおけるタイムアウト数のカウントにおいて、完全にタイムアウトする場合、０を返

す。ロックに成功した場合は、関数は１を返し、自身のスレッドにより、すでにロックされている場合は、関数は－１を返す。スレッドid（tid）を、パラメータとして、Quicklockに渡すことができ、その結果、スレッドidの再度の読み出しが必要でなくなる。インタープリタが、未だ、スクリプトコードを実行中に、一部の他のプロセスがロックされたスレッドを有している場合、このコードは、自動的にgMglock_timeoutメッセージを拡張可能である。診断のため、Quicklock構造は、ロック、スレッドid、衝突カウント、タイムアウト、再帰呼び出しカウント、前回のロックの行番号、及びモジュール・ファイル情報に関する情報を有している。

【０１０８】

コードのブロックならびにインライン表現に関する診断を行うgMg DbgDLLによって行われるデバッグ診断を、gMgLockは、制御することができる。DbgDLLは、ランタイム分析用に出される診断要求の列挙を渡す診断用の関数を有している。診断サポートに関して、gMgLockによるロックの設定が不成功の場合は、そのロックは、フェイルセーフ機構として動作し、その処理により診断用の関数を停止し、診断用関数の優先度を低くする。

【０１０９】

実際のホストアップリケーションからのイベントをモデル化するために、gMgシステムは、３つの主要なツリーを有することができる。１つは、ウィンドウ及びhtml要素などのGUIオブジェクト用、１つは、プロセス及びスレッドオブジェクト用、もう１つは、コンポーネント・モジュール及びメソッド用である。gMgシステムは、また、タイマー・オブジェクト・リストを有している。

【０１１０】

図７は、GUIオブジェクトツリー715を示している。このツリー７１５は、収集された総ての追跡データ用のGUIコンテキストを提供する根幹である。GUIオブジェクトツリー７１５は、総ての実行中のスクリプト７７５により共有されることが可能であり、各スクリプトに対してその状態を維持する。図７に示される３組のGUIオブジェクトが存在する。すなわち、セットP７２５、セットQ７２０、７３５、７４０、セットR７３０である。以下は、ホストアプリケーションのＧＵＩ７０５から取り出された動的構造を有するGUIオブジェクトツリー７１５を、更新する処理の説明である。

【０１１１】

UpdateTree関数は、ホストアプリケーションのGUI、或いは、他のプロセス及びモニタリングされている対象をモデル化する。gMgスクリプト記述エンジンに、イベントを与えるためには、動作環境において動的に変化するオブジェクトの正確な検出が必要とされる。これを実現するために、UpdateTree関数は、自身の内部構造に対して、総ての主要な更新作業を行い、制御スケジュール内における新旧のオブジェクト間の差異を、適切にモニタリングする。

【０１１２】

スケジュールの実行に関して、UpdateTree関数は、ホストアプリケーションとのオブジェクトの同期速度を制御するための多くの機構を有している。UpdateTree関数は、自身のタイマーを有し、過剰処理を回避する。このタイマーは、正常にツリーを走査するのに要する時間の関数である時間間隔で、動作する。アプリケーションが動作中であり、またgMgがインターセプトする任意のイベントにより、UpdateTreeコードが依然実行中であるため、タイマーイベントが開始されなくても、イベント検出中に、UpdateTree関数は呼び出される。処理の優先度は、より高いレベルに上げられ、アプリケーションがブロックされているように見える場合でも、UpdateTreeに、更新オブジェクトを走査させ、情報を獲得する。その間、UpdateTreeは、また、１つのスレッドを無視して、他のスレッドからの総てのウィンドウを処理する。例えば、各スレッドに対して１秒間に２回のポーリング速度で、３つのスレッドの実行している場合、パフォーマンス向上のために、１つのスレッドのみが選ばれて、その他のスレッドからのウィンドウをチェックする。オブジェクトのフィルタリングが、スレッドレベルで行われ、gMgスクリプトによって、ポーリング速度は、さらに調整可能である。さらに、オブジェクトツリーを取り除くような、或いは、現在

のスレッドだけに処理を限定するのとは対照的に全てのスレッドを処理するような、他の選択肢が多数存在する。

【０１１３】

gMgシステムは、ホストアプリケーションの動的な特性に対処するために、常に変化し、破棄及び作成されるGUIオブジェクトの状態変化を追跡する追跡機構を、有している。図７を参照すると、これらの変化条件に対応するために、破棄されるオブジェクト７６０のリストを保持する。gMg処理は、このメッセージを破棄し、スクリプト・インタープリタ７４５に送られるイベント７７０を生成する。旧リストからオブジェクトが移動されるときにオブジェクトが未だ存在する場合、SList７６０は、旧オブジェクトのリストを有している構造であり、オブジェクトは、新リスト７５５の最後尾に加えられる。以前のオブジェクトセットは、旧リスト７５０に存在するが、しかし、以前には存在しなかった新オブジェクトが、新リスト７５５に追加される。ポインタが、ツリー７２０の最上部において実際に存在するウィンドウの組に対して設定される。この設定の状態は、旧設定に変更され、次に、ツリー７２０の最上部が空にされ、この点から開始されて新リストが作成される。

【０１１４】

Enumwindows関数は、総てのウィンドウをスキャンして更新を実行し、旧リストを発見し、アプリケーションGUIをスキャンする必要があるのと同様に、新リストの中に旧リストを挿入し、或いは、アイテムがすでに存在しない場合は、ウィンドウ・アイテムを破棄リスト７６０に挿入する。次に、旧リストに残っているものは総て処理され、旧リストに残っている破棄されたどのウィンドウに関しても通知を受け取る。更新処理は、Enumwindowsproc７１０において行われる。コールバック関数Enumwindowsprocは、各トップレベルウィンドウ７８０に対して、Win32 APIにより、呼び出される。Lparamは、SList構造に対する、ポインタである。ウィンドウスレッドのプロセスidもまた、読み出される。この関数は、ウィンドウスレッドのプロセスidを取得し、旧リストを調べて、ウィンドウが存在するかどうかを検討する。

【０１１５】

オブジェクトが旧リスト中に存在するか確認される。オブジェクトツリー構造へのポインタのセットを有する旧リストが読み出され、未だ存在しているオブジェクトは、必要に応じて、新リストのメンバになるように、マークされる。その後、プロセスオブジェクトまたはスレッドオブジェクトは、ウィンドウ用に処理される。特定のスレッドに属する新しいウィンドウが検出された場合、システムは、問題となっているスレッドに関する、プロセスオブジェクトまたはスレッドオブジェクトの双方を作成する。既探索フラッグは、状態インジケータとして使用され、ウィンドウまたはスレッドにおける所定のプロセスに、オブジェクトがリスト(リストの一部として)されている場合にマークする。呼び出しフラッグが、確認され、破棄イベント７７０を経由してオブジェクトを終了するスクリプトに対する課題に使用される。

【０１１６】

通常の状態であるツリー構造に変化が何も生じない場合において、コードは効率的に動作する。変化が全く無い場合は、このコードは、このオブジェクトを切り離し、そのスレッドに対するフラッグ及びプロセスを更新し、ウィンドウを新リスト（例えば、ウィンドウが、旧リストから削除され、新リストに入れられる）にリンクする。このスレッドを使用して検査されたウィンドウは、現在のレベル以下のウィンドウはすべて検査されることは無い状態のレベルと同じである。この機構は、別のスレッドに属するどのトップレベルウィンドウをも跳び越して、一度に、１つのスレッドの処理のみ行う。好ましくは、この処理が制限されないならば、１つのスレッドの処理は、結果的になくなる、すなわち、自身の総てのスレッドを処理するために移譲されることになる。

【０１１７】

図８を参照すると、オブジェクトツリー８３２における動作が説明されており、次に、スクリプト８１２と同一のインターフェイス８０８を使用するウィンドウ８０６のクラス

がフェッチされる。このクラスは、gMgキャッシュ８１０にすでに存在している場合は、実行中のアプリケーション８０４から読み出される必要はない。読み出されたクラスがIEのクラス（ウィンドウの）に適合したものであるならば、このクラスは、オブジェクトツリーの構成要素に属する特定のスレッド（と、現在のスレッド８１４）からのものである。このオブジェクトツリー８３２におけるオブジェクトは、関連するスレッドidを有するため、このオブジェクトツリー８３２の各枝の部分は、自身のスレッドがオブジェクトツリーのオブジェクトのスレッド８４４に適合するウィンドウによって、使用可能である。

【０１１８】

関数notifydestroytreeは、破棄リスト８１６を作成し、旧リスト８２０を破棄する。この処理が、nottifydestroytreeの呼び出しに至らない場合は、旧リスト中の以前の旧オブジェクトツリーの状態のままである。しかし、枝のオブジェクトが、現在のスレッドに適合する場合は、処理は進行する。そして、空の新リスト８２２が存在し、子ウィンドウは総て、次のレベルにおいて列挙される。処理は再帰的に進行して、現在のウィンドウ構造を捉える。符号８３２は、更新前の、P 836，834，Q 838，842，848，850，852とR 840のオブジェクトのセットから構成されるオブジェクトツリーを表している。更新されたオブジェクトツリー８５４を参照すると、新IEウィンドウ、または、GUIオブジェクトが存在する場合、階層866，870における総てのHTML要素の子を取り出すために、GetIETreeに対する呼び出しが実行されて、次に、notifydestroytree()に対する呼びが行われ、このオブジェクトツリーにおいて残存しているオブジェクトを総て抹消する。これは、総てのウィンドウに関するオブジェクトツリーを列挙する基本的な処理であり、さらに、このオブジェクトツリーは、旧リストから新リストまで未だ存在している、ウィンドウまたはGUIオブジェクト856，858，862，860，868，872 を有している。オブジェクトが、一旦、旧リストにおいて破棄されたとして記録されると、新規に作成される総てのもの866，870は、今度は、グローバルリストに自動的にリンクされる。この位置に残存した総ては、オブジェクトツリーの分析中に、オブジェクトツリーの分析中に、バリアント（すなわち、関数Notifydestroytree８１８）を呼び出し、オブジェクトが存在しないことを検出する８６４ので、スクリプト８１２に対して破棄イベント８２８を発する。以前のオブジェクトツリー構造は、そのままの状態に未だあるので、破棄イベントは、旧オブジェクトツリーのコンテキストに送られて、削除された、或いは、保持されたオブジェクトを規則的に更新するために、gMgスクリプトの現在のコンテキストを維持する。

【０１１９】

関数SynchTree８２６は、実際に新オブジェクトツリーである実ウィンドウの組を、処理する。Dwindows構造８２４には、gMgシステムが使用する旧オブジェクトツリーが、含有されている。SynchTreeコードのこの部分は、総ての新オブジェクトを表し、旧オブジェクトツリー（スクリプトが後で使用するために、そのままの状態にしてある）のコンテキストにおける総ての破棄オブジェクトを、gMgに知らせる実ウィンドウの下で、同じオブジェクトから新オブジェクトツリーを作成している。SynchTreeに対する呼び出しが実行され、Dwindowsに実ポインタを再帰的にコピーすることで、gMgツリーの中へ現在の実オブジェクトツリーを転送する。この時点で、gMgスクリプト８１２は、今度は、新ツリーにアクセスし、旧ツリーはなくなる。さらに、破棄オブジェクトは総て、最早、ツリー８６４には存在しない。そして、新しく作成された総てのオブジェクトのグローバルリストが存在することになる。gMgスクリプトへ生成イベントを発行することにより、gMgスクリプトに、新しく発生する総ての新オブジェクトが通知される８６６、８７０。同時に、検出されたDOMにおけるHTML文書のオブジェクトが存在する場合は、関数IEEventは、サブスクリプションとして、構成される。

【０１２０】

ツリー更新処理が完了すると、新オブジェクトを総て格納するために使用される一時ファイルが、分離され、消去される。新オブジェクトのリストの終端ポインタは、リセットされて、リストの先頭に戻される。このリストは、このオブジェクトが作成され、発見される順序に、構築される。逆の順序では構築されない。関数Updatetreeは、アプリケーシ

ョンをスキャンし、生成／破棄イベントを生成する。破棄イベントは、先ず、消滅したままのオブジェクトに向けて発行され、次に、生成イベントは、検出された新オブジェクトに向けて発行される。

【０１２１】

単一のgMg.exe（エージェント）に属するgMgエンジンの一式内において、gMgエンジンの種々のインスタンス間の通信が、登録されたOSウィンドウのgMgメッセージによって、確立される。このメッセージは、フル・ディレクトリ・パス名とモジュール・ファイル名（すなわち、gMg.exe）を使用しても作成される、固有のgMgウィンドウ・メッセージである。フル・ディレクトリ・パス名は、gMgメッセージの識別子から構成され、現在実行中のエンジンによってアクセスのため、共有メモリ領域に格納される。アプリケーションが、そのコンピュータ・システムに対して個有のウィンドウ・メッセージを有している必要があるため、登録されたウィンドウ・メッセージはそのOSに登録される。gMgエージェントやそのエンジンンなどのアプリケーションは、OS API関数を呼び出し、識別文字列を渡す。その識別文字列から、OSは、識別文字列に関する内部テーブルをチェックし、存在する場合は、すでに割り当てられた識別子を戻す。最初に、アプリケーションによってメッセージが送られ、そのメッセージが、内部テーブルに追加され、OS内部で実行可能なそのアプリケーションに対するグローバルな固有のウィンドウ・メッセージを割り当てられるが、揮発性であるため、コンピュータが再起動された時には、この識別子は失われる。例えば、プログラム間の通信に関して、２つのプログラムが、それぞれ、OSの登録されたウィンドウ・メッセージを持っている場合は、固有の文字列は、プログラム・アプリケーションの各々を特定する。このOSは、各プログラムの対応する固有の識別子を返すことになる。登録されると、このウィンドウ・メッセージの情報は、gMgセンサにより投入されつつある総ての対象とするプロセスにより、容易に読み出される状態のgMgの共有メモリに、格納される。

【０１２２】

プライマリ・ディレクトリは、主要な実行可能ファイル（すなわち、gMg.exe）、DLL、.iniファイル、その他のコンポーネントを有するgMgエージェント・システムのコンポーネントを有している。プライマリ・ディレクトリのOSのフルパス名は、gMgエージェント及びエンジンのマルチインスタンス化に関する役割を演ずる。主要なエージェントの実行可能ファイルを有しているパスとディレクトリは、コンポーネント一式を有する内部のgMgインスタンスを入力するために使用される。単一のコンピュータに多数のエージェントが存在する場合は、各エージェントは、自身のディレクトリに存在している。さらに、gMgエンジンのランタイム・インスタンスは総て、フルファイル名及びフルパスにより、そのディレクトリに入力され、この結果、多数のインストールされたエンジンを同時に動作させることが可能となる。OSは、入力された共有メモリのブロックの一部を、各DLL名とそのパスへ割り当てて、個有性を確保する。このメモリブロックは、種々のインストールされたエンジンにわたり独立していて、衝突を回避する。総ての実行時の動作に対して、OSは、共有メモリを各エンジンDLLのプライマリ・ディレクトリに関連づける。

【０１２３】

図9を参照すると、各エージェント及び各エンジンは、１以上のスクリプト９３９を、個別のディレクトリから、或いは、リモートサーバから読み込むことができ、gMgマルチインスタンス機能をサポートする。同一のスクリプトが、N個のホストアプリケーション90，902，903において実行される、N個のプロセスにおいて実行される場合、918～936までの複数のスレッドを有するN個のプロセス909，912，915で実行中のスクリプト945，948，951のN個のインスタンスとコピーが存在することになる。異なるスクリプトも、各スクリプトが自身のプロセスのスレッドを操作可能な状況において、実行可能であるか、または、単一の連結スクリプトが、N個の総てのプロセスをモニタリング可能である。gMgスクリプト・レベルにおけるマルチインスタンス化は、主として、スクリプトのデータ領域によって、取り扱われる。単一エンジン内のN個のスクリプトを取り扱うために、関数LoadProgram９４２は、コマンドラインを介してロードを実行するか、または、gMgスクリプト自

身内部のコマンドとなる。ファイル名は、多数のオプション（例えば、ロード、アンロード、または、リロード）により、渡される。プログラムリスト（PGMLIST）９５７は、現在並列で実行中の総てのスクリプトについて、保持される。プログラムのロード中に、スクリプトのバイナリファイルのサイズが決定され、メモリは特定のサイズに割り付けられ、一旦ファイルのロードが成功すると、そのファイルは閉じられる。整合性のチェックが行われ、ブロックの開始時にシグネチャを探す。そのシグネチャが有効であれば、そのファイルはロードされ、ProgramList構造９５７における保持されたリストに追加される。

【０１２４】

プログラムリストの動作は、除去、追加、或いは、置換である。スクリプトが、既に実行中である場合は、スクリプトはロードの動作を実行しない。リロードの動作は、現在実行中のスクリプトを終了させ、同一ファイル名でスクリプトをロードするものである。スクリプト名が、プログラムリストにおける識別子であるような場合において、エンジンは、同一スクリプト名で、同時に１つのスクリプトしか実行することができない。その代わりに、ディレクトリが異なっている場合には、エンジンは、同一スクリプト名で２つのスクリプトを実行可能である。スクリプト名には、パス名も含まれているため、同一スクリプト名であるがパスが異なっている２つのスクリプトは、ロードが可能であり、衝突なしに実行できる。アンロード関数は、DirectoryNameのディレクトリから総てを削除するワイルド文字（例えば、unload DirectoryName/*）の使用を許可する。LoadProgramは、そのプロセス内でスクリプトが実行中には、実行できない。何故ならば、LoadProgram９４２は、実行中のスクリプトによって既に正常に設定されたgMglockの設定を必要とするからである。イベントの処理を完了していないスクリプトを、ロード、または、アンロードしようとするならば、別のスクリプトをロードできる前に、現在のスクリプトの動作が停止する。そして、スクリプトは、ロードを開始・実行する。

【０１２５】

各パスはちょうど１つのイベントを処理しているので、解釈スクリプトはイベントをシリアル化し、スクリプトは、非常に短時間、実行される。イベントが発生すると、スクリプトが実行され、通常、1または2ミリ秒以内に、イベント処理が終了する。故に、スクリプトは、実行されないことが多く、そのエンジンはアイドリングされる。最も簡単な形式で、高水準で実行可能な多くの異なる機構と構造が存在する一方、スクリプトは、“if"または“switch”命令のグループのごとくに動作する。

【０１２６】

予定通りにリモートの自動操作で更新するスクリプトが、要求される場合に、gMgスクリプトが実行中で、gMglockにより制御されるならば、新規の更新が実行される。エージェント・スクリプトにより制御が要求される場合において、必要に応じてスクリプトをロードし実行可能とする方法である、更新、削除、リロード等の様々なアクションが、実行可能である。指定されたプライマリgMgMainコンポーネント（すなわち、エージェント）は、プロセスにわたり、サーバまたは他のソースから、GLOBAl_RELOAD_SCRIPTメッセージ（RUN命令により生成された）を、受信できる。このメッセージは、プライマリgMgmainコンポーネントにより、異なるアプリケーション・プロセスにおいて実行されている、すべてのエージェント及びそれらの各エンジンに対して、変換されて直接転送することができる。コマンド・メッセージを受信すると、対象のエンジン及び／またはスクリプトの動作は、コマンド実行のロードまたは更新をすることになる。

【０１２７】

gMgモニタリングシステムにおけるさらなる２つの特徴は、応答、モデル化、及びマルチインスタンスGUIオブジェクトセットの処理を可能にする重要な、オブジェクト変数及びWhenObjectキーワードである。

【０１２８】

gMgスクリプト・キーワードのオブジェクト変数は、絶えず変化する環境内で、様々なオブジェクトのセットを追跡する機構を提供する。複数のスクリプトが存在している場合において、異なるプロセスのそれぞれの内部で動的な状態を追跡するために、各スクリプ

トは、自分自身の変数の組を有するのが好ましい。変数の各組は、他の変数の組から独立しており、各エンジンと個々のスクリプト・インスタンスをサポートする。オブジェクト変数は、複数のプロセス及びスレッドが並列で実行されている、GUIオブジェクトのセット及びサブセットの潜在的に異なるインスタンスに対して、リアルタイムで応答する機構を提供する。アプリケーションのランタイムのダイナミック性を、すなわち、マルチインスタンス構造を、反映してモデル化するために、スクリプトが書かれる。「オブジェクト変数」という用語は、gMgスクリプトの「オブジェクト」のことを言っている。動的なホストアプリケーションGUI構造に対して応答する及び関連付けられている、抽象的な操作、動作、或いは、動的な割り当てを、そのgMgスクリプトの「オブジェクト」は、対象としている。

【０１２９】

図１０を参照すると、gMgオブジェクト変数１０３５は、gMgのランタイムGUIオブジェクトツリー１０１５におけるGUIオブジェクト１０２０に加えられる、抽象的構造を規定している。オブジェクト変数は、対象のGUIオブジェクト、或いは、ホストアプリケーションのユーザインターフェイスを反映するGUIオブジェクトのセットを表すために使用される。

【０１３０】

さらに、オブジェクト変数内のオブジェクトに対するポインタを格納するオブジェクト変数の機能は、高度に動的なアプリケーションGUIオブジェクト環境において、複数のインスタンスと、複数のインスタンスの管理をサポートするものである。オブジェクトの参照は、オブジェクト内の変数として、或いは、他のオブジェクトまたはオブジェクト変数に対するポインタとして格納可能である。オブジェクト変数に対するポインタは、GUIオブジェクトツリー１０１５に存在する総てのオブジェクトにおいて、動的に作成可能であり、また、マルチインスタンスなホストアプリケーションの検出に応じて、自分自身の変数の組を、割り当て可能である。さらに、オブジェクト変数が、他の或る構造内のGUIオブジェクトツリーに存在するか、または、明確に、他の場所で独立的に宣言されようとも、オブジェクトのポインタは、任意のオブジェクトに格納可能である。例えば、同一タイプの２つのオブジェクト変数が存在する場合は、各オブジェクト変数は、それ自身のプライベートな変数のセットを有している。これらの変数は、スクリプトのデータ領域内にではなく、GUIオブジェクトツリーのオブジェクト内に存在する。別の例において、オブジェクトツリー内に２０のオブジェクトが存在する場合、gMgエンジンにより動的に割り当てられる２０組の変数に対応する、２０の位置が存在する。

【０１３１】

図１０を参照すると、所定のオブジェクト変数１００５はどれも、関数Setnumobjvar（ObjHandle, Progid）による関連性の処理において、変数１０１０のリストを追加する。コンパイル時に、固有idが、スクリプト（例えば、そのスクリプトに割り当てられるオブジェクト変数）におけるそのオブジェクト変数用に、生成される。gMgスクリプト１０４０において、物理的構造にキャストされたスクリプトのオブジェクト変数に、値が割り当てられる。オブジェクトに属する、数字のリスト、文字列のリスト、または他の型のオブジェクト変数のリスト１０２５、１０３０、１０３５は、現在のプログラム構造に属し、プログラムがそのオブジェクト変数用にコンパイルされた時点で割り当てられた照合idを含んでいるものが発見されるまで、スキャンされる。特定のオブジェクト変数のセット１０３９が存在し、プログラム・スクリプト１つにつき、それぞれの変数のリストを有するため、他の総ての並列のスクリプトが共有、または、参照を有するアプリケーションの同一のGUIオブジェクトへ参照１０４５するとしても、複数のプログラムまたはスクリプトを実行している場合、各スクリプトは自身のオブジェクト変数を有する。複数のインスタンスの実行のサポートのために、オブジェクトツリー内に含まれるそれぞれのオブジェクト変数1039に設けられた、各スクリプトは自身の独立変数の組において、１組のスクリプトは、実行中のプロセスを反映する同一のGUIオブジェクトツリー１０１５を共有する。好ましくは、同様なインスタンス支援構造を有する複数のツリーに拡大される単一のオブ

ジェクトツリーが存在するのが良い。

【０１３２】

状態変数のこの動的なセグメンテーションは、結果的に、アプリケーションのGUIの多数のインスタンスを同時に追跡する機能となる。オブジェクト変数の実装の一例として、同一のウェブページが、複数に分離したスクリプトにより、異なった観点から、同時に追跡可能である。この方式を完成するために、スクリプトが実行され、リスナー及びセンサが確立される。各スクリプトが独立的にイベントを操作する方法を決める場合、イベントが発生すれば、そのイベントは、各分離スクリプトに対して送られる。

【０１３３】

また、さらなる実施例において、総てのgMgオブジェクト変数（例えば、オブジェクトツリーまたはその他の構造に位置し、アプリケーションGUIオブジェクトまたは内部の抽象構造に相当）は、スクリプトで宣言される自分自身のオブジェクトのプロパティ値（例えば、文字列、ブール値、数値など）も有することができる。この変数により、比較、条件テスト、オブジェクト変数とそのプロパティの様々な状態に対する論理の実行が、可能となる。

【０１３４】

第２のgMgコマンド、すなわち図１１に示すWhenObjectキーワードは、もう一つの特徴であり、gMgモニタリングシステム１１３３におけるマルチインスタンス化機能をサポートするものである。図１１を参照する場合、WhenObjectは、ホストアプリケーション１１０３に属するGUIオブジェクトツリー１１１５における、任意のオブジェクト１１０６、１１０９、１１１２にも属するオブジェクトである。GUIオブジェクトが、ツリーにおいて、マルチインスタンス化されている場合、WhenObjectもまた、マルチインスタンス化される。WhenObjectコマンドの役割は、ホストアプリケーションGUIの変化に関連付けられた明確なイベント（OS、ホストアプリケーション、或いは、そのコンテナにより発出されたイベント）が無い場合に、プロパティを読み出すか、或いは、GUIオブジェクトのプロパティ変化を評価することである。時々、イベントが何も発生しない場合に、GUIのプロパティが変化することがある。逆に、GUIのプロパティが何も変化しない場合に、時々、イベントが発生する。またある時は、対象のプロパティの変化を検出ために、どのイベントを使用するかの識別が、さらに困難になるか、或いは、曖昧になる。WhenObjectコマンドは、軽量なポーリング機構を介して、GUIオブジェクトのプロパティにアクセスすることにより、これらの問題を解決する。

【０１３５】

特別なgMgスクリプト・コマンドであるWhenObjectは、呼び出されるPollWhenObjects１１４８のWhenObjectルーチンとなる。このルーチンは、スクリプトコード１１６０のブロックをWhenObject１１５７に関連付け、属するスクリプトがどれであるか追跡を続ける。各スレッドにおいて、Pollwhenobject()呼び出し１１４８がgMgコードにより独立して作成される、ホストアプリケーションのプロセス１１１８で実行中のスレッド１１２１、１１２４、１１２７がいくつか存在する。イベントのシリアライゼーション処理内において、スクリプト・エンジンが１つのスクリプトと１つのイベントのみ同時に実行するように、各スレッドは、最初に、gMglock１１２０を設定する。この時、WhenObject１１５１に関連するの全てのもののリストが、処理される。WhenObjectがGUIオブジェクトから切断される場合、WhenObjectは、使用されなくなり（即ち、GUIオブジェクトが削除される）、それに相当するWhenObjectに対応関係のある一方がWhenObjectリストから削除される、クリーンアップ動作がある。

【０１３６】

GUIオブジェクトは、現在実行中のスレッドに属する場合、そのスレッドは、適切なWhenObjectを実行する。現在のプログラムは、どのgMgプログラム・スクリプトが、このWhenObject要求を発出したのかを参照するために、セットアプされ、次に、WhenObjectの実行が開始されるかどうかを試験するスクリプトコード１１６０の一部において定義された通りのWhenObjectの条件式を指し示すスクリプト・インタプリタ・プログラム・カウンタで

あるコード・ポインタ１１５７を、セットアップする。スクリプト・プログラム・カウンタは、このスクリプトコード（すなわち、コンテキストを確立するプログラム構造）指摘する。スクリプトコード本体でコンパイルされた実際のブール式を、このポインタは指し示し、WhenObject Eventが発行される。WhenObjectが属するGUIオブジェクトツリー１１３６には、オブジェクト１１３９、１１４２、１１４５が存在する。Getnumber()が呼び出され、真であると判定される場合は、WhenObjectはアクティブとなる。スクリプト・プログラム及びイベント・オブジェクトのコンテキストにおいて、ブール式の判定を実行中であり、「実行コンテキスト」を設定する。このコンテキストは、正しいプロセス、スレッド、ツリー内のGUIオブジェクト等の内部で、処理作業が発生するのを確保する。グローバルな（現在の）実行中のイベントであるWhenObjectイベントを、gMgスクリプトが処理を行う。WhenObjectイベントのオブジェクトは、オブジェクトツリー中のオブジェクトであり、適切なスレッドにより現在処理中のWhenObject構造にリンクされる。このブール式が真であるならば、WhenObjectがアクティブとなる回数についての、保持されている統計値に加算される。コマンドは、停止の命令コードに遭遇するまで連続して処理される。

