República Federativa do Brasil
Ministério do Desenvolvimento, Indústria e do Comércio Exterior
Instituto Nacional da Propriedade Industrial.

(21) **PI0618412-0 A2**

BRPI0618412A2

(22) Data de Depósito: 10/10/2006
(43) Data da Publicação: 30/08/2011
(RPI 2121)

(51) Int.Cl.:
G06F 3/00

(54) Título: **MÉTODO PARA MONITORAR EVENTOS DERIVADOS DE UMA CAMADA DE APRESENTAÇÃO, MÉTODO PARA ANALISAR EVENTOS DE MÉTODO, SISTEMA PARA MONITORAR EVENTOS EM UMA CAMADA DE APRESENTAÇÃO DE UMA APLICAÇÃO ALVO DE COMPUTADOR, E MÉTODO PARA MONITORAR UMA FUNCIONALIDADE DE APLICAÇÃO HOSPEDEIRA**

(30) Prioridade Unionista: 11/10/2005 US 11/248,981

(73) Titular(es): Knoa Software, inc

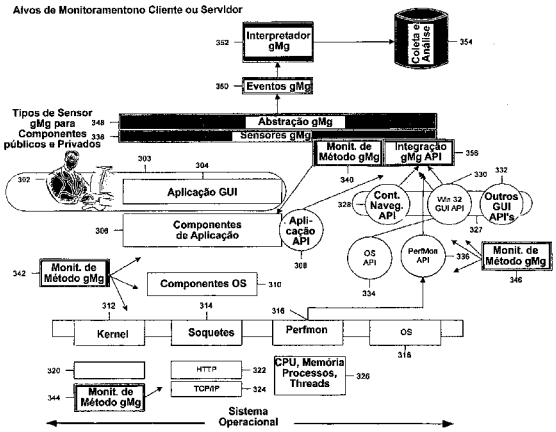
(72) Inventor(es): David Rayna, Philip Lui, Zbigniew Kopytnik

(74) Procurador(es): Antonio Mauricio Pedras Arnaud

(86) Pedido Internacional: PCT US2006039998 de 10/10/2006

(87) Publicação Internacional: WO 2007/044875de 19/04/2007

(57) Resumo: MÉDTODO PARA MONITORAR EVENTOS DERIVADOS DE UMA CAMADA DE APRESENTAÇÃO, MÉTODO PARA ANALISAR EVENTOS DE MÉTODO, SISTEMA PARA MONITORAR EVENTOS EM UMA CAMADA DE APRESENTAÇÃO DE UMA APLICAÇÃO ALVO DE COMPUTADOR, E MÉTODO PARA MONITORAR UMA FUNCIONALIDADE DE APLICAÇAO HOSPEDEIRA. A presente invenção apresenta um sistema e método para monitorar eventos derivados de uma camada de apresentação de uma aplicação alvo de computador incluindo as etapas de prover, independentemente de recompilar o código de fonte da aplicação alvo, um script executado em um nível dentro da aplicação alvo.O script escanea instanciamentos durante a execuçao de objetos da aplicação alvo, e aloca estruturas em tempo real para instanciamentos de objeto. Estas estruturas alocadas são adaptadas para criar uma reflexão da estrutura de aplicação alvo, usada junto com instanciações do objeto detectado correspondentes a uma pré-determinada estrutura de objeto, para capturar uma porção de espectro ambiental do objeto detectado. Ademais, o sistema pode processareventos de estado da máquina que ocorram pelo menos em um servidor e em uma máquina de cliente, correlacionar os eventos de estado da máquina com o espectro ambiental, e deduzir uma experiência de usuário com base nos eventos de estado de máquina correlacionados.



"MÉTODO PARA MONITORAR EVENTOS DERIVADOS DE UMA CAMADA DE APRESENTAÇÃO, MÉTODO PARA ANALISAR EVENTOS DE MÉTODO, SISTEMA PARA MONITORAR EVENTOS EM UMA CAMADA DE APRESENTAÇÃO DE UMA APLICAÇÃO ALVO DE COMPUTADOR, E MÉTODO PARA MONITORAR UMA FUNCIONALIDADE DE APLICAÇÃO HOSPEDEIRA".

Campo da Invenção

A presente invenção se relaciona a um sistema e método para monitorar uma interação do usuário com uma aplicação de computador, mais particularmente para monitorar e analisar a interação usuário-aplicação em uma sessão, em conexão com o desempenho e comportamento da aplicação, na extensão que estes afetem o usuário.

Histórico da Invenção

Nas empresas atuais, as aplicações comerciais provêem uma funcionalidade para executar processos comerciais em uma economia global complexa, quais aplicações se tornam extremamente complexas para atender os requisito de comerciais. Os esforços em adquirir capacidades para dominar as aplicações junto com mudanças de aplicação provocam um constante aumento de infraestrutura, cuja medição sistemática se constitui um desafio de ambos desempenho e medição do desempenho da aplicação.

Em geral, produtos de monitoramento de usuário podem ser divididos em categorias: Captura de Monitoramento de Desempenho, Monitoramento HTTP, Captura HTTP, Inserção de Código Fonte Durante Execução, Compilação ou Inserção de Código Fonte de Desenvolvimento de Aplicação Hospedeira. Os produtos a rastrear ilustram as respectivas categorias.

Há produtos que monitoram e capturam o desempenho de computador do usuário final, incluindo um produto da Reflectent Software, Westford, MA - "EdgeSight" (trademark) - para quantificar o desempenho e a disponibilidade de aplicações e serviços IT na perspectiva do cliente final, que permite visualizar o desempenho e a utilização quer em tempo real ou

historicamente. Se houver falhas, erros, e interrupções, as informações e evidências requeridas são imediatamente disponibilizadas para identificar a causa raiz da questão. Este produto não pode gerar dados de monitoramento a partir da camada de apresentação GUI de uma Aplicação Hospedeira e não pode monitorar uma atividade entre transações. Este produto não pode detectar subconjuntos de objetos GUI em conjuntos genéricos categorizados. Este produto não pode suportar multi-instâncias de Aplicações Hospedeira em nível GUI. Este produto não tem capacidade de realizar um Monitoramento de Método para chamadas de função. Este produto não pode suportar máquinas de estado distribuído local e remoto para um processamento rápido e eficiente.

Produtos convencionais são disponíveis para monitorar tráfego HTTP de um computador de cliente. Por exemplo, "End User Monitoring" da Mercury Corp. Mountain View, CA monitora proativamente websites e disponibilidade de aplicações em tempo real da perspectiva do usuário final. Este produto proativamente emula processos comerciais de usuário final contra aplicações. Este produto não gera dados de monitoramento a partir da camada de apresentação GUI de uma Aplicação Hospedeira e não monitora qualquer atividade entre transações. Este produto não pode detectar subconjuntos de objetos GUI em conjuntos categorizados genéricos, e ademais não provê capacidade para fazer um Monitoramento de Método. Este produto não pode suportar multi-instâncias de Aplicações Hospedeira em um nível GUI. Este produto não suporta máquinas de estado distribuído local e remoto para um processamento rápido e eficiente.

Um produto da Candle Corp (IBM/ Tivoli) - "ETEWatch" (trademark)- mede o tempo de resposta em nível de transação, cabo a cabo, que o usuário espera enquanto uma aplicação carrega ou executa uma ação. Este produto não pode gerar dados de monitoramento a partir da camada de apresentação GUI de uma Aplicação Hospedeira e não pode

monitorar qualquer atividade entre transações. Este produto não pode detectar subconjuntos de objetos GUI em conjuntos categorizados genéricos. Este produto não pode suportar multi-instâncias de Aplicações Hospedeira em nível GUI. Este produto não tem capacidade de realizar um Monitoramento de Método para chamadas de função. Este produto não suporta máquinas de estado distribuído local e remoto para um processamento rápido e eficiente.

Um produto da Compuware, Detroit, MI - "Vantage 9.7" (trademark) - correlaciona os dados de desempenho de aplicação e eventos com fatores comerciais importantes, como aplicações, transações, funções, e locais. Uma visualização adicional integra valores de tempo de resposta de usuário provendo uma maior capacidade para resolver problemas. Este produto provê que a organização IT resolva problemas de desempenho e proativamente administre aplicações e infra-estrutura. Este produto não pode gerar dados de monitoramento a partir da camada de apresentação GUI de uma Aplicação Hospedeira e não pode monitorar qualquer atividade entre transações. Este produto não pode detectar subconjuntos de objetos GUI em conjuntos categorizados genéricos. Este produto não pode suportar multi-instâncias de Aplicações Hospedeira em nível GUI. Este produto não tem capacidade de realizar um Monitoramento de Método para chamadas de função. Este produto não suporta máquinas de estado distribuído local e remoto para um processamento rápido e eficiente.

Há produtos que capturam tráfego HTTP em um servidor. "RealiTea" (trademark) da TeaLeaf Technology Inc - São Francisco, Ca captura passivamente as ações de cliente em tempo real, permitindo detecção, análise, e resposta imediata a estas questões. Este produto valida cada ação de cliente contra o resultado esperado em um processo multi-etapa. Este produto não gera dados de monitoramento a partir da de camada de apresentação GUI de uma Aplicação Hospedeira e não monitora qualquer atividade entre transações. Este produto não pode detectar

subconjuntos de objetos GUI em conjuntos categorizados genéricos. Este produto não suporta multi-instâncias de Aplicação Hospedeira em nível GUI. Este produto não tem capacidade de realizar um Monitoramento de Método para chamadas de função. Este produto não suporta máquinas de estado distribuído local e remoto para um processamento rápido e eficiente.

Há produtos que inserem códigos em Aplicação Hospedeira durante execução. "i³" (trademark) - da Veritas - monitora, analisa, e setup proativamente aplicações SAP, Siebel, etc.. Este produto provê uma visualização completa do desempenho da aplicação capturando, medindo e correlacionando medidas de desempenho de cada componente de infra-estrutura de aplicação (servidor Web, servidor de aplicação, banco de dados, e armazenamento). Embora este produto não veja eventos GUI, ele se restringe a aplicações Web e não é provido de conceito de detecção de conjuntos de objetos GUI genéricos. Ademais, este produto requer a inserção de códigos fonte no site da aplicação - um processo muito invasivo - e não um processo de implementação transparente. Este produto não pode detectar qualquer coisa diretamente no nível de sistema operacional. Este produto não pode detectar eventos ou recuperar propriedades diretamente dos métodos de componente (i.e. funções) ou diretamente interfacear Win32. Ademais, com este produto, não é possível fazer medições concorrentes como de aplicações múltiplas e diferentes, como Web e Win32. Ademais, falta persistência ao longo das páginas sem conceito de máquinas de estado localizado para execução de alertas em tempo real. Este produto não suporta máquinas de estado distribuído local e remoto para um processamento rápido e eficiente.

Há produtos que usam arquivos que resultam da compilação de código fonte no instante de desenvolvimento. "AppSight Black Box" (trademark) da Identify Software New York NY para servidores e/ou clientes, grava execução de aplicação em níveis sincronizados múltiplos com base

em um perfil dinâmico de gravação definido por usuário. Este produto não requer mudanças nos códigos fonte ou nos executáveis. No lado do cliente, Este produto tem uma captura estilo-Vídeo das ações de usuário e eventos de tela. Este produto não provê medições de resposta de transação. Ademais, enquanto este produto provê uma forma de monitoramento de método, ele usa arquivos gerados durante a compilação do código fonte de desenvolvimento e não suporta estritamente um processo de implementação transparente. A presente invenção provê um monitoramento de método sem requerer arquivos fonte compilados, e ademais, também monitora retornos de função. Este produto não gera dados de monitoramento a partir da camada de apresentação GUI no nível de objeto GUI a partir de modelos de objeto GUI. Este produto não pode detectar subconjuntos de objetos GUI em conjuntos categorizados genéricos. Este produto não suporta Multi-Instâncias de Aplicações Hospedeira em um nível GUI de aplicações GUI. Este produto não pode ser implementado de modo transparente e requer uma mudança no código fonte de aplicação e requer alguma resposta do processo de compilação. Este produto não suporta máquinas de estado local e distribuído para um processamento rápido e eficiente.

A patente U.S. N° 6.108.700 - "Application End-to-End Response Time Measurement and Decomposition" - se relaciona a uma medição cabo a cabo, geralmente em todas as camadas, uma estrutura (framework) para integrar um espectro de tipos de dados. A patente N° 6.078.956 "World Wide Web End User Response Time Monitor" descreve pedidos de monitoramento no nível HTTP, onde um tempo de resposta descreve pedidos de monitoramento em nível HTTP, onde é calculado um tempo de resposta associado a um primeiro pedido HTTP, e então passado para o servidor, e registrado para uso posterior. A patente U.S. N° 5.991.705 "End-to-End Response Time Measurement for Computer Programs Using Starting and Ending Queues" e

sua continuação a patente U.S. N° 6.202.036 se relacionam a um sistema para medir tempo de resposta cabo a cabo para uma transação realizada por computador. A patente U.S. N° 6.189.047 "Apparatus and Method for Monitoring Event Queue Operation with Pluggable Event Queues" descreve seqüências de mensagem para refletir as seqüências de mensagens usadas por uma aplicação, em conseqüência de uma interação GUI. Quais patentes estão incorporadas nesta por referência em sua totalidade.

Particularmente, as patentes de técnica anterior N^{os} 5.991.705, 6.108.700 e 6.078.956 falham em descrever um método ou sistema que possam gerar dados de monitoramento a partir da camada de apresentação GUI de uma Aplicação Hospedeira. Nenhuma técnica descrita na técnica anterior suporta máquinas de estado distribuído (local e remoto) para um rápido processamento, nem implementa qualquer modelamento ou estrutura de Árvore de Objeto GUI, além de simples filas, nem suporta o manuseio de Multi-Instância GUI. Ademais, nenhuma técnica descreve o uso de subconjuntos de objetos GUI em conjuntos categorizados genéricos, e ademais não provê capacidade para realizar Monitoramento de Método. A técnica anterior também não provê um contexto GUI para as outras formas de dados não-GUI.

A presente invenção se relaciona à matéria similar à descrita na patente emitida pelo depositante (N° 6.340.977 - "System and Method for Dynamic Assistance In Software Application Using Behaviour and Hospedeira Application Models") e sua continuação - N° de série 09/989.716 de 20 de Novembro de 2001. O depositante também tem implementações comerciais "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)", e "Knoa Compliance Measurement 2.0 (Knoa User Compliance Monitor). A presente invenção atende e melhora com um trabalho

anterior do depositante.

Falta na técnica um sistema de detecção e monitoramento GUI capaz de detecção automatizada de multi-instâncias de uma Aplicação Hospedeira, e detecção de multi-instâncias de subconjuntos de estruturas GUI. Ademais, falta na técnica um sistema capaz de detecção genérica de objetos GUI de Aplicação Hospedeira, ou subconjuntos de Aplicações Hospedeiras. A presente invenção preenche estas e outras necessidades.

Sumário da Invenção

Em um aspecto da invenção, provê-se um método para monitorar eventos a partir de uma camada de apresentação de aplicação alvo de computador. O método inclui as etapas de prover, independentemente de recompilar, o código fonte da aplicação alvo, um script sendo executado em um nível na aplicação alvo. O script escanea instanciamentos de objetos da aplicação alvo e aloca em tempo real estruturas de instanciamentos de objeto. Estas estruturas alocadas são adaptadas de modo a criar uma reflexão da estrutura da aplicação alvo que é usada junto com instanciamentos de objeto detectados, que correspondem a uma pré-determinada estrutura de objeto para capturar uma porção de um espectro ambiental do objeto detectado.

Em outro aspecto da invenção, o método processa eventos de máquina de estado, que ocorrem pelo menos em uma das máquinas - servidor ou máquina do cliente, correlaciona os eventos de máquina de estado com o espectro ambiental, e deduz a experiência de usuário a partir dos eventos de máquina de estado correlacionados.

Em ainda outro aspecto da invenção, um sistema configurado para monitorar eventos em uma camada de apresentação de aplicação alvo de computador compreende servidor e computador-cliente interconectados em uma rede de comunicação, onde as instruções operacionais no servidor incluem um programa de monitoramento incluindo um script operável executado em um nível abaixo da camada

de apresentação. O programa de monitoramento inclui instruções para escanear instanciamentos durante execução de objetos, inclui chamadas e retornos de método, e objetos GUI para a aplicação alvo, alocar estruturas em tempo real para os instanciamentos de objeto, adaptar as estruturas alocadas de modo a criar uma reflexão da estrutura de aplicação alvo, detectar um ou mais instanciamentos de objeto correspondente a uma pré-determinada estrutura de objeto, e capturar pelo menos o conteúdo dos objetos detectados.