【０１３７】

gMgインタープリタ１１５３（と、ホストアプリケーションのプロセスにおいて、実行中であることを示す１１３０）内で、関数ExceCOMmand()は、オペコードのgMgスクリプト・セットを指し示す。変数gPCポインタは、（式を指し示し、その式が、停止の命令コードに続く命令文のリストとして存在している直後の）スクリプトコードを指し示すグローバルなインタープリタ・プログラム・カウンタである。式を渡されたプログラムカウンタを移動する式が、評価される。その式が真となる場合、第１の命令文を指し示し、WhenObjectコマンドが存在している場合にコマンドを実行する。

【０１３８】

オブジェクトのプロパティの読み出しを実行する可能性がある時点で、WhenObjectコマンドを実行することにより、Getnumber式を経由してポール動作が行われ、次いで、スクリプト・コマンドが実行される。総てのスクリプトからの総てのWhenObjectは、集中型のリスト１１５１内に取り込まれる。さらに、スクリプト・オブジェクトは、各スクリプトとの関連を示す。そのスクリプトが終了すると、WhenObject（構造）は除去され、１つのオブジェクトが終了すると、それに対応する特定のWhenObjectもまた、除去される。gWhenObjectの構造は、並列な総てのマルチインスタンス化されたスクリプトからのWhenObjectの集約リストとなっている。

【０１３９】

スクリプト全体に及ぶスレッドにより、このWhenObjectが、実行される必要があり、どの時点でどのスレッドにおいて、どのスクリプトが実行されるか、スレッドの制御より、決定される。従って、現在のスレッドに関連したWhenObjectのみが、現在のスレッド中で処理される。このスレッドは、この関数を実行する前に、各スクリプト１１５４（これは、１１３０において、アプリケーションのプロセスの中に存在する、エンジンとスクリプトの拡張表現である）に対して、コンテキストをセットアップする。このコンテキストのセットアップによって、そのスレッドは、そのスクリプトに属し、他のスクリプトのオブジェクト変数へのアクセスを阻止する、オブジェクトを調べる。WhenObjectは、実行中の総てのスクリプトに対して、集約され得る。

【０１４０】

以下のコードのサンプルは、gMgシステムのマルチインスタンス機能の実施例を示す。このスクリプトは、無数のスレッドを検出し、それらのスレッドの各々は、現在無数のメッセージ・ウィンドウを有している１つのポップアップ・ウィンドウを並列的に制御する。gMgスクリプトが、アプリケーションにおける総てのGUIオブジェクトの寿命をモニタリングする場合、総てを完全にモニタリングするためには、多数のインスタンス構造を作成する必要があることになる。総てのオブジェクトに対して、gMgは、GUIオブジェクトのインスタンスを表す構造を自動的に保持するのである。オブジェクトツリーは、総てのオブジェクトに対する属性キャッシュを有しており、総てのオブジェクトに対するオブジェク

ト変数の格納場所を持っている。

```
//"ManyInstances"-このスクリプトは無制限の数のオブジェクトを処理する
objvar createtime:number;
if( create ) //生成に関して
{
createtime=sys_time;     //createtimeは、現在のイベントに関する生成された
                         //オブジェクト（従ってインスタンス）である。インスタンスは、
                         //イベントのアクションの結果として発生する。
var s=title;             //sは一時変数であり、gMgのインスタンス化機能を示す。
s=class;                 //ここの現在のオブジェクトに関するタイトル及びクラスは、
}                        //別々にGUIオブジェクトツリー内に保存される。
if( destroy )
{
//現在時刻からcreatetime（前に発生している）を引くことでオブジェクトの寿命を導き出せる。
dbg( objtypename(objtype) + '['+title+']['+class+'] existed for ' + string(sys_time-createtime)+'ms＼n');
}
//---------------------------------
//現在のオブジェクトがポップアップ・ウィンドウであるかを示すブール式
function ispop=iswindow && title=='blabla' && class=='whatever';
//現在のオブジェクトが注目しているメッセージ・ウィンドウであるかを示す
function ismsgwindow=iswindow && title=='blabla' && class=='whatever';
//生成イベント及びその生成物が、ポップアップであり、ポップアップの
//スレッド・オブジェクトにおいて、ptrをポップアップされるオブジェクトに
//格納するのであれば、オブジェクトは現在のオブジェクト、すなわち生成中の
//オブジェクト、を参照するキーワードである。オブジェクトが属するいかなるスレッドのスレッドオブジェクトにおいて、ポップアップへのprtを生成する。
objvar mypopup:object;
if( create && ispopup && thread.mypopup==none ) //create、スレッド値が空であれば
thread.mypopup=object            //ポップアップ・オブジェクトへポインタ
//破棄イベント及び破棄されるオブジェクトと、thread.mypopup内のオブジェクトとが
if( destroy && object==thread.mypopup )
 thread.mypopup=none;   //thread.mhypopupをクリアする
//メッセージ・ウィンドウが作成されると、WhenObjectがobjvar prevtitleに加えられる。
//以前のタイトルから変更（すなわちfnが変更）された場合はチェックを行う。
objvar prevtitle:string;
if( create && ismsgwindow )
 WhenObject( chanbed( live_title, prevtitle ) && thread.mypopup != none ) //スレッド値は０でない
```

数字のキーワードの定義、或いは、オブジェクト変数の定義は、現在のオブジェクトに総て対応するGUIオブジェクト（例えば、previtle, title, class）の動的な割当によって決定される暗黙の動作を有する。現在のオブジェクトは、現在のイベントとインタープリタにより処理中の現在のオブジェクトとを反映する構造となっている。

【０１４１】

変数mypopupは、オブジェクト変数である。関数Getnumobjjvar()の呼び出しが、処理中に実行される。初期化時に、第一のパスは、オブジェクトが存在していないことを見つけて、デフォルトをゼロに戻す。そして、関数SetObjObjVar()を呼び出すことにより、ポッ

プアップ・ウィンドウの位置に現在のオブジェクトの参照を置き、また、ポップアップ・ウィンドウへの参照を自分自身のスレッドオブジェクトに置く。

【０１４２】

live title値が前回のタイトルから変更されていれば真となるLive Titleは、現在のものであり、タイトル変更に際して、このタイトルを更新すし、前回のタイトルは、メッセージ・ウィンドウ・オブジェクトに格納されている。例えば、アプリケーションが１０個のメッセージ・ウィンドウを作成すれば、各メッセージ・ウィンドウは、自分自身のprevious title値を保有していることになり、総てのタイトル変更を追跡し続けることになる。このメッセージ・ウィンドウの各々に関連付けられた１０個のWhenObjectが存在する。

【０１４３】

Live Titleのコンテンツ、即ち現在のタイトルが、前回のタイトルから変更（Changedコマンド）されるかどうかを検出するため、各メッセージ・ウィンドウがモニタリングされている場合において、本システムは、メッセージ・ウィンドウをモニタリングする。仮に、値が真であり、ポップ・ウィンドウ・タイトルを「abc」であるとすると、各ポップ・ウィンドウが、このメッセージ・ウィンドウと同じスレッド中で実行されているかどうかが、チェックされる。従って、メッセージ・ウィンドウ・タイトルが変更され、表示されるポップアップ・ウィンドウ・タイトルが検出時に「abc」である場合、その条件が真となり、WhenObjectのアクションが実行される。

【０１４４】

WhenObjectは、ポップアップ・ウィンドウのテストがさらに限定する、検出されたメッセージ・ウィンドウ総てを実行する。各々のスレッドが、無数のメッセージ・ウィンドウを有することのできる無数のスレッドが存在可能であるが、しかし、スレッド当たり１つのポップアップ・ウィンドウのみが、許容される。このコードは、オブジェクト内のオブジェクト・ポインタを示す（すなわち、thread.mypopup=object）。一例として、各々のスレッドが自分自身のポップアップ・ウィンドウを有することができる２０のスレッドが、存在可能である。名前を付けられた２０のスレッド変数の総てに対して、mypopupは、各スレッド及びそれに対応するポップアップ・ウィンドウに関する情報を記録する。これらの構造から、各スレッド変数は、各々のポップアップ・ウィンドウが存在するか否かに関して、追跡が可能である。

【０１４５】

gMgコンテキストと、GUI構造のマルチインスタンス化の識別を行う場合のgMgコンテキストの役割とについて、説明する。図１２を参照すると、収集された追跡データの解析をサポートするために、サーバにコンテキスト情報を提供する、マルチインスタンス化のGUIアプリケーション構造部分を表す、階層的に組織化するエンティティを、gMgスクリプト・コンテキストのキーワードは、宣言する。コンテキスト・コマンドは、総ての型のオブジェクトに対して、使用可能である。トップレベルは、プロセス１２１５であるが、このプロセスが、複数のトップレベルのウィンドウを持っている場合は、現在の処理に関して、トップレベルのウィンドウで開始されるサブ・コンテキストは、サーバに送られる発生した追跡イベントに対するコンテキストを表し確認するために、ウィンドウの階層関係を作成する。種々のオブジェクト組織を説明する場合には、コンテキストの説明は、固定化されない。トップウィンドウが特定される場合は、追跡メッセージは、トップウィンドウとその親プロセスにより形成され、特定されない場合は、そのプロセスは、そのまま送信される。

【０１４６】

プロセスの範囲内にあるイベントを追跡するために、コンテキストをプロセスに設定可能であるが、しかし、複数のポップアップ・ウィンドウ１２０５などのオブジェクトを追跡する場合は、イベントが発生した特定のポップアップ・ウィンドウに、このコンテキストを関連付けることが可能である。この定義付けに応じて、このコンテキストは、サーバが解析する暗示的な階層をセットアップする。総ての追跡メッセージ記録と共に送信される、５レベル階層の任意の閾値が存在する。５レベルあれば、殆ど総ての場合については

十分であると考えられるが、しかし、このレベル閾値は、容易に拡大、または、縮小が可能である。

【０１４７】

図１３を参照すると、ホストアプリケーションのGUIツリー組織１３０５内に、複数の類似したGUIオブジェクトのインスタンス、或いは、GUIオブジェクトセットのインスタンス、或いは、オブジェクトのインスタンスが存在する場合は、コンテキスト・レベル値が重要となる。例えば、複数のポップアップ・ウィンドウ１３４０～１３５５を有する複数のスレッド１３２０～１３５５を有し（すなわち各スレッドにつき１つ）、さらに、各ポップアップが複数のタブ１３６０を貼られたウィンドウを有している、プロセス１３１５が与えられた場合、この処理は、レベル４のコンテキストとして、説明される。各インスタンスが識別可能（例えば、ウィンドウにおける一式のタブの中の各タブが、固有のものである場合）場合は、このコンテキストは、有用な追加情報を提供する。標準的なウィンドウGUIダイアログ・ボックスのタブのフォントサイズの変更におけるOKボタンを考えてみると、この例においては、同時に発生する類似のＯＫボタンに関する、複数の同じインスタンス及び構造が存在する場合、このコンテキストは、絶対的に必要である。特にウェブアプリケーションにおけるマイクロソフトのインターネット・エクスプローラは、他のスレッドとして、各スレッドが同一のウェブページを持っている複数のスレッドを許容する。各GUIオブジェクトが、複数のインスタンスを有することが可能な場合において、GUIオブジェクトの複雑な階層を有するアプリケーションにとって、コンテキストは、絶対に必要である。プロパティ、または、他の検出機構を使用して、コンテキストは、通常、識別可能であるので、５レベルで十分である。

【０１４８】

別の例において、ウェブアプリケーションに通常見られるような、複数のタブを有するウィンドウに対して、タブ制御名、または、他のラベルを送信すると、どんなタブが存在し、または、どんなタブがクリックされるかを、示すことができる。しかしながら、総てのタブ制御が、異なる場所におけるアプリケーションに対して、同一構造を持っているならば、他のコンテキスト・レベルの定義を追跡する動作に対する様々なタブと区別することが、必要とされる。付加的なコンテキスト・レベルは、固有のID及び／または参照などのデータを渡し、各タブにおける発生事象を反映する。

【０１４９】

また別の例において、コンタクト・アプリケーションは、顧客の情報記録を有していて、氏名、住所、電話番号、及びイーメールを含む複数の顧客記録を、ユーザに対して表示する。この場合、総ての記録は、個々のスレッドと個々のポップアップ・ウィンドウとで構成される、同一の内部構造を持っているが、各顧客に関する固有のコンテンツ情報を持っている。ポップアップ・ウィンドウは、同一の構造を持っているが、各々は固有のコンテンツを持っている。トップレベルの構造においては、ポップアップ・ウィンドウは、追加情報を持たないgMgシステムと同一に見える。この不明瞭さを解決するために、総ての追跡メッセージと共にサーバへ送られる再構成されたコンテキスト情報となる、様々なポップアップ・ウィンドウを区別するために、別枠のコンテキスト・レベルが使用される。サーバにおいては、共通のコンテキスト・レベルが、共にグループ化されて、１つのウィンドウでクリックが行われた場合に、別のウィンドウにおけるクリックと混同されないように、各GUIウィンドウ構造内で発生するイベントを追跡する。

【０１５０】

GUIオブジェクトに加えて、スレドッド・オブジェクトも、また、識別処理を目的とすることが可能である。マルチスレッド化されたGUIアプリケーションにおける各アプリケーションのスレッドが、自身のコンテキストを割り当てられている場合において、スレッドはコンテキスト的に識別可能である。複数のスレッド及び複数の同一のトップウィンドウを有するアプリケーションを与えられて、付加レベル（プロセスにおける、スレッド・オブジェクト）が導入され、識別用のスレッドidとなる。各スレッドが同一のトップウィンドウを有している複数の同一スレッドが存在する場合、他の同一のトップウィンドウに

おける動作から、各トップウィンドウのグループにおける動作の追跡を個別に継続する必要があるので、これにより、マルチスレッド状態を処理する。複数のスレッドが、同一に見える可能性があるので、他のオブジェクト、または、関連性（例えば、継承元）が定義されて、様々なスレッドと、それらが関連するオブジェクトとを識別する。GUI構造の識別時、従ってイベントの識別時に、業界標準の顧客関連管理アプリケーション（「CMR」）におけるような多くのウェブアプリケーションにおいて、この曖昧な状況に陥る。

【０１５１】

マイクロソフトのCRMアプリケーション（CRMは、レッドモンドのマイクロソフト社の製品である）などの様々な実装系（処理系）において、新スレッドを作成する新しいどの動作に対しても、ポップアップを開くことが可能であることが判明したため、作成されたスレッドと同じだけ、追加のポップアップ・ウィンドウが必要とされる。この状況において、gMgスレッド・コンテキストを使用することにより、個々のポップアップ・ウィンドウが、識別される。しかしながら、レベルが１つしかなく、ポップアップ中で、ウィンドウの追加的に複数の同一のインスタンスが作成可能である場合は、追加されるコンテキスト・レベルが必要である。結局のところ、マイクロソフトのCRMの例は、単に２つのレベルを必要とする。レベル数の拡大は、利便性が増大することになり、将来的な拡大に対処することが可能である。この代案として、ポップアップ・ウィンドウ内で、その他の属性は、この属性を区別するために利用可能である。

【０１５２】

十分な深度（分解能）のGUIオブジェクトのコンテキスト階層を定義することにより、別な面において同一となるGUIオブジェクト構造が識別可能となる。もし、同一である複数のGUIオブジェクト構造が存在する場合、トップウィンドウのコンテキストは、対象のGUIオブジェクトに関連する論理セットにおけるイベントの総てをグループ化するために、使用可能である。gMgコンテキストの特徴は、オブジェクトを区別するマルチインスタンス機能をサポートする。

【０１５３】

議論を、識別の問題及びイベント検出の問題に移す。この識別とイベント検出は、gMgシステムに関するイベントの主要な区分の１つを提供するフックを確立するために、マルチインスタンスをサポートする目的を有している。

【０１５４】

図１４を参照すると、ChangeHookとlockgMgEngineは、スレッドIDを渡され、必要に応じて、アプリケーションのプロセス１４０６をフックする。gMgメッセージ１４２４が、ポストされて、関数PostgMgMsg()を持つアプリケーションに、登録される。PostgMgMsg関数は、渡されたスレッドIDを使用して、そのスレッドIDを使用するウィンドウを見つけ出し、ウィンドウ・メッセージを設定する。前述のごとく、複数のgMg.exeの個有性を確立するための.exeのフルパス名を使用する、自身の固有のウィンドウ・メッセージを、gMgは登録する。通常、PostgMgMsgが動作し、このメッセージの複数のコピーをポストし、登録処理を完了する。この登録処理においては、継承する第１のメッセージがシステムで使用され、残りのメッセージは捨てられる。共有メモリ１４５４が、総てのgMgエージェント、或いは、エンジン・インスタンスに対して、gMgウィンドウ・メッセージの滞在時間を制御し、追跡する。診断サポートのため、lparamが共有メモリ中に在る診断データへポイントしている状態で、tmdiagnostics値が、メッセージwparam中に含まれている。診断情報が、或るgMgコンポーネントにより必要とされる場合は、gMgウィンドウ・メッセージは、winhookルーチンのうちの１つに到着するdiag_excportsメッセージを送信する。

【０１５５】

ＯＳ１４０３がクロック１４５７を駆動することで、実行中の総てのフック・ルーチン１４３３～１４４２は、最終的に、このメッセージをインターセプトして、ルーチンHandlemsg()１４４５を呼び出す。Handlemsg内のgetmessagehookルーチン１４４５において、getmessagehookは、このシステムにより現在設定されているフック数に基いて、数字Ｎに連結される。関数handlegetmsghookmessage１４３０を呼び出す０～Ｎのマクロのセット

１４２７中で、このフックは実行される。各インスタンスにおいて、gMgMsgルーチンが、何度も複製させることを防止するために、コンパイル中のautoinlineはオフに設定される場合、フッキング動作が終了した時点で、これらのルーチンの各々に対するポインタの配列も存在している。さらに、HandleMsgHookMsg１４５１を呼び出す、sendmessage１４４８のメッセージも処理するＮ個のSendmessagehooksがHandlemsg内に存在する。これらのルーチンは、そのメッセージを、それらの起源に関係なく、インターセプトする。

【０１５６】

処理作業をモニタリングする専用のワーカースレッドを作成する代わりに、ホストスレッド１４０９、１４１２、１４１８、１４２１を使用するのが好ましい。コールバック関数をセットアップするプロセスにおいて作成されたタイマーのみによって、ウィンドウの作成は、最小限に抑えられる。さらに、フッキングの目的は、透過的で最小限の侵入にすることにあるため、ワーカーウィンドウは作成されない。たとえ、ワーカーウィンドウを作成するとしても、ウィンドウは、依然として、プロセス制御のためにタイマーを必要とする可能性は高い。

【０１５７】

gMgシステムは、ホストアプリケーションの既存のウィンドウに、メッセージを送信後、そのメッセージをインターセプトする。メッセージは、プロセスにわたり送信されることは滅多にないが、仮にこのような事が起これば、ホストアプリケーションにおけるＮ個の異なるウィンドウに対して、探索が開始される。一旦、配置されたgMgが、各ウィンドウにフックを設定すると、メッセージが対象の各ウィンドウに到着した時に、gMgはそのメセージをインターセプトする。

【０１５８】

tidマクロは、スレッド毎に１フックが存在する場合、現在のスレッドIDを獲得する。通常、メッセージは、現在のスレッドを持っているウィンドウから受け取るが、時々、他のウィンドウスレッドからメッセージを受け取ることもある。Sendmessageを実行するスレッドは、同一のプロセスに属していなければならず、さもなければ、DLLが、明瞭にフックされない別の無関係なプロセスに挿入されなければ、ならないことになる。Sendmessageが別のプロセスにより開始される場合は、Sendmessageは、対象のプロセス中に入れられ、フック・ルーチンを呼び出すそのプロセスにおいて、スレッドを割り当てられる。Handlemsgルーチンは、Getmsghook()とSendmsghook()１４４５の双方から呼び出される。メッセージの処理は一定の動作パターンを持っている、すなわち、メッセージが到着しても、gMgはインターセプトを行おうとする。

【０１５９】

SecondMessageを呼び出し中に、追加パラメータと、プロセス間ウィンドウ・メッセージ（例えば、スクリプトのロード、エンジンのアンロード等）を有するプロセス全体にわたる情報とを、渡すために、GetExtParams()が使用される。GetExtParams()は、インデックスを加えたバージョン番号にから構成される、共有メモリにおいて、ロックを持っている。この関数は、バージョン番号を覆い隠し、共有メモリ内のアレイの中にインデックスを付けて、インデックス付加後のメッセージにより渡される追加データを取得することができる。（バージョンを有している）共有メモリIDが要求IDと一致しない場合は、共有メモリのIDは、使用しないエントリである。32ビット数のIDは、絶えず増加している。下位の6ビットは、ExParams構造に入る環状バッファの中のインデックスである。このバッファ内の32ビットの IDが、渡されたパラメータに一致しない場合は、そのIDは再利用されたか、または、不要なものであると判断される。GetExtParams()は、lparamをフェッチし、それが不要なものであれば、nullを戻す。lparamが不要でない場合は、別のパラメータによりメッセージを処理する。エンジンのリリースバージョン用のコードを削除するために使用するマクロがある。dbgDLLがロードされたかどうかを知るために、テストも行われる。

【０１６０】

gMgシステムにおけるGUIオブジェクトのセットを検出する、GUIオブジェクト検出機構

を、汎用検出と呼ぶ。汎用検出法は、クライアント及びサーバでの分散処理のGUIイベントを、判定し検出する。GUIオブジェクト操作には、特定操作及び汎用操作の２種類がある。一般的な方法で、非常に少数の操作記述子により、オブジェクトが検出され、GUIオブジェクトのセット、または、GUIオブジェクトのクラスが作成される。関数に入力される入力パラメータは分析可能であり、ただ１つの特定のGUIオブジェクトよりはむしろ、GUIオブジェクトのクラスを検出する。

【０１６１】

クライアント側では、図１５のGUI検出の範囲１５１５において、gMgスクリプト式を、意図的により汎用１５１０（例えば、関連するGUIオブジェクトの全種類を検出するために）にする、或いは、より特定１５０５（例えば、単一GUIオブジェクトを検出するために）にする。検出式は、この概念的範囲のどの場所にも存在するGUIオブジェクトを、検出することができる。gMgセンサは、ホストアプリケーションのGUIオブジェクトデータを収集して、これを使用して、gMgスクリプトはGUIオブジェクト検出用の式を一般的に（または、分類して）定義する。その結果として収集されたGUIオブジェクトの詳細及びプロパティは、一括されてサーバに送信される。サーバは、これらの分類されたイベントと、それに伴うプロパティを受信し、さらに、それらを処理して、特定のGUIオブジェクト・イベントを判断する。サーバは、追跡メッセージ・レベルと、予め定義されたセマンティクス及び仕様とを使用して、ユーザコンテキストを構築する。特定的GUI検出のために、追跡メッセージは、オブジェクトに関する特定のプロパティを含有する識別された各GUIオブジェクトに送信される。汎用検出を行う場合は、サーバに対する十分なデーを有する追跡メッセージが送信されて、特定GUIオブジェクト解析及び識別をリアルタイムの後で判定する。

【０１６２】

以下の特定及び汎用結合コードの例において、第一実施例は、クライアントが、追跡中の４つの定義されたGUIボタンの各々に関する異なる追跡メッセージを、フィルタリングして送信する場合における、特定結合（単一のGUIオブジェクトを対象）を示す。

【０１６３】

１．クラスが「button」、タイトルが「OK」、トップウィンドウが「オープンウィンドウ」であれば、チェックが行われる。全てが真であれば、Open OKが押されたことを示す追跡メッセージが送信される（Open OK）。

【０１６４】

２．Saveダイアログ内のOKボタンであれば、Save OK追跡メッセージが送信されることになる。

【０１６５】

３．さらに、openダイアログ内のCancelボタンであれば、Open Canceled追跡メッセージが送信されることになる(Cancel Open)。

【０１６６】

４．Save/Cancelメッセージを持つSaveボタンとsaveダイアログであれば、適正な追跡メッセージが送信されることになる（Cancel Save)。

【０１６７】

以下に、４つの特定型のボタンが、検出される方法を示す。

```
/*特定及び汎用結合コードのサンプル*/
contextprocess=processserver"someserver"
context thread=thread in process
context topwindow=topwindow in thread

//特定結合 - クライアントが、特定オブジェクトを判定し各々に固有の追跡メッセージを送る

if ( wlbutdown && class=='BUTTON' && title=='OK' && topwindow (title=="Open" &&
```

```
class=="332767" ))
track ( priority, pressed, topwindow, "Open Ok");

if ( wlbutdown && class=='BUTTON' && title=='OK' && topwindow (title=="Save" &&
class=="332767" ))
track (priority, pressed, topwindow, "Save Ok");

if ( wlbutdown && class=='BUTTON' && title=='Cancel' && topwindow (title=="Open"
 && class=="332767" ))
track (priority, pressed, topwindow, "Open Cancel");

if ( wlbutdown && class=='BUTTON' && title=='Cancel' && topwindow (title=="Open"
 && class=="332767" ))
track (priority, pressed, topwindow, "Save Cancel");

//汎用結合 - クライアントが、総てに対して１つの追跡メッセージを送るが、しかし
、サーバがどれかを把握するための情報を有するメッセージとして

if (wlbutdown && class=='BUTTON')
track (priority, pressed, topwindow, "Button", title=title, toptitle=topwindow.t
itle, topclass=topwindow.class);
```

プレス・イベントに結合される、クラス「button」に関する総てのボタンダウン・メッセージのいずれも、ボタン（どのボタンでも良い）が押されたことを示す追跡メッセージを送信する。topwindowタイトルとtopwindowクラスもまた、サーバへ転送可能であり、４つのボタンの内どのボタンが動作しているのかを、解析し検出する。汎用コードにおいて、複数の追跡メッセージ構造は同一であるが、但し、サーバへ通信する前に、対象のGUIオブジェクトを明確に識別する追跡メッセージとは対照的に、追跡メッセージ構造を識別する様々なコンテント・データをその構造が包含可能である場合を除く。

【０１６８】

特定及び汎用の双方の場合、イベントは、インタープリタにより受信され、イベントのインタープリタ処理内で、特定及び汎用の双方におけるスクリプトに記述されている条件に基づいて、プロパティの読み出しを行い、条件式の実行を可能にする。プロパティの読み出しは、条件か、または条件の命令文（例えば、if(Q) [R]、ここでRは、１以上の追跡メッセージを包含する）の本体の一部のどちらか一方の一部分である。さらに、サーバは、種々のGUIオブジェクトを処理または検出し、Save, Cancel, OKなどのボタンの種類を探すか、或いは、文字列照合を行うためにテキストを分析するかのいずれかを行う。後者の場合は、検出処理の権限をクライアントからサーバに移動する。

【０１６９】

gMgシステムのメソッドのモニタリングは、コンポーネント内部に含まれるメソッドモニタリングするために使用される特別のセンサを実装する。この特別で柔軟なセンサは、対象とするホストアプリケーションの範囲にわたって、クライアントまたはサーバのOSの内部へ、様々なイベントを発生し統合に関する発明の範囲において、実現させる。メソッドのモニタリングは、関数の呼び出しをインターセプトし、DLLとして、パッケージ化されたホスト・アプリケーションのコンポーネントに戻すセンサ型をサポートするコンポーネント・メソッドをインターセプトする機能を有する。関数呼び出しのインターセプトによって、追跡とモニタリングのために、論理スクリプトへ渡されるコンポーネントからイベントとプロパティを、gMgシステムが、読み出すことを可能とする。

【０１７０】

いかなる関数も、他の関数、或いは、自分自身を再帰的に呼び出すことが可能であれば、gMgメソッドモニタリングは、様々な状況及びコンポーネント環境における２つの基本

的な方針に基づいている。第１に、対象メソッドのアドレスを配置すること。第２に、対象メソッドがインターセプトされることである。対象メソッドのアドレス配置のために使用する技術は、COM、非COMオブジェクト、エクスポート機能、或いは、タイプライブラリ・ヘッダで利用可能な情報、関数パラメータ、関数の戻し、メモリ、変数、その他、などを実行する場合に、どのようなコンポーネント技術を使用するかにより決まってくる。COMメソッド(図１６の１６２４～１６３６まで)は、gMgの機能の一例として検討するが、Corba（マサチューセッツ州ニーダム、オブジェクト・マネジメント・グループから利用可能）、IBMのSOM（ニューヨーク州アーモンク、IBM）等の他社からの様々なコンポーネント技術と、その他、ソフトウェア構築に使用される多数のコンポーネント・システムとを有する本原理に、適用可能である。gMgシステムに対してこれらの付加的技術のサポートをさらに拡大することは、異るコンポーネント型のアダプターを構築するタスクである。本技術における当業者にとって容易に理解できるごとく、この説明に提示される技術は、他のコンポーネント技術に適用可能である。

【０１７１】

別の実施例において、gMgメソッドセンサの起動時に、対象メソッドを含むオブジェクトが存在することを、モニタリングシナリオが命令する場合に、gMgのテンポラリ・オブジェクトを使用するCOM内の付加的要素及び技術的変化が、発生する。しかしながら、オブジェクトが、未だ存在しない場合は、ＯＳが作成する新オブジェクトをモニタリングすることにより、新オブジェクトのメソッド位置を与えられる。gMgは、ホストアプリケーションの連続して実行すること（すなわち、メソッドモニタリング中は、実行が中断されない）が維持されるように、動作する。ＯＳ、或いは、ホストアプリケーションにおいて利用可能なインターフェイスに基づく種々のコンポーネント技術内に、メソッドの対象は、埋め込まれている。

【０１７２】

COMコンポーネントが既に作成された後に、gMgメソッドセンサが起動される場合に、COMメソッドのインターセプトをサポートするために、VTtableが最初に検索され、メソッドアドレスを読み出す。VTableまたは仮想テーブルは、クラスに属するメソッドまたは関数へのポインタを有しているCOMクラスにおけるテーブルである。gMgメソッド・シグネチャ・インターセプト技術は、モニタリング作業を確立するために使用される。ここで留意すべきは、COMインターフェイスは、公開された場合には、TypeLibsとして記述され、DLL/OCX内で見出される。

【０１７３】

エクスポートされた関数名がプログラム・モジュール・ヘッダ内のコンポーネントのエクスポート・テーブルにリストされる場合において、別のコンポーネントの種類であるエクスポートされた関数が、DLLコンポーネントにおいて公開されたメソッド・インターフェイスを有する。一旦検索されたgMgメソッドシグネチャモニタリング（図16の1612，1621，1639参照）動作が、メソッドから及びメソッドへのホストアプリケーションの呼び出し及び戻りの間、ポインタの補足を行う。

【０１７４】

一旦、コンポーネントの型（COM、または、エクスポートされた関数）が決定されると、図16AのgMgメソッドシグネチャモニタリングである１６５７は中枢であり、対象のメソッドが検索されると、多種多様のシナリオのgMgメソッドモニタリングにより全面的に適用される。メソッドのエントリポイントの命令コード・パターンとして、メソッドシグネチャは定義され、そのエントリポイントは、gMgシステムにおいて記述され、gMgの検査ツールにより発見される。一旦、定義されると、これらの命令コード・パターンは、呼び出しと戻りとのインターセプトに使用されるメソッドのエントリ・ポイントを検索し、確認するために使用される。メソッドシグネチャモニタリングは、そのパッケージ化、環境、或いは、位置に関係なく、特定されたどの関数に対しても、広く適用される。一旦、対象のメソッドアドレスが、COMのコンポーネント型の中の１つの内部に、或いは、エクスポートされた関数の内部に見つかると、gMgメソッド・シグネチャ・メソッドモニタリング

が、メソッドのインターセプトを行うために適用される。

【０１７５】

別の構成において、呼び出しが、検索されたVTableメソッドから開始され、コンポーネント内のメソッドへの他の内部呼び出しが、VTableメソッド内に存在するとき、gMgメソッドシグネチャモニタリングが、内部メソッド呼び出しを追跡するために使用される場合も同様に、メソッドシグネチャモニタリングが発生する。さらに、メソッドシグネチャモニタリングは、COMオブジェクト内部のVTableメソッドモニタリングのセットアップの一部として使用される。

【０１７６】

gMgメソッドモニタリングの概略を示す図16を参照すると、ＯＳ環境1609内で実行中のホストアプリケーション1603は、ＯＳコード関数LoadLibraryExW1618に対する呼び出しにより、そのDLLコンポーネント1615のロードを開始する。gMgモニタリングは、カーネル関数LoadLibraryExWのインターセプトをセットアプし、対象のDLLの総てのインスタンスが確実に検出されるようにする。DLLは、エクスポートされた関数1620、または、COMのタイププライブラリ1624を有している。COMメソッドモニタリングに対して、対象のCOMオブジェクトの作成の直前である。gMgモニタリングは、インターフェイス・メソッド1636総てに対するポインタを有する、COMのVTable 1633を特定するために、COM作成1627の様々な段階をインターセプトする。これらの段階における実行は、次のように行う。gMgがDLLGetClassObjectメソッドをインターセプトする。DLLGetClassObjectはホストアプリケーションにより呼び出され、IClassFatoryを戻す。次に、gMgは、IClassFactory::CreateInstanceをインターセプトする。次に、IClassFactory::CreateInstanceは、ホストアプリケーションにより呼び出されて、新しく作成されたCOMオブジェクトを戻す。そして、新しく作成されたCOMオブジェクトによる検査と操作により、そのVTableの特定を行う。

【０１７７】

COMオブジェクトが、既にインスタンス化され、その「遅延束縛」（コンパイル時とは対照的に、実行時において、メソッドが、インスタンス化され、VTableのような構造に登録される）メソッドを有しているような場合、コンポーネント内の他の対象（既に束縛されている）メソッドにより使用される同一のVTableにアドレスを与える、自身のテンポラリCOMオブジェクトを、gMgシステムは作成する。一旦、VTableが検索されると、gMgシステムは、自身のメソッドシグネチャモニタリング1639（１６５７において拡張を示す）を、透過的に目立たないように対象メソッドをモニタリングする。エクスポートされた関数１６１８をモニタリングするために、gMgメソッドシグネチャモニタリングは、また、関数の呼び出しをインターセプトする。

【０１７８】

図16Aの1657を参照すると、一旦、メソッドが検索されると、アプリケーションのメソッド呼び出し側1606により、直接的に、または、間接的に、作成される対象メソッドの呼び出し1642及び戻り1645に対するシグネチャメソッドのインターセプトを、gMgモニタリングは、セットアップする。gMgインターセプト・コードは、元の関数の整合性を保持し、ホストアプリケーションの呼び出し及び／または戻りに基づいて、適切なgMgイベントをgMgインタープリタ1660に送り、スクリプトにより要求された関連情報を転送する。

【０１７９】

メソッドの戻りのモニタリングは、汎用モニタリングと特定モニタリングのいずれかに分類される（注：GUIオブジェクトにおけるものとは異なる）。図16Aに示すのは、以下のセクションで説明される、汎用的な戻りを捕捉するための、スタックコピー1648及びスタック割り当て1651（ある場合には、スレッド毎のスタックと呼ばれる）の２つの技術である。特定モニタリング1654は、タイプライブラリのような公開された場所に、自身のパラメータが発見されるメソッドに参照する。しかしながら、汎用メソッド・インタセプションは、関数のパラメータ、或いは、戻り値に関する事前情報なしに、任意の関数によっても動作する。この場合、関数からの呼び出し、或いは、関数からの戻りが発生する場合に、特定情報なしに、イベントはただ生成されるのみである。汎用インタセプション・メソ

ッドの呼び出し及び戻りにおいて、サンクとそれらのコードが実行される。

【０１８０】

gMg追跡モニタリングシステムにおいて、３つの重要なツリー構造（図7、8のGUIオブジェクトツリー、プロセス／スレッド・ツリー）のうち、第三の、すなわちコード・モジュール・ツリーは、モジュール、関数モニタリング、メソッドモニタリングをサポートするために動的に検出されるCOMオブジェクト・インスタンスを、含んでいる。

【０１８１】

メソッドモニタリングセンサにより使用される、モジュール・ツリーを説明する図17を参照すると、モジュール・リスト1705を有する、分離したモジュール・ツリーがルートに存在している。関数モニタツリーの項目1710，1725，1730は、ツリーのどの場所にあってもよい。インターフェイスモニタ1715からのfunction_callイベントに関して、引数（function_callイベント（P1,Ptr_to_COM,・・・））の内の１つがCOMオブジェクトを指し示す場合に、COMオブジェクト1720は、function_callイベントの子になることができる。スクリプトによって起動されて、選択された関数のみがモニタリングされる。モジュール・ツリーは、スクリプトモニタリング要求に応えるために必要なもののみを、選択的に有している。さらに、スクリプト中のキーワードにより駆動されて、gMgインタープリタは、GUIオブジェクトツリーとモジュール・ツリーを更新し、実際のGUIアプリケーションをモデル化し、アプリケーションにおける関数イベントをサポートする。

【０１８２】

開始時には、モジュール・ツリーは、プログラム・モジュール1705で満たされる。ツリー内のインターフェイスと関数オブジェクトは、スクリプトからの要求によってのみ、作成される。一旦、実行されると、モジュール・ツリーは、LoadLibraryExWがインターセプトの呼び出し及び戻りを使用して維持される。

【０１８３】

エクスポート関数において、モニタへのスクリプトの要求は、モジュール・ツリーにおけるモジュール・オブジェクトの子関数Call Objectに、加えられる。注目するCOMオブジェクトのインターフェイス・オブジェクトはまた、参照のために、モジュール・オブジェクトの子オブジェクトとして加えられ、オブジェクトの前記に示した型の作成をモニタリングする。望ましい型のCOMオブジェクトは、それが作成されると、インターフェイス・オブジェクトの子として加えられる。スクリプトがインターフェイスのこれらメソッドのモニタリングを要求する場合は、メソッド関数呼び出しオブジェクトは、インターフェイス・オブジェクトの子、または、COMオブジェクトの子として加えられる。インターセプトされたメソッド呼び出しから発見されるCOMオブジェクトは、スクリプトを参照するために一時的に作成され、常にモジュール・ツリー内に存在するわけではない。使用後に、COMオブジェクトは、エンジンにより、ガベージコレクトされる。

【０１８４】

メソッドシグネチャモニタリングは、関数の呼び出しをモニタリングするための技術を提供し、gMgメソッドイベント作成し、プロパティの読み出しを行う様々な状況に適応される。メソッドシグネチャモニタリングは、メソッド関数の認識されたメソッドシグネチャを表すバイト命令コード・パターンを識別する。一旦、探索され検出された関数は、アプリケーション・コンポーネント内で発生する他の内部イベントまたはプロセスと同様、GUIイベント及びプロパティなどの項目に対して、インターフェイスを提供する。メソッドシグネチャは、任意の種類の呼び出しに関するどのコンポーネント環境と、捕捉関数パラメータと、に対して広く適用される。一旦、対象関数へのすべての呼び出しを追跡する手続きのアドレスが決定されると、gMgは、呼び出しインターセプト・ルーチン及びINFO構造インスタンスによるジャンプ命令によって、そのアドレスをリンクする。このINFO構造は、関数コードに対するバイト・ポインタと、インターセプト手続きに対するボイドポインタと、そのインスタンスを持っているInterceptMultEdxInfoと呼ばれるINFO構造と、を有している。マルチインスタンスのサポートの一部として、CPUレジスタの使用（すなわち、インスタンス・レジスタ）、或いは、他の機構のような、関数呼び出しのインター

セプトに関する情報を、運んで渡す種々の技術が存在する。

【０１８５】

図18を参照すると、そこには、メソッド・モニタの作成について、説明されている。gMgスクリプト1833に含まれるgMgスクリプトモニタコマンド1836により開始されて、先ず、関数のアドレスが、MakeFunctionMonitor機構により特定される。Ｎ個のメソッド1827を含むモジュールN1806を与えられて、関数のアドレスを特定し、インターセプトの要求を実行する用意として、適切なモジュール・リストは、既に見つけられ、Createtoolhelp32snapshot 1815、Mod32First 1809のような、モジュールを列挙するAPIを使用して、インターセプトされたLoadLibraryExWの呼び出し中に、このモジュールリストは、自動的に保持される。本技術における当業者ならば容易に理解するであろうが、これらのAPIは、更新されるＯＳで利用可能な、公表されているような関数により、置き換えが可能である。モジュール・リスト1803は、インタセプションの要求時に、既に準備されている。一旦、エクスポートされた関数名またはCOMメソッドコードが検索されると、MakeFunctionMonitor1851は、関数名と、モジュール・オブジェクトと、インターフェイス・オブジェクトまたはCOMオブジェクトを受取り、MakeFunctionを呼び出す。そして、オブジェクトが関数インターセプト構造を有しているgMgオブジェクト・クラスの派生クラスとして、関数モニタオブジェクト1860が、作成される。使用されたプロパティは、ポインタを元の手続きに戻すgMgオブジェクト・インスタンスを有しており、成功すれば真に設定される。

【０１８６】

MakeFunctionMonitorは、ウィンドウ、HTML DOC、或いは、タイマと違い、対象関数の呼び出された回数のような様々なプロパティを有することができる関数オブジェクト型である「関数モニタオブジェクト」型である。関数のインスタンス及び割り当ての総ては、グローバルに追跡可能である。インターセプト要求がある場合は、所望の関数を有するモジュールは、インターセプト要求中、或いは、既に決定されたことが認識されたことを通知される。インターセプト要求があった場合は、所望の関数を有するモジュールは、インターセプト要求中、或いは、既に要求が決定されていることのいずれかが通知される。gMgスクリプトが、モジュールを渡し、GetProcAddress 1824が、関数本体（例えば、モジュール・オブジェクト用のMakeFunctionMonitorが、InterceptCallMultEDX1818の呼び出し内部の関数名で、関数GetProcAddress1824を使用する）の開始アドレスを発見するために使用される。

【０１８７】

Callintercept 1848は、gMgスクリプト・コマンド1839の結果として間接的に呼び出されて、インターセプションを実行し、関数オブジェクト1860に位置されるINFO構造1863にアクセスする。再帰を避けるため、gMgスクリプト・コマンドが、インターセプト処理中に、不注意にインターセプトされた関数を呼び出すならば、gMglockにおけるlockDが実行される。インターセプションが試みられる前に、ロッキング機構は、Callintercept関数1848が同一スレッドによりロック中でないかについて、1842でチェックし、ロック中の場合は、1845でCallinterceptはスキップされる。インターセプションが成功したと見なして、インターセプション・コードが、関数モニタオブジェクトにキャストされるこのINFO構造に対するグローバルポインタを渡す。

【０１８８】

関数Functionintercept 1866は、gMgの関数オブジェクト1860を持っている。一旦、インターセプションが成功すると発行されるgMgイベント1869は、そのイベントと、実行中の総てのスクリプト1875とを処理する関数呼び出しDo_gMgEvent 1872に渡される。Do_gMgEventは、関数呼び出しイベントを受け取り、そのイベントに基づいて動作し、動作完了すると、グローバルポインタとグローバルロックを除去する。

【０１８９】

GetstringとGetNum関数が使用するそのメソッドイベントは、gMgインタープリタに受け取られて、メソッドEventProperty1869と呼ばれる。これらのGetstringとGetNum関数は、gMgオブジェクトの各型に対して暗黙的であり、仮に自身の型が識別されない場合は、お

互いを呼び出す。gMgスクリプトは、インデックスにより、ストリング・イベントのプロパティを要求して、プロパティが特定されない場合は、スクリプト・インタプリタ・コードは、gMgオブジェクトの各型に対する、一連の関数における他の型の内の１つを呼び出す。

【０１９０】

Event.s1などのスクリプトの内部において、解釈は、getstringイベントのプロパティを探す一連の呼び出しとなる場合は、s1を表すインデックスが渡される。Nullポインタに対するチェックが行われ、グローバルイベントの選択値は、関数呼び出しイベントが処理中であるかどうかを判定する。処理中のcreateイベントなどのgMgイベントの別の型を受け取った場合、ホスト・アプリケーション・ツリーの読み取りとなるように、オブジェクトツリーの読み取りの優先度が高くなることが、好ましい。イベントが、関数呼び出しを処理している場合、元の呼び出しパラメータが存在している状況において、スタックにおけるオフセットであるポインタが設定される。各パラメータは、このスタックの位置から、読み出すことができる。

【０１９１】

１回の実行により、文字列の長さが未知のパラメータを処理する関数が、与えられる。この関数は、文字列の位置が未定であるかどうかを判定し、見つけた値が、各々１バイト文字であるか、２バイト文字であるかを、プラス、または、マイナスの値としてそれぞれ戻す。文字列が何も存在しない場合には、関数は、ゼロを戻す。クエリー中である文字列プロパティが、スクリプトで処理中である現在の呼び出しインターセプト・イベント1869に、適するかどうかを、判定することができる。適する場合は、関数call Do_gMgEvent 1872を処理している最中のcall Interceptargument1878が存在する。これは、文字列プロパティ変数が、ゼロではなく、スタック（すなわち、スタック・アドレスに在る）からINFO構造のアドレスを指し示す場合のみに、存在するのである。

【０１９２】

物理的にオブジェクト構造内に在り、自身とホスティング・クラス（使用される位置にのみ存在している）に対するポインタが存在しており、このポインタはインターセプションに関する情報と、関数オブジェクト1860であるgMgオブジェクトに対するポインタとを、有している。これら総てのクラスは、メソッド・インターセプション・センサに対してローカルであり、内蔵されている。

【０１９３】

メソッドシグネチャを説明する図19を参照すると、gMgメソッドセンサを実行するために、gMgメソッドモニタリングシステムは、メソッドシグネチャ1930と呼ばれる対象メソッドの捕捉されたバイト・パターン1920の配列を経由してループし、そのパターンを、メソッド1905の位置を検出するための検査ツールにより発見される、実行時の対象関数バイト命令コード・パターン1910と、比較する。シグネチャを定義するために、種々の命令コードを表すバイトの文字列であるマクロが、作成される。定義が、関数エントリポイントにおける標準的な命令コードの作成に、使用される。シグネチャは、また、don't_careコードとsignature_endコードのような、特別に定義されたコードから構成されている。例えば、コンパイルされたコードの特別バージョンを有する特別のライブラリは、様々なバージョンにおいて異なるDLLのバージョンにより決定される関数において、ある種の命令の固定された順序を持っている。関数のコードは、エントリ・コード1910が認識可能であるかどうかを判定するために分析され、認識可能であれば、一致するシグネチャに対するポンタが戻される。この評価により、ジャンプ命令を挿入する前に、どこか他の場所でどれだけの数のコードがコピーされる必要があるかを示し、また、コードを継続して実行するために、何処へ戻るべきかを示すものである。ジャンプ命令がインターセプトされた関数の一部として既に見つけられている場合と、検出されたgMgがアプリケーション関数の呼び出しを既にインターセプトしている場合とは、例外である。gMgモニタリングプロセスは再帰的になり得ず、プロセスは異常終了される。

【０１９４】

この命令コード分析技術は、また、様々の派生的CPUのクラス内で動作する。単一スレッドのアプリケーションにおいて、これに代わる技術が適用できる場合もあり得る。すなわち、gMgコードに入る前に元の対象コードを復帰させ、インターセプションを効果的に無効にし、そして、gMgコード（インターセプション・ハンドラ）から戻った後、ジャンプ命令を対象関数コードの開始時に戻すことにより、インターセプションを再び可能とする技術である。文書化された総てのシグネチャに対して、Signature_terminator 1935が検索されるまで、各バイト・コードを経由する処理が繰り返される。その値が一致しない場合は、次のシグネチャの処理が行われるが、Signature_terminator値に至るまで一致状態が見つかる場合は、開始位置の値が保存され、その長さが戻される。

【０１９５】

一旦、シグネチャが発見されると、インターセプションを成功させるために、何もない命令コード境界1955上へオーバーレイされるgMgコードのジャンプ命令を、特定し配置する。ジャンプ命令は、命令コードの真ん中にジャンプすることは出来ないので、正確な命令コード境界の位置を知る必要がある。５バイトのオーバーレイの場合、最初の４バイトは、２つの命令コード1960と1965から構成され、５番目のバイトは、関連するアドレス1970を有している。この関連するアドレスは、元関数においてオーバーレイがどの場所から実行されたかに関連するため、オーバーレイの移動及び実行を防止する。この防止作用は、関連する命令コードを絶対位置に移すことによって、打開される。

【０１９６】

対象関数1940において、対象とするオーバーレイに関して、５バイトが最小でも必要であり1955（signature_endを加えて）、32ビットのCPUに、ジャンプ命令を設定する。このオーバーレイは、また、７または10バイトであることが可能であるが、16バイトを超えることはない。一旦、５バイトの命令コードが発見されると、関数コードの置き換え時に、どの場所にジャンプして戻るかを示す特定値signature_endを挿入する。元の命令がコピーされて、命令コードの全列がオーバイレイ1945され、インターセプション・ルーティン1975へのジャンプ命令に置き換えられる。コピーされた命令コードは、短い値1980の配列に保存されているバイト値である。インターセプション・ルーティン1975からの戻りが行われると、プログラムフローは、このコードinterception/continuationポイント1950に戻る。

【０１９７】

関数は１以上のパラメータを渡すので、モニタリングされるメソッド（すなわち、Getstringfunctioncall()）のパラメータをフェッチする作業を行うgMg関数が、また、存在する。この場合、命令コードは、数値型と文字列型との双方であってもよい。例えば、save event.s1（ここで、s1は文字列のプロパティである）が与えられると、N-Eventプロパティ・リスト（NEP）から得られるワード定数である１つのパラメータを持つ関数Getstringeventproperty()が存在する。NEPリストは、総てが文字列イベント・プロパテイである実行可能なワード定数のリストを、定義する。Getstringeventprops()が呼び出されて、イベント・プロパテイ番号を渡す。イベント・プロパテイ・セルは、ブラウザURL S1～SnにおけるURLフレーム・ヘッダのような、文字列イベント・プロパテイである。このイベント・プロパテイは、引数やブール・イベント・プロパテイとして列挙され、命令コードにおいて引数として使用される。数値イベント・プロパテイは、同様の方法で発見され、引数として、ブール・イベント・プロパテイにおいて列挙される。文字列イベントは、数値プロパティとして“１”で開始され、ブール・イベント・プロパテイは、実行時において1000で開始される。

【０１９８】

パラメータが、ASCIIであるか、または、Unicodeであるかを識別するために、API（例えば、IsTextUnicode）が呼び出されるか、或いは、ユニコードのstrlengthパラメータと共に、ASCII文字やunicode文字について、カウントが行われる。ASCII対象をループし、全ASCIIの合計長及び文字列数を見つけることによって、ASCII文字について、カウントが行われる。ユニコード文字の大部分がASCIIコード文字であり、その文字列が長過ぎない

ならば、それは有効なユニコード文字列である可能性もある。この条件と異なる場合は、この文字は、ユニコード文字列の可能性はないと考えられて、ゼロとされる。負カウントがASCII文字列であることを示す場合において、大部分の文字列は戻され、またヌル文字列を示す場合は、ゼロが戻される。文字列が何も存在しなければ、空の文字列が戻される。このルーチンは、また、ポインタを、ワイド文字一式を含むスタックに戻す。

【０１９９】

インターセプションが、開始時に、或いは、対象関数コードの実行途中に、発生する可能性がある。gMgのモニタリングは、ゼロのオフセットを基本とする関数を使用し、関数の実行途中にインターセプション・ポイントが発生する場合、ゼロでないオフセットを基本とする関数を実行する。同一関数のマルチインスタンスのインターセプションが発生でき、それを検出することも可能である。

【０２００】

コードが、gMgサンクへのジャンプ命令のために、検査される。コードが見つかった場合は、関数は既にインターセプトされており、インターセプションのセットアッププロセスは、異常終了される。さもなければ、関数は、インターセプションを行うためにセットアップされる。対象メソッドコードを修正するためには、コードを有しているメモリブロックを制御する機能が必要とされる。gMgのChangememory関数は、インターセプトされるべき関数を有しているメモリブロックの許可を変更する。gMgのchangemeoryルーチンは、実行オプションを削除しread/writeオプションを設定することによって、このメモリブロックの属性を変更する。関数インターセプションは、メモリブロックのプロパティの変更に依存しており、ホスト・アプリケーション・コードにおいて、そのプロパティの変更を行うことなく失敗する。元の保護機能を戻し、そして、属性を回復するために、ハンドルが使用される。メモリの変更が頻繁に起きる可能性があるので、その処理は、メモリ変更とメモリ復旧の双方に分割される。

【０２０１】

サンクの利用は、実行時において、識別子の遅延束縛から実在するオブジェクトに至る、サポート機構の範囲で使用される１つの技術であり、或いは、メソッドの呼び出し中にマルチインスタンスを追跡する技術である。別の状況で適用される場合は、サンクは、コードに対するポインタとして、実関数に対するポインタであることが可能であり、或いは、実関数へ戻るサンク空間に存在することがあり得るし、或いは、元の対象実関数のアドレスでもあり得る。書き換え及び実行可能な属性を有するメモリの一部であるインターセプション・サンクを指し示すために、ポインタが、モニタリングされるコード内に確立される。サンクは、複数の呼び出しを、または、戻りインスタンスを、追跡するために使用される。多数の関数のインターセプトのサポートを容易にするために、インターセプト用コードは、インターセプト・サンクとレジューム・サンクを除いて、共用することが可能である。各関数のエントリ・コードは、関数毎に分離したインターセプト・サンクへのジャンプと置き換えられる。その一方で、インターセプト・サンクは、特定のインターセプトを表すエンジン・データ構造(“INFO”)に対するポインタを供給し、多数のインターセプトされた関数と共有される共通ハンドラ関数を呼び出す。サンクの役割とその利用は、仮想テーブルとCOMモニタリングのセクションで検討する。

【０２０２】

（関数のパラメータ、または、関数のフォーマットが未知である）汎用関数呼び出しインターセプションの場合、スクリプト処理後にハンドラ関数は戻り、インターセプト・サンクは、インターセプトされた元の関数のオーバーレイされたエントリコードのコピーを有しているレジューム・サンクへ、ジャンプする。インターセプトされた元の関数には、インターセプトされた関数の残り部分へのジャンプが後続する。特定関数インターセプト（ここでは、関数のパラメータと関数のフォーマットが、既知である）に対しては、ハンドラ関数は、実関数と正確に同じフォーマットでアプリケーションにより渡されるパラメータを、受け取り、さらに、ハンドラ関数は、インターセプト・サンクにより共給されるインターセプトされた関数用のINFOpointerを受け取る。スタックに追加することにより

、または、使用されていないCPUレジスタに配置することにより、追加的INFOpointerパラメータは、インターセプト・サンクにより渡されることが可能である。ハンドラ関数は、（INFOstructureを経由して特定可能である）レジューム・サンク（INFOpointerを経由して特定可能である）を経由して、元の関数を呼び出し、最初の関数の結果を、呼び出しアプリケーションへ戻すことが出来る。ハンドラ関数は、呼び出されている物は何かを説明するINFOを参照しながら、呼び出しのスクリプトを通知する。ハンドラ関数は、また、スタック上のパラメータの位置に注目し、スクリプトが、数値、または、文字列として、そのパラメータを参照することができる。インターセプトされた関数のパラメータが既知である場合は（どちらかの公開された関数、COM型ライブラリにおけるCOMメソッド、タイプされたパラメータ等）、そのパラメータは、COMオブジェクト（例えば、“この”ポインタ）または（ウィンドウ・ハンドルから導かれた）ウィンドウ・オブジェクトとして、スクリプト中で、参照可能である。特別の場合（例えば、LoadLibraryExW,、DLLGetClassObject、そしてIclassFactory::CreateInstance）、ハンドラ関数は、元の関数（新モジュール・オブジェクト、intercept IVlassFactory::CreateInstanceと、各戻り値の検査）の呼び出しの前、または、後に別の動作をする。

【０２０３】

サンクの場合、サンクの割り当てが各スレッドを経由してロックされる必要のある複数のＮ個のスレッドが、サポートされる。各スレッドが自分自身の戻りサンクを必要とする場合、例えば、２つのスレッドは、２つの異なる関数を呼び出すことが出来るか、或いは、同一の関数であってもよいので、スレッドによるロックが必要である。ある状況においては、呼び出しの各インスタンスは、同一関数を、再帰的呼び出しを行うことも可能である。さらに、処理作業は、関数がC型、または、パスカル型呼び出しを使用するか否かにかかわらず、各呼び出しの間に、サンクを使用して急いで作成されたものを、動的に解放できるならば、総てのスレッドに対する並列スタックの作成は容易になる。gMgメソッドモニタリング内におけるサンクの適用は、ホスト・アプリケーション・コンポーネント・メソッド呼び出しと戻りからの広範囲のイベントを発生するために用いられる。

【０２０４】

メソッドモニタリング用のサンクの割り当てを示す図20を参照すると、有効長５以上のメソッドシグネチャが発見される場合が示される。有効値が見出される場合、Virtualallocation関数2006は、rewriteの属性を有するメモリ2009のブロックを割り当て、24バイトを動的に一度で割り当てる。サンクを割り当てするために、バッファメモリの再利用可能なブロックが使用される。総ての文字列式に対するプログラムには、１つの固定バッファが存在し、字列式の計算中に、総てのgMg命令後、空にする必要のある使用されていない一時的な文字列を使用する。このオーバーヘッドは、定期的にリセットする必要がある。プライベートgMgヒープは、また、アプリケーション・ヒープを使用する代わりに使用されて、何かのgMgエラーが起こる場合に、さらに隔絶されて、ホストアプリケーションへの影響を回避する。プライベートヒープ関数は、ヒープを作成し、そして、総ての新しい演算子が新しいプライベートヒープによって使用されねばならないように、総ての割り当てを行なうために、ハンドルを提供する。InterceptcallmultEDX 2003は、メモリの２つのブロックを割り当てる。１つは、コードが呼び出し2012をインターセプトし、もう１つは、対象関数に対して、処理の継続を確保するために元のコード2015を保存する。gMgのインターセプト実行部の内部で、一連のコードが挿入される。すなわち、総てのデータのレジスタをプッシュするPUSHAD命令2018、その後、インターセプトのインスタンスと関連付けられる必要のあるINFO構造のアドレスである長い値のプッシュが実行される。