Estes e outros aspectos, componentes, e vantagens da presente invenção poderão ser apreciados em conexão com os desenhos anexos a partir da descrição de certas configurações ilustrativas.

Definição de Termos

Os termos usados nesta especificação geralmente têm seu significado ordinário na técnica em um contexto específico, onde cada termo é usado. Certos termos serão discutidos para prover um guia adicional para descrever os dispositivos e métodos, como fazê-los e usá-los.

Por conseguinte, uma linguagem alternativa (e correlatos) poderá ser usada para qualquer um ou mais termos dados nesta, não tendo qualquer significância especial se o termo foi elaborado ou discutido nesta. Sendo apenas providos sinônimos para certos termos. Os um ou mais sinônimos usados nesta especificação não exclui o uso de outros sinônimos. O uso de exemplos nesta especificação, incluindo exemplos dos termos tratados, tem um propósito meramente ilustrativo, e não limita o escopo e o significado da invenção ou de qualquer termo exemplar. Ademais, a invenção não se limita às configurações dadas.

"Agente" denota uma unidade de código primário associado a uma ou mais instâncias de execução de um Engine DLL.

"Componente" denota um pacote de sistema operacional geral que compreende dados e código de programação a serem executados. O código de programação de componente contém chamadas de função ou métodos que são usadas para

executar a funcionalidade de uma aplicação ou serviço e facilitar a reutilização do código e arranjo modular de uma Aplicação Hospedeira mais complexa. Componentes implementam muitas tecnologias diferentes criadas por empresas para promover uma reutilização e publicação de interfaces para aplicações externas. Componentes contendo códigos de programa ou classes orientadas a objeto podem ser implementados em tecnologias de componente, tal como da Microsoft (Microsoft Corp, Redmont, WA), Modelo de Objeto de Componente (COM), uma estrutura (framework) para desenvolver e suportar objetos de componente de programa COM ou outros tipos de tecnologia de componente.

"Espectro Ambiental" denota condições variáveis presentes de modo manifesto na aplicação alvo, mas não se limitando a, objetos, objetos GUI, mensagens, chamado de Método, e retorno de Método.

"Modelo de Evento" denota um conjunto de eventos, onde a detecção e coleta de eventos refletem e representam um processo real que ocorre em uma Aplicação Hospedeira.

"Função Export" denota uma interface de método exposta em um componente DLL, onde o nome de função Export está inscrito nos cabeçalhos de módulo de programa.

"Estrutura" (Framework) denota uma estrutura fundamental de trabalho para modelar estados, e a detecção de objetos GUI e eventos de chamada e retorno de Método.

"Genérico" denota uma operação para detectar um conjunto de alvos. Este conjunto pode se referir a uma classe ou categoria de objetos similares, e aqui é usado para descrever conjuntos de objetos GUI, métodos, e outros eventos. Em oposição ao termo "Genérico" é usado o termo "Específico", que usualmente se refere a um único objeto GUI, método, ou a um outro evento.

"Engine gMg" denota um interpretador de script que processa arquivos em script binário contendo descrições lógicas e condicionais que executam um monitoramento seletivo em alvos monitorados.

"Árvore de Objeto GUI gMg" denota uma estrutura

hierárquica usada nesta para modelar e representar objetos GUI ativos, compreendendo a interface de usuário de uma Aplicação Hospedeira.

"Contexto GUI gMg" denota, em um sistema gMg, a intercalação de qualquer tipo de dado em um fluxo de dados GUI, onde os dados GUI provêem um contexto GUI (ou de usuário) com propósito de análise, para refletir melhor a experiência do usuário em uma aplicação.

"Solução gMg" denota um arranjo binário de um script compilado que descreve condições, lógica, alvos seletivos executados para monitorar a Aplicação Hospedeira.

GUI - (de Graphical User Interface (Interface Gráfica de Usuário)) uma interface controlada por uma Aplicação Hospedeira - é a interface primária entre o usuário e a Aplicação Hospedeira. O termo "Objeto GUI" se refere a um objeto que faz parte de uma estrutura GUI programada normalmente em RAM, que compreende a interface de usuário apresentada ou que reage ou é controlada por um dispositivo de entrada de computador (teclado) através de um código de Aplicação Hospedeira.

"Hook" denota a parte do sensor que realiza a captura de mensagens de sistema operacional geradas por uma Aplicação Hospedeira, que serão processadas por um sistema externo para filtrar ou alterar ações com base no envio de mensagem em tempo real de Aplicação Hospedeira.

"Aplicação Hospedeira" (Host Application) denota uma aplicação alvo de monitoramento.

"IE" - abreviatura de "Internet Explorer" - é o navegador de Internet da Microsoft.

"Instância" denota uma funcionalidade de programa, cujo código operacional pode ser levado para um conjunto derivado, onde uma única derivação pode instanciar uma ou mais unidades operacionais na memória para implementar a funcionalidade. O código de programa na memória iniciada e executada em conseqüência da operação de derivação é chamado instanciamento.

"Eventos Lógicos" denota eventos processados em uma

máquina de estado que representam um estado alvo que pode ser processado em um ou mais níveis, i.e. em estágios sucessivos de processamento de máquina de estado em diferentes locais físicos em clientes ou servidores.

"Método" denota conotação com uma função, por exemplo, uma função de programa e vice-versa. O Método/Função se encontra em um componente ou biblioteca.

"Eventos de Método" denota eventos gerados monitorando um método ou função de componente (por exemplo, método ou função de programa).

"Multi-instância" denota mais que uma instância de uma designada que estão concorrentemente na memória.

"Variável de Objeto" denota um objeto especializado da invenção que se liga a um membro de objeto GUI existente na Árvore de Objeto GUI.

"OS" é abreviatura de "Sistema Operacional" (Operating System").

"Sensor" denota uma unidade de código especializada, que recupera dados, em tempo real, a partir de componentes de aplicação usando interfaces públicas ou privadas.

"Específico" denota uma operação para detectar uma condição lógica, tal como a detecção de um objeto GUI único.

"Máquina de Estado" denota uma unidade de processamento com eventos de entrada e saída que determinam estados distintos e discretos em seqüência ou progressão, através de um processamento condicional dos eventos que chegam, acoplados a um dispositivo de armazenamento persistente.

"Thunk" denota uma estrutura dinamicamente alocada usada para executar um código ou armazenar informação de instância no monitoramento de método. A estrutura Thunk é aplicada a uma variedade de cenários para capturar chamadas de função ou retornos de função.

"Função Virtual" denota uma função que, se sobreposta por uma subclasse, será chamada pela classe base.

"VTable" denota uma tabela virtual em uma classe COM,

contendo apontadores de métodos para métodos ou funções, pertencendo a uma classe.

Descrição Resumida dos Desenhos

SISTEMA DE COMPUTAÇÃO E GUI

A figura 1 mostra um hardware cliente e servidor, de acordo com uma configuração da invenção;

A figura 2 mostra uma camada de apresentação GUI em um exemplo de aplicação Web GUI, de acordo com uma configuração da invenção;

MONITORAMENTO DE ALVOS

A figura 3 mostra alvos monitor em um ambiente operacional de computador no cliente ou servidor (não GUI), de acordo com uma configuração da invenção;

VISÃO GERAL DE COMPONENTE DE SISTEMA gMg (cmd);

A figura 4 mostra uma Visão Geral de Componente de Sistema gMg;

FLUXO DE ÈVENTO DE SISTEMA GMG

A figura 5 mostra uma Visão Geral de Fluxo de Evento e Propriedade no Sistema gMg, de acordo com uma configuração da invenção;

A figura 6 mostra Fontes de Evento Principal, de acordo com uma configuração da invenção;

OBJECT TREE GUI

A figura 7 mostra atualização da Árvore de Objeto GUI, de acordo com uma configuração da invenção;

A figura 8 mostra Eventos gMg de Árvore de Objeto: atualização da Árvore de Objeto GUI e Eventos Cria/Destrói (Create/ Destroy) gMg, de acordo com uma configuração da invenção;

FUNÇÕES PARA SUPORTE MULTI-INSTÂNCIA

A figura 9 mostra uma funcionalidade de programa de carregamento, de acordo com uma configuração da invenção;

A figura 10 mostra Variáveis de Objetos, de acordo com uma configuração da invenção;

A figura 11 mostra a estrutura "WhenObject", de acordo com uma configuração da invenção;

A figura 12 mostra um Contexto de Janela com Processo e

diferentes elementos de janela, de acordo com uma configuração da invenção;

A figura 13 mostra Contextos e Múltiplas Janelas Popup de acordo com uma configuração da invenção;

A figura 14 mostra um processo Hook, de acordo com uma configuração da invenção;

DETECÇÃO GENÉRICA DE OBJETOS GUI

A figura 15 mostra Espectro Específico para Genérico, de acordo com uma configuração da invenção;

TÉCNICAS DE SENSOREAMENTO DE MÉTODO

A figura 16 mostra uma Visão Geral de Monitoramento de Método, de acordo com uma configuração da invenção;

A figura 17 mostra uma Árvore de Módulo usada para Monitoramento de Método, de acordo com uma configuração da invenção;

A figura 18 mostra a criação de um Monitor de Método, de acordo com uma configuração da invenção;

A figura 19 mostra Assinatura de Método e uma Sobreposição de Código de Método, de acordo com uma configuração da invenção;

A figura 20 mostra Alocações Thunk para Monitoramento de Método, de acordo com uma configuração da invenção;

A figura 21 mostra o Monitoramento de um Método de Função Virtual, de acordo com uma configuração da invenção;

A figura 22 mostra um Monitoramento de Retorno de Método Cópia de Stack, de acordo com uma configuração da invenção;

A figura 23 mostra o Retorno de Método de Monitoramento, Alocação de Stack, de acordo com uma configuração da invenção;

ANÁLISE

A figura 24 mostra a Hierarquia de Máquina de Estado, com Máquina de Estado local e remota, de acordo com uma configuração da invenção;

A figura 25 mostra uma Avaliação Estática e Instanciada para Transições em Máquinas de Estado, de acordo com uma configuração da invenção;

A figura 26 mostra um Processamento de Evento de Máquina de Estado, de acordo com uma configuração da invenção;

A figura 27 é um fluxograma ilustrando um processo, de acordo com uma configuração da invenção;

A figura 28 é um fluxograma ilustrando um processo, de acordo com outra configuração da invenção; e

As figuras 29A e 29B são fluxogramas ilustrando um processo, de acordo ainda com uma outra configuração da invenção.

Descrição Detalhada das Configurações Ilustradas

Para efeito de introdução, apresentam-se configurações de sistemas e métodos para prover um rastreamento e monitoramento temporizado de eventos estruturados, específicos ou genéricos, em múltiplas instâncias de uma Aplicação Hospedeira. O rastreamento e monitoramento resultam de uma camada de apresentação de aplicação, por exemplo, em GUI, ou no nível de função ou método. A geração de eventos e a subseqüente análise de detecção de padrão provêem avaliações e estatísticas. O monitoramento de usuário inclui, sem se limitar, medição de tempo de resposta, detecção de erros, e coleta de eventos de aplicação. O monitoramento de usuário provê informações a partir das quais se distingue o contexto e interação de usuário.

A medida que a aplicação se torna mais complexa em um ambiente empresarial, compreender e medir a experiência de usuário se torna mais difícil. Uma gama de problemas pode ocorrer, onde os usuários podem experimentar tudo, desde carregamentos ou tempos de resposta lentos, mensagens com erro de encriptação, páginas web parcialmente convertidas (renderizadas), a registros de banco de dados incompletos, etc.. Departamentos IT focalizam infra-estruturas de estágios finais, ainda que na verdade sejam usuários empresariais, responsáveis pelas rendas e lucros das empresas.

Para determinar a medida quantitativa de uma experiência de usuário, há potencialmente muitas fontes de informação

disponíveis em ambientes de usuário e servidor. Uma vez havendo muitos componentes que executam interações e transações com fontes de dados centralizadas, cada componente se torna um link uma cadeia longa de eventos que compreende a Experiência de Usuário. Tais componentes podem se encontrar na camada de rede, na camada de hardware (CPU, Memória, Memória Virtual), na camada de apresentação GUI, no servidor de aplicação, etc..

Em virtude de o usuário executar processos de negócio críticos a medida que interage com a camada GUI de uma aplicação, o problema de contexto e identificação no nível GUI de todas ações e atividades em qualquer nível de infra-estrutura de uma aplicação idealmente deve ser levado de volta para uma ação de usuário em algum contexto de usuário. Eventos de camada GUI provêm apenas tal contexto. O sistema rastreia este nível e "vê" o que o usuário vê". O GUI é organizado em uma estrutura hierárquica e contém tantos subconjuntos de objetos GUI quantos possa replicar durante execução dependendo da ação do usuário e da lógica de organização da aplicação que controla como as estruturas de objeto GUI são geradas e mantidas durante execução. O sistema provê mecanismos de identificação de suporte que diferenciam as estruturas que, outrossim, pareceriam idênticas. Em particular, se um usuário encontra uma mensagem de erro gerada em qualquer nível da infra-estrutura, a instância de erro é capturada para análise posterior para corrigir a aplicação.

Na Internet atual, é muito comum que as aplicações sejam multi-instanciadas, onde as janelas têm estruturas idênticas operadas simultaneamente, criando problemas para a detecção precisa de objetos GUI em Multi-Instâncias da aplicação. Particularmente, sendo problemático distinguir interfaces que parecem diferentes ao usuário, mas que, internamente, parecem ter estruturas idênticas. Aplicações mais recentes podem gerar numerosas janelas com diferentes identificações na interface de

usuário, e ainda que tenham estruturas internas idênticas de objeto. Este multi-instanciamento também é um efeito posterior das práticas de programação mais modernas que se esforçam em maximizar o compartilhamento de procedimentos, e a reutilização de componentes para obter ambos um tempo de execução mais longo e eficiência.

Quanto ao problema da Transparência, um sistema de monitoramento precisa ser não-invasivo e transparente à Aplicação Hospedeira, e operar como se o componente de monitoramento não existisse. Muito freqüentemente, a abertura da aplicação não provê acesso ao código fonte para prover interfaces e produzir eventos customizados a aplicações externas, por exemplo, um programa de monitoramento. Com um sistema de monitoramento, o problema passa a ser estender uma aplicação para monitoramento seletivo sem alterar a aplicação. Para conseguir tal transparência, o sistema de monitoramento deve interceptar e filtrar os eventos que chegam, e adicionalmente ler e processar os mesmos. Opcionalmente, o sistema também pode filtrar os eventos passados, em seguida da Aplicação Hospedeira.

Quanto ao problema de Fontes de Medição disponíveis, os processos de rastreamento em Aplicações Hospedeiras empresariais podem ser instados a executar um Monitoramento de Desempenho de Usuário real com as Fontes de Medição disponíveis. No nível GUI, a prospecção em processos complexos requer mais dados a partir de mais dimensões prontamente disponíveis em forma padrão. Conquanto muitas API's sejam publicadas pelos desenvolvedores, sua maior parte é mantida oculta, e, freqüentemente, alguns eventos ou propriedades podem ser recuperados somente de componentes privados. Através de um acesso aos componentes privados, é possível melhorar substancialmente precisão e acessibilidade.

Quanto ao problema de Cobertura de Monitoramento, com a complexidade e tamanho das aplicações empresariais correntes, a cobertura de monitoramento pode implicar em

um risco enorme. Há centenas mesmo milhares de módulos, e milhares de operações possíveis pela natureza das permutações e combinações de grandes bancos de dados que podem gerar interfaces de usuário dinamicamente baseadas em condições derivadas de dados. Para monitorar mesmo uma utilização básica, um sistema deve detectar centenas de estados ou condições, e capturar dados eficientemente em tempo real. Ademais, a implementação do sistema deve ser suficientemente rápida para acomodar as demandas do negócio e prover um retorno e cobertura adequados.