【０２０５】

InterceptmultEDX()関数は、効率が向上するように、レジスタを最初に確保することなしに、インスタンス・レジスタ（例えば、EDXレジスタ）が、呼び出された関数により使用可能であるとの仮定に基づいている。CPUスタックの変更は、避けるべきであるので、インフォ・データに対してmoveEDXレジスタを書き込む。そして、データEDXレジスタを移動させ、次のコードの一部へインスタンス情報を渡す。この方法は、どの関数がインター

セプトされている最中かを知る場合に使用することができ、また、Loadlibraryのなどのカーネル関数のような正確に同一パラメータを有する同じテンプレートの作成に使用される。

【０２０６】

RAM（すなわち、HEAP）に保存された情報は、同じEDX目的に対しては、使用できない。それは、共有のアクセスに起因して、他のスレッドとの衝突が起こる可能性があるからであり、従って、スレッドが安全ではない。レジスタ、或いは、スタックの使用のみが、同期や他の手間のかかる操作を必要とすることなくインスタンス情報を渡す、スレッドの安全を確保する方法である。RAMが、インスタンス情報を渡すために使用可能なこともあるが、しかし、かなり効率の悪い可能性がある。各呼び出し時にRAMを使用することにより、インターセプタは、インターセプトされた関数のアドレスに基づいて、検索を実行する。また、検索テーブル（インフォ・インスタンスに対する関数のアドレスマップ）の保持は、手間のかかる操作であるスレッドの安全性を確保するように同期する必要がある。

【０２０７】

ある場合は、スタックが使用可能であるが、RAMを使用する場合において、この方法は、透過性の維持に関して不都合がある。インターセプション関数は、元の関数と完全に置き換わる必要があるが、処理中に、元の関数を呼び出す可能性がある。このことは、制御の観点から困難である。この場合、インターセプション処理は、元の関数と同様にスタックを一掃し、正確に同一パラメータを持つ必要があり、さらに、同時に、１以上のパラメータ、すなわち、インスタンス、または、コンテキスト情報を、モニタリングシステムのインターセプションをサポートするために追加する必要がある。いくつかの対象関数が、ジャンプ命令を介して起動され、スタックはプッシュにより変更できない（この場合、プッシュは、対象関数のパラメータの戻りアドレスと戻りアドレスの後に配置される特別の情報となる）。この状況において、インターセプション・ルーチンのコードは、元のパラメータには、容易にアクセスはできない場合があるため、元の関数に行ったものと同じ方法では、そのスタックを一掃出来ない。例えば、

```
Stack:
…
dwFlags
hFile
lpLibFileName
Return-address
//プッシュは、パラメータを「追加」しない。プッシュは、return-addressに従う:
dwFlags
hFile
lpLibFileName
Return-address
Info //インターセプション・ルーチン用
```

EDXの代替に関して、“MOV EDX, infoAdrr”の代りに“PUSH infoAddr"を使用することは、スレッドセーフな実行可能なオプションである可能性はあるが、この代替は、CPUレジスタを使用する場合より速度が遅い。

【０２０８】

CPUレジスタを使用するメリットがある。例えば、レジスタを経由してインスタンスのアドレスを渡すことは、インスタンスのコンテキスト情報を渡すために最適である。この方法は、レジスタに渡すのと同じINFOを経由して、いかなるパラメータも、モニタリングされたメソッド/関数へ渡されない場合に限り動作する。EDXレジスタを経由して渡されるパラメータは、エクスポートされた関数用としては、決して使用されないことが発見されている。しかしながら、EDXを使用しているgMgモニタリングにおいて、ライト・クラッシュを生じさせる可能性のあるコードを書くことも可能であるが、しかし、アッセンブラを多分必要とするであろうし、利用可能なコンパイラと共に使用される可能性はない。この

クラッシュは、エクスポートされたCOM関数に対して、発生の可能性は殆どないといえる。さらに、“EDXを使用することは、“sysenter”命令を通してなされるカーネル・レベルの呼び出しインターセプションに対しては、或いは、EAXとEDXをパラメータとして頻繁に使用するその他の内部呼び出しに対しては、有効でない可能性がある。しかしながら、これらの条件では、別のレジスタが置き換えられる。EDXを使用すると、また、コード・サイズが小さくなる。本発明は、EDXの使用は、スタック渡しが合理的である汎用のインターセプトに対する必要条件であることを、それほど制限はしない。しかしながら、元の複数のインスタンスに対して新しいハンドラ関数を置き換える場合には、CPUレジスタにおける付加的なコンテキスト情報を渡すことは、スタック渡し、または、メモリ渡しである場合よりも、はるかに優れている。

```
Multiple Instance:
HRESULT STDMETODCALLTYPE iIClassFactory_CreateInstance(
        IClassFactory*This,
        /*[unique][in]*/IUnknown    *pUnkOuter,
        /*[in]*/REFIID        riid,
        /*[iid_is][out]*/ void       **ppvObject)
        HRESULT_stdcall iDLLGetClassObject( REFCLSID rclsid,
        REFIID  riid,
        LPVOID* ppv)
Single Instance:
        LoadLibraryExW
```

再び、図20を参照すると、一度、対象となった関数のパラメータ2036が知られると、関数はインターセプトされ、それから、gMg「プロキシ」（又は、インターセプト）関数2045で置き換えられる。元の呼び出し元2024は、関数結果待ちでブロックされた状態のままとなる。gMgプロキシ関数2045は、元の関数2048、2042を呼び出し、戻り値を受取り、そして、その後、結果2051と共に元の呼び出し元2024に戻る。gMgは、元の関数2030と全く等しいパラメータを有するプロキシ関数2045を生成するが、インスタンス情報に対し、少なくとも１つの別のgMgパラメータ２０３３を加える。これは、異なるDLL2060、2063、2066内の複数の関数を、すべて同じメソッド名2072、2075、2078で、及び同一の関数シグネチャで、フックする必要がある場合になされる。インターセプト・ルーチンに情報を転送するため、EDXレジスタは、呼び出し傍受が行われる間、元のアプリケーションのスタックの変更を防ぐ特別なgMgINFOパラメータを渡すためにも用いられる。

【０２０９】

gMgインターセプト関数は、INFO構造のインスタンスのアドレスのプッシュをアセンブルし、それからインターセプト２０４５、２０４８を行う間、呼び出されるように特定化される呼び出しインターセプト手続きを構築する。gMgは、置き換えられる関数の元のコードのコピーを処理し、そして、それをセーブする２０４８。その後、すべてのレジスタは、サンク（アドレス又はポインタを保持する構造）２０５４内で、元の呼び出しルーチンへ戻るために取り出され、コードにジャンプを行う。元のルーチンに対するコードは、実際のルーチンを指し示すprealprocを用いてアクセスされる。

【０２１０】

ホストアプリケーションのコードからの呼び出しの間、ジャンプ命令２０３９は、サンクを、元の対象関数のコード内のインターセプト点から、関連するgMgインターセプトルーチン２０４５にアセンブルされる。元のコードは、コピーされ２０４８、それから、インターセプトルーチンが戻るところで、サンクが連続ポイント２０４２（対象コード内で）に戻るように、別のジャンプ命令がアセンブルされる。

【０２１１】

これにより、２つのサンクの構築が完了する。１つはインターセプト用、そしてもう１つは戻り用である。別の実施形態において、２つのサンクは、より大きな１つのサンクに統合される。そのサンクは、内部gMgインターセプトから元のルーチンを直接呼び出すこ

とを許可する、又は、元のルーチンにジャンプするアドレスprealprocへの呼び出しを許可する結果、インターセプトをバイパスする。

【０２１２】

セットアップ、またはメソッドモニタリング処理、及びサンクを使用する際に生じる多くの課題がある。すなわち、マルチスレッド・アプリケーション間のスレッドの衝突、及びgMg GUI オブジェクトツリー更新に関するイベント処理である。置き換えられたコードを不注意に実行する可能性を許可しているため、別のスレッドがそれを実行するサンクを呼び出している場合、元のサンクコードを変更するか又はオーバーレイする間に、問題が生じる。この問題点を回避するために、gMgシステムは、異なる状況に適応するために動的に用いられる技術、すなわち、スリープ、又はスレッドサスペンションを使用する。

【０２１３】

メソッドモニタリング動作中のスレッドの不一致を回避するため、gMgがコード変更を行う時間に、複数のスレッドが同じコードを実行することを試みる場合、スリープ技術は、不一致を可能な限り避ける試みとして、Nミリ秒に対するスリープ関数を発行する。スリープは、スレッドを一時停止し、このスレッドに対し、タイムスライスを再起動する。これは、エラーが生じる可能性を減少させるため、スレッドが、その操作を完了するための完全なタイムスライスを有することを意味する。上述のように、ジャンプ命令は、gMgインターセプトルーチンにジャンプする元のコードの変更の一部として、アッセンブルされる。これは、オフセットアドレスをジャンプコード内に置き、そしてオフセットアドレスを計算するマクロ、又はインライン関数として実行される。

【０２１４】

別の技術は、API呼び出しにより、処理中の他のすべてのスレッドを、一時的に停止することである。スレッドが例外を生じさせる場合、例外は、フックを用いて幾つかのメッセージに応答してインターセプトされることで、スレッドに例外を生じさせるが、スレッドを巧みに再起動させる。問題が生じる可能性をさらに減少させるため、スリープに加え、スレッドの優先を「高」にすることができる。別のスレッドが起動するが、衝突する機会はより少なくなる。複数のCPUを有する環境でモニタリングを行う場合を除いて、スリープ技術単独で、十分であろう。複数のCPUで、ＯＳは、通常、関数／メソッドのインターセプトが行われる間、処理の実行を、異なるプロセスに割り当てる。

【０２１５】

インポートテーブルを通して呼び出される関数に対するマルチ・スレッドの実行に関し、インポートテーブルを用いる特別なインターセプト機構は、基本的に呼び出しをロックし、インターセプトに対するコード変更を行い、それからインポートテーブルの値を変更することにより呼び出しを解除する。関数Getprocaddressは、インポートテーブル内にアドレスを提供する。

【０２１６】

さらに、メソッドモニタリングは、GUIオブジェクトツリーのスキャンとインタリーブされ、ホストアプリケーションと調整し、GUIオブジェクトツリーと同期化されるように保持する。ObjectInterfaceEPTで始まる関数のチェーンを呼び出すイベント及びツリースキャン制御機構がある。GetStringEventProperty()は、最初に呼び出され、そしてインデックスを渡すが、プロパティが存在しない場合は、GetStringWinEventsProperty()への呼び出しが行われ、次いでStringGetWebEventProperty()への呼び出しが行われる。モニタリングされる関数が呼び出される前に、インターセプトされる関数を呼び出すアプリケーションに先立ち、gMgが、現在のイベントで潜在的にモニタリングを行っているか判定するために、テストが実行される。この時に他のイベントが処理されており、検出されるイベントが存在する場合において、ホストアプリケーションGUIツリーは、アプリケーション・ツリーの走査の優先順位を更新する方法として、再スキャンされる。さもなければ、アプリケーション・ツリーがスキャンされる時間のみ、通常、タイマイベント上にある。アプリケーションがビジーである場合、インターセプトされている多くのイベントが生成される可能性がある。さらに、チェック中に、アプリケーション・ツリーへの変更を保持

するＯＳws_timerイベントが制御される。幾つかの場合、またgMgイベントを処理するときに、アプリケーション・ツリーへの変更最近なされない場合、生成イベントまたは破棄イベントを、別の型のイベントの処理内部において生じさせるツリーが再スキャンされる。この理由により、チェックは実行されなければならない。

【０２１７】

異なる種類のアプリケーション・コンポーネントのパッケージングは、異なるメソッド構造を提供し、したがって、対象のメソッド／関数及びそれらのインターフェイスを特定するために、異なるメソッドが必要となる。COMは、マイクロソフトウィンドウズ(登録商標)上でエンタープライズ・ホスト・アプリケーションを構築するために用いられる重要なコンポーネント技術であり、プロパティの読み出し又はイベント検出用のＶｔａｂｌｅを介して行われるCOMメソッドの検出は、コンポーネントのモニタリング用に要求されるgMg技術である。

【０２１８】

クラスの特定インスタンスであるCOMオブジェクトのインスタンス内において、まずgMgの主要な目的は、オブジェクトのテーブルを見つけることである。オブジェクトが存在しない場合、VTableは、その構築段階の間、読み出されないが、オブジェクトが既に存在する場合、gMgは、自身の一時的なオブジェクトの生成することで、VTableを検索する。任意の新しいオブジェクトが生成される時、そのオブジェクトは、既にインスタント化されている既存のオブジェクトとして、同じVTableをアクセスする。gMgは、存在しているVTableを検索するために使用される一時的なオブジェクトを生成する。
そのVTableは、コンポーネントに属する一組のメソッドによって共有される。一度、VTableが見出されると、その後、gMgメソッド・シグネチャ・インターセプトルーチンが、モニタリング処理を実行するために立ち上げられる。

【０２１９】

クラスは、メモリ内にオブジェクトをインスタンス化するために使用され、ＧＵＩＤ（マイクロソフト・ウィンドウズ(登録商標)において任意のプログラミングコンポーネントを識別するために一般に使用される個有の識別子）及びインターフェイス名の両方により識別される。タイプライブラリにおいて見出されることで、gMgスクリプト・ライタは、gMg検査ツールを用いて、インターフェイス名を介し、ｉｄを提供する。DLLによって公開されることで、ｉｄは、gMgtool VTree、又はマイクロソフトのOLE Viewのようなユーティリティ内において見つけられる。DLLは、COMオブジェクトが記述されたタイプライブラリを含む。一度ｉｄが見つけられると、オブジェクト内のクラスに関連するインターフェイスのＧＵＩＤが見つけられる。その後、実行時において、オブジェクトのインターフェイスのｉｄは、スクリプトから生成される。

【０２２０】

クラスは、複数のインターフェイス（例えば、IUnknown、IDispatch、等）を実装することができる。オブジェクト内で実装される、クラスの他のインターフェイスすべては、一般には、レジストリ内で公開される。クラスのｉｄは、注目するオブジェクトにインターフェイスを生成するために使用される。逆に、インターフェイスのｉｄが、一度知られると、そのインターフェイスを実装するクラスのｉｄを得ることが可能となる。

【０２２１】

クラスｉｄ及びインターフェイスｉｄが与えられることで、オブジェクトのインスタンスが生成される。インスタンスは、非常に短時間で生成され、そして、オブジェクトが破棄される前に、VTableのアドレスは、それが、メモリ内に、発見されるように保存される。インターフェイスは、インターフェイスを実行するモジュールがロードされて初めて、モニタリングされる。モジュールオブジェクト上に、gMgを生成するイベントは、モジュールが存在する意味を受け取る。インターフェイスは、今後、VTableが、メモリ内にインスタンス化されるように、モニタリングされる。コンパイラは、予め生成され、そして、実行時間にソースコードにあるすべての可能なVTableを、コンパイル時間において、生成する。

【０２２２】
コードへのポインタを有する固定化されたテーブルとして、コンパイルされるVtableは、すべてのインスタンスにより共有される。オブジェクトのインスタンスが生成される時、メモリブロックは、VTableを指し示す第一のdwordを、割り当てられる。例えば、すべて同じ型の１０個のオブジェクトが存在する場合、このオブジェクトは、総て同じVTableを共有する。ここでは、VTableはインターフェイスと同義語である。

【０２２３】
様々なコンピュータ・プログラム言語におけるコンパイルに関し、例えば、標準Ｃプログラミングインターフェイスが、そのメソッドを実行するためにデータ構造を用いる一方で、Ｃ＋＋で構築されるコンポーネントは、仮想関数を用いる。検討した様に、仮想関数は、VTableとして実行され、対応するメソッドへのポインタを有する。さらに、VTableは、COM定義の一部として（すなわち、IUnknown、IDispatch、及び他の選択されるカスタムセット）インターフェイスの多数の組を有する。複数のインスタンスを有することのできるCOMオブジェクトが与えられることで、すべてのインスタンスのセットは、IUnknownから生成される。さらに、その生成は、VTableコンテントを決定する。例えば、インターフェイスが、Invokeから生成される場合、VTableは、メソッドに対するInvokeポインタへのポインタを有する。しかし、インターフェイスが、IDispatchから生成されない場合、そしてQueryInterfaceが実行される場合、異なるVTableが構築されることになる。

【０２２４】
複数のクラスの継承のような、様々な要因に依存するIUnknownの一部として、図２１の２１０６のVTableの最初の３つのエントリは、常に、IUnknownインターフェイスのメソッドを指し示す。ここで、IUnknownインターフェイスは、AddRefが次に続くQueryInterface、及びReleaseである。IUnknownから生成される次のインターフェイスのセットは、Invokeを含むIDispatchであり、Invokeは、コンポーネント内で、一組のメソッドにアクセスするための汎用メソッドである。

【０２２５】
複数の継承の場合のように、同じメソッドを指し示す１つ以上のVTableが存在し得る故に、VTableは、必ずしも１つとは限らない。さらに、同じ組のメソッドに接続される別の場所に位置する他のVTableが存在し得る。例えば、戻される必要のある２つのVTableが存在し得る（例えば、メソッドＸ）。メソッドＸは、特定のインターフェイスにより、VTable内のインデックスＮとして有効な場合も在り、又は、異なるVTableへのInvokeを通じての発送ID Qとして有効である場合も在る。

【０２２６】
VTableを使用して動作する時、多くの課題に直面する。関数へのポインタを有するVTableをインターセプトすることは、メソッド呼び出しをモニタリングする１つの方法である。しかしながら、VTableポインタのVTableインターセプトは、或る状況では不適切であることがわかっている。なぜならVTableをバイパスできる関数呼び出しの実行中に、内部呼び出しが起こり得るからである。他の課題とは、次のことである。すなわち、ホスト・アプリケーションが、関数を検索するためにVTableを使用することなく、直接関数を呼び出すことができ、又、同じ関数をすべて指し示す複数のVtableが存在し得ることである。これは、モニタリングシステムが、所定の対象関数に対し、すべて関連するVTableをインターセプトしなければならないことを、暗示している。ここで、所定の対象関数とは、禁止されていることが判明し、仮に取得できたとしても、すべてのVTableにアクセスをすることが困難なものである。ある既知のインターフェイス（iKnown）について、QueryInterfaceは、複数のVTableを検索するために用いられるが、かなりの処理のためのオーバーヘッドを生じることもある。しかしながら、QueryInterfaceは、デジタルシグネチャ関数のインターセプト技術が使用される代替方法である。

【０２２７】
IDispatchはアプリケーションのコンポーネントにインターフェイスを提供するものであり、マイクロソフト社から利用可能な製品である自動化の一部であり、このアプリケー

ションのコンポーネントは、遅延束縛を実行するために、用いられる（すなわち呼び出される関数は、コンパイル時ではなく、実行時間において決定される)。IDispatchは、IUnknowから発生する。VTableに含まれることで、IDisptachは、Invoke及び直接メソッド・インターフェイスの両方が利用可能な場合、２つのインターフェイスを有する。

【０２２８】

メソッドIDは、汎用Invokeメソッドを用いて、インデックスを付けられたメソッドを、呼び出す自動化（マイクロソフトのコンポーネント技術）において、特定化される。ここで、Invokeは、IDispatchの一部である。Invokeは、特定のメソッドを実行し、すべてはプロパティの検索、又はセッティングのためのメソッドに、マッピングされる。２つのインターフェイスが実行されない場合、Invokeは、すべてのパラメータを用意する必要のあるファクタのために、又は、呼び出しに先立ち、パラメータのリストのセットアップの必要があるために、より遅くなる。しかしながら、２つのインターフェイスが利用可能な場合、仮想関数が存在し、実際のコードに設定されるメソッドを指し示すVTable中に、設定される関数に対するポインタが存在する。

【０２２９】

第一の目的は、新しいCOMオブジェクト・インスタンスの生成をインターセプトすることであり、iKnownインターフェイスを設定すること、仮想関数が、IDispatchから導入されない場合でも、仮想関数が存在するかどうかを定めることである。クライアントは、エキスポートされ、クラス・ファクトリ・インターフェイスを要求する関数DLLGetClassObject()を呼び出す。一旦、この関数が、クラス構造インターフェイスを獲得すると、この関数はインスタント化されるオブジェクトに対し、インターフェイスのGUIDのIDを割り当てるパラメータを必要とするメソッドCreateInstance()を、呼び出す。CreatInstanceへの呼び出しもまた、gMgメソッドモニタリングの一部としてインターセプトされる。このメソッドは、パラメータとして、生成されるべきインスタンスが実行するインターフェイスのGUIDを必要とする。DLLGetClassObjectが処理を完了するとき、DLLGetClassObjectは、IKnown、インターフェイスのGUID、及び、VTableへのポインタが見出されるThisPointerも、戻す。

【０２３０】

インターフェイスが、IDispatchメソッドから導出されない場合、アプリケーションが、初期のバインディングを使用すること、すなわち、呼び出される関数が既知であり、コンパイル時に互いにリンクされることのすべてが、gMgシステムによってサポートされることを暗黙的に意味している。多くの場合、IDispatchに対するポインタは、オブジェクトのインスタンスと同義である。インターフェイスが、IDispatchから導かれる場合、それは、自身のVtableを生成するが、逆に、インターフェイスがIDispatchから導かれない場合は、完全に分離しているVTableが、生成される。

【０２３１】

新しいCOMの生成が見出された後、次の目的は、COMコンポーネント内に含まれる機能を実行する実際のメソッドコードを、検索し、このコードにアクセスすることである。すべてがIDによりswitch命令文を介してアクセス可能である場合、実際のCOMメソッドは、特定の動作を実行し、直接にアクセスされるか、Invokeを介してアクセスされるかの、何れかである。ある場合において、メソッドは、表に出されて、直接の呼び出しが可能である。タイププライブラリにおいて、オブジェクトが仮想テーブルで２つのインターフェイスとして宣言され、さらに、仮想テーブルへのポインタが存在する場合に、このテーブルは対象関数のアドレスを特定するために検査される。

【０２３２】

メソッドは、コンポーネント・カレンダー制御例におけるように、特定のインターフェイスのコンテキスト内でのみ、実在する。インターフェイスのGUIDが存在する場合、VTable及びそのインターフェイスが知られる。IKnownは、もし、それが利用可能である場合、メソッドのVTableを提供する。さらに、QueryInterfaceは、異なるVTableを戻すことが可能なIDispatchを得るために、実行されなければならない。

【０２３３】
図２１は、さらにCOMオブジェクトと、それらの内部自動化インターフェイス、及びVTableを示す。COMオブジェクト２１０９，２１１２，２１１５の多くのインスタンスが存在する一方、これらのインスタンスは、同じコード２１３５を共有し、同じVTableをもまた共有する。DLL２１１５内に、DLLのVTableを指し示すポインタ２１０６が存在する。VTable2136に位置する仮想関数は、メッソド（例えば、２１３９、２１４２、２１４５）を、間接的に見つける構造の一部であるポインタである。仮想であることにより、新しいクラスを形成するために、新しいクラスが、種々のクラスから生成される。例えば、新しいクラスは、IDispatchインターフェイス2124から生成される。コンパイルの間に、新しく生成されるInvokeクラスを指し示す新しいポインタが存在する場合を除いて、元のVTableがコピーされる。純粋な仮想メソッドと同様に宣言されるクラスから生成する固定化したクラスは、コンパイル時において、これらの実際に定義されるメソッドを実行しなければならず、それら各々のインターフェイス定義に優先することが可能である。メソッドが提供されない場合、コンパイル・エラーが生じる。図中の「VTableの詳細」と記載された部分は、VTable2136の詳細像２１１８である。COMのIunknown 1121、及び、QueryInterface2127を含むIDispatch2124、Addref2130、Release2133、そしてInvoke2134は、生成されているクラスに特定され、そして、コンポーネントのコンパイル時にリンクされて、各々が実行される。

【０２３４】
図２１は、仮想関数のモニタリングの一部として、Invokeメソッドに対する特定のインターセプト・ルーチンを示す。元の関数と全く同じパラメータがコピーされ、暗黙的なgMgパラメータと、モニタリングされる関数と、に対するgMgの呼び出しの間に、使用される。実装設計に応じて、コンポーネント内に含まれる、様々な生成されたInvokeが多数存在する。例えば、実行されるInvokeクラス・メソッドの何れかに対する多くの呼び出しと共に、IDispatchから生成される多数のInvoke (例えば、クラスＡ Invoke 2148、クラスＢ Invoke2154、クラスＣ Invoke2157等)が存在し得る。各Invoke(A、Ｂ、Ｃの内のいずれか)への種々の呼び出しは、gMgインターセプト・ルーチンに、異なる呼び出し２１７５に対する分離コンテキストのサンク２１６９、２１７２の各々を、割り当てさせる。

【０２３５】
Invoke Aクラスを一例として考慮すると、gMg呼び出しインターセプトは、プログラムフローを、２１６３における出力先変更から移動させて、特定の呼び出しインスタンスのデータ構造に対してコンテキストデータを含んでいる呼び出し用の適切なクラスＡサンク２１６９に到達させる。IDispatchから導出された異なるInvokeは、例えば、Invoke Ｂの情報２１７２を有する異なるサンクを備えることも可能である。しかしながら、gMgインターセプト・ハンドリング・ルーチン２１６６は、すべての呼び出しインスタンスに対し同一であり、そして、すべてのVTableメソッド2135に使用される。さらに、gMg処理は、適切な呼び出しコンテキスト情報及びコード２１７８の読み出しにより、インターセプトルーチン２１６６から、適切なメソッドへのあて先２１４８への経路を決める。これは、コンテキストのデータ構造の中に含まれる適切なあて先ポインタである。コンテキスト情報は、サンク、（gMgインターセプト処理により挿入される入力ジャンプ命令の後の）モニタ関数連続記憶位置２１６０、及び他の情報へのポインタを含むgMgオブジェクトの中に保持される。

【０２３６】
gMgインターセプトの手順は、以下のように行われる。インターセプトコードのセットアップは、一時的なgMg生成オブジェクトを使った技術を通じて行う。その後、呼び出し元２１０３が、対象関数を呼び出す場合に、インターセプトのセットアップ処理は、２１４８においてモニタリングされる関数２１６３のエントリポイントにおいて元のコードの一部をオーバーレイするが、２１７８に保存されるオーバーレイされたコードのコピーを保持する。実関数を実行するため、インターセプトの呼び出しは、２１７８からエントリコード部分を読み出し、連続記憶位置２１６０にジャンプする前に、まず、このエントリ

コード部分を実行する。gMgが、メソッド呼び出しからの、いくつかのプロパティを読み出すようなタスクを達成するために、自身の呼び出しを実行することで、既にインターセプトが行われたインターフェイスを介して、２１７８において、呼び出しが実行される。モニタリングされるメソッドが、ホストアプリケーションにより呼び出される場合に、gMgイベントが、生成される。しかしながら、呼び出しが、グローバルなgMgロック（図６を参照）をチェックすることでgMgインタープリタによって初期化される場合、gMgは、呼び出しのインターセプトを妨げる保護機構を有する。呼び出しを起こすホストアプリケーションだけが、gMgメソッドイベントとなる。

【０２３７】

モニタリングされる関数へのエントリで検出される元のパラメータは、gMgインターセプトルーチンにより捕捉され、関数の元のコードへ渡される。その結果は、gMgインターセプト・ルーチンが、スタックから実際のパラメータを読み出すことになる。メソッドからの戻りに際し、パラメータは、レジスタ、又はレジスタに含まれる間接ポインタから、最初に読み出される。

【０２３８】

メソッドモニタリング法のセクションにおける先の検討は、メソッド呼び出しを設定及びモニタリングするための、メモリ・オブジェクトの生成前、又は生成後に焦点を当ててきたが、gMgモニタリングシステムにおいては、メソッド戻りのモニタリングも、その重要性は等しい。呼び出しの捕捉に加えて、メソッド戻り値を捕捉する技術は、さらに処理するために、メソッドの結果を、捕捉して、gMgシステムに渡す場合に有用である。メソッドの戻りのモニタリングは、多くの環境における操作に関して、gMgスクリプトが利用可能となるメソッド呼び出しを行った結果を得られる。関数の戻りインターセプトの原理は、コンポーネント・パッケージにおける多くの異なる種類の対象を狙うことができ、ここで検討される関数の戻りを捕捉する2つの方法がある。まず第１は、スタック・コピー法であり、スタックのコピーを作成する。他方、第２は、スタック割り当て方法で、コンテキストの呼び出し及び戻りを保存するために、サンクの割り当て及び割り当て解除を、動的に行う。

【０２３９】

メソッドの戻りモニタリングのため、第一の「スタックコピー」技術は、（呼び出し元のスタックフレームを用いて）スタックの一部のコピーを行うことと、gMg呼び出し及び戻りのインターセプト・ルーチンから及びgMg呼び出し及び戻りのインターセプト・ルーチンへ対象関数のプログラムフローに影響を与えるために、コピーした物を用いることに基づいている。スタックのコピー法は、元のスタックを変更することなく、スタックをインターセプトする。スタックの一部のコピーを行うこと及び関数の戻りの実行ポイントをインターセプトすることにより、スタックの動きを検出し、スタックのgMgコード・バージョン内においてスタック操作を実行するgMgインターセプト・ルーチンへ、結果的に、プログラムの実行動作が移動する。gMgシステムが、透過的で、アプリケーションのソースコードに変更を必要とせず、そして、コンポーネントがプライベートなら、対象のモニタリングされる関数に利用可能なドキュメント化は、存在しないと考えられ、パラメータがいくつ関数に渡されるかは、分らない可能性がある。したがって、コピーが行われるとき、この（Ｎ個のパラメータを捕捉するために十分大きい）捕捉されたスタックの一部は、おそらくモニタリング処理とは無関係で、かつ必要な関数パラメータ及びいくつかの付加的なデータの両方を含んでいる。

【０２４０】

スタックコピー技術は、どれだけのスタック・ポインタが動いたかを認識するgMgインターセプトコードへの戻りを指し示すアドレスを有する。例えば、gMgコードが、（プリセットの設定として）20のワードを加える場合、gMgコードへの戻った後、スタック内の適切な場所にスタックポインタを位置させるため、スタック・ポインタに20を加える。スタック上のすべての関数パラメータのコピーが行われ、関数は戻りアドレスに、呼び出しを戻す。最後に、メソッドへの呼び出しが完了すると、スタック上のパラメータのコピー

は、元の呼び出し元に戻る前に、破棄される。この例の方法は、すべての戻りにおいて、２０ワードのパラメータを、常にコピーする。このパラメータが渡されている最中であり、デストラクタから呼び出される場所において、UninterceptcallEDX()は、関数をアンフックし、いかなる使用されたメモリも解放し、必要な場合、いかなる保存されたメモリも元のメソッド状態へ回復する。そしてgMgオブジェクトの参照カウントが０になる場合、オブジェクトを破棄して、インターセプトを解除する。インターセプト・ルーチンを確立する間、タイミングの問題を、考慮しなければならない。

【０２４１】

スタックコピー技術によるモニタリング戻りを示す図２２を参照すると、呼び出し元２２０５が、モニタリングされているメソッドを呼び出す。到着時に、メソッド呼び出しの完了後、プログラムフローが続くとすぐに、呼び出し元の戻りアドレス２２２０にと共に、スタック２２１０は、メソッドのパラメータ２２１５を有することができる。

【０２４２】

しかしながら、gMgモニタセットアップ段階から（図１８参照）、メソッドがモニタリングされる場合、モニタリングされる関数の呼び出しエントリアドレスは、既に設定され、エントリは、インターセプトの呼び出し段階の間に、ジャンプ命令２２３０で、変更される。元のコードをオーバーレイする前に、オーバーレイされた関数の命令コードは保持され、元の関数コード呼び出し２２４５に対する、インターセプトされた呼び出しに利用される。したがって、アプリケーションが、モニタリングされる関数２２２５を呼び出すとき、プログラムフローは、関数のコードアドレスに入る。変更されるエントリのため、プログラムフローをgMgインターセプト・セットアップ・ルーチン２２４０に切り替えるgMgジャンプ命令２２３０が、実行される。最初に、インターセプト・セットアップ・ルーチンは、スタック部分のコピー２２５０を作る。gMgシステムは、捕捉のためのスタックのバイト数を定めるモニタリングされている関数呼び出し中の、パラメータの最大数に関して推測を行う。この推測に基づき、gMgは、関数へのエントリに、スタックの一部分を捕捉するために、設定されたサイズ制限を使用する。このスタック部分は、関数パラメータ値２２５５、元の戻りアドレス２２６０、及び幾つかの付加データを有する。この戻りセットアップ段階の間に行われる別のステップは、スタックコピーにおける戻りアドレス２２６０を変更したものであり、そのスタックコピーは、戻り捕捉の実行２２８０へのポインタとなる。その戻り捕捉の実行は、呼び出し２２０５への戻るプログラムフローに先立って行われる。この時点で、戻り捕捉のセットアップが行われ、メソッドの戻り値の捕捉は、以下のように、実行され進行していく。

【０２４３】

モニタリングされる関数への呼び出しを実行するため、gMgインターセプト・ルーチンのセットアップの間、ジャンプ命令２２３０によりオーバーレイされた命令コードは、メソッドコード２２３５の継続ポイントにジャンプする前に読み出され、実行される。モニタリングされる関数の残りコードは、元の呼び出し元２２０５からパラメータを渡すために必要なスタックの実行と共に、（インターセプト・ルーチンのジャンプ命令によってオーバーレイされる）初期エントリ命令コードとを、取り込むことを行う。スタックの動作は、アプリケーションの元のスタック２２１５の代りに、スタック２２５５のインターセプト・ルーチンコピー上で、行われる。完了時点で、プログラムフローは、メソッドの最後に達し、gMgインターセプト・ルーチンのスタック・コピー２２６０内に設定される戻り命令を実行する。この命令は、実行フローを、インターセプト・ルーチンの戻り捕捉コード２２８０に移動する。この時点で、戻り値は、レジスタから、又は、スタックコピー２２５０から見出されるポインタにより捕捉され、呼び出し元がアクセス可能な記憶装置に保存され、その後、gMgイベントはインタープリタ２２７０に送られる。gMgスクリプト・レベルにおいて、スクリプトが、関数呼び出しインターセプトの現在のイベント・オブジェクトを参照する場合、そのスクリプトはスクリプト変数に割り当てられた後、その変数は参照される。割り当てられた変数が、（呼び出しまたは戻り検出のためのどちらかの）条件の一部である場合、メソッド呼び出し（又は戻り）イベントが受け取られるならば

、スクリプトのロジックは、適切な順序でスクリプトを実行する。最終ステップは、呼び出し元２２０５のアドレスを含む元のスタックへジャンプすることにより、元の呼び出し元２２７５に戻るための動作である。戻りコードを実行すると、通常の方法で、すなわち、インターセプト処理の開始においてコピーされた量により、スタック・ポインタ・レジスタを調整することで、スタックをポップし、その結果、プログラムフローがホストアプリケーション内で再開する。

【０２４４】

あるいは、スタックを割り当てる方法は、戻り値を捕捉するための別のメソッドを提供する。スタックコピー方法とは違い、スタック割り当てメッソドにおいては、呼び出し元のスタックのコピーは、行われない。その代わりに、呼び出しを行っているコンテキスト及びそれらの各々の戻りの追跡を続けるために、小さなサンクの動的な割り当てが存在する。関数の戻り値を追跡するため、サンクは、戻り値を調べる別のサンクを指し示すスタック上に変更された戻りアドレスを有している。呼び出しのモニタリングをサポートするサンクと異なり、戻りのモニタリングには、呼び出し及び戻り毎に、割り当て及び割り当て解除がなされる、動的に管理されるサンクが使用される。このスタック割り当て方法には、多くの段階がある。第１は、呼び出し元からモニタリングされる関数へのエントリで、次には、モニタリングされる関数に対する呼び出し間に生じる戻りインターセプト・ルーチンのセットアップであり、元のモニタリングされる関数への呼び出し、戻り、及び戻り値の捕捉であり、最後は、クリーンアップ及び呼び出し元への戻りである。

【０２４５】

図２３に示されるスタック割り当て戻りモニタリング方法は、サンクメモリを、動的に割り当て、そのメモリにおいて、サンクは、コードの出力先変更と、ホストアプリケーションの継続性を保持するための、呼び出し及び戻りのコンテキストの保存とのために使用される。関数がインターセプトされる時、オーバーレイされる元のコードは、サンクスペースに保存され、その後、モニタリングされる関数の実行を再開するジャンプ命令が実行される。幾つかのサンクがセットアップされ、１つは、インターセプトを行うため、もう一つは、インターセプトの初期部分へ入った後の再開実行のため、そしてさらに別のものは、戻りを処理するためである。

【０２４６】

任意の所定のメソッドを共有する形態のため、インターセプト・コンテキストは、異なる呼び出しインスタンスの詳細を表す個々のデータ（すなわち、呼び出しサンク２３３０、２３２７、２３５４、２３５７、２３４２、２３５１）を用いて、ホストアプリケーションの適切な経路指定を処理するために、必要である。例えば、メソッドＱを呼び出す２つのホストアプリケーション関数Ｘ及びＹが与えられる場合、gMgインターセプト・ルーチンＰは、２つの異なる戻りアドレスを、１つは関数Ｘに対し、１つは関数Ｙに対して、追跡を行う。プログラムフローの保持を確実にするために、様々なコンテキストの追跡の保持することと、異なるホストアプリケーションの関数に対する継続性とは、gMgサンクの鍵となる役割を担う。

【０２４７】

呼び出し元２３０３は、モニタリングされる関数２３１５を呼び出し、モニタリングされるメソッドへのエントリでオーバーレイされるジャンプ命令２３１８に遭遇する。このジャンプ命令は、インターセプトをうまく行うために必要な動的情報を取り扱う戻りインターセプト・ルーチンのセットアップ処理を始めるコードに達する。戻りアドレス２３１２は、サンクを指し示すパラメータ２３０９をも有するスタック２３０６中で、変更される。このサンクは、gMg共有サンク割り当て解除及び、戻り捕捉ルーチン２３６６の両方に、ジャンプ接続２３４５を提供する。戻り捕捉ルーチン２３６６は、異なる呼び出しインスタンスを追跡する、メソッド呼び出しコンテキストの戻り部分２３４８を接続する。この動的に割り当てられたサンクは、元の戻りアドレス２３４８を有する。呼び出し中の戻りインターセプト・ルーチンのセットアップのもう１つの動作は、進行中の呼び出しインスタンスに対するコンテキスト情報を提供し、また、戻り段階でのセットアップの部分

でもある（戻りは、インスタンスごとに、ディダイレクトされる）別のサンク２３３０の割り当てである。処理は、その後、進行中のインスタンスに対する呼出元の元の戻りアドレスを保存するサンク２３４２を割り当てる、共有割り当てルーチン２３３６に達し、そして戻り捕捉処理２３６０へのリンクをセットアップする。

【０２４８】

モニタリングされるメソッドに対する実行ルーチン２３３３呼び出しは、エントリポイントにおけるメソッド２３１８へのジャンプ命令によりオーバーレイされた、モニタリングされるメソッドの命令コードを保存したサンク２３５４を呼び出すことにより、２３３９で実行される。メソッドの実行は、継続ポイント２３２１において再開し、戻り命令２３２４の完了するまで続く。インターセプト戻りのセットアップ段階の間に、元の戻りアドレスが変更された２３１２ことを再び呼び出すことで、プログラムの実行は、呼び出し元２３０３への戻りを行うことの代りに、戻り捕捉コードへのリンクを有するgMgルーチン２３４５に達する。そこで、プログラムフローは、捕捉処理２３６６の最後まで達し、そこでは、サンクが割り当てられ２３６３、戻り値は、捕捉され２３６０、そしてgMgイベントは、gMgインタープリタに発行される。gMgスクリプト・レベルにおいて、スクリプトが、関数呼び出しインターセプトの進行中のイベント・オブジェクトを参照する場合、それは、スクリプト変数に割り当てられ、その変数は参照とされる。割り当て変数が、（呼び出し検出のためであるのか、又は、戻り検出のためであるのかの）条件の一部であるのである場合、メソッド呼び出し（又は戻り）イベントが受け取られるならば、スクリプトのロジックは、適切な順序でスクリプトを実行する。その出力が捕捉された状態で、プログラムフローは、呼び出し元２３４８の元の戻りアドレスに続いて進行し、そして、間接ジャンプにより、呼び出し元２３０３に戻る。

【０２４９】

スタック割り当て方法において、gMgサンクは、スタックの順序で、サンクを割り当て及び解放するスレッド毎のスタックのように振舞い、このサンクは、互いに補完し合う必要があるが、それは、情報が戻りアドレスと組み合わされる必要があるためである。効率的な動作のために、メモリブロックは、予め割り当てられて、増分メモリの割り当てに使用され、それらの特性は、書き換えと実行の双方が同時に行われる。

【０２５０】

要約すると、gMgメソッドモニタリング法は、動作システムからアプリケーションへのコンポーネントの幅広い範囲へのアクセスを提供し、（他の非gMgツールに対し）公開されドキュメント化される機能性、または、公開もドキュメント化もされない機能性のどちらかへのアクセスを実現する。gMgインタープリタと通信を行い、統合と、拡張と、ホストアプリケーションの動作を記述するために使用される固有のGUIコンテキストと付加的な情報源との相関関係とを、提供するgMgシステムに対する付加的なセンサとして、メソッドモニタリング法は、GUIセンサと共同して、実行される。COM及びエキスポートされる関数は、ここに説明されるが、gMgメソッドモニタリングシステムに適用される原理は、他の多くのコンポーネント技術に適応し、gMg抽象化モデルに、容易に適用可能である。

【０２５１】

この段階で、すべてのgMgモニタリングソースは、異なるデータ種類の複合体を生成できるgMgイベント形式、特にGUIコンテキスト内のすべての形式、で生成される。低レベルイベント、すなわち、gMgソースのポイントにおいて検出されるイベントは、最も適用可能であり、効率的な位置で、処理の異なる段階での状態機械により順々に処理される。原則として、何かが認識されるとすぐに、必要条件が与えられた状態の処理のために、直ちに考慮されるべきである。この目的のため、gMgシステムにおいて、状態機械の実行は、ローカル及びリモートの両方に対して有効である。

【０２５２】

図２４は、gMg状態機械の階層を示す。ローカル及びリモートの、状態機械機構の両方が与えられることにより、gMgシステムは、軽量で分散された状態機械の階層を、構築する能力を有し、ネットワーク全体に及んで、最大限の柔軟性を有し、リアルタイムでタイ

ムリーな応答の送出が行われる。モニタリングされる対象ホスト・アプリケーション２４１８、２４２１、２４２４は、イベントを検出し、収集するために、内蔵されるgMgセンサを有しているものとして、ここでは表される。一旦イベントが検出されると、それらは、gMgメッセージとして、それら各々のインタープリタ２４１２、２４１３、２４１４に送られ、そのインタープリタは、各々、スクリプトを実行中である（例えば、２４０９）。例えば、スクリプトは、（gMgセンサによって検知される）低レベル・トリガ・イベントを論理イベントに変換可能な、小型かつ高速のローカル状態機械２４０３を定義できる。一旦、処理されると、これらの論理イベントは、１つ以上の追跡メッセージ２４１５としてパッケージ化される。そのメッセージは、リモート（現時点で、処理の第2のレベル）状態機械により、通常、gMg収集サーバ２４０６上で、受け取られる。これらの論理イベントは、順に、他の状態機械のセットにより、さらに処理されて、検出応答２４２７として、渡される継続状態に基づいて、マルチレベル論理イベントを生成する。

【０２５３】

この処理は、階層２４３０内の状態機械２４３２により、さらに説明される。この（詳細な状態の）単一の状態機械２４４２は、一連のトリガ・イベントの入力２４４５、２４４８、２４５１、及び出力２４５４。この出力は、順に、同一のスクリプト内に、ローカル２４０９、あるいは、リモート２４０６に存在する他の状態機械に、供給される。状態機械階層ツリー２４３０は、gMgイベントに基づく柔軟な状態検出機構を形成するため、出力と入力とが互換性を有するようにした、ローカル及びリモートの状態機械の任意の組み合わせである状態機械の集まり（すなわち、２４３０、２４３３、２４３６、２４３９）を表す。分散型の状態機械をリンクする機能により、集中型サーバ上のイベントの迅速な検知及び経路指定が可能となる。なぜなら、処理状態の大部分が、ユーザ又はいくつかのローカル・ホスト・アプリケーションの動作の結果として、ローカルで直ちに検知可能だからである。

【０２５４】

必要に応じて、状態機械は、大規模のユーザベースにおけるサーバリソースの必要性を著しく減少するように、クライアント上で（つまりローカル）実行される。イベント間の状態を保存するコマンドは、ネットワークトラフィックを減少させるのみならず、リアルタイム警告のための最適化処理、及び他の時間依存の動作を促進する。これを実現するために、gMgスクリプト言語は、クライアントコンピュータ、または、ターミナルサービスのような仮想クライアント・スペース上の、状態を取り扱うシーケンスコマンドを、有する。gMgシーケンスコマンドは、シーケンスコマンド構文内に含まれる論理式の命令文に説明されるイベントとして、対象ホスト・アプリケーションの様々な状態を検出する。

【０２５５】

以下のコードの例は、単一の動作をもたらす状態のグループを追跡するシーケンスコマンド、すなわち、追跡コマンドの使用を説明する。

コード例

```
init
{
function filesavemenuitem=…;
function savedialogmydocuments=…;
function savedialogsavebutton=…;
function savedialogfilename=…;
var filename=nostring;
};
if (sequence (
 wlbutdown && filesavemenuitem,          // (1)
 wlbutdown && savedialogmydocuments:filename=_ // 命令文は、次の行に続く
   findsibling (savedialogfilename).title,   // (2)
 wlbutdown && savedialogsavebutton))    // (3)
```

track (saveend, topwindow, 'File Save To My Documents COMpleted', file=filename)
;

コード例において、「左のボタンを押し下げる」は、異なるオープン・ダイアログ・メニューアイティム上にて押される。状態コマンドは、一連の「ファイ・ルメニュー・オープン」アイテム（１）上の「左のボタンを押し下げる」順序を、続いて、ダイアログ（２）の「ファイル保存を開く」を、そして、最後に「実行されるコマンドを保存」を、Save Endイベント及びSavedMy DocumentsCOMlpletedイベントのトリガで探す。この例に提示されるif命令文は、一連のイベントとして検出されるＧＵＩオブジェクトを表す個々の条件を有する一連の条件を備える。インタープリタは、個々のイベントを受け取り、そして、イベント間の状態の追跡を続け、そこで、シーケンスは、そのコマンドにより定められるある順序で、状態を集約する。すべての条件が完了すると、条件は真となり、その後、シーケンス・コマンドの本体が実行され、そのコンテンツを実行する。通常、コンテンツは、シーケンス・コマンド又は他のアクションに適合する状態及び条件を反映する単一の追跡メッセージである。

【０２５６】

シーケンス・コマンドの他の特徴は、順序化されたシーケンス、または任意の順序シーケンスである、OR動作、または複合動作のように、より複雑な動作を行うことが可能なことであろう。All_Ofシーケンスに対し、イベントＡ，Ｂ，又はＣに対するシーケンス・グループの条件は、すべてのイベントＡ、Ｂ、及びＣが生じる場合に、真となる。Any_Ofシーケンスに対し、イベントＡ，Ｂ、又はＣのいずれかが、一旦生じる場合、グループ・シーケンス条件は、真となる。シーケンス・レベルにおいて、動作は、シーケンス・コマンドを、論理ブール演算と組み合わせることができる。OR、AND、NOT等は、どんな組み合わせでも使用される。シーケンス・コマンドは、また、ネストされる。すなわち、１つのシーケンス・コマンドは、以下のコードの一部として、別のシーケンス・コマンドを含んでも良い。

```
fragment:
If(Sequence(S1a,Sequence(S2a,S2b,S2c),S1b))
{
Action…
SendTrackMsg(Q)
}
```

この例では、第２のシーケンス・コマンドは、第１のシーケンス・コマンド条件の項目の中に含まれる。処理を行う場合、インタープリタは、第１のシーケンス１と遭遇し、その条件に対する追跡状態を設定する。その条件の１つは、別のシーケンス２のコマンドで、そのコマンドは、それ自身の分離した追跡状態を、順々に設定する。シーケンス２のコマンドは、それが、真になると場合の、その定義された論理シーケンスが完了するまで、イベントを待ち受ける。シーケンス１が、もう一度、評価される場合、その数式により自身のロジック及び処理の継続を完了する。一般に、動作全体は、コンピュータ環境が許容する容認可能なリソースレベル（メモリ、CPU、等）において、N回繰り返し動作する。

【０２５７】

さらに、Javaコード・ジェネレータなどのコード・ジェネレータは、gMgスクリプトとして、または、サーバの状態機械のコンポーネントを追跡しモニタリングするgMgとして、様々な動作環境におけるクライアント・コンポーネント上において、入力とするgMgスクリプトで、状態機械を生成し配置するために使用される。このローカルの状態機械及び柔軟な形態の総合的な利点は、サーバ・リソースの要件及びネットワーク・トラフィックの両方が飛躍的に軽減されることである。

【０２５８】

再び、配置されたコンポーネントの概略を示している図４を参照すると、gMgアプリケーションが、gMg収集サーバ及び追跡データベースにより収集され、受け取られる追跡メッセージまたはgMgイベントを、生成する状況が説明されている。この段階では、柔軟性

があり効率的処理が実行されるように設計されたリモート状態機械に基づいて、gMg解析処理が実行される。処理機構への入力の際、イベントは、データベースに保存され、その後、メモリ内の状態機械により直ちに処理されるか（リアルタイムで）、または、スケジュール（ほとんどリアルタイムで実行される、または、事後のバッチ処理を行う）状態機械のイベントに基づき、後ほど実行するためにバッチ処理が行われる。さらに、状態機械解析の中には、GUIオブジェクトセットの複数のインスタンスは、分離したgMgスクリプトから、又は単一のスクリプトからのGUIオブジェクトセットであるのか、についての検出を行う多くの機構が存在する。

【０２５９】

状態機械の初期構成は、鉄道ダイアグラムの論理的枠組みを使用するXMLファイルで定義される。鉄道ダイアグラムは、並行処理と、または、簡明な形式の条件とを、説明し、当技術分野では、周知とされている。XMLがファイルから最初にロードされて、初期状態機械セットが作成される場合に、初期のルーチンが呼び出される。解析ルーチンは、各要素、プロパティ、子要素等に対して、チェックを行う。XMLファイルは、状態機械のコンテンツ及び遷移を定義するブロックにより、サーバの状態機械のソリューションのプロパティを記述している。ブロックは、論理的関係のために、多数の要素の組み合わせをサポートする、XMLの構造であるコンテナの概念を、具現化する。コンテナは、ORのようなブール構造であり、ここで、ORは、遷移に対するコンテナである。コンテナは、その子に対し、固有のものである。

【０２６０】

ブロック内のプログラムの命令文は、イベント処理の動作、又は、link、block、goto、または遷移などの論理動作を表す。遷移は、ブール数式、またはタイムアウト終了で構成される条件により、保護される。

【０２６１】

一般的に、状態は、任意の所定の追跡メッセージまたはイベントが、所定の状態機械内で、現在処理されている位置を継続して追跡をすることにより、状態機械の中で定義される。状態が与えられると、次の遷移先は、追跡メッセージの記録領域内で定義される、その後のイベントのコンテンツによって決定される。受け取られたイベントは、その状態となる処理位置を有する。イベントが、遷移基準に適合する場合、状態位置に到達し、その状態位置は、新しい状態となる。したがって、状態は、遷移を実行した最終的な結果となる。進行中の状態は、先に遭遇した状態イベントに依存し、遷移基準が、任意の所定の進行中イベントと適合する場合、変更されるる。

【０２６２】

その状態機械に関するgMg構文中に、遷移基準は、link及びgoto命令文にリンクされる。gotoは、状態の変化を実行するが、linkは実行しない。リンクは、追加の枝に似ているが、gotoは状態を変える。linkは、システムへ入る入力イベントを評価して、ルートを決めるために並列に分割されたパス（又は並列なif命令文）のように動作する。linkは次の遷移に対して、可能な経路とトラバーサルとを作成するが、どんな状態変化も引き起こさないであろう。ラベルは、goto命令文（状態）からの行き先、又はlink命令文（非状態）からの行き先として用いることができる。

【０２６３】

もう１つのキーワード・コマンドは、ブロックの終端にあるリスナーに類似しているAnytime命令文であり、その条件が入力イベントと一致する毎に実行され、その現状態がブロック内に存在する。Anytime命令文は、すべての遷移に対してORと比較でき（例えば、終端にAnytimeを備えた所定のブロック）、状態がブロックのどこにでもあるとき、処理は、遷移に加え、実行されるすべてのAnytime命令文をチェックする。所定のブロック内の正確な状態に関係なく、何か実行される必要があれば、Anytimeは、どのような状況においても用いられる。Anytimeは、リスナーとして動作する。処理がブロックに達し、イベントが受け取られ、イベント内に注目がある場合には、状態は、条件及び基準に依存するAnytime命令文の実行の影響を必ずしも受けない。所定の状態において、もう１つのイ

ベントが受け取れられるとき、所定のブロック内のすべてのAnytimeとすべての親ブロックが、チェックされる。次に、イベントを適合させるすべてのAnytimesに対して呼び出しが行われ、次の可能性のあるすべての遷移が、チェックされる。一致する元の遷移が、実行される。それぞれの新規イベントに関して、状態遷移を行い、多くのAnytimeブロックを実行する可能性が存在する。Anytimeコマンドは、ブロックリスト内のすべての条件に対し一度評価され、その結果の動作は、すべての一致したものに関して一度実行する。

【０２６４】

例えば、数ページが、モニタリングされて、応答時間が測定される場合、Anytime命令文は、より少ないプログラム命令文で、測定のプログラミングを可能にする。対象のページは、ブロック命令文内に記載され、Anytime命令文が加えられる。Anytime文は、次に、その測定プロセスを、規定されたイベントを経由して、すべての対象ページに適用し、ユーザがその対象のページを行き来する順序に関係なく、ブロック命令文におけるリストに列挙される。シーケンスが、topic＝page1、topic＝page2、topic＝page3、などのページで規定され、そのページが遷移のシーケンスを表す場合、ユーザがたとえどのようなページにあっても、Anytime命令文は、それぞれのページのロード時間の測定を行うことになる。Anytimeを用いるアプリケーションのもう１つの例は、所定のプロセスを数えることである。Anytimeは、ブロック内のイベントごとに１回実行され、状態機械は、１度に１つずつイベントを評価する。Anytimeは、ブロックと、所定のブロック内にのみ関連する

多重スクリプトの存在の検出には、エージェントとスクリプトの識別子の組み合わせを必要とする。エージェント・セッションIDは、固有でなければならなく、その中で、エージェント・セッションIDもまた、スクリプト・セッションIDと一意的に識別されるスクリプト・セッションである。その識別子は、インスタンスの検出に導く。例えば、登録またはエラーを検出するために用いられる状態機械において、エージェント・セッションIDは、また識別可能にして一意的になれた状態機械を組織化するために用いられる。通常、グローバルな状態機械は、グローバルな可視性で作成されないが、必要とされる場合には、それは可能である。通常、全ユーザにわたる状態は存在しないが、代わりに、各ユーザ又はプロセスは、状態機械のインスタンスにより追跡される自身の状態を必要とする。

【０２６５】

明らかな識別子を有することのほかに、イベントの効果的処理を可能にするために、もう１つの重要な処理構造の機構は、gMgデータ・ソース・ジェネレータで送られた追跡メッセージによって生じるスコープの定義である。スコープは、GUIオブジェクト群のマルチインスタンスの検出、イベント、又はgMgデータ・ソース（クライアント）に発生する他のプロセスを、識別しさらにサポートする。

【０２６６】

種々のセッションからのイベントは、キューに集められる。フィールド値に基づいて、スコープが、イベントのオブジェクトを割り当てる。通常、スコープもまたセッションであり、代わりにそのスコープもまた、トピック・フィールドなどの、いくつかの他の追跡メッセージフィールドを用いることによって定義できる。それぞれのスコープ内では、状態機械の１つ以上のインスタンスがある。例えば、タスクがエラーを検出すると、そのセッションに対する１つの状態機械は、複数の状態機械を要求せずに、メッセージを決定するために、作成可能である。

【０２６７】

新規のイベントが初期遷移と一致する場合には、状態機械の急増（及びサーバ資源に対する高い需要）を避けるために、状態機械がすでにそのスコープ内に存在する場合、新規状態機械を作成する必要はない。新規イベントは、既存の状態機械への処理に対し渡されることができる。例えば、ページがアプリケーションに現れれば、ページ・イベントは、ページが作成され表示される初回のみに発生する、起動遷移として利用される。この時にのみ、新規状態機械が作成される。作成された状態機械は、固有であり、ここで固有とは、１つのスコープ内での１つのインスタンスを意味する。

【０２６８】

状態機械のインスタンスは、その作成を誘起したイベントのスコープに基づいて、スコープに割り当てられる。通常、そのスコープは、セッションIDによって、識別される。ここでスコープは、追跡メッセージレベルの定義L1からL5で定義される。セッション及びスプリクトレベルのスコープがある。スプリクトレベルのスコープが含まれるなら、スクリプトIDを検出するために状態機械が存在するであろう。そのスクリプトIDは、エージェント・セッションIDに関して固有であり、クライアントは、エージェントIDの作成について、サーバにより通知され、それは、すべてのクライアント機器に関して、繰り返される。