Quanto ao problema de Detecção Agregada em Tempo Real, uma detecção complexa de eventos para grandes bases de usuário pode ser um desafio formidável em uma programação eficiente em um ambiente com recursos restritos. Freqüentemente, aplicações empresariais podem gerar tantos eventos quanto o processamento e detecção de processos ou operações empresariais seletivos se mostrem desafiadores com as restrições de tráfego de dados, velocidade de suprimento, e monitoramento transparente. Para contemplar estes problemas, a invenção propõe um número de mecanismos.

Para contemplar o problema de contexto e identificação no nível GUI, no Método Multi-Instância Genérica e Sistema de Detecção GUI (gMg), provê-se um Contexto de Usuário que é um fluxo de eventos GUI, que pode ser intercalado com qualquer outro tipo de dado disponível no ambiente e correlacionado com outras fontes externas, em particular com dados de servidores ou da infra-estrutura ambiental. Eventos detectados na camada GUI provêem um mapeamento direto da experiência de usuário e interação com a camada de apresentação de uma aplicação. O sistema inclui condições de contemplar uma ampla variedade de fontes de dados com configurações de diferentes de sensores, i.e. Sensor de Método, em uma vista integrada em um Contexto GUI. Usando um sensor GUI transparente especializado, todas as outras formas de dados podem ser intercaladas no fluxo de evento GUI para

prover correspondência direta de eventos com uma experiência de usuário em tempo real.

No Sistema gMg, os modelos dinâmicos são sincronizados com estruturas de Aplicação Hospedeira em tempo real para contemplar o problema de multi-instanciamento de aplicação e prover um suporte às complexas interfaces gráficas de Aplicação Hospedeira, onde múltiplas instâncias de estruturas GUI similares ou idênticas são comuns. Árvore de Objeto GUI dinâmica modela objetos GUI de Aplicação Hospedeira. Variáveis dinâmicas podem ser afixadas para selecionar objetos GUI na árvore para suportar Multi-Instâncias de objeto GUI.

A complexidade de aplicações empresariais e sua iniciação fazem que quaisquer mudanças às aplicações de empresariais, sejam extremamente lentas e desajeitadas, se possível. O Sistema gMg inclui configurações de vários assistentes de sensores para transparência de aplicações, para recuperar eventos e propriedades da aplicação empresarial e seu ambiente, sem alterar o código fonte da aplicação empresarial. Assim, estes sensores têm a capacidade de estender a funcionalidade da aplicação sem recompilar quaisquer componentes da Aplicação Hospedeira. Os sensores intervêm minimamente de modo não-invasivo para capturar os dados necessários para popular os modelos da aplicação empresarial. Destes modelos podem ser gerados eventos precisos para permitir análise.

A funcionalidade de monitoramento de Método do Sistema gMg expande grandemente as fontes de medição disponíveis. No ambiente de Aplicação Hospedeira há duas categorias de interfaces: pública e privada. As interfaces públicas são publicadas e documentadas para integrar sistemas externos. Interfaces de Programação de Aplicação (API de "Application Programming Interfaces") através de uma gama de convenções de linguagem de programação provêem a capacidade de integrar a funcionalidade de componente com as funções externas de um sistema externo. Interfaces privadas não são abertamente documentadas, mas se o seu

comportamento for descoberto, as mesmas poderão prover propriedades ou eventos valiosos e por vezes críticos para construir contextos adicionais para medição que podem ser enquadrados em um contexto GUI. Abaixo são descritos sensores com a capacidade de inspecionar de modo transparente (i.e., sem mudar o código de fonte da Aplicação Hospedeira) ambos componente público e componente privado, e integrar seus eventos com outros tipos em um único ambiente de programação de desenvolvimento, assim abrindo dimensões adicionais mais ricas para o processo de medição.

Para monitorar a cobertura da camada GUI, a técnica anterior requer uma implementação manual de um correspondente descritor para cada objeto GUI específico, assim como uma extensiva instrumentação seletiva do GUI da aplicação. Para eficientemente obter uma cobertura suficiente de uma aplicação para um monitoramento efetivo, portanto sendo necessário um mecanismo mais efetivo para prover uma extensiva cobertura, usando um número razoavelmente pequeno de operações. O sistema gMg faz isto através de uma detecção Genérica de Métodos ou estruturas GUI otimizando e reduzindo grandemente os esforços de implementação de soluções de medição.

Na camada GUI e em outras estruturas, faz parte da natureza da organização interna da aplicação dispor de conjuntos objetos relacionados de para implementar e converter (renderizar) a camada de apresentação visível. Com esta organização própria, obtêm-se Eventos a partir da detecção de estruturas de objetos GUI relacionados em tempo real, que geram Eventos categorizados e são executados como operações genéricas para monitorar conjuntos de objetos GUI. Um grande benefício da abordagem de sistema, é que durante o tempo de desenvolvimento, os desenvolvedores da solução de monitoramento não precisam descrever cada resultado possível produzido, mas somente analisar a expressão de operador estruturado para detectar todas estruturas

similares, quer no presente ou futuro. Genericamente, os dados colhidos então poderão ser integrados com outras fontes do sistema para uma visualização multidimensional da interação de usuário com a Aplicação Hospedeira.

Quanto à Detecção Agregada em Tempo Real, no Sistema gMg são apresentadas configurações alternativas de máquinas de estado distribuído local ou remoto que processam os eventos de primeiro nível detectados. Na fonte, em um fluxo de dados Contextuais GUI, máquinas de estado podem processar eventos localizados em um estado lógico mais alto e emitir uma mensagem de rastreamento para o servidor de coleta. Recebendo um estado lógico diferente, as máquinas de estado remoto podem subseqüentemente processar adicionalmente este tipo de evento lógico, junto com outros tipos de eventos lógico similares recebidos de outros computadores. Esta configuração de processamento distribuído provê taxas de detecção muito rápidas para grandes bases de usuário.

Usando modelos GUI local (cliente) ou remoto (servidor), gM^2g (aqui também chamado "gMg") é um sistema de Rastreamento de Método e Multi-Instanciamento Específico e Genérico de GUI automatizado para modelar Aplicações Hospedeiras de Software e Multi-Instanciamento ou subconjuntos de Aplicações Hospedeira concorrentes. As capacidades gMg podem ser aplicadas de modo transparente sem precisar recompilar o código fonte de aplicação alvo. O modelo gMg no cliente local ou servidor remoto, é um GUI e estrutura (framework) de Objeto de Método que reflete a estrutura real da Aplicação Hospedeira e o mecanismo de sistema que reconstrói e interpreta os dados colhidos na análise de saída. Objetos GUI lógicos são rastreados por comandos de código que filtram Eventos e propriedades de objeto GUI para detectar conjuntos de objetos GUI reais. Objetos de Método são rastreados rastreados por comandos de código para detectar Eventos criados pelo Engine interpretador em resposta a chamadas e retornos de Método de Aplicação Hospedeira para os

componentes. GUI e quaisquer outros Eventos e propriedades disponíveis podem então ser recuperados de seus componentes. Como um sistema dinâmico usando Modelos gMg prescritos, o processamento gMg aloca e adapta estruturas em tempo real à existência Multi-Instância de GUI real e Objetos de Método em Aplicações Hospedeiras. Estas estruturas são processadas para produzir eventos lógicos e remontados para refletir estruturas Multi-Instância em Contextos distintos, sem precisar ter um conhecimento prévio de todos os relacionamentos ou propriedades da Aplicação Hospedeira.

Organizando dinamicamente os dados extraídos em tempo real da Aplicação Hospedeira nos Objeto, Processo, e Árvores de Módulo de Código, o Sistema gMg manipula estruturas complexas múltiplas e correspondentes conjuntos de múltiplas propriedades (i.e., texto) e correlaciona estas estruturas em conjunto para obter uma análise precisa. Uma análise para reportar, alertar, e responder pode ser feita por manipulação Multi-Instância, por Máquinas de Estado conectadas que operam no servidor e/ou cliente em uma variedade de configurações.

Para detectar ocorrência em instâncias múltiplas relacionadas de uma única Aplicação Hospedeira, provêem-se membros de código diferentes e extensivos para os diferentes tipos de sensores que detectam Categorias de Evento que aparecem em vários pontos da aplicação, ou em seu ambiente operacional do nível GUI para as chamadas de Método de Componente. O sistema inclui um componente Interpretador Multi-Instância que parses um script proprietário, que, especificamente ou genericamente, filtra e interpreta os Eventos que passam nos sensores. Dependendo das condições de propriedade e evento, o interpretador gera Mensagens de Rastreamento. Opcionalmente, os Eventos Interpretados podem ser analisados em um fluxo no componente de Máquina de Estado Local antes de gerar a Mensagem de Rastreamento (Rastreamento). Um componente de Comunicação então

arranja (empacota) quer o interpretador ou as Mensagens de Rastreamento de Máquina de Estado Local e as transmite ao componente de Máquina de Estado Remoto, que a rastrear analisa as Mensagens de Rastreamento recebidas. Alternativamente, a Máquina de Estado Remoto pode distribuir diretamente mensagens a uma outra Máquina de Estado para um console e apresentação.

O componente Reporte pode realizar uma análise para pontos de vista agregados, direções, agrupamentos, e categorias. A análise pode ser transformada em uma variedade de formas textuais, tabulares, ou gráficas e disponibilizadas para uma variedade de funções, que podem se destinar a um grupo suporte de infra-estrutura, gerenciamento de string, suporte de usuário, e outros.

Os sistemas e métodos descritos são aplicáveis a muitas áreas. Por exemplo, em um ambiente de produção, podem ser utilizados em um amplo espectro de aplicações de Desempenho de Usuário, onde há necessidade de monitoramento contínuo. Em Suporte de Infra-Estrutura, o sistema pode ser usado para medir tempos de resposta e detectar as mensagens de erro encontradas pelo usuário. Em Treinamento, a informação de utilização pode detectar os usuários que enfrentam dificuldades com partes da aplicação. Em Conformidade, os processos de monitoramento podem detectar quando os usuários saem das políticas aceitas e gerar alertas. Para aplicações de Segurança, a detecção de uma atividade suspeita em uma ou mais aplicações pode gerar alertas. Em Planejamento de Recursos, informações de utilização determinam como são aplicados os recursos e revelam perfis de tempos de pico. Para um desenvolvimento de programação, retro-alimentação, tempos de resposta precisos, e erros provêem correções e novas edições mais rapidamente. Eventos GUI são medidos para transações e entre transações intercaladas com diferentes tipos de dados, dadas como único conjunto em um contexto GUI de usuário. Em Suporte, as informações de utilização, erro e tempo de resposta

provêem um perfil automático de incidentes e facilitam o processo de encontrar e resolver problemas.

A rastrear é dado um resumo de hardware e ambientes GUI. A descrição de sistema de monitoramento gMg começa tratando diferentes tipos de monitoramento de alvos, seguindo uma visão geral do componente instalado. O Modelo de Evento então é rastreado pela exploração da estrutura de dados GUI (Árvore de Objeto GUI gMg). A descrição a respeito de métodos de implementação de Multi-Instanciamento de engines e scripts gMg em um multi-processo e ambiente de encadeamento (thread) e então apresentados. Então, se discute a detecção de objeto GUI genérico *versus* específico, seguindo uma descrição a respeito de Monitoramento de Chamada de Retorno de Método usando Assinaturas de Método gMg e funções virtuais em objetos COM. A seção final explora análise usando Máquinas de Estado de distribuição Hierárquica para rápida ação e detecção em tempo real.

A figura 1 ilustra a incorporação de uma configuração de computador-cliente 100 e servidor 105 capaz de implementar e operar o sistema e método de Monitoramento de Método GUI Genérico. O Sistema e Método gMg pode ser um software ou programa de aplicação executado por um computador-cliente 100 que pode ser um micro-computador, uma estação de trabalho, ou qualquer outra plataforma de computação (por exemplo, um assistente pessoal (PDA), ou dispositivos eletrônicos pessoais (PED) incluindo celulares, aparelhos MP3, câmaras digitais, e gravadores de vídeo). O servidor 105 ou qualquer computador tendo software de servidor coleta dados de muitos computadores e complementa as tarefas de processamento de cliente. O conector de rede 110 provê conectividade a computadores externos, i.e. servidores. Um protocolo de conexão padrão transmite e recebe dados em qualquer formato entre computadores remotos. O computador-cliente 100 pode incluir subsistemas conectados por um barramento. As instruções incluídas no Sistema RAM 130 são executadas

pela Unidade de Processamento Central 150.

Numerosos dispositivos externos são conectados ao Controlador de Entrada/Saída (I/O) 115 via Portas (não mostradas). O display de gráficos 2D e 3D e vídeo, ou qualquer imagem móvel, é operada pelo Adaptador de Vídeo 120 que mostra um gráfico na Tela 125. O processamento de som é feito pelo hardware de som 135. O Mouse 155 e o Teclado 145 transformam entradas de usuário em dados de entrada, sendo que as entradas de usuário geradas pelo usuário interagem com as informações de tela. Um Armazenamento Permanente 160 retém dados no estado desligado do computador, mantendo os dados acessíveis durante execução.

A figura 2 ilustra uma típica aplicação GUI que no caso é uma aplicação de Internet. Na verdade, há dois tipos de objetos GUI monitorados: Win 32 e Controle Web IE. A janela principal 205 contém todas estruturas GUI de subseções, (i.e. navegadores), assim como o conteúdo de páginas do navegador que pode incluir muitas páginas web (i.e. elementos html). As aplicações de navegador consistem de elementos Win32 (i.e. a janela principal 205, título da janela, controle de janela principal 215, itens de menu 225, tela de endereço URL 220, e ícones de navegação e operação 230). Elementos adicionais da janela principal provêem indicador de status 235 e indicador de estado de segurança 240. A própria página web principal, representa a página html de Aplicação Hospedeira transmitida pelo servidor e compreendida de várias interfaces funcionais para interação de usuário e recuperação de informações.

A figura 2 representa uma função simples calendário, onde há vários elementos que provêem visualizações e operações de filtragem incluindo: ícones e controles de botão para as seções organizacionais principais de aplicação 245, 250, Visualização de Projeto 255, Filtros de Usuário, Projetos e Categorias 265, Operadores de Adição 260 e Busca 275, Tela de Data 270, Tela de Calendário Gráfico

280, e um ícone de status de projeto 285. Este exemplo ilustra como a GUI é convertida (renderizada) com uma combinação de dois tipos de processo de aplicação GUI para prover dois tipos de alvo para sensores GUI Win32 e WebIE. A interface de usuário provê um fluxo critico de eventos GUI, onde muitos outros tipos de registradores de dados estatísticos, tempos de resposta, contadores de desempenho de computador, para Eventos de Método podem ser colocados em Contexto GUI para conectar diretamente uma experiência de usuário para precisamente analisar o desempenho de aplicação ou processo. A função de calendário é dada como exemplo de Aplicação Hospedeira. A invenção, contudo, não se limita a um tipo ou natureza particulares de Aplicação Hospedeira.

Um Alvo de Monitoramento é qualquer entidade ou processo organizado em tempo real em Aplicação Hospedeira cujas propriedades, Eventos ou comportamentos podem ser convertidos pela detecção em um Evento lógico para interpretação e análise pelo sistema gMg. Há uma ampla gama de Alvos de Monitoramento potenciais em um ambiente alvo que é dividido em métodos privados ou públicos API's no sistema operacional ou aplicação. Cada Alvo de Monitoramento contém seus próprios valores e tem Eventos detectáveis que podem ser usados para formar perfis de comportamento em áreas de interesse.

Referindo-se à figura 3, um Usuário 302 interage com a Aplicação Hospedeira GUI 304, que por sua vez controla os Componentes de Aplicação 306 atendidos pelo servidor. A GUI e camadas de componente, todas por sua vez, quando necessário, interagem com os Componentes do Sistema Operacional 310-326 para obter plena funcionalidade. Embora haja muitos aspectos para o sistema operacional representado aqui, há poucas funções criticas com respeito ao Kernel 312, rede 314, 322, 324, operações de núcleo, tal como CPU, Memória, Processos, Thread 326, Arquivo 320 e Componentes de Sistema Operacional diversos 310, 318. Os Componentes de Aplicação Hospedeira 306

interagem com qualquer ou todos estes subsistemas OS, para executar funcionalidades básicas, tal como comunicação de rede, tela gráfica, e etc..