【０２６９】

次の２つの実施例は、アプリケーションのスコープ特性を説明する。第一の実施例では、アプリケーションへのログイン手順は、簡単な初期遷移のみを要求する。安全なログイン手順に関して、ユーザは、本人識別情報を入力し、次にその入力を送る。ログイン画面の最初の表示に際して（セッションのライフタイムにおいて、通常一回）、イベントが発生される。サーバは、イベントを受け取り、初期遷移における状態エンジンは、新規状態機械を作成する。もう１つの実施例では、ウインドウズのアプリケーションは、２つのスレッドを有し、１ウィンドウにつき１つのスレッドを備え、ここでは、gMgのモニタリング法は、２つのウィンドウのそれぞれの処理を追跡する。このシナリオで、ユーザは、セッション内又は状態機械内で、本人確認される。同じユーザに対する同じセッションには複数のスコープも存在する。サーバ解析は、各ウィンドウに対し１つのスコープを与えて、２つのスコープを利用する。しかし、サーバ解析は、依然として同じユーザと関連している。さらに、状態機械の収集機能内では、各ウィンドウに対して異なる状態機械が存在できる。

【０２７０】

図２５を参照して、遷移を含む状態機械の定義のセット２５０５内に、シリアルイベント２５５０を与え、イベント（例えば、２５１０から２５２０まで）を参照する状態機械の定義のセット２５６５内のそれぞれの静的条件は、EvalStatic２５５５で処理される。そして、状態機械のインスタンスとの関連において評価されなければならない状態機械のインスタンス変数２５２５～２５４５を参照する残りの条件のセット２５７０を、EvalStatic２５５５は、残しておく。そのインスタンス条件は、それぞれの状態機械のインスタンスに関する関数EvalInstance２５６０を用いて調べられ、最初に一致する遷移が、実行される。これは、既存の状態機械か、あるいは新たに作成された状態機械のどちらかに起こる。初期遷移は、新規インスタンスを作成しない限り、部分的に一致しないが、他のどの遷移とも類似したイベントを処理する。一致した遷移が、現在の状態からアクセス可能の場合、パスを解放するためのリンクがあり、遷移条件が真の場合、イベントに対するインスタンス・レベルにおける条件が存在することも可能である。

【０２７１】

例えば、追跡メッセージが到着し、その状態のスコープが５に等しい場合には、この値はインスタンスに保存されるが、インスタンスがまだ受け入れ可能でないため、この時点では確認することができない。処理がインスタンスへのアクセスに達するときだけ、値を評価して、そのステータスが受け取ったイベントに一致するかどうかを調べることができる。これは、静的条件とは対照的に、インスタンス条件の一例である。その条件が受け入れることが可能で、その遷移条件が真の場合、静的（false）評価は、インスタンス条件の評価となる。サーバの状態機械のコンパイラは、静的条件とインスタンス条件とを区別し、適切な処理ルーチンを呼び出す。これらの遷移イベント条件が真の場合、その遷移と関連したコードの動作が、選択されて実行される。次に、遷移処理は、遷移の後段階に移動するか、又はgotoがある場合には、gotoの行き先には、状態処理が存在する。

【０２７２】

図２６を参照すると、処理の大部分は、評価処理にあり、非常に高い割合で、何千もの取得イベント２６０３の経路指定又は発送作業である。状態のあいまいさを避けるために、この大きなセット内では、すべてのセッションのイベントは、２６０９でシリアライズされ、状態機械のセット２６２１に入力する。状態エンジン２６２７は、取得イベントを

規定された状態機械（例えば、２６２４）でチェックする。前のセクションで論じたように、状態機械の定義と状態機械のインスタンスの区別が、ここでは適用できる。状態機械は、状態検出の機構の定義であり、すべて定義された状態機械は、あらゆる入力イベントを処理する。（静的な）イベント条件を、２６１２で一致させる場合には、状態機械のエンジンは、そのスコープに対してイベント２６３３をチェックし、一致するスコープ２６３９内で、インスタンスでのイベントを処理するのみである。ある特定のインスタンスの状態の条件は、より高度な状態機械の定義レベルで、決定することができない。状態機械が取得イベントに向けられている場合、そのイベントは、状態機械のインスタンスによって捕捉されなければならず、それぞれのインスタンスの状態に依存する。すべての遷移のセットを与えられた場合、全体的な目的は、最初に静的条件２６１５をテストし、次にすべてのインスタンス条件２６１８をテストすることである。静的条件のチェックは、効率を良くするために最初に行われる、比較的高度なレベルの操作である。

【０２７３】

状態機械は、スコープによってグループ化され、所定のイベントに関し、状態機械のインスタンスを鍵とするイベントスコープを用いて、マップで検索される。静的構造のセット内に、入力イベントに一致する、遷移及びすべてのAnytimes２６１２が、読み出される。静的評価は、イベントに一致しない遷移を排除するために、クイックテストとして使用できる。各インスタンスに関して１度イベント処理を繰り返す代わりに、処理は、トップレベル（すなわち、状態機械の定義レベル２６１５）でなされることができ、その条件における現在の入力イベントで始動しない遷移は、実行遷移リストに現れないであろう。しかし、そのイベントが遷移、又はAnytime命令文に一致する場合には、それぞれの一致の型は、それぞれの一致する遷移又はAnytimeリスト２６１２に追加される。

【０２７４】

ある条件を満たせば、次のステップは、状態機械のインスタンスを作成することである。遷移リストが初期遷移を含み（すなわち、初期遷移が一致する）、状態機械が、作成されるために必要なスコープ内に、必要とされる多くのインスタンスとして、固有のものでなく、又はそのスコープで作成されたインスタンスが何もない場合、新規インスタンスは、２６３６で作成される必要がある。言い換えれば、たとえ遷移が一致したとしても、状態機械が固有のものでないか、あるいは、スコープ内にインスタンスが何もない場合、インスタンスはスコープ内に作成されなければならない。真であれば、新規の状態機械のインスタンスが作成される。

【０２７５】

有用な方法もまた、状態、タイムアウトなどによって保持される。遷移が一致する場合２６４２には、遷移がアクセス可能であることが確認される（すなわち、関数 IsAnytimeValidは、このAnytimeが現在の状態からアクセス可能であることを確認する）。関数（遷移）の実行は、遷移の内部で発生し、有効な遷移２６４８内に呼び出される。そのチェックによって、遷移が２６５１で現在実行されているということが決定される。前述したように、一般のgMgインスタンスは、トップレベルで効率よく確認できる（状態機械の定義）が、しかし、条件が既存の状態機械のインスタンスを参照すれば、各状態機械は、それらのインスタンスに対して異なった値を有することができる。したがって、各状態機械は、正確なインスタンスでチェックされなければならない。

【０２７６】

遷移処理が完了した後、別の遷移が存在する又は存在しないかもしれない。データ・ソースを駆動するプロセスが中断される場合、又は状態機械がその最後の遷移に達した場合には、このような条件が発生する。遷移のどれも現在の状態と一致していない場合、一定期間を経て、そのとき遷移は、gMgシステムが動かなくなるのを防ぐためにタイムアウトする。

【０２７７】

任意の遷移が実行される場合、タイムアウトは、チェックされる２６５４。タイムアウトは、後でトリガーされる可能性のあるタイムアウトのバッファに登録されなければなら

ない。すべての状態において、タイムアウトは、許容可能な最短タイムアウト位置を登録する。例えば、２つのタイムアウトがある場合（例えば、5分及び１０分）、より小さい時間が登録される。５分のタイムアウトは、最初に終了するので、１０分のタイムアウトは、捨てられ、その次に続く状態変化は、最初に実行される。どの時点でも、最短タイムアウトである状態機械には、可能なタイムアウトがただ１つある。したがって、遷移が実行される場合には、タイムアウトは、リフレッシュされなければならなく、新規状態は移動され、新規状態からタイムアウト可能なすべてが検索され、新規タイムアウト要求が登録される。イベントが到着する際、タイムアウトがチェックされる。登録されたタイムアウト要求を有するタイムアウト要求のバッファがある。イベントのタイムスタンプに基づいてイベントが到着する際、バッファは、どのタイムアウトに関しても調べられ、登録方法は呼び出され２６５４、状態機械についての情報は、更新される。例えば、そのシステムが所定の状態にあり、３０分のタイムアウトが存在し、３０分の終了前に異なった状態に移動する場合には、タイムアウトの別のグループにアクセスすることも可能である。しかし、タイムアウトが何もなければ、タイムアウトの登録は、削除される。そのシステムがイベント待ちで、イベントの到着がいつか確かでない場合、ある時間間隔（例えば、１分）の並列タイムアウトが作成できる。次にイベントが、１分以内に到着する場合、状態処理が開始する。しかし、処理が阻止されるのを防ぐためにイベントを到着させないようにする場合、そのシステムは１分のタイムアウトの登録を始めることができる。遷移実行後に、現タイムアウトの登録は、２６５４によって除去されるであろう。新規タイムアウトだけが、受け入れ可能である、すなわち、現在検査された位置から固有の遷移パスがある場合に登録される。他方では、イベントが現在位置で、１分後に取得されない場合、タイムアウトのバッファがチェックされ、次いでタイムアウトが使用され、遷移が実行されてから、次の状態に移動する。この次の状態から、新規タイムアウトの遷移の検索が実行され、新規タイムアウトの遷移が登録される。

【０２７８】

好ましくは、イベント処理後に、タイムアウトが対処される。遷移の実行はタイムアウトを排除してもよいので、正規処理後、タイムアウトが処理される。タイムアウトはバッファに記録され、遷移のように取り扱われる。ここでは、その条件は、タイムアウトがコードアクションを有することである。遷移が何もないある時間間隔後に、遷移がトリガーされる場合において、タイムアウトは、実際には、イベントの不足によりトリガーされるため、タイムアウトは、イベントなしの特別な遷移である。コードアクションは、通常、外部的イベントのためにパラメータを有するが、タイムアウトには何もない。状態エンジンレベルで用いられる１つのタイムアウトバッファ２６３０がある。バッファ入力は、数分に対応し、各入力に対して、その時間でタイムアウトする予定の遷移のリストがある。処置が完了した後に、現在のイベントの時間は、タイムアウトバッファで対応する時間を、検索するために用いられる。タイムアウトの発生が、状態機械を削除するであろう場合において、この時間を含み、この時間までにエントリで検索されたすべての遷移は、タイムアウトできる。保留中のタイムアウトすべてに対するループは存在しない。そして処理は、すべての状態機械に関して、遷移に対する呼び出しが非常に速い。遷移が実行された後、エンジンは、現在処理された状態関数に対してIfEndState（）を呼び出し、最終状態にある場合には、次にそのソリューションは終了される。

【０２７９】

最後に、公共と個人コンポーネント・メソッド・インターフェイスのための、センサ構造によって提供される、広範囲でモニタリング可能な対象を備えたgMgモニタリングシステムは、企業アプリケーションをサポートする基礎的システムのためのパフォーマンス・プロファイルを反映するために、豊富なGUIユーザコンテキストにおいて、解析可能なデータソースを作成する。GUIオブジェクトの汎用の検出、又は特定の検出機能を備えたマルチインスタンス・アプリケーションのGUIをサポートするgMgシステムの機能は、その動作環境内部の動的アプリケーション・プロセス内に、最大の柔軟性及び適応性を提供する。さらに、ローカル及びリモート分散型の状態機械は、遅い応答時間の検出、必要以上の

エラー状態、ビジネスプロセス解析などの、アプリケーションの範囲内におけるイベントの形式に対してタイムリーな解析を提供する。

【０２８０】

図２７を参照して、メソッドイベントを解析するためのプロセス２７００を説明するフローチャートを、以下のように示す。ステップ２７１０では、独立モニタリングプログラム４３６（エージェント４１４、エンジン４１６、ソリューション４２６を含む）は、（GUIオブジェクトを識別するための）ホストアプリケーションの階層的GUIオブジェクトツリー６６０、（コードモジュール及びメソッドモニタリング、COMオブジェクトを識別するための）図１７のコードモジュールツリー、及び（ホストアプリケーションの処理及びスレッドを識別するための）プロセス／スレッドのツリーを含む、種々のツリー構造形式において、モデルオブジェクトに対して動作可能である。ステップ２７１５では、動作環境スペクトラムの表示（すなわち、オブジェクト、GUIオブジェクト、メッセージ、メソッド呼び出し及びメソッド戻りで明らかにされる対象アプリケーションの条件）が、アプリケーション・ウィンドウに対し導出される。ステップ２７２０では、重要なイベントのタイムリーな配信に対して、２つ（またはそれ以上）のレベルで発生可能な状態機械のイベント（例えば、第一レベルのローカルユーザのデスクトップ２４０３、及び第二レベルのリモートサーバ２４０６）が処理される。ステップ２７２５では、gMgシステムのセンサ、又はホストアプリケーションの動的な動作環境スペクトラムの表示５１５（例えば、ユーザ、アプリケーション、OS及びメソッドイベント）に応じて処理する状態機械により生成される、１つ以上の状態機械が発生したイベントが、相互に関連づけられる。ステップ２７３０では、相互関連した状態機械の結果に基づいて、ユーザの経験の表示を推測する。ステップ２７３５では、動作環境スペクトラムの表示と共に動作する論理イベントデータを、状態機械イベントから、サーバ５５７に集める。ステップ２７４０では、バイト命令コードパターン１９２０は、識別され、コンポーネント関数のメソッド・エントリ・ポイントを検索し検出するために用いられる。ステップ２７４５では、バイト命令コードパターン間のインターフェイス機能は、識別され１９１０、ここではインターフェイス機能は、イベント及びプロパティにインターフェイスを提供する。ステップ２７５０では、インターフェイス機能のアドレスが、決定される。使用される技術は、インターセプトされた機能のタイプに依存する（例えば、エクスポートされた機能１６２０か、あるいはCOM タイププライブラリ１６２４のどちらか）また、モニタリングプロセスは、オブジェクト作成前に（１６２７）、それとも、その後に、開始されるのか（１６３０）により決定される。命令コードパターンは、メソッド呼び出し、メソッド戻り及びGUIオブジェクトを含むオブジェクトを識別するために、呼び出し１６４２と戻り１６４５の傍受の実行を可能にする。ステップ２７５５では、プロセス２７００は、機能呼び出しを追跡するためにインターセプトルーチン(図１８)１８４８を用いてインターフェイス機能に関連づける。

【０２８１】

図２８は、ホストアプリケーションの機能をモニタリングするプロセス２８００のフローチャートを、説明している。ステップ２８１０では、対象とされたメソッドの一連の捕捉されたバイトパターン、すなわち、メソッドシグネチャ１９３０から、ホストアプリケーション内にメソッド関数を表すバイト命令コードパターンを識別する。ステップ２８１５では、プロセス２８００は、ランタイム関数のバイト命令コードパターン１９１０を、メソッドの位置１９０５を決定するためにメソッドシグネチャリストと比較する。ステップ２８２０では、対象とされたメソッド関数と、呼び出しインターセプトルーチン２０１２と、ジャンプルーチン命令２０３９を用いる情報構造のインスタンス２０４５との間に、リンクが確立される。ステップ２８２５では、プロセス２８００は、選択的に２つのスタック操作技術、すなわち、戻り捕捉スタックのコピー１６４８及び戻りスタックの割り当て１６５１を用いて、呼び出しから、メソッド関数への戻りをモニタリングする。その結果は、アプリケーションの内部処理に関する追加情報を取り出すために利用可能な戻り値の捕捉である。

【０２８２】

図２９A、２９Bを参照して、コンピュータ・対象アプリケーションのプレゼンテーション層から導かれたイベントをモニタリングするプロセス２９００のフローチャートを、以下に説明する。ステップ２９０４では、「ソリューション」４２６の形式でのスクリプトは、対象アプリケーションに対してローカルなクライアントコンピュータ４３６に関して提供される。スクリプト４２６は、対象アプリケーション内で動作可能である。対象アプリケーション５４５のマルチインスタンス化からオブジェクト５０９のランタイムインスタンス化を、ステップ２９０６でスキャンする。そのオブジェクトをスキャンすると、１組のリアルタイムイベント５１５を誘起し、そのリアルタイムイベント５１５は、対象アプリケーションのメソッド呼び出し及びメソッド戻りから、ユーザ、アプリケーション、GUIオブジェクトのOSイベントと特定値、及びメソッドプロパティから導かれたイベントである。ステップ２９０８では、内部オブジェクトツリー８３２は、データ構造を組織化するために割り当てられる。そのデータ構造を、ステップ２９１０では、次に、実際のアプリケーションのGUIに、リアルタイムな動的変化８５４を反映するように適応させる。その適応からのイベントの「マルチタイプ化」ストリームは、実際のアプリケーション１１１５、１１０３の現状を反映する。このGUI構造は、インターリーブ及び参照するために、他のすべての型のデータに対してコンテキストを提供するイベントの一種を反映する。ステップ２９１２では、オブジェクトのインスタンス化を、関連オブジェクトのカテゴリーを検出できるスプリクト表現を用いることによって（コードサンプル参照）、所定のオブジェクト構造を適合させる対象アプリケーションのインスタンス化から検出する。図１５は、関連したGUIオブジェクトのカテゴリー全体を検出するために、汎用１５１０に（又は、単一のGUIオブジェクトを検出するために、特定１５０５に）することが可能なgMgスクリプト表現を示す。検出表現は、アプリケーション内に、ランタイム構造を見い出す多様なバリエーションを反映するために、この概念的スペクトルのどこでもGUIオブジェクトを検出できる。ステップ２９１４では、検出したオブジェクトの動作環境スペクトラムの少なくとも一部は、対象アプリケーションのインスタンス化から、検出したオブジェクトのコンテンツから捕捉される。

【０２８３】

ステップ２９１６では、イベントからのスクリプト論理シーケンス表現（ローカルの状態機械）、又は状態機械（リモートの状態機械）の処理を用いて、動作環境スペクトラム（すなわち、メソッド、OS、ユーザ、アプリケーションイベント）から受け取ったgMgセンサ４３８、４３９、４４０から論理イベント５２１を導出する。ステップ２９１８では、処理されたコンテンツは、gMgセンサによって検出されたマルチインスタンス化構造を反映するために再構築され（すなわち、データはサーバ４４６に集められる）、GUIオブジェクトツリーによって反映される。gMgシステムの階層的な状態機械処理２４３０、２４４２は、コンテキスト中でユーザが経験したように、アプリケーションの状態を再現するイベントのシーケンスを検出する。ステップ２９２０では、アプリケーションの元の状態を反映するために、アプリケーションのインスタンスとGUIオブジェクト群のインスタンスとをユーザのセッションに分離する固有の状態機械に対して規定されたスコープ２６３９を用いて検出されるデータ群へ、再構築された構造を相互に関連付ける。ステップ２９２２では、対象アプリケーション構造の反映（すなわち、多重スクリプトは、同じGUIオブジェクトツリー６６０を用いる）を、複数のスクリプト９４５、９４８、９５１間で共有する。ここでは、各複数のスクリプトは、対象アプリケーション９０１、９０２、９０３における、それぞれのインスタンス化と関連する。中間プロセスは、受け取られて、リアルタイム表示及びリアルタイム検出のために処理されるか、あるいはさらに解析するためにデータの記憶場所に収集されるかどちらかの、追跡メッセージ（４３６と４４６の間）の形式で捕捉イベントを用いる。ステップ２９２４では、ユーザ経験の条件に対する知見を再現して提供するために、ソース（例えば、クライアント）を送り先（例えば、サーバ）に同期させるような、一般に理解されるセマンティックな定義を用いる追跡データ群に関する履歴から分かる傾向の複雑な解析は、追加リソースにより可能となる。

【０２８４】

図２９Bに特に関連して、プロセス２９００は、ステップ２９３２に継続する。ステップ２９３２では、対象アプリケーションの一部を表し、コンテキスト情報（例えば、コンテキストレベル１２０５、１２１０、１２１５）を含むgMgコンテキストを用いた階層的に組織化する構成要素（図１２）を公開する。ステップ２９３４では、階層的に組織化する構成要素によってサポートされた捕捉された動作環境スペクトラム（例えば、オブジェクトツリー１０１５と、そのオブジェクト変数特性、例えば１０３５、又はオブジェクト変数のセット１０３９）を解析する。ステップ２９３６では、プロセス２９００は、図１４のフックを用いて、ユーザと対象アプリケーションとのインタラクション、又はプロセス１４０６又はスレッド１４０９、１４１２、１４１８、１４２１に現れるイベントを発生させる対象アプリケーションとのインタラクションを検出する。ステップ２９３８では、論理的スクリプト表現変数を用いて（「特定及び汎用の結合コード例」と題する前述のコード例と、オブジェクト変数１００５を参照）、検出ステップの結果に対応して、スクリプトの実行に基づいたイベントを発生させる（例えば、WhenObjects１１５１などの機構を有するgMgイベントの発生）。

【０２８５】

ステップ２９４０では、発生したイベントのうちの１つまたはいくつかを、インタープリタ７４５へ移す。ステップ２９４２では、その選択的に発生したイベントを解析する。選択は、スクリプトにおける論理表現などの、gMgシステムの様々な機構を用いて、GUIオブジェクトツリー構造内に、検出条件、変数、及びオブジェクト変数、WhenObjectを記載する。ステップ２９４６では、その解析を外部ソースに関してパラメータ、特に、サーバ又は動作環境のインフラからのデータと関連づける。そのスクリプトを、ステップ２９４８では、対象アプリケーションにおけるウィンドウの単一プロセス１３１５と関連づける。ステップ２９５０では、スレッドＩＤと関連する固有のメッセージを検出する。ステップ２９５２では、アクティブフックルーチン１４３３～１４４２により、そのメッセージをインターセプトする。ステップ２９５４では、スクリプト表現及び動作を用いて（コード例「ManyInstances」参照）、対象アプリケーションに関する情報を得るために、そのメッセージのコンテンツを評価し、イベント及びプロパティを導出する。ステップ２９５６では、プロセス２９００は、解析と相関ステップの結果とを含むレポート４５４、警告４５０、相関関係４５２、動作／応答４５６を発生させる。ステップ２９５８では、アプリケーションの繰り返されたスキャンは、検出されたイベントによって呼び出された所定間隔６１５（例えば、OSの時計による駆動、もしくは特定条件による駆動）で、アプリケーションGUIを反映する構造を有するgMg GUIオブジェクトツリー６６０を継続的に投入し、動的に割り当て、適応し、検出し（例えば、図１１のWhenObject機構と図１４のフックによる断続的なポール操作）、捕捉するステップを繰り返す。

【０２８６】

本発明が詳細に説明されてきたように、さまざまな改善点、代替点、等価点は、当業者に明らかになるであろう。そのため、詳細な説明が限定としてではなく一例として提供されたことは理解すべきである。詳細な構造の多数の変化と、コンポーネントの組み合わせと配設は、以下に請求項を記載するように、本発明の思想と範囲から逸脱することがない限り用いてもよい。１つの実施形態から別の実施形態への要素の置き換えもまた、十分に意図され熟考される。本発明は、本明細書に添付されたクレームと、詳述の等価点に関してだけに規定される。

【図面の簡単な説明】

【０２８７】

コンピューティングシステム及びGUI

【図１】本発明の実施形態に係るコンピュータと、クライアント及びサーバのハードウェアを示した図である。

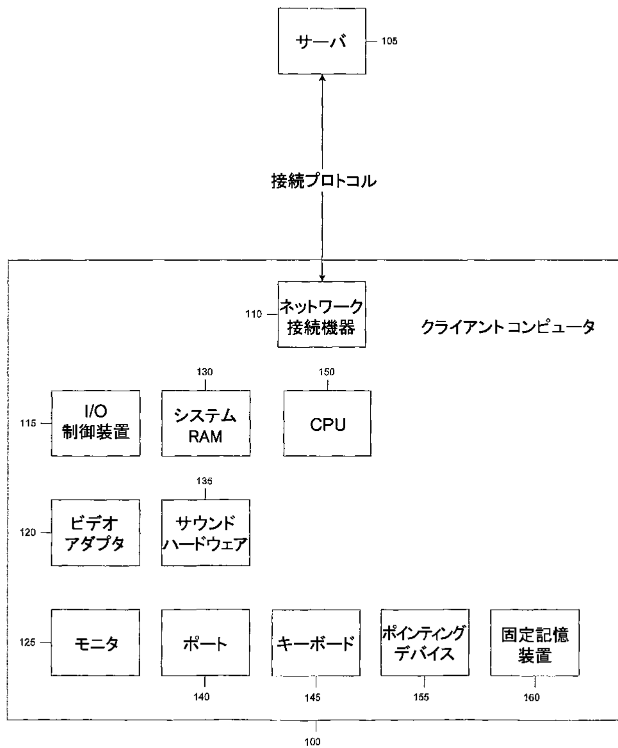
【図２】本発明の実施形態に係るGUIウェブアプリケーションの例におけるGUIプレゼンテーション層を示した図である。モニタリング対象

【図３】本発明の実施形態に係るクライアントまたはサーバ（非GUI）におけるコンピュ

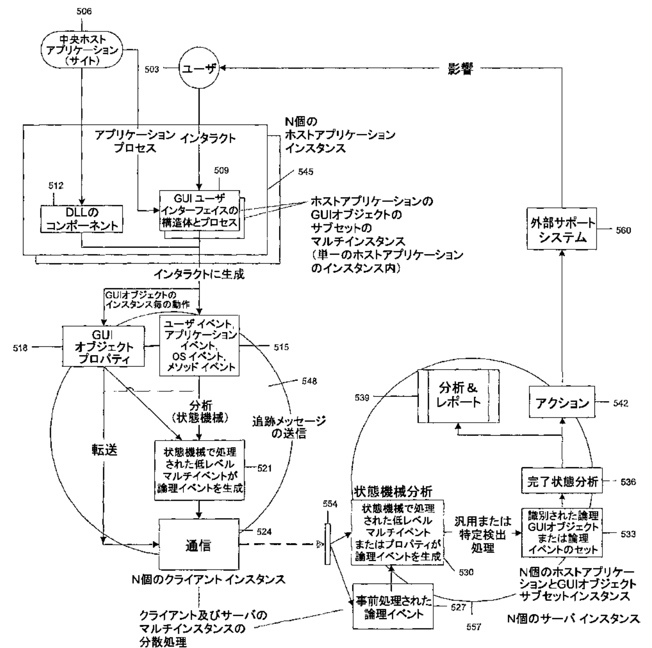
ータの動作環境内のモニタ対象gMgシステムコンポーネントの概略
【図４】本発明の実施形態に係るgMgシステムコンポーネントの概略を示す図である。gMgシステムイベントフロー
【図５】本発明の実施形態に係るgMgシステムにおけるイベント及びプロパティのフローの概略図である。
【図６】本発明の実施形態に係る主要なイベントソースを示す図である。GUIオブジェクトツリー
【図７】本発明の実施形態に係るGUIオブジェクトツリーの更新を示す図である。
【図８】本発明の実施形態に係るオブジェクトツリーのgMgイベントである、gMgオブジェクトツリーの更新、及びgMgがイベントを生成及び破棄を示す図である。マルチインスタンスをサポートする機能
【図９】本発明の実施形態に係るプログラムのロードに関する機能を示した図である。
【図１０】本発明の実施形態に係るオブジェクト変数を示す図である。
【図１１】本発明の実施形態に係るWhenObject構造を示す図である。
【図１２】本発明の実施形態に係るプロセス及び異なるウィンドウ要素を有するプロセスウィンドウコンテキストを示す図である。
【図１３】本発明の実施形態に係るコンテキスト及び複数のポップアップウィンドウを示す図である。
【図１４】本発明の実施形態に係るフック処理を示す図である。GUIオブジェクトの汎用検出
【図１５】本発明の実施形態に係る汎用スペクトラムに対する特定を示す図である。メソッド検出技術
【図１６】本発明の実施形態に係るメソッドモニタリングの概略を示す図である。
【図１７】本発明の実施形態に係るメソッドモニタリングに使用されるモジュールツリーを示す図である。
【図１８】本発明の実施形態に係るメソッドモニタの生成を示す図である。
【図１９】本発明の実施形態に係るメソッドシグネチャ及びメソッドコードオーバーレイを示す図である。
【図２０】本発明の実施形態に係るメソッドモニタリングに関するサンク割り当てを示す図である。
【図２１】本発明の実施形態に係る仮想関数メソッドのモニタリングを示す図である。
【図２２】本発明の実施形態に係るメソッド戻りースタックコピーのモニタリングを示す図である。
【図２３】本発明の実施形態に係るメソッド戻りースタック割り当てのモニタリングを示す図である。分析
【図２４】本発明の実施形態に係るローカル及びリモート状態機械処理を有する状態機械階層を示す図である。
【図２５】本発明の実施形態に係る状態機械において、遷移に関する静的及びインスタンス評価を示す図である。
【図２６】本発明の実施形態に係る状態機械のイベント処理を示す図である。
【図２７】本発明の実施形態に係る処理を示すフローの概略図である。
【図２８】本発明の別の実施形態に係る処理を示すフローの概略図である。
【図２９Ａ】本発明のまた別の実施形態に係る処理を示すフロー1の概略図である。
【図２９Ｂ】本発明のまた別の実施形態に係る処理を示すフロー2の概略図である。
【符号の説明】
【０２８８】
100　クライアント（ローカル）
105,446　サーバ（リモート）
340,342,344,346　gMgメソッドモニタリング（監視プログラム）
350　gMgイベント（イベント）