Diferentes subsistemas padrão no ambiente operacional são disponibilizados por provedores de OS por interfaces de programação de aplicação pública (API) 308 e 328 a 336, integrados com o Sistema gMg em 356 e sensores gMg 338. Uma Aplicação API 308 pode prover Eventos e propriedades a partir de um amplo espectro de categorias de informação interna, tipicamente publicado para parceiros de aplicação para estender uma funcionalidade da aplicação com propósito de customização por clientes ou serviços. Um Sistema Operacional API 318 provê uma funcionalidade a respeito de todos aspectos de uma OS a partir de seus processos, threads, memória, alocação de memória etc.. O monitoramento de desempenho (performance) ("PerfMon") é especificamente formado API 316 publicado no Microsoft Windows, que provê contadores e dados a respeito de muitos aspectos de hardware e serviços, por exemplo utilização de CPU, memória virtual, etc.. O Controle de Navegador API 328 provê propriedades e Eventos a respeito de modelo de objeto de documento de aplicação de navegador (DOM) que é usado para suportar aplicações de monitoramento GUI da web. A contrapartida do controle de navegador é o interfaceamento da mensagem API GUI Win32 que monitora aplicações de cliente Win32 GUI. Há também outros GUI APIs 332, como para Java (Sun MicroSystems, Santa Clara, CA), VisualBasic, e.NET (ambos da Microsoft, Redmond WA) em aplicações de cliente. O sistema gMg pode ser usado em todas as categorias de interface dadas acima para criar diferentes fontes de dados de monitorar processos de coleta e análise.

O Sistema gMg realiza a geração e integração de fontes de dados com seus vários tipos de sensores 338 ligados por seus adaptadores às várias interfaces alvo monitoradas disponíveis (308 e 328 a 336). Estes sensores são iniciados e detectam eventos que são supridos para a

camada de Abstração gMg 348 que produz Eventos gMg 350. Estes Eventos resultantes são então supridos para o Interpretador gMg 352 para uma análise condicional em tempo real que cria uma outra camada de Eventos de rastreamento que são alimentados para o Servidor de Coleta e Análise 354. A Análise produz uma gama de dados a partir de alertas, correlações, para dirigir ações. Como parte do Sistema gMg, há um Sensor de Método gMg especializado (i.e., Monitoramento de Método gMg 340, 342, 346) que realiza um monitoramento geral das chamadas de método alvo em uma gama de componentes a partir da Aplicação Hospedeira para o Sistema Operacional. O Sensor de Método pode extrair Eventos e propriedades de componentes públicos e privados em condições e cenários adequados. Dado sua ampla gama de sensores extensíveis e adaptáveis, o Sistema gMg pode dispor de dados de muitos tipos de fontes para prover um monitoramento abrangente da Aplicação Hospedeira e processos de ambiente, todos em contexto GUI. Em ambiente servidor, no qual, em operação, não há interação direta de usuário através de interface GUI, todas as outras interfaces de monitoramento públicas e privadas se mantendo válidas para gerar fontes de dados de monitoramento gMg. Em um servidor (sem Usuário 302), não há camadas de execução GUI 303, 304, 327. No entanto, GUIs para ferramentas de administração de servidor podem estar presentes e ser monitoradas.

Referindo-se à figura 4, onde são ilustrados os componentes gMg abertos em um típico ambiente empresarial, um Usuário 402 pode acessar uma ou mais Aplicações Hospedeiras 404, 406, 408 ou uma ou mais instâncias de mesma aplicação supridas pela Aplicação Hospedeira de Servidor 410.

As Ferramentas de Desenvolvimento de Sistema gMg 412 podem criar e agrupar Soluções gMg 426 preparadas para serem abertas no ambiente de computação de usuário. Ferramentas de desenvolvimento incluem ferramentas de inspeção de Método e GUI, compilador, debugger,

ferramentas de Máquina de Estado analíticas de servidor, administrativas, e abertura de utilidades de empacotamento e administrativos, e um ambiente de desenvolvimento integrado. Os gMg executáveis incluem um único agente gMg.Exe 414 e um ou mais Engines Dlls 416, 418, 419, 420. A coleta de agente gMg.exe, um ou mais Engine.DLL, e uma solução gMg compreende uma instância do sistema gMg 436. A figura 4 ilustra múltiplos engines monitorando Aplicação Hospedeira 406 e Engine.DLL 419. O mecanismo de Memória Compartilhada 442 tem uma função de coordenar e compartilhar recursos nos vários Engines. O Sistema gMg dinamicamente rastreia múltiplos processos em uma Aplicação Hospedeira, se necessário, e os Sensores 438, 439, 440 detectam chamadas de método ou Eventos GUI nas Aplicações Hospedeiras para formar Contextos de Usuário para os dados colhidos. O contexto de usuário incluindo o contexto de sistema (por exemplo, chamada de método, retorno de método, e mensagens) é capturado para criar uma "fotografia" (snapshot) do espectro ambiental da Aplicação Hospedeira e Sistema.

O agente gMg.exe e os engines são abertos no ambiente alvo, quer um computador de mesa, um servidor, (quer em GUI ou não-GUI) ou um servidor terminal (um servidor que executa componentes de cliente em um servidor terminal que apresenta na tela dados rasterizados). Usado durante uma fase de desenvolvimento, o Debug opcional 422, 424 se destina a debugger, em tempo real, usado em conexão com ferramentas de desenvolvimento para prover um ponto de descontinuidade, telas variáveis, strings, e outros aspectos típicos de ferramenta de debug. Soluções gMg 426, 428, 432, 434, pequenas e facilmente dinamicamente atualizadas consistem de scripts compilados que definem e descrevem os Alvos de Monitoramento separados ou sobrepostos e executam operações de rastreamento. Estas soluções são executadas por diferentes instâncias engine.DLL e podem rastrear Alvos de Monitoramento separados ou sobrepostos, se requerido.

Através da detecção de Eventos de Método ou GUI, são criadas mensagens de rastreamento que a rastrear são enviadas a um Servidor e Banco de Dados de Rastreamento 446 pelo Módulo de Comunicação 444. O Servidor de Coleta faz uma análise em tempo real, ou próxima de um tempo real, ou para mais tarde dependendo dos requisitos disponíveis e recursos de computação. O módulo de Processamento e Análise 448 produz Alertas 450, Correlação com fontes de dados externos (por exemplo, estatísticas de monitoramento de estágios finais 452) ou Reportes 454. Os Reportes 454 podem ser procedentes de uma ampla variedade de perspectivas com dados processados agregados revelando grupos ou comportamentos individuais detalhados a respeito da interação com os processos de aplicação em Contexto de usuário GUI real. Em funções empresariais diferentes 458 a partir de suporte técnico, treinamento, suporte de infra-estrutura etc. as informações ou alertas podem ser acessadas para serem transmitidos para prover melhoramentos e ações corretivas 456, ou suportar usuários finais.

Referindo-se à figura 5, onde um Usuário 503 interage com uma ou mais Aplicações Hospedeiras concorrentes em serviços de terminal ou computador de mesa com um Site de Aplicação Hospedeira 506, componentes relacionados 512, e uma ou mais instâncias 545 da Apresentação de Aplicação Hospedeira GUI 509. Elementos da Aplicação Hospedeira podem ser localizados na camada de apresentação e permitir que o usuário solicite operações e receba respostas de aplicação. A Aplicação Hospedeira pode executar mais que uma instância concorrentemente, ou concorrentemente executar multi-instâncias de conjuntos GUI, que são subconjuntos da Aplicação Hospedeira.

O GUI pode ser organizado na memória como uma hierarquia e ter muitos subconjuntos de objetos GUI que replicam durante execução dependendo da ação de usuário e da lógica de organização da aplicação que controla como são geradas as estruturas de objeto GUI. Ao lado da

camada de apresentação, uma aplicação também contém muitos componentes ou DLLs 512 que executam funcionalidade (i.e., comunicação com um servidor, cálculo, recuperação de propriedades de ambiente, interação com o Sistema Operacional do computador, etc.). A biblioteca de Componentes que contém os métodos para executar uma funcionalidade e a detecção gMg dos Eventos de Método provê acesso a propriedades, entrada, saída, Eventos e outras informações a respeito da função dos componentes em suporte à Aplicação Hospedeira. Eventos de Método são detectados a partir de componentes de biblioteca que provêem GUI ou outras funcionalidades de Aplicações Hospedeiras subjacentes.

O sistema gMg utiliza propriedades de objeto GUI 518 e informações de estado de objeto para suportar a análise. Um jogo de pares valor e nome, e sua relação com outros objetos refletem o estado do objeto em um dado instante. O valor da propriedade pode ser recuperado programaticamente em qualquer instante usando busca (polling). O objeto também pode ter propriedades customizadas associadas, definidas e controladas pelo sensor pelo script. Informações adicionais de estado podem resultar da combinação de estados e eventos anteriores. O Engine Sensor gMg 548 em reação com o fluxo de processo da Aplicação Hospedeira pode gerar um conjunto de Eventos em tempo real 515, que são Eventos produzidos por Usuário, Aplicação, Eventos OS, e valores particulares de propriedades de Método e objeto GUI. Os Eventos refletem condições que são satisfeitas em um dado momento, ou um estado corrente da aplicação (i.e., um espectro ambiental). Estes Eventos se encontram em diversas categorias: eventos de usuário gerados com um dispositivo de entrada, tal como mouse ou teclado; eventos indiretamente produzidos por uma ação de usuário; eventos de aplicação gerados por uma aplicação de modo autônomo (i.e. atualizando (refreshing) uma janela que sofreu atualização), uma ação de registro, ou rede; e

Eventos de Chamada e Retorno de Método que são setup para monitorar funções componentes específicas. Eventos posteriores podem ter propriedades que proporcionem detalhes ou qualifiquem adicionalmente os eventos. A partir desta entrada de Eventos de baixo nível, opcionalmente Eventos Lógicos 521 podem ser produzidos usando um processamento lógico condicional para determinar estados via Máquinas de Estado. Ambos, eventos Lógicos e não-processados quando alcançam o módulo de Comunicação 524 são preparados para transmitir para um servidor remoto de coleta 557. Este módulo controla buffering e compressão ou segurança antes da transmissão. A acumulação também provê um mecanismo para armazenar dados temporariamente no evento de coleta, ou rede, interrupções ou em ambientes não-conectados.

Uma troca de informações é provida entre o cliente 548 e o servidor 557. O módulo de comunicação de cliente fornece valores de evento lógico ou propriedades de objeto GUI selecionadas e também recebe comandos de controle do servidor. No estágio de coleta e análise, os dados são supridos para a Instância de Servidor de coleta 557 na forma de mensagens de rastreamento que podem ser processadas ou armazenadas em banco de dados. O fluxo de Eventos é desmultiplexado em contextos lógicos separados de usuário determinados dinamicamente por parâmetros de análise.

Dependendo se os dados chegam como propriedade de objeto GUI no Evento de nível mais baixo ou Evento de nível lógico, determina-se como os dados são roteados 554. Se chegarem como multi-eventos de nível mais baixo, então o processamento da Máquina de Estado produz Eventos Lógicos 530 e contextos. No entanto, se o dados chegarem como Eventos Lógicos de máquina de estado previamente processados 521, então novos contextos poderão ser produzidos por um subseqüente processamento da Máquina de Estado 527. Como saída, quer através de Detecção Genérica ou Específica, são formados conjuntos de objeto GUI,

assim como Conjuntos de Evento Genérico ou Específico 533. Uma análise subseqüente termina padrões identificadores de estados e valores pré-determinados 536 para produzir um conjunto completo de dados processados. Este dados completamente processados são supridos para o estágio de Análise e Reporte 539 para processamento agregado, direção, reporte para apresentação gráfica, e uso em consoles de reporte gMg.

Para um amplo espectro de usuários, os consoles são usados para propósitos tais como controle de programa, reparos de programa, debugging, otimização, etc.. Alternativamente, os eventos processados podem ser diretamente roteados para o componente de Ação do sistema 542 que inclui display e comunicação de condições limite, que demandam uma prioridade mais alta, tais como erros ou tempos de resposta mais lentas. Ações podem incluir comunicação com sistemas externos 560, que provem funções, tais como emissão de protocolos de suporte, reparos automáticos, treinamento, e outras ações, que ao cabo afetem o usuário.

Referindo-se à figura 6, o Sistema gMg usa diversas categorias importantes de fontes de Eventos. No Evento 610, pode ser encontrado Windows32, Eventos de sistema operacional recuperados via API, funções Hook e "callbacks". Estes Eventos provêem acesso a muitas aplicações diferentes de mensagem, propriedades e funções que provêem acesso a muitos diferentes aspectos de processos de interação entre o usuário, a aplicação, e o sistema operacional. Outra fonte de Eventos pode ser provida pelo navegador (Eventos de Internet Explorer) que exemplificam como uma aplicação recipiente expõe Eventos e propriedades a respeito de modelos e tipos de dados, tal como Modelo de Objeto Documentado (DOM) para processamento externo recuperado com listener e objetos COM disponíveis. Um navegador web, tal como Internet Explorer, exemplifica a natureza dinâmica de aplicação de Internet moderna de GUI que mostra estruturas complexas

de janela multi-instanciadas. Dentro do sistema gMg, Listeners COM são habilitados e usam funções callback (IEEvents.cpp) para detectar Eventos IE. Classes são registradas com Internet Explorer, e um método no qual uma classe é chamada sempre que ocorrer um Evento com a recuperação de propriedades de objeto IE é provida via interfaces COM. Um outro exemplo do tipo Evento é o Evento de método chamar e retornar que provê um amplo espectro de Eventos de Sensor criados pelos componentes de monitoramento de Sensor gMg.

Todos estes Eventos são transmitidos para o script interpretado gMg 625. Os Eventos, antes que cheguem no interpretador, são serializados pelo mecanismo de bloqueio 620. Ademais, o bloqueio também é diferenciado para N diferentes instâncias do Engine e/ou Script 625, 630, 635. Uma vez analisado o script e serializado o Evento, a função executa uma operação poll intermitente (com aplicação Scan 640) no GUI da Aplicação Hospedeira 605. Concorrentemente, há eventos timer de sistema operacional que chegam via procedimento callback timer 615, que programa escaneamento da Aplicação Hospedeira GUI 640 em taxas configuráveis e ajustáveis. Durante o processo, a Árvore de Objeto GUI gMg 660 - estruturas de gravação de ramificação 655, 660, 665 - é atualizada, deletada, ou copiada em snapshot da vista corrente do sistema do estado de aplicação de seu GUI. O script gMg também ajusta outros timers 645, que são manuseados, mas em um nível mais baixo de prioridade para comandos "WhenObject" gMg. O processamento "WhenObjet" 650 é um processo que cria Eventos Cria/ Destrói gMg para objetos GUI que requeiram sincronização em 640, onde não há Eventos produzidos pela Aplicação Hospedeira e sistema operacional. "WhenObject" provê um timer simulado para buscar conjuntos específicos de objetos GUI.

O conjunto gMg de diferentes processos de monitoramento provê uma ampla gama de categorias de fonte através de um ambiente de aplicação e sistema operacional. A taxa de

interação pode ser controlada para operações somente-leitura (read-only) que apropriadamente recupera, para extrair detalhes e formar registros para mensagens de rastreamento e subseqüente análise.

A rastrear são dados mecanismos de suporte de GUI, Multi-Instanciamento de processo, processamento de script durante execução que são instalados durante o processo de iniciação. O componente gMg inclui Árvore de Objeto GUI, carregamento e organização de múltiplos scripts, componentes de script que suportam componentes de manuseio Multi-Instância de Aplicação Hospedeira (i.e., Variáveis de Objeto, e comando "WhenObject"), rastreado pela provisão de Contextos, Hooks, e finalmente a detecção Genérica de conjuntos de Objetos GUI.

O Sistema gMg pode ser configurado em um número de maneiras diferentes para sua abertura. Na configuração cliente-servidor, um Agente local seus engines e soluções podem ser instalados em um computador local para detectar Eventos. Em um ambiente web, um código de Solução gMg pode ser injetado instalado em um servidor, e o código de Solução gMg suprido dinamicamente ao computador de cliente, onde o Agente gMg e o engine já foram instalados. Embora haja vantagens para cada abertura, ambas detectarão Eventos com base em uma estrutura comum e serão capazes de obter propriedades específicas, quando necessário. As configurações de abertura gMg, no entanto, não se limitam a isto, e especificamente outras configurações poderão ser usadas dentro do escopo desta, desde que o sistema gMg seja operável.