352,745,812,1153,1660,2412　gMg（スクリプト）インタープリタ（インタープリタ）
404,406,408,901,902,903,1103,1603　ホストアプリケーション（対象アプリケーション）
450　警告
452　相関関係
454　レポート
456　是正措置（応答）
506,909,912,915,1118,1215,1315,1406　アプリケーションプロセス（プロセス）
518　GUIオブジェクト
518　プロパティ
515,2550,2603,2606,2609　イベント（イベント）
527　事前処理された論理イベント（論理イベント）
625,1154,1130　gMgスクリプト（スクリプト）
775,939,1040,2409　スクリプト
808　インターフェイス
814　スレッドid
2030,2033,2215,22552309　パラメータ
2210,2306　呼び出し元スタック（スタック）
2250,2306　スタックコピー（スタック）
2415　追跡メッセージ（メッセージ）
2430,2432,2433,2436,2439　状態機械

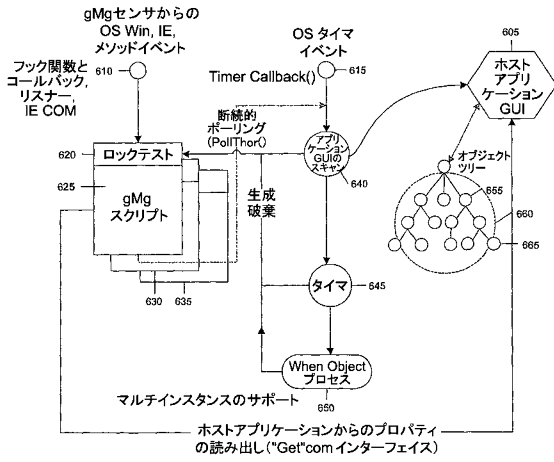
【図１】



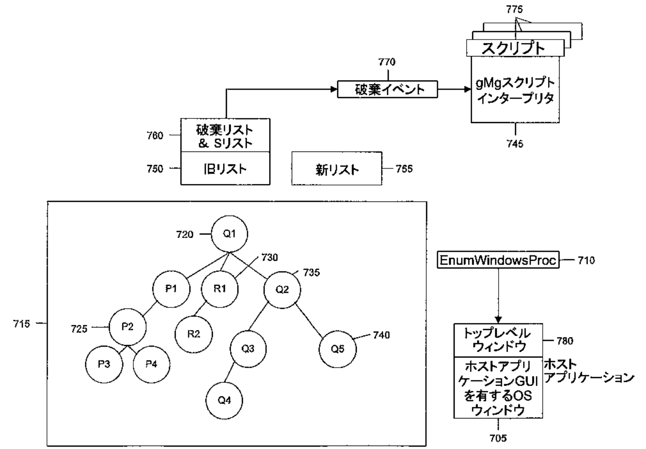
【図５】



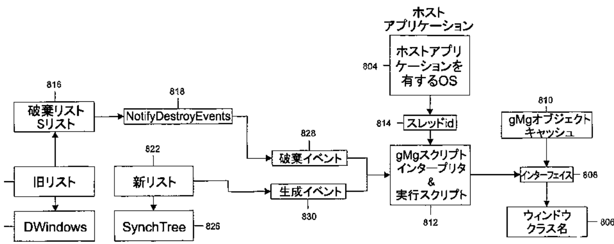
【図６】

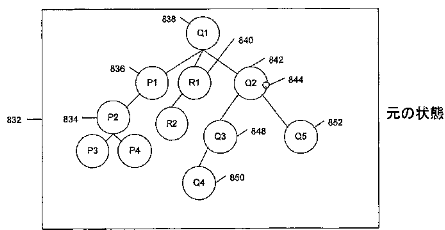


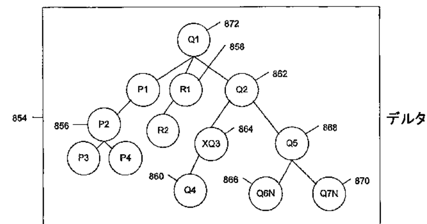
【図７】



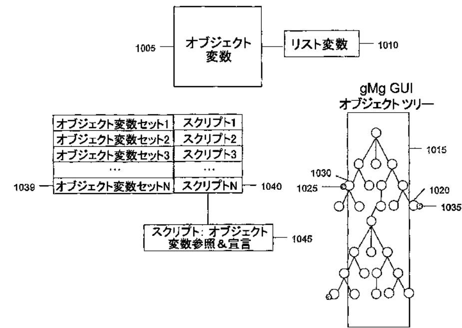
【図８】



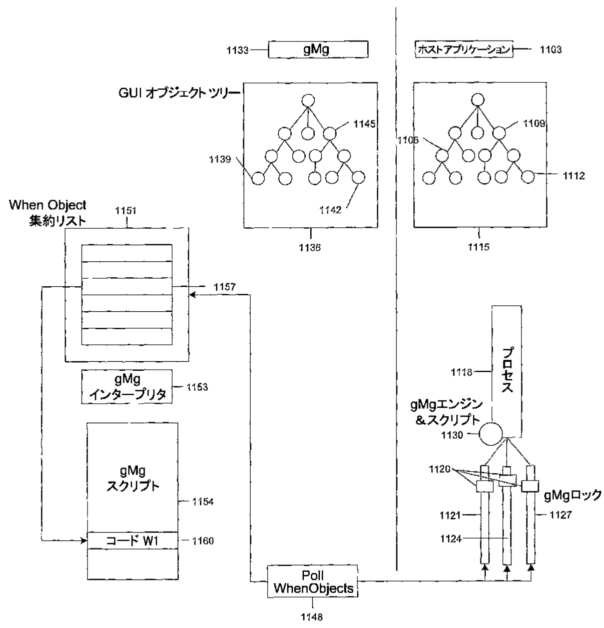




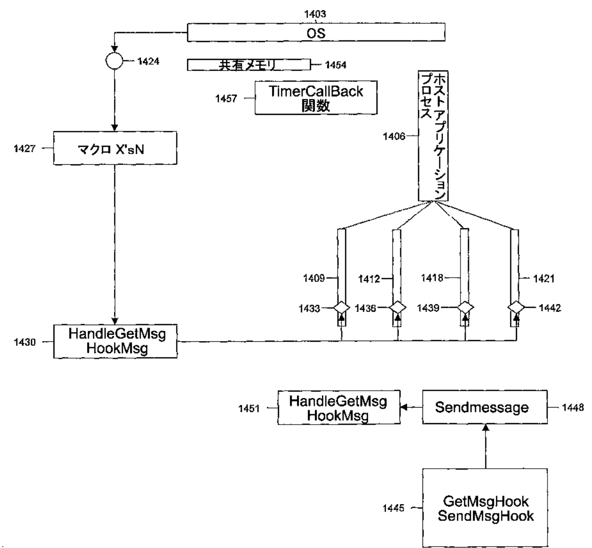
【図１０】



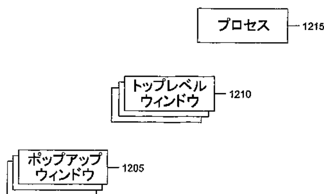
【図１１】



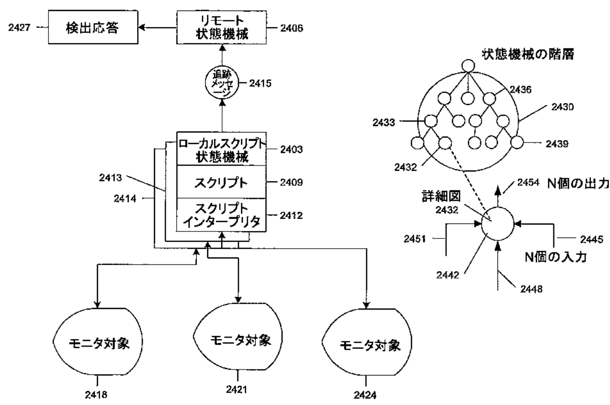
【図１４】



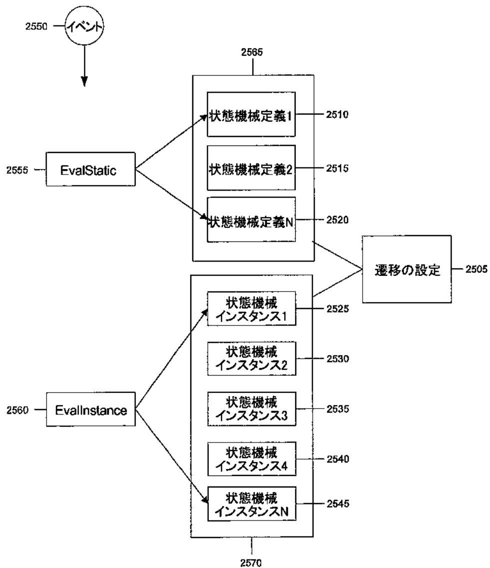
【図１２】



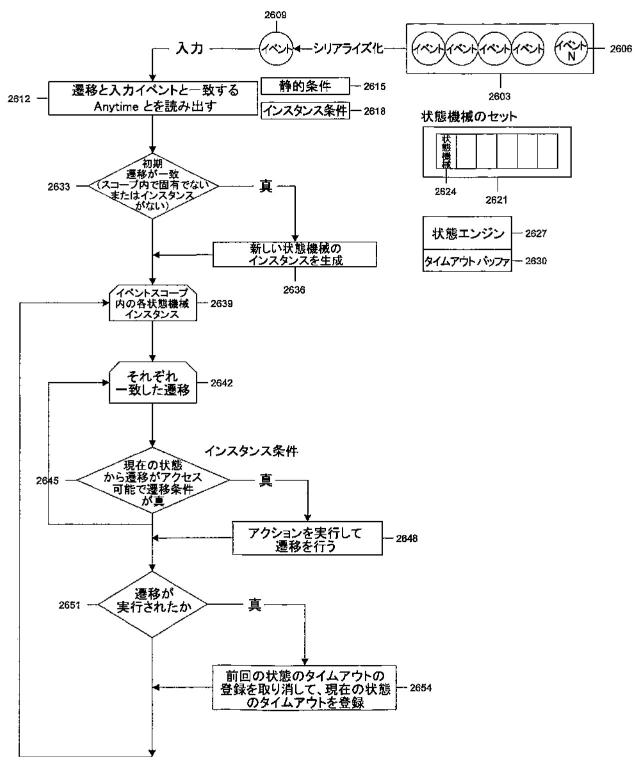
【図２４】



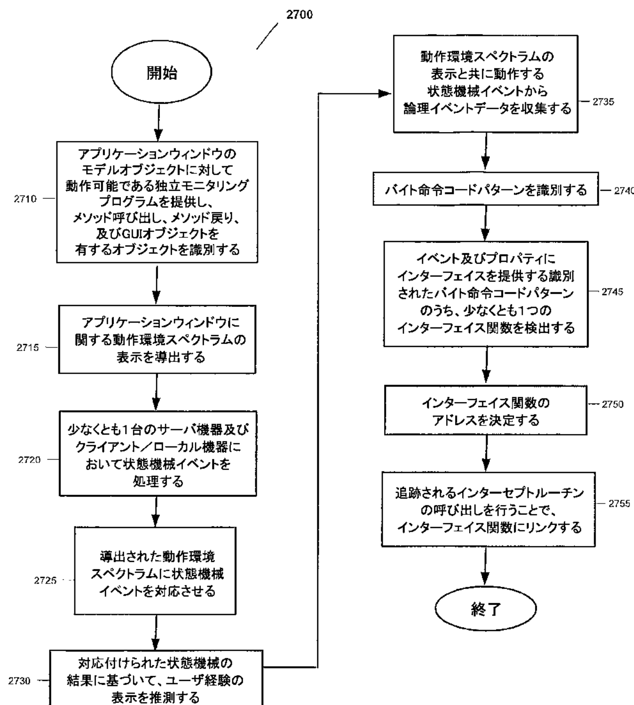
【図２５】



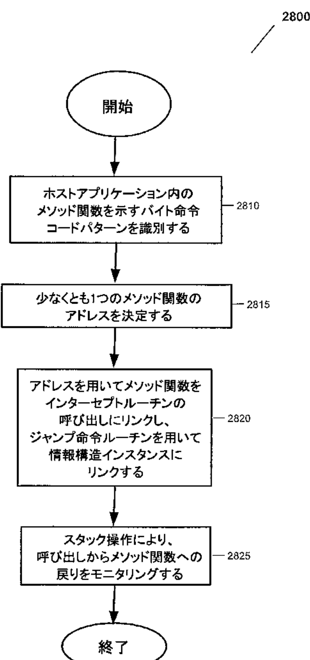
【図２６】



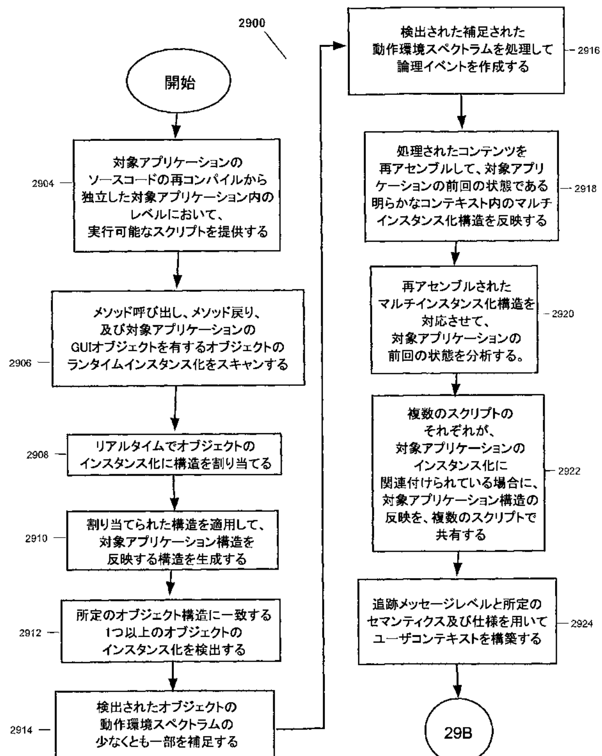
【図２７】



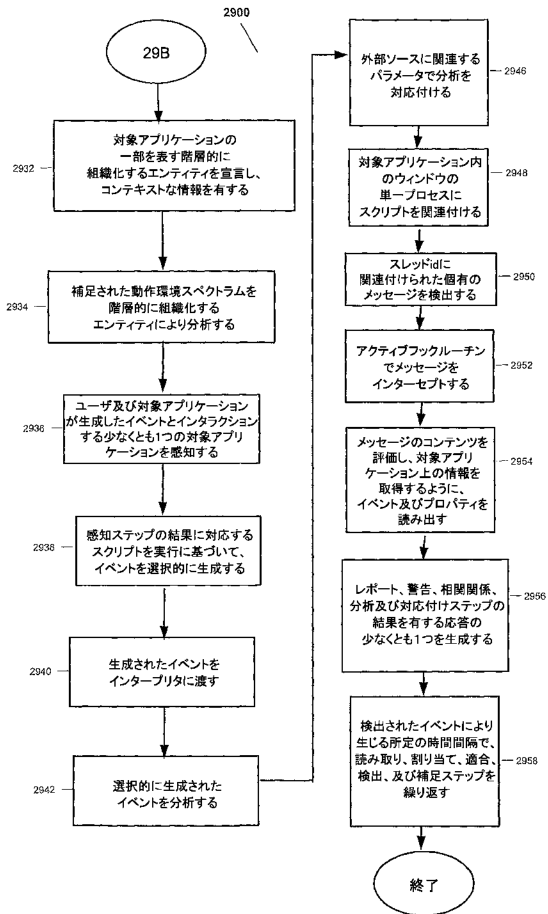
【図２８】



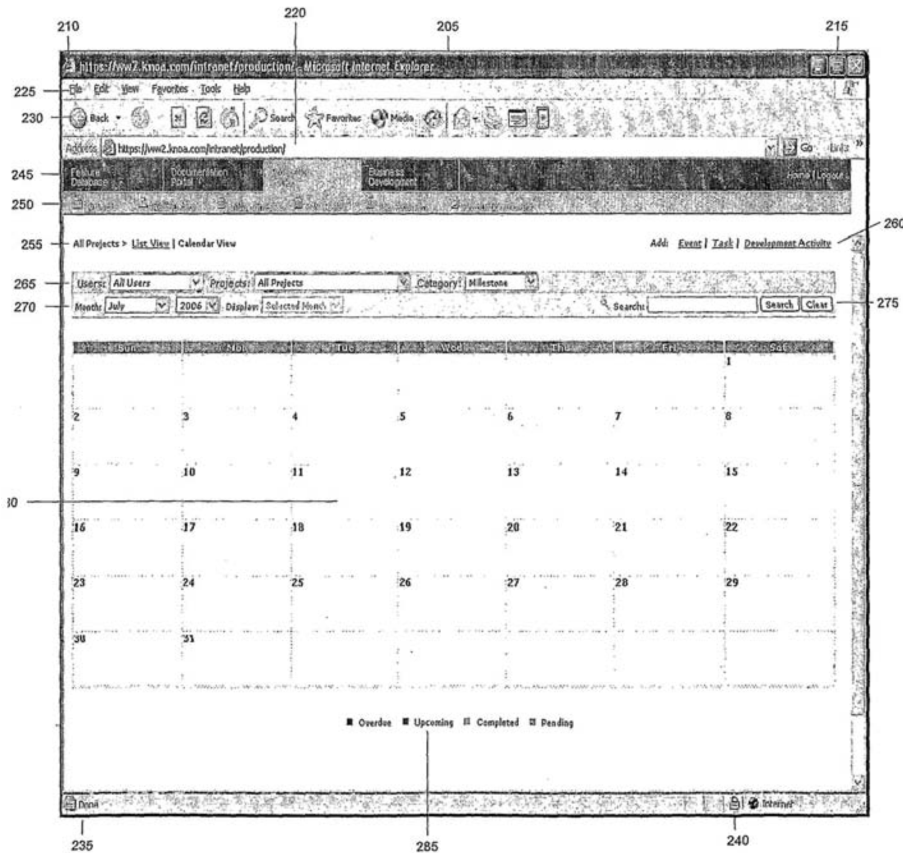
【図２９Ａ】



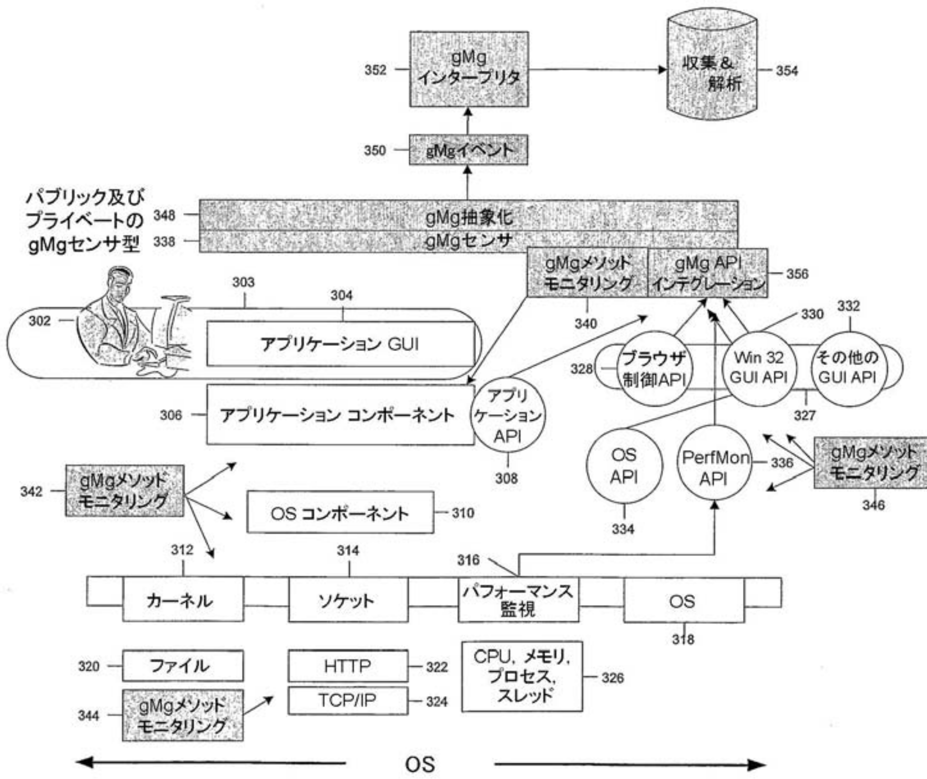
【図２９Ｂ】



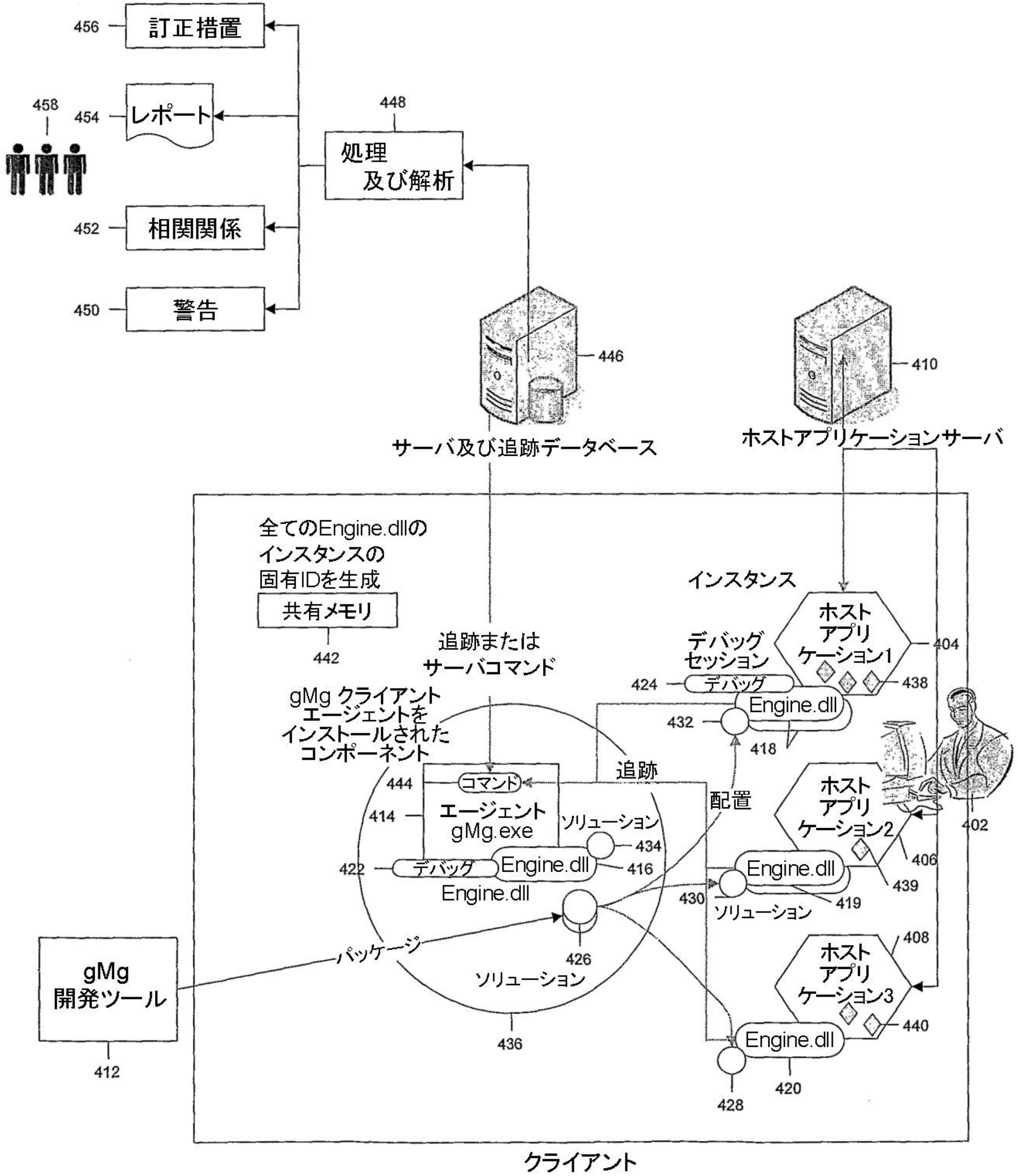
【図２】



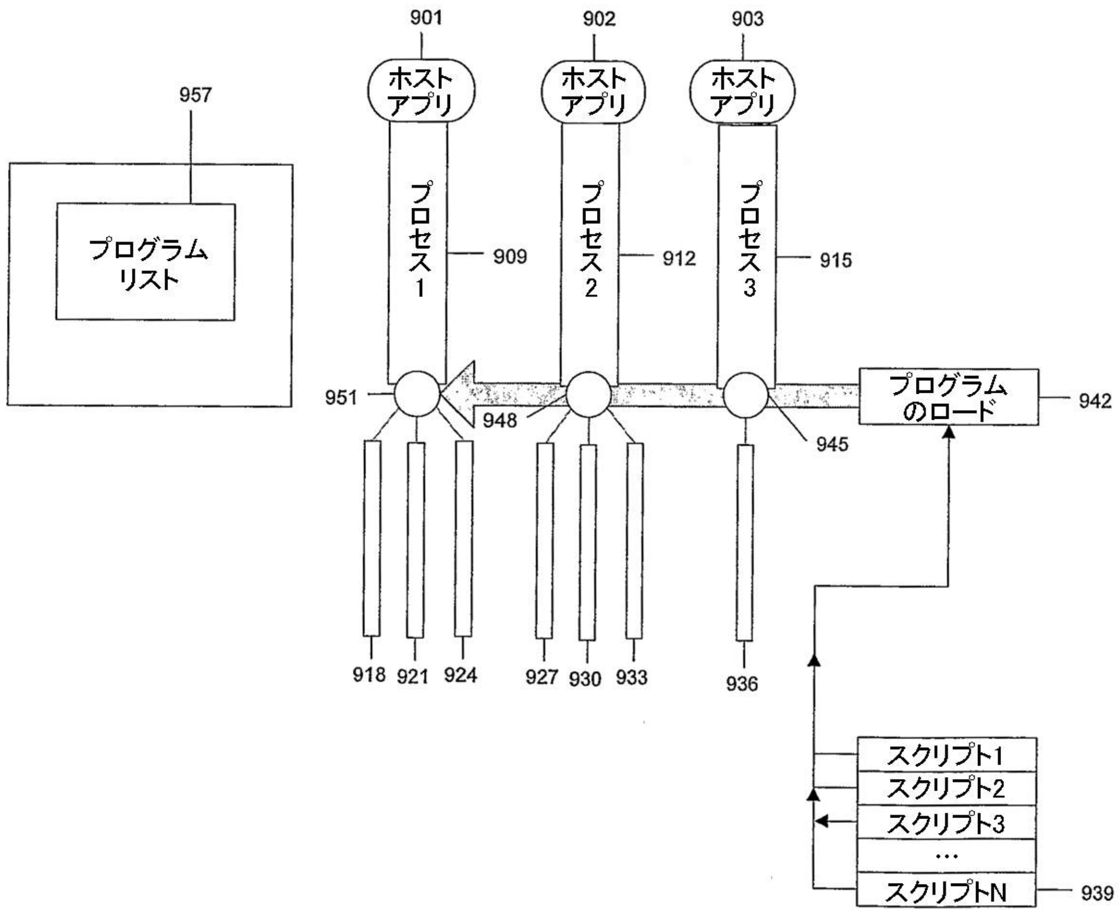
【図３】



【図４】



【図９】



【図１３】

ホスト アプリケーション

1305

ホストアプリケーションプロセス

1315

1320

1325

1330

1335

ポップアップA

ポップアップB

ポップアップC

ポップアップD

1340

1345

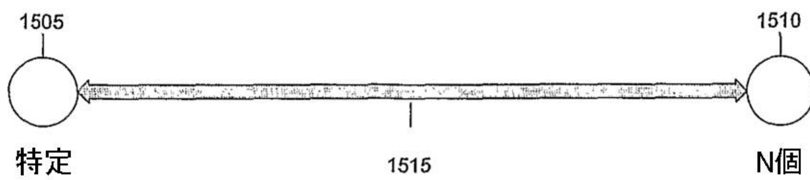
1350

1355

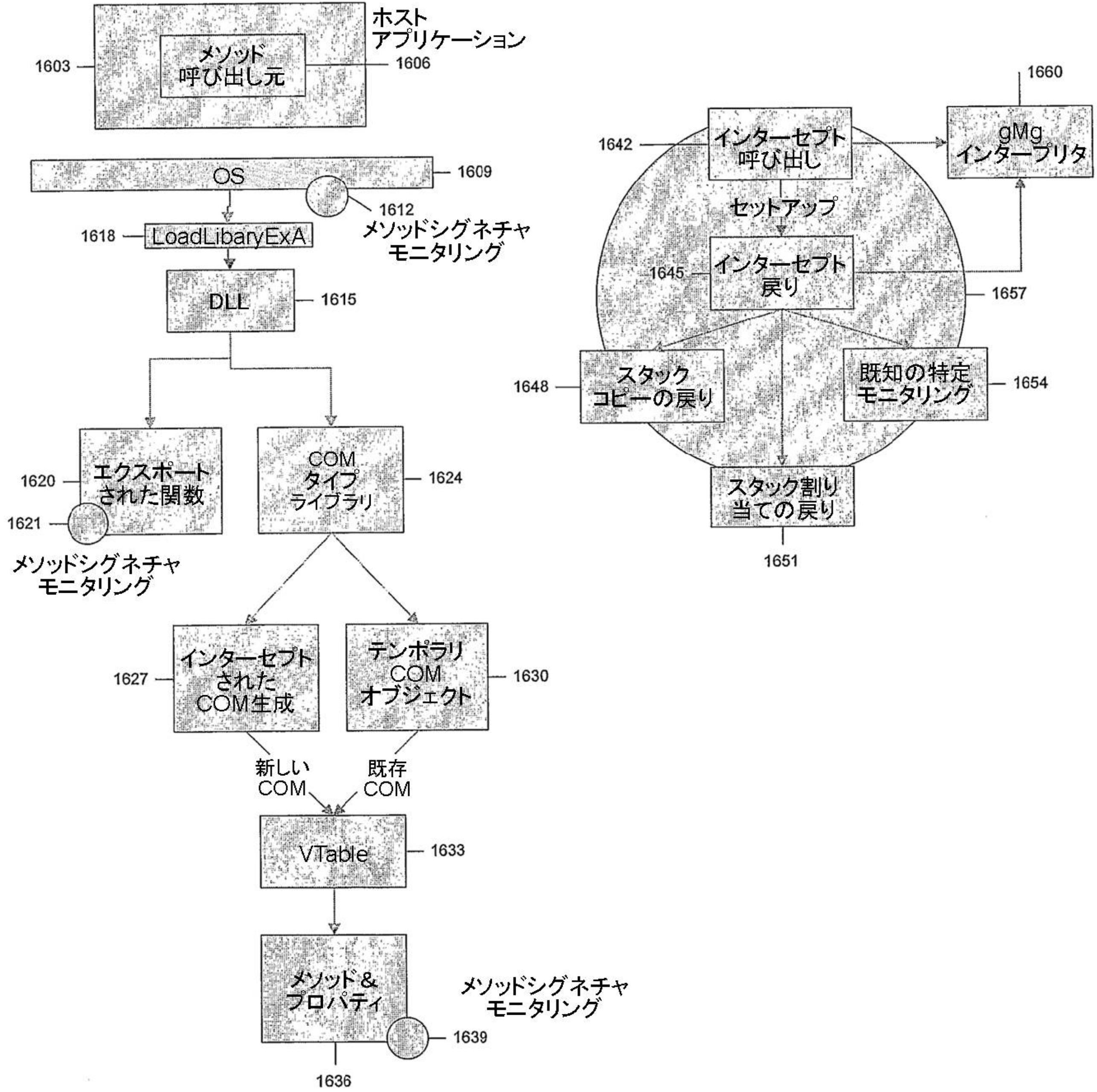
Tab 1 Tab 2 Tab 3 Tab 4

1360

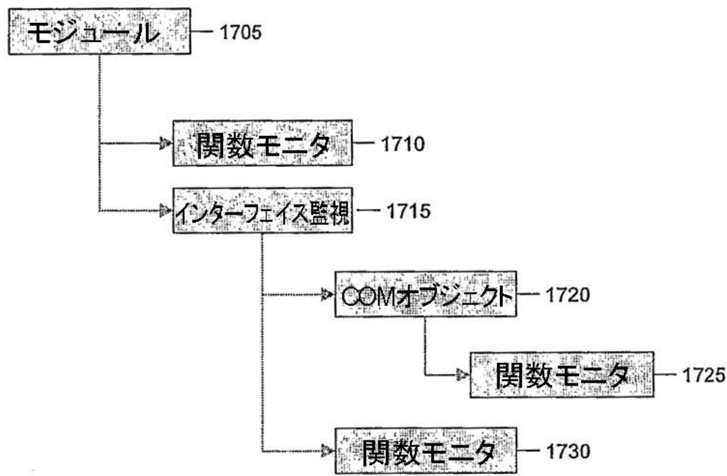
【図１５】



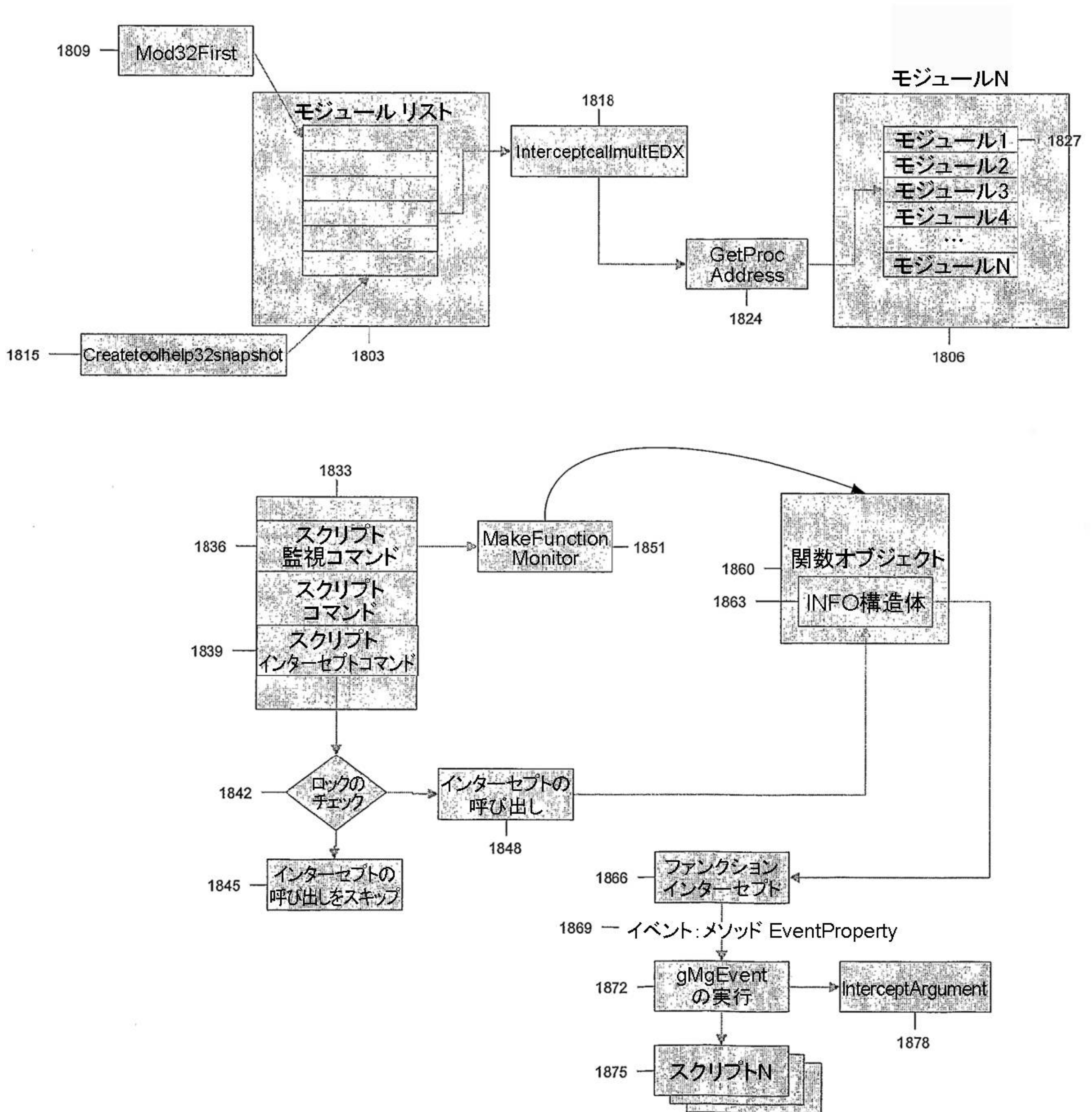
【図１６】



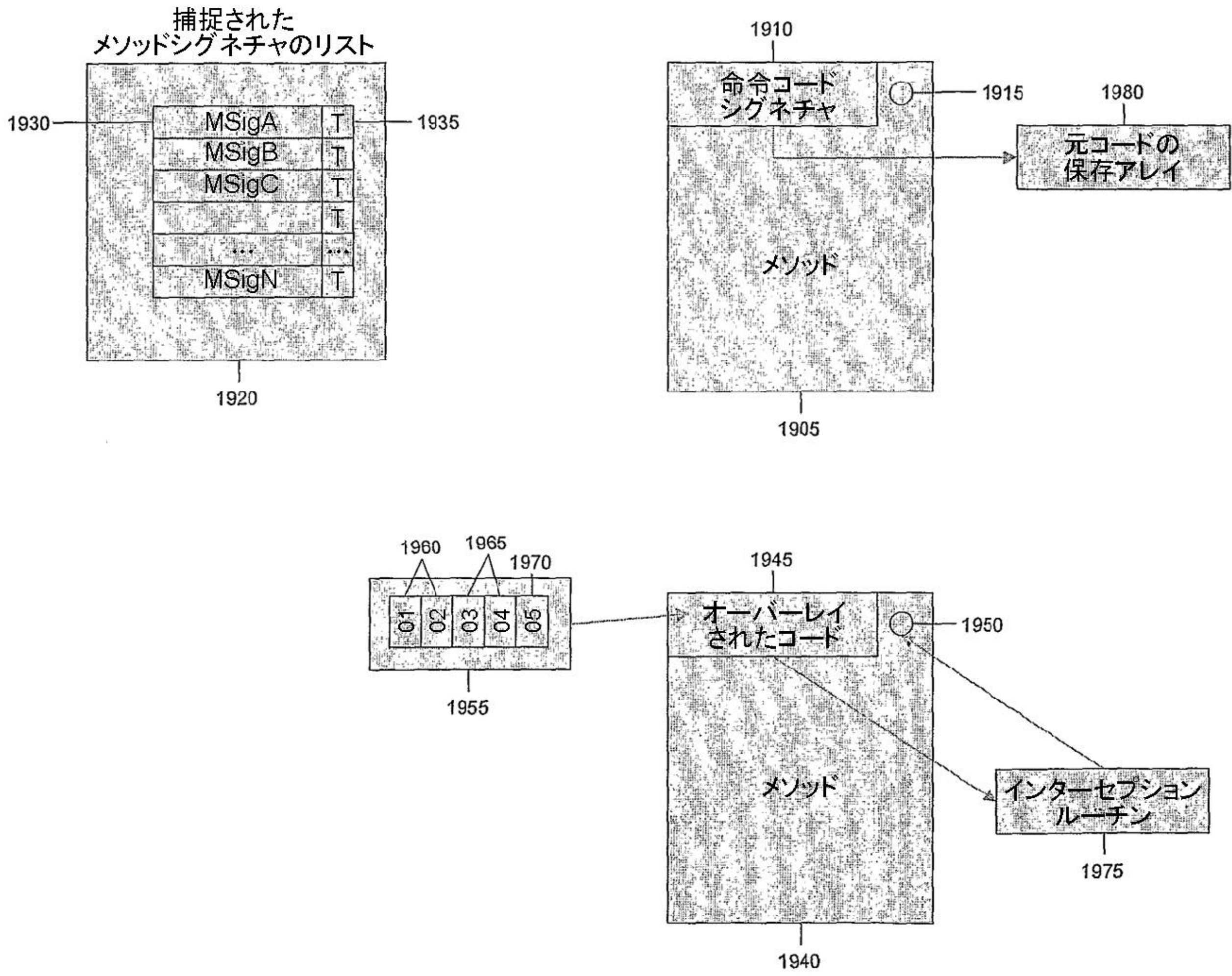
【図１７】



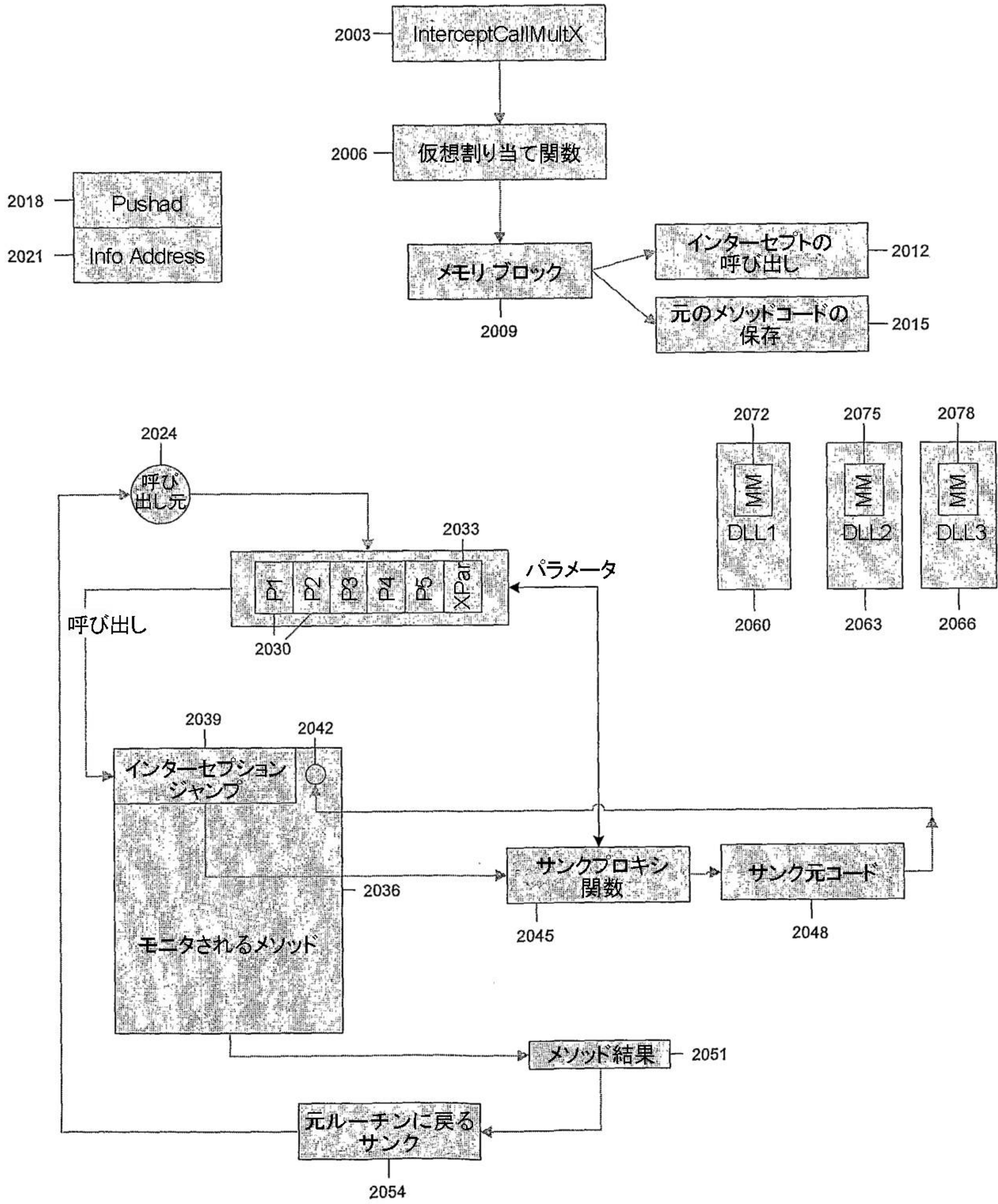
【図１８】



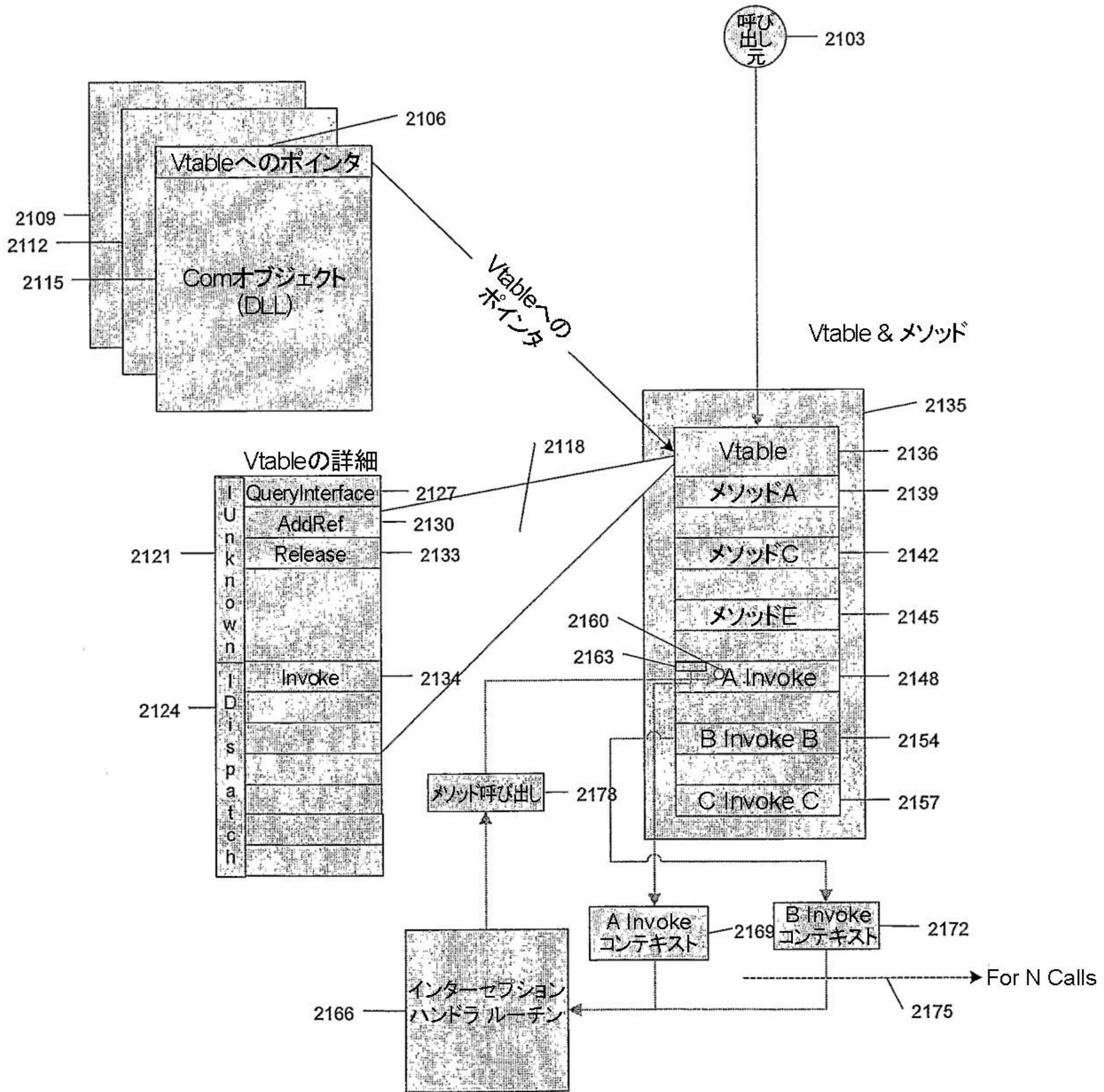
【図１９】



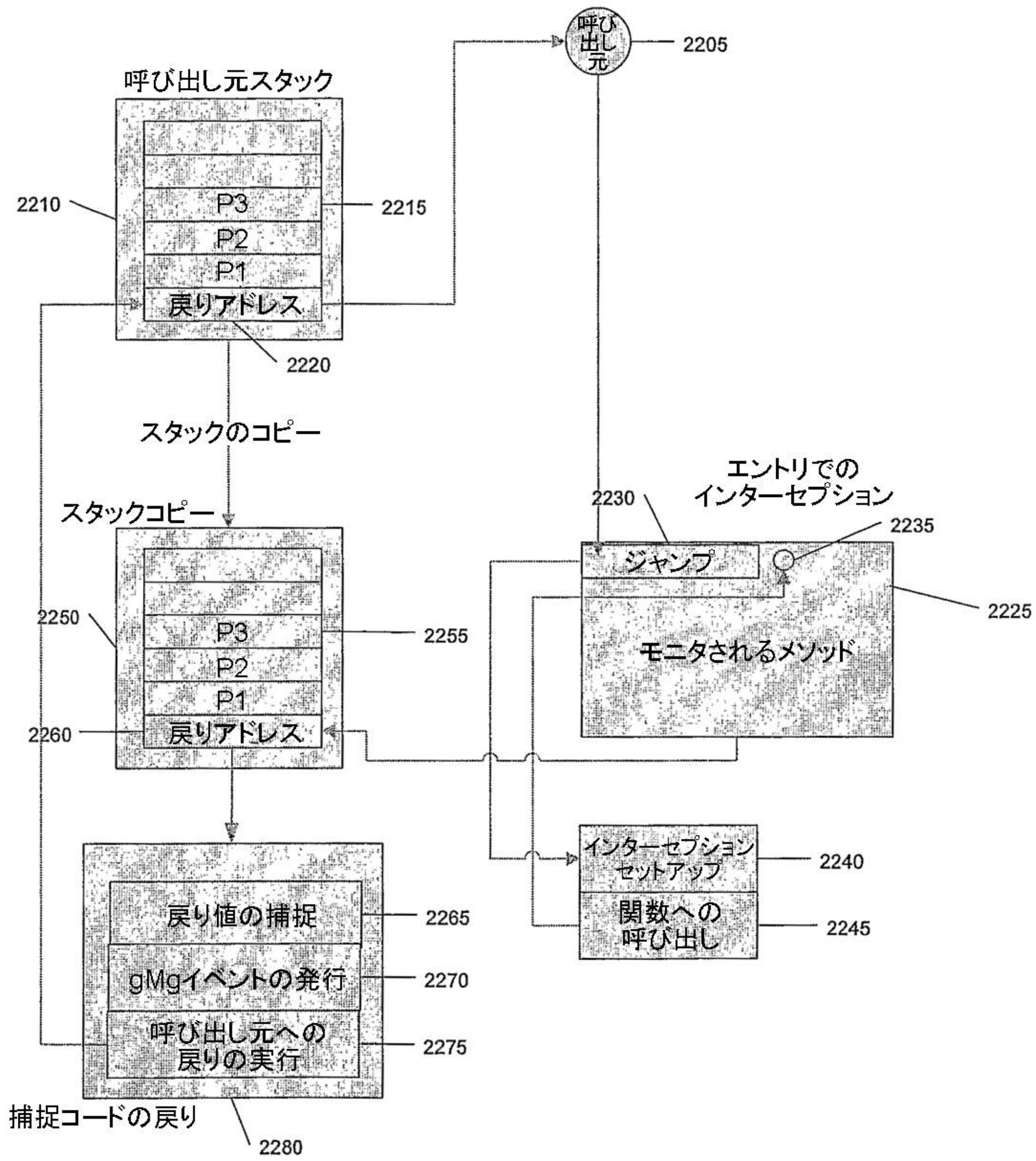
【図２０】



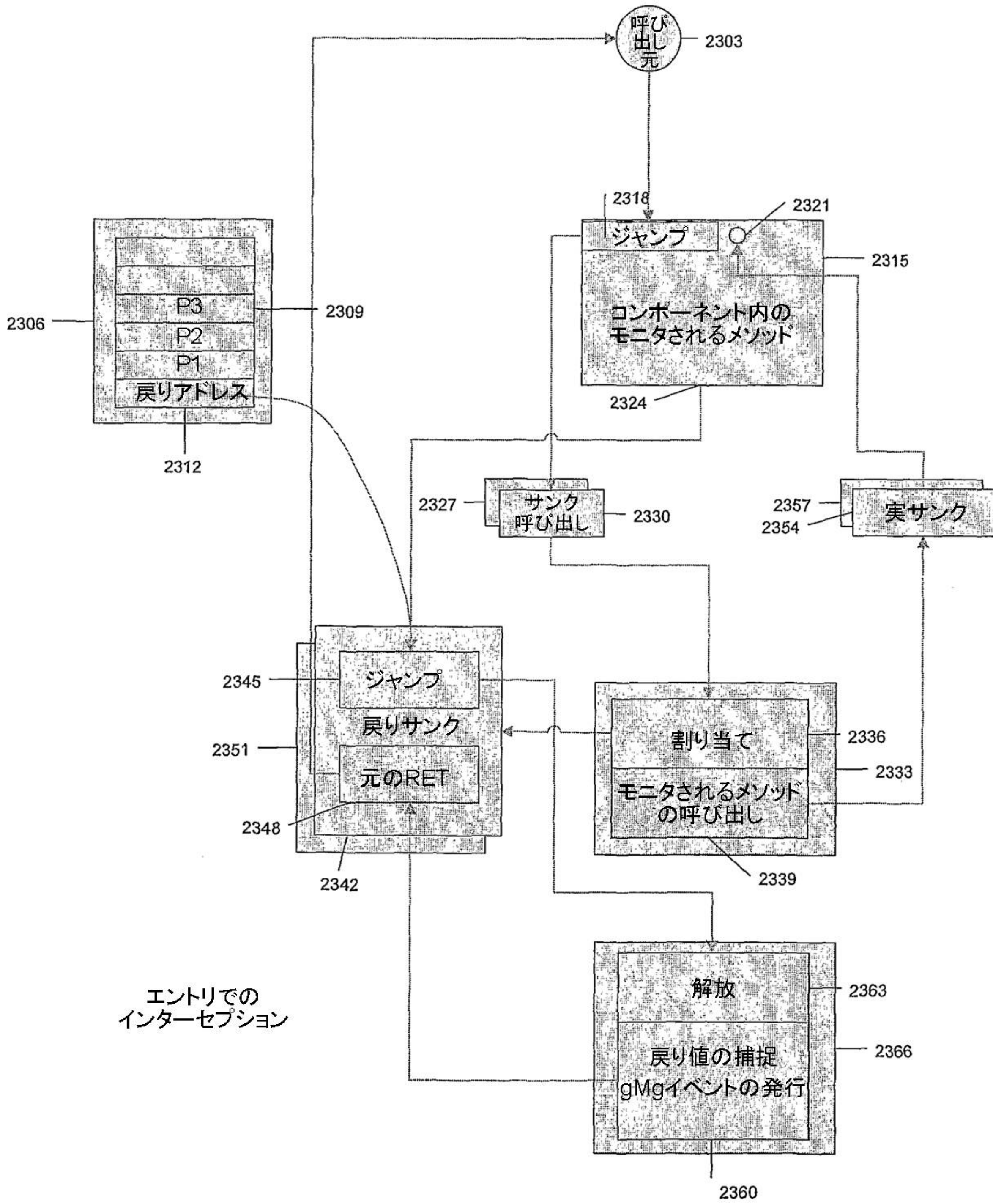
【図２１】



【図２２】



【図２３】



フロントページの続き

(81)指定国 AP(BW,GH,GM,KE,LS,MW,MZ,NA,SD,SL,SZ,TZ,UG,ZM,ZW),EA(AM,AZ,BY,KG,KZ,MD,RU,TJ,TM),EP(AT,BE,BG,CH,CY,CZ,DE,DK,EE,ES,FI,FR,GB,GR,HU,IE,IS,IT,LT,LU,LV,MC,NL,PL,PT,RO,SE,SI,SK,TR),OA(BF,BJ,CF,CG,CI,CM,GA,GN,GQ,GW,ML,MR,NE,SN,TD,TG),AE,AG,AL,AM,AT,AU,AZ,BA,BB,BG,BR,BW,BY,BZ,CA,CH,CN,CO,CR,CU,CZ,DE,DK,DM,DZ,EC,EE,EG,ES,FI,GB,GD,GE,GH,GM,HN,HR,HU,ID,IL,IN,IS,JP,KE,KG,KM,KN,KP,KR,KZ,LA,LC,LK,LR,LS,LT,LU,LV,LY,MA,MD,MG,MK,MN,MW,MX,MY,MZ,NA,NG,NI,NO,NZ,OM,PG,PH,PL,PT,RO,RS,RU,SC,SD,SE,SG,SK,SL,SM,SV,SY,TJ,TM,TN,TR,TT,TZ,UA,UG,US,UZ,VC,VN,ZA,ZM,ZW

(72)発明者 ルイ，フィリップ
アメリカ合衆国，ニューヨーク州　１００１１，ニューヨーク，ウエスト　１６ス　ストリート　５５
(72)発明者 コピトニック，ズビグニュー
アメリカ合衆国，ペンシルベニア州　１８３０１，イースト　ストラウズバーグ，ブラックベリー　テラス　５３８
(72)発明者 レイナ，デイビッド
アメリカ合衆国，ニューヨーク州　１１５６５－２１３６，マルヴェルヌ，レキシントン　アベニュー　８９

Ｆターム(参考) 5B042 GA05 HH20 HH30 LA19 LA24 LA25 MA14