Uma vez o Sistema gMg aberto, na ativação de início, a linha de comando é parsed quanto a opções, tal como script de carregamento, Stop, etc. e estabelece comunicação com a fila de mensagem. As opções de executar no arquivo ini., que podem incluir scripts ou diretórios de scripts, são inspecionadas e enviadas à rotina do string de comando parse principal. Uma lista de scripts no arquivo.ini pode ser executada na iniciação e injetada

em múltiplos processos. Um chamada para a função ChangeHookTidl, faz o gMg.exe auto-injetar o engine através de sua própria janela, e executar seu próprio interpretador (engine gMg), como se monitorasse outra aplicação. Alternativamente, a janela associada gMg também pode servir de comunicação entre engines, junto com o protocolo gMg para outros engines gMg associados ao mesmo Agente.exe, injetado em outras aplicações.

Uma trajetória de arquivo é usada para formar e registrar uma mensagem de janela especial, que registra um processamento gMg no sistema operacional. O nome de toda trajetória de arquivo do módulo gMg.exe é usado e será o identificador único da mensagem de janela para distinguir das demais instâncias, cópias e versões de agentes e engines executadas em um mesmo instante, assim permitindo que instâncias únicas dinâmicas sejam executadas. A capacidade de distinguir instâncias permite separar gMg.exe's, instalados em múltiplos diretórios, para estabelecer suas próprias estruturas durante execução para uma apropriada operação simultânea. Se estiverem sendo executadas mais que duas Aplicações Hospedeiras e, conseqüentemente, mais que uma instância de engine gMg, e ambos engines fizerem a mesma solicitação (tal como, listas Export), pode haver conflitos no Diretório Primário, tal como um arquivo que se recuse a abrir ou um arquivo que se sobreponha a outro. Preferivelmente, para resolver isto, os nomes de arquivo são ligados às instâncias. Isto se torna parte da base para um suporte Multi-Instância dinâmica. Nesta condição, não pode haver dois arquivos de mesmo nome no mesmo lugar.

Na iniciação (startup), o objetivo é setup os Sensores de Monitoramento gMg usando hooks, que são necessários para interceptar as mensagens de janela antes de a aplicação recebê-las. Os Hooks estão no nível de thread, onde há um hook *per* thread. Monitorar um thread para mensagens de janela faz um gMg DLL ser injetado no espaço de processo

associado, onde há um gMg DLL injetado (via CALLWNDPROC, hook GETMESSAGE com SetWindowsHookEx) *per* processo. O script gMg engancha (hooks) outros processos, se necessário, e injeta o DLL nos espaços de processo. Dado o handler, o objeto thread do objeto monitorado é recuperado, e produz um objeto thread para encontrar a instância (i.e., o thread-id para enganchar ou desenganchar o thread-id). Para adicionalmente controlar o enganchamento do thread, há múltiplas versões da função ChangeHookTid bloqueado e desbloqueado. A versão chaveada é usada quando um bloqueio precisa ser feito. Depois de inicializada a memória compartilhada, o código de iniciação também chama loadDBGDII para carregar o debugging DLL se este estiver presente (opcionalmente) em um diretório designado no mesmo espaço de processo. Como em geral há múltiplos thread em uma única aplicação, o gMglock implementa um processo bloqueado quando se usa o interpretador de script, e faz todos eventos terem um único thread. Se dois threads em uma aplicação tentam gerar um Evento que interessa ao gMg, ou se houver N eventos de diferentes fontes, o engine força uma serialização de eventos, que se refere a eventos processados em seqüência. O código de execução opcode de interpretador é primeiro bloqueado por gMglock e então é chamada DogMgEvent(t) para processar cada evento.

Para a função LoadProgram chamada por gMg.exe são transmitidos thread-id, nome de arquivo de script, e opção (carregar-descarregar). O Sistema gMg examina os caracteres e converte o nome de arquivo a unicode. Se o thread-id específico chama ChangeHook() com asterisco "*", provê-se uma operação para soltar todos os ganchos (hooks). Em virtude de o sistema suportar Multi-Instâncias de engines, múltiplos scripts poderão ser executados, no entanto, quando forem usados mais que um script, o sistema controla tantos scripts quantos poderão ser iniciados (não sendo permitido iniciar scripts com um asterisco curinga). Se o script não tiver curinga,

o setup de enganchar avança, e engancha o thread que passa por ele. A função ChangeScript então envia uma mensagem ao thread que provê o nome de arquivo, seu próprio thread-id junto com possíveis opções: carregar/descarregar o script; carregar o programa diretamente; inspecionar a lista de todos os programas sendo executados; carregar um único script com a opção descarregar; ou recarregar um script.

Executando o script, a função D0_gMgEvent() passa um objeto em um evento, e se o bloqueio estiver estabelecido, conta o número de eventos processados. Árvore de Objeto GUI (figuras 7 e 8) deve ser construída na iniciação antes de ser processado o primeiro evento pelo script. Preferivelmente, cada script é processado, mas se for estabelecido um flag descarregar, é realizada a operação de descarregar script, que inclui enviar uma mensagem de rastreamento de fim de script. No entanto, se o script ainda não tiver sido incializado, uma mensagem de rastreamento de inicialização (objeto zero Evento nulo) é primeiro enviada para o script inicializá-lo junto com Eventos Existe para todos objetos existentes. Esta ação solicita o script que agora começa a ouvir os eventos. Este loop de iniciação começa para cada script em sucessão para quer remover um script, inicializar, ou passar seus eventos.

Um mecanismo para sincronizar sensores e engines é um bloco centralizado de memória compartilhada contendo barras de janelas de todos os módulos executáveis principais gMg. O código gMg verifica terminação de executáveis, ou se há outra versão sendo executada, e se uma janela de executável deixa de existir, um flag é dado. Uma verificação de status é feita e se outro flag for emitido, uma caixa de mensagem será opcionalmente mostrada para dar o status e indicar se há outro gMg.exe sendo executado oculto. Se outro executável for detectado, o gMg usa a mensagem WM_COPY para passar a linha de comando inteira como bloco de dados a todas

outras instâncias de engine, e então sair. No entanto, se nenhuma outra instância estiver sendo executada, então o processamento avança para criar a janela principal. A janela principal da gMg.exe, nunca mostrada, é usada para criar mensagens e permitir que o gMg.exe se auto-monitore como uma aplicação. Uma barra na memória compartilhada é ajustada para apontar a própria instância do gMg.exe.

Durante a fase de terminação, o manuseio do gMg Multi-Instância ainda se faz necessário. No fechamento, quando DLLProcessDETACH é chamado, um flag no processo impede que as tarefas correntemente em execução façam um trabalho adicional, uma vez que outras operações ainda estão sendo executadas sincronicamente. A função Cleanthread() que chama CleanIEthread() em aplicações web, comanda desconexão ou quaisquer callbacks ou releases ou objetos COM deixados no IE podem ser descartados. DLLProcessDETTACH (e sua contraparte DLLProcessATTACH) é recursiva, de modo a suportar ambientes dinâmicos de Aplicação Hospedeira e múltiplos instanciamentos.

Há pelo menos um script gMg *per* processo de Aplicação Hospedeira. Alternativamente, múltiplos scripts gMg podem monitorar um único processo. Scripts são projetados para operar todos thread de um processo, mas um único script pode monitorar qualquer número de threads, tais como aqueles de aplicações de uma janela-*per*-thread, tal como tipicamente encontrado em aplicações web. O script é processado muito rapidamente e a recuperação de atributos é pesadamente cacheada para prover um melhor desempenho. O cacheamento é feito sempre que houver uma dependência de propriedades de busca, ou na recuperação de atributos com segurança no mesmo thread.

Itens potenciais ocorrem durante um acesso thread transverso ou quando um thread é acessado a partir de um objeto COM, e cujo comportamento de ordenação é desconhecido particularmente quando a execução estiver

sendo feita em um ambiente IE. No entanto, itens a respeito a um acesso a thread transverso (especialmente, em ambientes IE) serão menos significativos, porque um Evento gMg gerado por um thread específico, contempla apenas os objetos relacionados àquele thread, e raramente reage a objetos em diferentes threads. Quando se manipulam objetos de diferentes threads, o processamento gMg tem que terminar o processamento do thread corrente, antes de contemplar outros objetos do thread. As colisões são evitadas serializando o processamento do Evento e impedindo ambigüidades, tal como quando dois threads compartilham e modificam uma variável comum. O processamento em gMg é guiado pelos princípios de problemas de concorrência comum, como sabido na técnica. O princípio de serialização se aplica a ambos ambientes de Componente, tal como aplicações em ambos ambientes COM ou aplicações não-COM (componente). Em ambientes COM, os métodos de interface podem ser usados em múltiplos threads. A despeito do modelo de thread (i.e. thread seguro, thread único, etc.) o monitoramento de thread é serializado com gMglock. Um Evento de processamento IE pode ser feito *per* thread, mas o script gMg precisa ser determinístico e portanto serializado *per* Evento. Um processamento *per*-thread é importante para Eventos que são relacionados, mas relacionamentos são determinados pelo conteúdo do script gMg. O Sistema gMg usa um único threading para processamento de Evento e estende isto para qualquer Evento. Todos parâmetros transmitidos a Invoke têm um apontador, que será usado mais tarde para recuperar as propriedades de Evento. Se um script pedir as propriedades de Evento, ele usa um dos apontadores globais para os parâmetros de Evento e recupera os detalhes do Evento. Este mecanismo mantém a recuperação centralizada, e a impede de se distribuir a muitas localizações que pode causar uma excessiva quantidade de operações mover/mostrar stack. A serialização de Eventos e script deve ser eficiente. Por exemplo, se uma

aplicação GUI for processada por um único thread, será facilmente manipulada com Sensoreamento gMg, mas se uma aplicação tiver muitas janelas, e cada janela seu próprio thread, ocorrerão colisões de thread. Preferivelmente, este problema potencial é evitado tendo thread A processando um evento enquanto o thread B aguarda no gMgLock. Enquanto as mensagens são processadas pelo thread A, o thread B aguarda no gMglock para liberação.

Um sistema gMg usa um mecanismo de bloqueio para estabelecer a serialização dos Eventos gerados. BeginLock- o bloqueio Macro - realiza uma troca de interlock do parâmetro, e se voltar a 0 - o valor anterior é 0 - então foi provido o bloqueio. O thread-id é estabelecido e a localização do chamador é determinada, encaminhando para onde o bloqueio foi usado e o thread-id que estava fazendo o bloqueio. Enquanto BeginLock registra o thread, outra macro gTid usa o thread-id que está armazenado no gMglock. Uma vez estabelecido o gMglock, o gTid fica muito mais rápido, porque só precisa buscar o thread-id armazenado no bloqueio.

Para debugging deadlocks, o beginlockD pode manipular com ações específicas, se houver um bloqueio recursivo. BeginlockD também é capaz de manipular com ações específicas para bloqueios recursivos, onde, em certos lugares, se permite um nível de recursão com diferentes ações realizadas para recursão, em contrapartida a apenas executar uma ausência ou inadvertidamente provocar uma falha. O beginlockD realiza uma troca de interlock e armazena o thread-id. O mecanismo de bloqueio do Sistema gMg é equivalente a uma seção critica, mas trabalha nos processos, quando o bloqueio estiver em uma memória compartilhada, e que será muito rápido quando não houver colisões. Mas, se houver colisões, preferivelmente o processamento gMg força um chamada fazer um Quicklock, que testa para verificar se este é o thread-id corrente. A contagem de recursão sendo mantida por razões estatísticas e retorna a -1, caso contrário é colocado um

flag na recursão e a contagem de colisão incrementada. Um loop controlado então é executado, quando o loop "dorme" 1,0 milisegundo, e então "acorda" com uma ausência de 10 segundos. Uma verificação é feita de modo que se for provido um bloqueio no gMgLock global, há uma contagem de loop (i.e. uma contagem de script) das instruções sendo executadas. Se o script gMg for grande, onde se buscam uma quantidade de parâmetros e todas as propriedades de cada objeto na árvore, então as ausências se estendem de um valor *default*. Se estiver sendo aguardado o gMgLock e o código de script estiver sendo executado, o QuickLock será chamado somente se houver colisão. Ocasionalmente, há um bloqueio recursivo, (função BegLockD), e uma mensagem é emitida para diagnóstico. O QuickLock volta a 0 se houver uma ausência completa, quando for feita uma contagem do número de ausências no bloqueio. A função volta a 1, se estabelecido com sucesso o bloqueio, e -1 se já feito o bloqueio com seu próprio thread. Deve ser notado que o thread-id (tid) pode ser transmitido como parâmetro para o QuickLock, de modo que este não tenha que ser recuperado novamente. O código pode automaticamente estender a mensagem de ausência gMgLock, se algum outro processo tiver o thread bloqueado, com o interpretador ainda executando o código de script. Para diagnóstico, a estrutura QuickLock contém informações a respeito do bloqueio, thread-id, contagem de colisões, ausência, contagem de recursão, número de linha do último bloqueio, e informações de arquivo de módulo.

O gMgLock pode controlar o diagnóstico debugging provido pelo DbgDLL que provê um diagnóstico para expressões em linha, assim como para blocos de código. O DbgDLL contém um diagnóstico de função que passa o pedido de enumeração de diagnóstico emitido para análise durante execução (run time). Para suporte de diagnóstico, se o gMgLock não se estabelecer com sucesso, ele atuará como mecanismo a prova de falha, e o processo pára a função de diagnóstico e a coloca em baixa prioridade.

Para modelar Eventos da Aplicação Hospedeira real, o Sistema gMg pode incluir três árvores principais: uma árvore para objetos GUI, tal como elementos html; uma árvore para processo e objetos thread; e uma árvore para rastrear métodos e módulos de componente. O Sistema gMg também inclui uma lista de objetos timer.

A figura 7 representa a Árvore de Objeto GUI 715, que é o fundamento para prover o Contexto GUI para todos os dados de rastreamento coletados. Árvore de Objeto GUI 715 pode ser compartilhado por todos scripts de execução 775 e mantém os estados para cada script. Há três conjuntos de objetos GUI na figura 7, i.e. conjunto P 725, conjunto Q 720, 735, 740 e conjunto R 730. A rastrear se descreve o processo de atualização das estruturas de Árvore de Objeto GUI 715 com estruturas dinâmicas extraídas de GUI 705 da Aplicação Hospedeira.

A função Árvore de Atualização modela a GUI da Aplicação Hospedeira ou de outros processos e alvos monitorados. Para prover Eventos ao engine de script gMg, faz-se necessário uma detecção precisa de objetos dinamicamente alterados no ambiente. Para isto, a função Árvore de Atualização leva todas atualizações primárias a sua estrutura interna para monitorar apropriadamente as diferenças entre novos e velhos conjuntos de objetos em um programa controlado.

Com respeito à programação, a função UpdateTree tem um número de mecanismos para controlar a taxa de sincronização de objeto com a Aplicação Hospedeira. A Função UpdateTree tem seu próprio timer para evitar um sobre-processamento, qual timer atua em intervalos, em função do tempo normalmente requerido para escanear uma árvore. A função UpdateTree também é chamada durante a detecção de Evento, se o Evento timer não for disparado porque a aplicação está ocupada, e também quaisquer Eventos que gMg estiver interceptando pode fazer o código UpdateTree ainda executar. A prioridade de processamento é elevada e faz a UpdateTree escanear, atualizar objetos,

e capturar info, mesmo se a aplicação parecer estar bloqueada. UpdateTree também deixa de lado um thread para processar todas as janelas dos outros threads durante aquele tempo. Por exemplo, se três threads estiverem executando com uma taxa de busca para cada thread de duas vezes per segundo, somente um thread será incumbido de verificar as janelas dos outros threads para melhorar o desempenho. A filtragem de objeto é feita em nível thread, e as taxas de busca são adicionalmente ajustáveis para melhorar o desempenho. Adicionalmente, há um número de outras opções, tal como limpar Object Tree ou processar todos threads, em vez de limitar o processamento apenas ao thread corrente.

Para manipular a natureza dinâmica da Aplicação Hospedeira, o sistema gMg inclui um mecanismo de rastreamento, que rastreia o status de mudança de objetos GUI que muda constantemente, sendo ora deletados ora criados. Referindo-se à figura 7, para acomodar estas condições de mudança, se mantém uma lista dos objetos a serem destruídos 760. O processamento gMg destrói as mensagens e gera Eventos 770 que são enviados ao interpretador de Script 745. A Slist 760 é uma estrutura que contém uma lista de objetos velhos a medida que os objetos são tirados da Lista Velha 750, se ainda existirem, e então adicionados no fim da Lista Nova 755. O conjunto anterior de objetos permanece na Lista Velha, mas os objetos novos que não existiam antes são adicionados à Nova Lista 755. Um apontador é colocado no conjunto de janelas existentes no topo da árvore 720, o status do conjunto é alterado para "velho" esvaziado, e uma Nova Lista é formada a partir deste ponto.

A função EnumWindowsProc realiza a atualização escaneando cada janela, encontra a Lista Velha e coloca na Lista Nova conforme necessário, a medida que a aplicação GUI é escaneada, ou se os itens deixarem de existir, coloca o item janela na Lista Destrói 760. O que quer que tenha sido deixado na Lista Velha então é processado. Uma

notificação é recebida de qualquer janela destruída deixada na Lista Velha. O processo de atualização ocorre no EnumWindowsProc 710. A função callback EnumWindowsProc é chamada pelo win32 API para cada janela de topo 780. Lparam é um apontador para estrutura SList. O processo thread de janela id também é recuperado. A função obtém o id de processo thread da janela, e corre a Lista Velha em busca da janela.

Objetos são verificados para determinar se os mesmos estão na Lista Velha. A Lista Velha, que inclui um conjunto de apontadores para estrutura Object Tree, é recuperada, e os objetos ainda existentes são marcados para serem membros da Lista Nova, conforme necessário. Quando feitos, objetos de thread ou processo são processados para janela. Se uma nova janela for detectada pertencente a um certo thread, o sistema cria ambos objetos de thread e processo para o thread em questão. Um flag visitado é usado como indicador de status para marcar se um objeto está listado (como parte da lista) em um dado processo em uma janela ou thread. O flag visitado é verificado e usado para emitir para o script que os objetos foram terminados via Evento Destrói 770.

O código opera eficientemente, onde não houver mudanças na estrutura de árvore, que é a condição usual. Se não houver mudanças, o código desvincula os objetos, atualiza os flags para os seis threads, e processa e vincula a janela à Lista Nova (i.e., a janela passa da Lista Velha para a Lista Nova). Usando o thread, as janelas ficam no mesmo nível, sem nenhuma janela abaixo do nível corrente ora examinado. Este mecanismo desconsidera qualquer janela de nível superior que pertença a outro thread, e somente processa um thread por vez. Preferivelmente, se o processamento não for restrito, o processamento de um thread deve cair a zero, o que denota que o processo foi incumbido de processar todos seus threads.

Referindo-se à figura 8, onde são ilustradas operações da

Árvore de Objeto 831 a rastrear da classe da janela 806, que usa a mesma interface 808 enquanto o script 812 é buscado. A classe não precisa ser recuperada da aplicação corrente 804 se já estiver no cachê gMg 810. Se o thread corrente corresponder à classe IE (do windows) que procede do thread específico (e thread corrente) pertencente aos membros de Árvore de Objeto. Os objetos na árvore 832 contêm threads ids associados, conseqüentemente cada ramificação somente poderá ser atendida pela janela cujo thread corresponde ao thread de objeto 844.

A função NotifyDestroyTree constrói uma lista Destroy 816 e descarta a Lista Velha 820. Se o processamento não produzir uma chamada para NotifyDestroyTree, a árvore velha anterior na Lista Velha permanece intacta. Mas se os objetos de ramificação correspondem ao thread corrente, o processamento prossegue. Há agora uma Lista Nova 822 limpa, e todas as janelas "filho" são enumeradas no próximo nível. O processamento avança recursivamente para capturar a estrutura de janela corrente. O elemento de figura 832 representa o Object Tree que consiste dos conjuntos de objeto P 836 e 834, Q 838, 842, 848, 850 e 852, e R 840, antes da atualização. Referindo-se à árvore atualizada 854, se houver uma nova janela IE ou objetos GUI se chama GetIETree para recuperar todos os elementos HTML "filho" na hierarquia 866, 870, rastreando uma chamada para NotifyDestroyTree() para destruir todos objetos velhos deixados na árvore. Este é o processamento GetIETree para recuperar os elementos "filho" na hierarquia 866, 870 rastreada de uma chamada para NotifyDestroyTree() para destruir os objetos antigos da árvore. Este é o processamento básico que enumera a árvore para todas janelas, incluindo as janelas ou objetos GUI que ainda existem 856, 858, 862, 868, 872 da Lista Velha para a Lista Nova. Uma vez os objetos marcados e descartados na Lista Velha, tudo que foi criado recentemente 866, 870 agora será automaticamente

conectado a uma lista global. Tudo que é deixado neste lugar está chamando a variante (i.e. chama a função NotifyDestroyEvents 818), enquanto parse a árvore, emite um Evento Destroy 828 ao script 812 quando detecta que não existem mais objetos 864. A estrutura de árvore anterior ainda se mantém intacta, de modo que os Eventos Destroy sejam enviados no contexto da árvore antiga para manter o contexto corrente do script gMg para atualizar de modo ordenado os objetos deletados ou mantidos.

A função SynchTree 826 processa o conjunto de janelas real que na verdade é a nova árvore. A estrutura DWindows 824 contém a árvore velha que é a árvore usada pelo Sistema gMg. Esta seção de código SynchTree formou uma nova árvore fora dos mesmos objetos sob a mesma janela que representa todos novos objetos e notifica ao gMg todos os objetos destruídos no contexto da árvore velha (intacta para uso posterior pelo script). Uma chamada para SyncTree é feita para copiar recursivamente os apontadores reais para DWindows e resulta na transferência da árvore corrente real para a árvore gMg. Neste ponto, o script gMg 812 agora tem acesso à nova árvore, e a árvore velha é descartada. Em seguida, os objetos destruídos são descartados da árvore 864. Agora há uma lista global de todos objetos novos criados. A emissão de um Eventos Cria 830 para o script gMg notifica ao script todos objetos novos 866, 870 que passam a existir. Simultaneamente, se houver um objeto documento HTLM no DOM descoberto, a função IEEvent será estabelecida como uma assinatura.

Na completação do processo de atualização de árvore, a lista temporária usada para armazenar todos os novos objetos é desvinculada e limpa. O novo apontador de fim de lista de objeto então é recolocado na frente da lista. A lista é formada na ordem em que os objetos são descobertos a medida que são criados, e não em ordem inversa. A função UpdateTree escaneia a aplicação, e gera Eventos Cria/Destrói. O Evento Destrói é emitido primeiro

para os objetos descartados, e o Evento Cria é emitido para objetos novos que forem detectados.

Em um conjunto de engines gMg pertencente a um único gMg.exe (Agente), a comunicação entre diferentes instâncias dos engines gMg pode ser estabelecida com a mensagem gMg da janela de OS. Esta mensagem é uma mensagem única da janela gMg criada usando o nome de trajetória de diretório completo (i.e., gMg.exe). O nome de diretório completo compreende o identificador da mensagem gMg armazenada na área de memória compartilhada para ser acessado pelos engines em operação. A mensagem de janela corrente é registrada no OS, porque se requer que a aplicação tenha uma mensagem de janela única no sistema de computador. Aplicações, tais como Agente gMg e seus engines, chamam uma função API OS, passando um string de identificação, a partir do qual o OS busca em uma tabela interna o string identificador que, se presente, retorna o identificador já designado. A primeira vez que a mensagem é enviada por uma aplicação, se adiciona à tabela interna e se designa uma mensagem de janela globalmente única para aquela aplicação executável no OS para aquela aplicação executável no OS, que em virtude de ser volátil, o identificador se perde se o computador for reiniciado. Por exemplo, com respeito à comunicação interprograma, se 2 programas tiverem mensagens de janela registradas OS, um único string identifica cada aplicação de programa. O OS retorna um identificador único para cada programa. No instante do registro, a informação de mensagem de janela é armazenada na memória compartilhada de gMg a ser recuperada por todos processos alvo injetados pelos sensores gMg.

O diretório primário contém todos componentes de sistema Agente gMg incluindo o executável principal (i.e., gMg.Exe), DLLS, arquivos .ini, e outros componentes. O_nome de trajetória completo do Diretório Primário desempenha uma função nos agentes e engines de Multi-

Instanciamento gMg. A trajetória e o diretório contendo o arquivo executável de Agente são usados para chavear instâncias gMg internas contendo conjuntos de componente. Se houver múltiplos agentes em um único computador, cada agente terá seu próprio diretório. Ademais, todas instâncias durante execução dos engines gMg são chaveadas para o diretório pelo nome de arquivo completo e trajetória, que permite que os múltiplos engines instalados sejam executados simultaneamente. O OS aloca uma porção do bloco de memória compartilhada chaveada a cada nome DLL e a sua trajetória para garantir unicidade. Este bloco de memória é independente nos diferentes engines instalados, para impedir conflitos. Para operações durante execução, o OS associa a memória compartilhada com os diretórios primários de cada engines DLL.

Referindo-se à figura 9, onde cada agente e engine podem carregar um ou mais scripts 939 a partir de diretórios separados ou a partir de um servidor remoto para suportar uma capacidade Multi-Instância gMg. Se o mesmo script for executado em N processos para executar em N Aplicações Hospedeiras 901, 902, 903, então haverá N instâncias e cópias do script 945, 948, 952 executadas em N processos 909, 912, 915 com múltiplos threads 918 a 936. Diferentes scripts também podem ser executados em cada processo, onde cada script pode manipular threads de seu respectivo processo, ou um único script concatenado pode monitorar todos N processos. O Multi-Instanciamento no nível de script gMg é manipulado primariamente por uma área de dados de script. Para manipular N scripts em uma única função de engine, o LoadProgram 942 executa a operação de carregar com um string de comando, ou resulta de um comando no próprio script gMg. O nome de arquivo é dado por um número de opções (carregar, descarregar, ou recarregar). Uma lista de programa (PGMLIST) 957 é mantida de todos scripts que estão sendo executados concorrentemente. Durante o processo de carregar

programa, o tamanho do arquivo binário de script é determinado, e a memória alocada ao tamanho específico, e uma vez o arquivo carregado com sucesso, o arquivo será fechado. Uma verificação de integridade é realizada e busca uma Assinatura no inicio do bloco. Se válido, o arquivo é carregado e o arquivo adicionado à lista mantida das estruturas ProgramList 957.

As Operações de Lista de Programa incluem remover, adicionar, ou substituir. O script não carrega, se o script estiver executando. Uma operação de recarregar termina o script correntemente executado e carrega um script com mesmo nome de arquivo. O engine pode executar apenas um script por vez com mesmo nome de arquivo, onde o nome de arquivo é o identificador na lista de programas. Alternativamente, um engine pode executar dois scripts com mesmo nome, se em diretórios diferentes. Uma vez que o nome também contém os nomes de trajetória, dois scripts com mesmo nome, mas em diferentes trajetórias, podem ser carregados, sem provocar conflito. A função descarregar também permite o uso de caracteres curinga (ou seja, Descarregar DirectoryName*) que remove tudo do diretório DirectoryName. O LoadProgram não pode ser executado, enquanto o script estiver sendo executado naquele processo, porque o LoadProgram 942 requer que um gMgLock estabelecido, que normalmente é feito pelo script em execução. Se houver uma tentativa de carregar/ descarregar um script que não terminou o processamento de Evento, uma operação interrompe o script corrente antes de um outro script puder ser carregado. O script então inicia e executa a operação de carregar.

Scripts interpretativos serializam Eventos, e um script é executado por um breve período de tempo, uma vez que a cada passagem processa um único Evento. Quando o Evento ocorre, o script executa e usualmente processa o Evento em 1 ou 2 milisegundos. Então, os scripts freqüentemente não são executados e o engine fica ocioso. Embora haja muitas diferentes organizações e estruturas possíveis,

em um nível alto, ou em sua forma mais simples, o script atua como comando "IF" ou "SWITCH".

Onde se requer automaticamente atualizar scripts em uma programação, se um script gMg estiver sendo executado e governado por gMgLock, pode ser provida uma atualização de uma nova versão. Várias ações controladas por script de Agente poderão ser executadas, tal como atualizar, remover, recarregar, etc. que é a forma na qual o script pode ser carregado e solicitado, conforme necessário. Um componente gMgMain primário (i.e. Agente) pode receber a mensagem GLOBAL_RELOAD_SCRIPT (gerada por um comando RUN) a partir do servidor ou de alguma outra fonte pelos processos. Esta mensagem pode ser transformada ou diretamente encaminhada pelo componente primário gMgMain a todos agentes e respectivos engines, em diferentes processos de aplicação. Quando recebe esta mensagem de comando, a operação de script e/ou engine alvo resulta na execução do comando de carregar ou atualizar script.

Dois aspectos adicionais no sistema de Monitoramento gMg são as senhas WhenObjet e Variáveis de Objeto - importantes para permitir resposta, modelagem, e manipulação de conjuntos de Objeto GUI Multi-Instância.

A senha de script gMg Objeto Variável provê um mecanismo que rastreia diferentes conjuntos de objeto em um ambiente que varia constantemente. Preferivelmente, para rastrear estados dinâmicos em cada um dos diferentes processos, onde há múltiplos scripts, cada script tem seu próprio conjunto de variáveis. Cada conjunto de variáveis independe de outros e suporta cada engine e instância de script individualmente. A Variável de Objeto provê um mecanismo para responder em tempo real a instâncias de jogos potencialmente diferentes e subconjuntos de objetos GUI executados em múltiplos processos e threads concorrentemente. Scripts podem ser escritos de modo a refletir e modelar estrutura de Multi-Instanciamento dinâmica durante execução da aplicação. O termo "Variável de Objeto" se refere a "objetos" de script gMg, onde os

objetos se destinam a manipulações abstratas, operações, ou alocações dinâmicas em resposta e associada a estruturas dinâmicas de Aplicação Hospedeira GUI.

Referindo-se à figura 10, onde uma Variável de Objeto gMg 1035 define uma estrutura abstrata ligada a um objeto GUI 1020 na Árvore de Objeto GUI 1015. O termo "Variável de Objeto" é usado para representar objetos GUI ou conjuntos de objeto GUI que refletem a interface de usuário da Aplicação Hospedeira.

Ademais, a capacidade da Variável de Objeto de armazenar apontadores para objetos em Variáveis de Objeto adicionalmente suporta múltiplas instâncias, e o gerenciamento de múltiplas instâncias em ambientes de objeto GUI de aplicação muito dinâmica. Uma referência de objeto pode ser armazenada como variável em um objeto ou como apontador para outro objeto ou variável de objeto. O apontador para uma Variável de Objeto pode ser dinamicamente criado em cada objeto na Árvore de Objeto GUI 1015 e ser designado a seu próprio conjunto de variáveis em resposta à detecção de Aplicação Hospedeira Multi-Instância. Ademais, um apontador de objeto pode ser armazenado em qualquer Variável de Objeto, se esta Variável de Objeto estiver na Árvore de Objeto GUI em uma outra estrutura, ou ser explicitamente declarado independentemente em algum lugar. Por exemplo, se houver duas Variáveis de Objeto do mesmo tipo, cada uma delas tem seu próprio conjunto de variáveis "privado". Estas variáveis ficam em objetos na Árvore de Objeto GUI, e não na área de dados de script. Em outro exemplo, se houver vinte objetos na Árvore de Objeto, então haverá vinte locais correspondentes para vinte conjuntos de variáveis dinamicamente alocadas pelo engine gMg.

Referindo-se a figura 10, onde qualquer dado Objeto Variável 1005 pode ser vinculado a uma lista de variáveis 1010, onde a associação é manipulada pela função SetNumObjVar (ObjHandle, Progid). No momento da compilação, um único id pode ser gerado para aquela

Variável de Objeto no script (i.e., o valor de nome de Variável de Objeto designado para o mesmo). No script gMg 1040, um valor pode ser designado para aquela Variável de Objeto incluído em uma estrutura física. A lista numérica ou string de outros tipos de Variáveis de Objetos 1025, 1030, 1035 ligados a objetos é escaneada até encontrar o objeto pertencente à estrutura de programa corrente e contenha o correspondente id designado no instante em que o programa foi compilado para aquela Variável de Objeto. Pelo fato de haver conjuntos específicos de Variável de Objeto 1039, cada um deles com sua respectiva lista de variáveis per script de programa, quando se executa múltiplos scripts ou programas cada script tem suas próprias Variáveis de Objeto, ainda que as mesmas referenciem 1045 os mesmos objetos GUI da aplicação que todos os outros scripts concorrentes compartilham ou referenciem. Um conjunto de scripts compartilha a mesma Árvore de Objeto GUI, refletindo um processo de execução no qual cada script tem seu próprio conjunto de variáveis independente ligado a cada Variável de Objeto 1039 contida na Árvore de Objeto para suportar múltiplas instâncias. Preferivelmente, há uma única Árvore de Objeto que pode ser expandida para múltiplas árvores com estruturas similares de suporte de instância.

A segmentação dinâmica das variáveis de estado resulta na capacidade de rastrear concorrentemente muitas instâncias de uma GUI de aplicação. Como exemplo de implementação de Variável de Objeto, a mesma página web pode ser rastreada simultaneamente a partir de um diferente ponto de vista, por múltiplos scripts separados. Para fazer isto, um script é executado e listeners e sensores são transmitidos. Quando ocorre um Evento, este é enviado a cada script separado, onde cada script decide como manipular o Evento independentemente.

Alternativamente em uma configuração adicional, cada Variável de Objeto gMg (na Árvore de Objeto ou em outras estruturas, que corresponda a objetos GUI da aplicação ou

a organizações abstratas internas) pode por sua vez conter seus próprios valores de propriedade de objeto (i.e. string, número Boolean) dados no script. Isto provê uma adicional profundidade de processamento para fazer comparação, testes de condição, ou realizar lógica para diferentes estados de Variáveis de Objeto e suas propriedades.

O segundo comando gMg, o comando WhenObject descrito na figura 11, é outro componente que suporta uma capacidade de Multi-Instanciamento no sistema de monitoramento gMg 1133. Referindo-se à figura 11, onde o WhenObject é um objeto vinculado a um objeto 11106, 1109, 112 na Árvore de Objeto GUI 115 pertencente à Aplicação Hospedeira 1103. Quando objetos GUI são Multi-Instanciados na árvore, o WhenObject também será Multi-Instanciado. A função do comando WhenObject é recuperar propriedades ou avaliar mudanças de propriedade de objeto GUI, quando houver Eventos não liberados (emitidos por OS, Aplicação Hospedeira, ou seus recipiente) e associados a mudanças na Aplicação Hospedeira GUI. Algumas vezes há mudanças de propriedade GUI quando não ocorrem eventos. Ao contrário, algumas vezes ocorrem eventos quando nenhuma mudança de propriedade GUI ocorrer. Em outras vezes, identificar quais eventos usar para detectar mudanças da propriedade visada se torna difícil ou ambíguo. O comando WhenObject resolve estas questões acessando propriedades de objeto GUI através de um mecanismo de busca mais leve.

O comando de script gMg especial WhenObject resulta na rotina WhenObject a ser chamada PollWhenObjects 1148. Esta rotina associa um bloco de código script 1160 a WhenObject 1157 e mantém o rastreamento do script a que ele pertence. Há diversos threads 1121, 1124, 1127 sendo executados no processo de Aplicação Hospedeira 118, onde em cada thread é separadamente feita uma chamada PollWhenObject() 1148 pelo código gMg. Cada thread estabelece gMgLock 1120, primeiro de modo que um engine de script execute apenas um script, um Evento em um

instante no processo de serialização de Evento. Neste instante, é processada a lista de todos WhenObjects 1151 relevantes. Há uma operação de limpeza na qual se o WhenObject for desconectado de um objeto GUI, ele se torna obsoleto (i.e. um objeto GUI é deletado) e seu correspondente WhenObject de contrapartida sendo removido da lista WhenObject.

Se um objeto GUI pertence a um thread correntemente em execução, então este thread executa o apropriado WhenObject. O programa corrente é estabelecido de modo a se referenciar a qualquer script de programa gMg que emitiu o pedido WhenObject e instalar o apontador de código 1157, que é o contador de programa de interpretador de script que aponta para a expressão de condição WhenObject 1157, como definido em um fragmento do código de script 1160, que testa se a execução de WhenObject deve ser acionada. O contador de programa de script aponta para o código de script (i.e., a estrutura de programa que estabelece um contexto). Este aponta a expressão Boolean real compilada no corpo do código de script, e um Evento WhenObject é emitido. Existe um objeto 1139, 1142, 1145 na Árvore de Objeto GUI 1136 ao qual WhenObject foi ligado. Uma chamada é feita a GetNumber( ), e se verdadeira, então WhenObject é acionado. A expressão é avaliada no contexto do programa script e objeto GUI, e estabelece o contexto de execução. O contexto garante que o processamento ocorre dentro do processo correto, objeto GUI na árvore thread, etc.. O script gMg processa um Evento WhenObject que é o Evento global (corrente) que está sendo processado. O objeto do Evento WhenObject é o objeto na Árvore de Objeto que é ligado à estrutura WhenObject que está sendo processada pelo thread apropriado. Se a expressão for verdadeira, ela será adicionada às estatísticas, com respeito a quantas vezes o WhenObject é acionado. Os comandos são continuamente processados até encontrar o limite opcode.

Dentro do interpretador gMg 1153 (mostrado em execução 1130 no processo de Aplicação Hospedeira) a função ExeCommand() aponta para o conjunto de script gMg de opcodes. O apontador gPC variável é o contador de programa de interpretador global que aponta para o código de script (apontando para a expressão de condição e imediatamente depois da expressão de condição ser uma lista de comandos rastreado de um parada opcode). A expressão é avaliada, que leva o contador de programa passar a expressão. Se a expressão for verdadeira, então ela aponta para o primeiro comando e executa o comando, em qual caso há um comando WhenObject.

A execução do comando WhenObject realiza uma operação de busca através da expressão Getnumber neste ponto, que pode fazer uma recuperação das propriedades do objeto e subseqüentemente os comandos de script são executados. Todos WhenObject de todos scripts entram em uma lista centralizada 1151. Adicionalmente, os objetos script indicam sua associação com os respectivos scripts. Quando termina o script, o WhenObject (estrutura) é removido, e quando termina um objeto, aquele WhenObject específico correspondente também é removido. A estrutura WhenObject é uma lista coletiva de WhenObjects de todos scripts Multi-Instanciados concorrentes.

O WhenObject precisa ser executado por threads nos scripts, qual script será executado quando, em cujo thread, for determinado pelo controle de thread. Assim, somente os WhenObjects relacionados ao thread corrente serão processados no thread corrente. O thread estabelece o contexto para o respectivo script 1154 (representação expandida de engine e script no processo de aplicação em 1130) antes de realizar a função. Estabelecendo o contexto, o thread vê a variável de objeto pertencente àquele script e bloqueia o acesso às variáveis de objeto de outros scripts. WhenObjects podem ser centralizados para todos scripts sendo executados.

A amostra de código que se rastreia ilustra configurações

da capacidade Multi-Instância de um Sistema gMg. O script detecta um número ilimitado de threads, onde cada thread controla uma janela popup concorrentemente com um número ilimitado de janelas de mensagem. Se um script gMg monitora o tempo de vida de cada objeto GUI em uma aplicação, para pleno monitoramento muitas estruturas de instância precisariam ser criadas. Para cada objeto o gMg automaticamente mantém estruturas que representam as instâncias de objetos GUI. A Árvore de Objeto tem um cachê de atributos de cada objeto, e um dispositivos de armazenamento de Variável de Objeto para cada objeto.

```
//"ManyInstances" - este script opera um número ilimitado de objetos
objvar createtime:number,
if (create)// on creation
createtime=sys_time;
        //createtime é um objvar criado (portanto
        //instância) para o Evento corrente, Onde uma instância
        // é uma ação resultante do Evento.
Vars=title;
        // é uma variável temporária e ilustra gMg
S=class
        // é uma capacidade de instanciamento,
        // aqui título e classe para o
        // objeto corrente são armazenados separados
        //na Árvore de Objeto GUI
If (destrói)
{
//tempo corrente - createtime (que ocorreu antes,
// determina o tempo de vida dos objetos
dbg (objtypename (objtype)+[+título+][+classe+] existiu para +
string(sys_time-createtime) +ms/n');
{
//.....
//Expressão boolean que diz se o objeto corrente é uma janela popup
função ispopup= iswindow && title==="blabla" &&c class== 'whatever'
//indica se o objeto corrente é uma janela de mensagem de interesse
função ismsgwindow= iswindow && title - blabla &&class = 'whatever'
```

```
//se um evento cria, o cria um popup, o objeto thread do popup
//armazena um ptr para o objeto popup, o objeto uma senha para
//objeto corrente, i.e. o objeto criado, cria um ptr que surge
//no objeto thread, qualquer que seja o thread a que o objeto
pertença objvarmypopup:object;
if(create &&ispopup && thread.mypopup==none)// se um create, thread
cal seria vazio
thread.mypopup-object// aponta para popup object
//se evento destrói e objeto são destruídos e o objeto se encontra
em thread.mypopup
if(destroy&&object == thread.mypoopup)
thread.mypopup = none / desimpede thread.mypopup
//quando janela de mensagem é criada, WhenObject é ligado a objvar
//prevtitle; verifica se o titulo mudou do título anterior, i.e.
//changed fn
objvar prevtitle:string
id(create && ismsgwindow)
WhenObject (changed(live_title, prevtitle) && thread.mypopup= none
//threadval é diferente de zero
```

Numerosas senhas ou definições de Variáveis de Objeto têm comportamentos implícitos, determinados por alocações dinâmicas de objetos GUI (i.e., prevtitle, title, class) todos respondendo ao objeto corrente, onde o objeto corrente reflete o Evento corrente, e o objeto corrente é processado pelo interpretador.

A variável mypopup é um variável de objeto. Durante o processamento é feita uma chamada para a função GetNumberObjVar(). Quando da inicialização, a primeira passagem descobre que o objeto não está presente e volta o *default* para zero. Então uma chamada para a função SetObjVar () coloca uma referência no objeto corrente na localização da janela popup e também coloca uma referência na janela popup em seu próprio objeto thread.

Título Ativo (Live Title) é corrente e renova o título em caso de mudança de título, que será verdadeiro se o valor do Título Ativo mudar de um título anterior, qual título anterior é armazenado no objeto de janela de mensagem.

Por exemplo, se uma aplicação cria dez janelas de mensagem, cada janela tem seu próprio título anterior. Há dez WhenObjects associados a cada janela de mensagem. O sistema monitora as janelas de mensagem, onde cada janela é monitorada para detectar se o conteúdo de Título Ativo, i.e. o título corrente, muda (muda o comando) a partir do título anterior. Se verdadeiro, e o título de janela popup é "abc", é verificado se a respectiva janela popup está executando no mesmo thread que a janela de mensagem. Por conseguinte, quando um título de janela de mensagem muda, o título da janela popup representada é "abc", e na detecção a condição é dada como verdadeira, e o WhenObject executado.

O WhenObject é acionado para cada janela de mensagem detectada, e a rastrear qualificado pelo teste para janela popup. Pode haver um número ilimitado de threads, onde cada thread pode ter um número ilimitado de janelas de mensagem, mas sendo permitido apenas um popup per thread. Este código também ilustra um apontador de objeto dentro de um objeto (i.e., thread.mypopup=object). Por exemplo, pode haver vinte threads, onde cada thread pode ter sua própria janela popup. Para cada uma das vinte variáveis de thread nomeadas, mypopup grava informações a respeito de cada thread e sua correspondente janela popup. Para estas estruturas, cada variável de thread pode ser rastreada, quer a respectiva janela popup exista ou não.

A descrição agora se refere ao contexto gMg e seu papel na identificação de Multi-Instanciamento de estruturas GUI. Referindo-se à figura 12, a senha de contexto de script gMg apresenta uma entidade de organização hierárquica que representa uma seção de uma estrutura de aplicação GUI Multi-Instanciamento para prover o servidor de uma informação contextual para suportar a análise de dados de rastreamento colhidos. O comando Context pode ser usado para todos tipos de objetos. O nível de topo é um processo 215, mas se o processo tiver múltiplas

janelas de nível de topo 1210, será especificado um sub-contexto, que começa com a janela de nível de topo, para o processo corrente construir um relacionamento hierárquico da janela para representar e identificar um contexto para Eventos de rastreamento gerados que é enviado para o servidor. A descrição contextual é flexível na descrição de uma variedade de organizações de objeto. Se uma janela de topo for especificada, a mensagem de rastreamento será formada com ambos, a janela de topo e seu processo "filho", caso contrário somente o processo será enviado.

Para rastrear um Evento no escopo de um processo, o contexto pode ser estabelecido para o processo, mas rastreando objetos, tal como múltiplas janelas popup 1205, então o contexto pode ser estabelecido em relação à específica janela popup, na qual os Eventos ocorreram. Dependendo da definição, Context estabelece uma hierarquia implícita que o servidor analisa. Há um limite arbitrário para hierarquia de cinco níveis, que é enviado com cada registro de mensagem de rastreamento. Cinco níveis são considerados suficientes na maioria dos casos, mas este limite de nível pode ser expandido ou reduzido.

Referindo-se à figura 13, onde o valor de nível de contexto é significativo se em uma organização de árvore GUI de Aplicação Hospedeira 1305 houver múltiplas instâncias similares de objetos GUI, conjuntos de objetos GUI, ou objetos. Por exemplo, se dado um processo 1315 com múltiplos threads 1320 a 1335, com múltiplas janelas popup 1340 a 1355 (i.e., uma janela para cada thread) e cada popup contendo uma janela com múltiplos botões 1360 - isto é descrito como um contexto nível quatro. O contexto provê uma informação extra que se revela útil, quando cada instância pode ser identificada (i.e., cada botão em um jogo de botões em uma janela é único).

Considerando o botão OK no botão para mudar tamanho de fonte de uma típica caixa de diálogo de janela GUI, neste exemplo, o contexto é absolutamente necessário quando

houver múltiplas instâncias e estruturas idênticas de botões OK ocorrendo simultaneamente. O Internet Explorer da Microsoft, particularmente em aplicações web, permite múltiplos threads, onde cada thread pode ter a mesma pagina web que outro thread. O contexto é absolutamente necessário para uma aplicação que tenha tal resolução hierárquica de objetos GUI, onde, por sua vez, cada GUI pode ter múltiplas instâncias. Cinco níveis são suficientes, porque os contextos em geral são identificáveis usando propriedades ou outros mecanismos de detecção.

Em outro exemplo, para uma janela com múltiplos botões como é normal em aplicações web, apenas enviar o nome de controle de botão ou outros nomes indica que o botão existe ou está sendo clicado. No entanto, se todos os controles de botão tiverem estruturas idênticas para aplicação em diferentes lugares, para distinguir entre diferentes botões para atividade de rastreamento, será necessária outra definição de nível de contexto. O nível de contexto adicional passa dados, tal como id único e/ou referências, e reflete as ocorrências de cada botão.

Em outro exemplo, uma Aplicação "Contact" contém registros de cliente, e mostra registros contendo nome, endereço, telefone, e mail. Neste caso, todos registros têm estruturas internas idênticas, que são threads separados e janelas popup separadas, mas contendo informações de conteúdo único para cada cliente. As janelas popup têm estruturas idênticas, mas cada uma delas tem um conteúdo único. Na estrutura de nível de topo, elas parecem idênticas ao sistema gMg sem informações adicionais. Para resolver esta ambigüidade, um nível de contexto extra é usado para diferenciar as diferentes janelas popup, que produz uma informação de contexto reconfigurada enviada com cada mensagem de rastreamento ao servidor. No servidor, os níveis de contexto comum são agrupados nos Eventos de Rastreamento que ocorrem em cada estrutura de janela GUI, de modo que

se houver um clique em uma janela, este não será confundido com um clique em outra janela.

Junto com objetos GUI, os objetos thread também podem ser visados pelo processo de identificação. Os threads podem ser contextualmente identificados, onde cada thread de aplicação em uma aplicação multi-thread GUI é designado seu próprio contexto. Dada uma aplicação com múltiplos threads e múltiplas janelas de topo idênticas, é introduzido um nível adicional (um objeto thread em um processo) e um thread-id usado para identificação. Isto manipula uma situação multi-thread onde há múltiplos threads idênticos, cada thread tendo janelas de topo idênticas, requerendo manter rastreamento da atividade de cada grupo de janela de topo separado da atividade em outras janelas de topo idênticas. Uma vez que múltiplos threads podem parecer idênticos, outros objetos ou relacionamentos (i.e., um predecessor) são definidos para identificar threads diferentes e objetos associados. Esta situação ambígua, quando se identificam estruturas GUI e Eventos, é encontrada em muitas aplicações web, (aplicações padrão de gerenciamento de relacionamento de cliente de indústria (CRM)).

Em diferentes implementações, tal como aplicação CRM da Microsoft (Microsoft Corp. de Redmont), descobriu-se que um popup pode ser aberto para qualquer nova atividade que, por sua vez, cria um novo thread, então seriam necessários tantos threads quanto pudessem ser criados como janelas popup adicionais. Nesta situação, o uso de Contexto de thread gMg distingue as janelas popup separadas. No entanto se houver apenas um nível e dentro dos popups for possível criar instâncias idênticas múltiplas de janelas, então um nível de contexto adicional será necessário. A aplicação CRM da Microsoft dado como exemplo requer apenas dois níveis. A extensão do número de níveis pode se mostrar mais conveniente e ser reservada para uma futura expansão. Alternativamente, outros atributos podem ser utilizados nas janelas popup

para distingui-las.

Definindo uma hierarquia contextual de objeto GUI de profundidade (resolução) suficiente, estruturas de objeto GUI, que de outra forma seriam idênticas, podem ser distinguidas. Se houver múltiplas estruturas de objeto GUI, janelas no mesmo contexto topwindow poderão ser usadas para agrupar todos Eventos em conjuntos lógicos associados aos objetos GUI alvo. O componente Context GUI suporta uma capacidade múltiplas instâncias para diferenciação de objeto.

A discussão agora se volta aos problemas de detecção de eventos e identificação com propósito de suportar Multi-Instâncias para estabelecer Hooks que proporcionem uma das principais categorias de eventos para o Sistema gMg. Referindo-se à figura 14, onde ChangeHook e LockgMgEngine passam o thread-id, e engancham o processo de aplicação 1406, conforme necessário. Uma mensagem gMg 1424 então é encaminhada e registrada para a aplicação através da função PostgMgMsg(). A função PostgMgMsg() usando o thread-id, encontra a janela usando o thread-id e então determina a mensagem de janela. Como mencionado acima, o gMg registra sua única mensagem de janela usando o nome inteiro de trajetória do.exe que determina a unicidade para múltiplos gMg.exe's. O PostgMgMsg trabalha genericamente e encaminha múltiplas cópias da mensagem para completar o processo de registro, onde a primeira mensagem a ter sucesso é usada no sistema e as remanescentes descartadas. A memória compartilhada 1454 é usada para controlar e rastrear a obsolescência da mensagem de janelas gMg para cada instância de engine ou Agente gMg. Para suporte de diagnóstico, o valor tmdiagnostics é contido na mensagem wparam com Ipram apontando para os dados de diagnóstico localizados na memória compartilhada. Quando a informação de diagnóstico for requerida por algum componente gMg, a mensagem de janela gMg envia a mensagem diag_exports que chega em uma das rotinas winhook.

Controlado por um relógio 1457 a partir do sistema operacional 1403, todas rotinas Hook em execução 1433 a 1442, eventualmente interceptam a mensagem e chamam a rotina HandleMsg() 1445. Na HandleMsg, nas rotinas GetMessageHook 1445, o GetMessageHook é concatenado com um número N baseado no número de Hooks correntemente determinados pelo sistema. O Hook é implementado em um conjunto de macros 0-N 1427, que chama a função HandleGetMsgHookMessage 1430. Há também um arranjo de apontadores para cada uma destas rotinas, quando estiver sendo feito o enganchamento (hooking), onde o autoinline durante a compilação é desativado, para evitar que a rotina gMgMsg se duplique múltiplas vezes, em cada instâncias. Ademais, na HandleMsg há N SendMessageHooks que também manipulam a mensagem SendMessage 1448 que chama HandleMsgHookMsg 1451. Estas rotinas interceptam as mensagens, a despeito de suas origens.

Preferivelmente, em vez de criar threads de trabalho dedicados a monitorar processamento, são usados threads hospedeiros 1409, 1412, 1418, 1421. A criação de janela é minimizada com somente com um timer criado no processo para determinar uma função callback. Ademais, nenhuma janela de trabalho é criada uma vez que o objetivo de realizar Hook é ser transparente e minimamente invasivo. Ainda que nenhuma janela de trabalho seja estabelecida, a janela ainda precisa de um timer para o controle de processo.

O sistema gMg envia uma mensagem às janelas existentes da Aplicação Hospedeira e então a intercepta. Mensagens raramente são enviadas pelos processos, mas se isto ocorrer, se empreende uma busca de N diferentes janelas na Aplicação Hospedeira. Uma vez localizado o gMg, se estabelecem Hooks em cada um deles e a mensagem é interceptada quando chega na respectiva janela alvo.

A Macro tid adquire o thread-id corrente, onde houver um Hook *per* thread. Normalmente, as mensagens são recebidas a partir de janelas que possuem o thread corrente, mas

ocasionalmente as mensagens podem ser recebidas de outros threads de janela. Um thread que executa um SendMessage deve pertencer ao mesmo processo, caso contrário um DLL terá que ser injetado em outro processo externo, que não esteja explicitamente enganchado. Se o SendMessage for iniciado por um processo diferente, SendMessage é colocado no processo alvo e designado um thread no processo que chama uma rotina Hook. A rotina HandleMsg é chamada a partir de GetMsgHook() e SendMessageHook(). O manuseio da mensagem tem um comportamento uniforme, a despeito de quantas mensagens chegarem, o gMg tenta uma interceptação.

O GetExParams() é usado para passar parâmetros e informações adicionais em processos com mensagens de janela inter-processo gMg (i.e., carregar script, descarregar engine, etc.) em chamadas SendMessage. Ele tem um bloqueio na memória compartilhada que consiste de um número de versão mais um índice. Esta função mascara o número de versão e pode ser indexada em um arranjo na memória compartilhada para adquirir dados adicionais a serem transmitidos com a mensagem após a indexação. Se a memória compartilhada id (que inclui a versão) não corresponder ao id requerido, então é uma entrada obsoleta. O número id de 32 bits é continuamente incrementado. O seis bites de base são índices em um buffer circular na estrutura ExParams. Se o id de 32 bites no buffer não corresponder ao parâmetro dado, então se determina que o id é reutilizado ou obsoleto. GetExParams() busca Iparam e retorna zero, se obsoleto. Há um macro usado para remover o código para uma versão release do engine. Um teste também é feito para ver ser dbgdll é carregado.

O mecanismo de detecção de Objeto GUI que detecta conjuntos de objetos GUI no Sistema gMg é chamado detecção genérica. O método de detecção Genérica determina e detecta Eventos GUI com processamento distribuído em ambos cliente e servidor. Há dois tipos de

operações de objeto GUI: Específico e Genérico. Em "Genérico", os objetos são detectados por um número muito menor de descritores operacionais para criar um conjunto ou classe de objetos GUI. Parâmetros de entrada fornecidos a uma função podem ser analisados para detectar uma classe de objetos GUI, ao invés de apenas um objeto GUI específico.

No cliente, o espectro 1515 de detecção GUI (figura 15), as expressões de script gMg são deliberadamente feitos mais genéricos 1510 (i.e., para detectar todas categorias de objetos GUI) ou mais específicas 1505 (i.e. para detectar objetos GUI únicos). As expressões de detecção podem detectar objetos GUI que se encontram em algum lugar neste espectro conceitual. Os sensores gMg colhem os dados de objeto GUI de Aplicação Hospedeira, e através destes, o script gMg define expressões genericamente (ou categoricamente) para detecção de objeto GUI. A coleta resultante de detalhes e propriedades de objeto GUI é colocada em pacote e enviada ao servidor. O servidor recebe estes Eventos categorizados e as correspondentes propriedades, e em seguida os processa para determinar Eventos de objeto GUI. O servidor constrói contextos de usuário usando níveis de mensagem de rastreamento, semânticas e convenções pré-definidas. Em uma detecção GUI específica, uma mensagem de rastreamento é enviada a cada objeto GUI identificado contendo propriedades específicas a respeito do objeto. Em uma detecção genérica, uma mensagem de rastreamento é enviada contendo dados suficientes para o servidor determinar identificação e análise de objeto GUI Específico em tempo real posterior.

A rastrear são dados exemplos de código de Ligação (Binding) Específica e Genérica, a primeira configuração mostra uma ligação específica (visando um único objeto GUI) onde o cliente filtra e envia diferentes mensagens de rastreamento a cada um dos quatro botões GUI definidos que são rastreados.

1. verifica-se se a classe é "botão", o título é "OK", e a janela de topo é "open window", se positivo, então uma mensagem de rastreamento é enviada, indicando que "Open OK" foi apertado - ("Open OK");
2. se houver um botão "OK" no diálogo "Save", uma mensagem de rastreamento "Save OK" será enviada - ("Save OK");
3. se for um botão "Cancel" em uma caixa de diálogo aberta, a mensagem de rastreamento "Open Canceled" será enviada - ("Cancel Open");
4. se for um botão "Save" e o diálogo Save tendo uma mensagem "Save/Cancel", a mensagem de rastreamento apropriada será enviada -("Cancel Save").

A rastrear se demonstra como os quatro tipos específicos de botões estão sendo deletados.

```
/* Amostra de Código Ligação Especifica e Genérica */
processo de contexto = servidor de processo "someserver"
thread de contexto = thread em processo
contexto topwindow = top window em thread
//Ligação Específica - cliente determina objeto específico e envia
uma mensagem de rastreamento para cada
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 == "Save" && class == "332767"))
track(priority, pressed, topwindow, "Save Cancel");
//Ligação Genérica - cliente envia mensagem de rastreamento a todos,
mas com info para servidor verificar
if(wlbutdown &&class == BUTTON)
track(priority, pressed, topwindow, "Button", Title=Title,
```

toptitle=topwindow.title, topclass=topwindow.class);

Qualquer mensagem por botão ou qualquer coisa com a classe "botão" com um Evento de apertar, envia uma mensagem indicando que um botão (qualquer botão) foi apertado. O título topwindow e a classe topwindow também podem ser encaminhados para o servidor para analisar e detectar qual dos quatro botões está ativo. Na mensagem de rastreamento de múltiplos códigos genéricos, as mensagens podem ser idênticas, exceto por conterem diferentes dados de conteúdo, que as distingue como contrárias à mensagem de rastreamento, que claramente identifica os objetos GUI alvo antes de da transmissão para o servidor.

Em ambos casos - Específico e Genérico - os Eventos são recebidos pelo interpretador e no processamento do interpretador de Evento, e com base nas condições descritas no script em ambos casos Específico e Genérico realiza a recuperação de propriedades para permitir uma execução condicional de expressões. A recuperação de propriedades é parte da condição ou parte do corpo do comando de condição (i.e., if(Q) {R}, onde R pode conter uma ou mais mensagens de rastreamento). Ademais, o servidor manipula ou detecta diferentes objetos GUI e quer busca os tipos de botões, tal como Save, Cancel, OK etc, ou parse o texto para proceder à correspondência de string. Este último caso muda a responsabilidade do processamento de detecção do cliente para o servidor.

O monitoramento do Método de Sistema gMg implementa um sensor especializado usado para monitorar os métodos contidos nos componentes. Este sensor especializado flexível permite a possibilidade de a invenção gerar e integrar muitos tipos de eventos no espectro de Aplicação alvo no sistema operacional quer no cliente ou servidor.

O monitoramento do Método compreende a interceptação dos métodos de componente para suportar um tipo de sensor que intercepta chamadas e retornos de função em componentes de Aplicação Hospedeira em pacotes DLL. A interceptação

das chamadas de função permite que o Sistema gMg recupere Eventos e propriedades de componentes, que são transmitidos aos scripts lógicos para rastreamento e monitoramento.

Dado que qualquer função pode chamar outras funções ou a si mesmo recursivamente, o Monitoramento de Método gMg se baseia em dois objetivos básicos em uma variedade de ambientes de componente e situações: 1. o endereço do Método alvo deve ser localizado; 2. o Método alvo é interceptado. A técnica para localizar o endereço do Método depende de qual tecnologia de componente é usada em sua implementação, tal como COM, objeto não-COM, função Export, ou informação disponível no cabeçalho typelibrary, parâmetros de função, retorno de função, memória, tabelas, etc.. Embora Métodos COM (figuras 16, itens 1624 a 1636) sejam discutidos como exemplo de capacidade gMg, o princípio é aplicável a métodos contidos em diferentes tecnologias de componente de outras companhias, tal como Corb (Object Management Group, Needham, MA), IBM's SOM (IBM, Armonk, NY) e muitos outros sistemas de componentes usados para construir aplicações de software. Para o sistema gMg, um suporte estendido destas tecnologias adicionais se refere à tarefa de construir um adaptador diferente tipo componente. As técnicas apresentadas podem ser aplicadas a outras tecnologias de componente, como deve ser prontamente aparente àqueles habilitados na técnica.

Em outra implementação, um fator adicional e uma variação técnica no COM usando um objeto temporário gMg, ocorrem quando o cenário de monitoramento determina que o objeto contendo os Métodos alvos exista no instante de ativação do Sensor de Método gMg. No entanto, se o objeto não existe, o monitoramento da criação do sistema operacional preserva a continuidade de execução da Aplicação Hospedeira (i.e., provê uma execução ininterrupta enquanto monitora Métodos). Os Métodos alvo são incorporados em várias tecnologias de componente, que

por sua vez se baseiam nas interfaces disponíveis no sistema operacional ou Aplicação Hospedeira.

Para suportar interceptação de método COM se o sensor de Método gMg for ativado depois de o componente COM já tiver sido criado, as Vtables são primeiro localizadas para recuperar endereços do método. Uma VTable é uma tabela em uma classe COM contendo apontadores de método para métodos ou função pertencente à classe. A técnica de Interceptação por Assinatura de Método gMg então é aplicada para estabelecer um monitoramento. Deve ser notado que as interfaces COM, se publicadas, são documentadas como TypeLibs e encontradas em DLLs/OCXs.

Outra categoria de componente, a função Export tem interfaces de método expostas em um componente DLL, onde os nomes de função Export são listados na tabela Export de componente em cabeçalhos de módulo de programa. Uma vez localizado, o monitoramento de Assinatura de Método gMg (figuras 16, itens 1612, 1621, 1639) estabelece pontos de captura durante chamadas e retornos da Aplicação Hospedeira de/para o Método.

Uma vez determinado o tipo de componente (função Export ou COM), o monitoramento de Assinatura de Método gMg (figura 16 item 1657) é giratória e aplicada difusamente no Monitoramento de Método gMg em uma ampla variedade de cenários, uma vez localizado o Método alvo. Uma Assinatura de Método é definida como padrão opcode de um ponto de entrada do Método descrito no Sistema gMg e descoberto com ferramentas de inspeção gMg. Uma vez definidos, os padrões opcode são usados para localizar e verificar os pontos de entrada de Método usados para interceptação de chamada e retorno. O monitoramento da Assinatura de Método pode ser universalmente aplicado em qualquer função localizada, a despeito de pacote, ambiente, localização. Uma vez encontrados os endereços de Método alvo, então o Monitoramento de Assinatura de Método de gMg é aplicado para realizar interceptação de Método.

Em outra configuração, o Monitoramento de Assinatura de Método ocorre se, quando uma chamada começa com um Método VTable localizada, e dentro do Método VTable há outras chamadas internas a Métodos e componentes. O Monitoramento de Assinatura de Método também pode ser usado para rastrear as chamadas de método internas. Ademais, o Monitoramento de Assinatura de Método é usado como parte do arranjo de Monitoramento de Método VTable dentro de Objetos COM.

Referindo-se à figura 16, onde é mostrada uma visão geral do Monitoramento de Método gMg, a Aplicação Hospedeira 1603 executada no ambiente de sistema operacional 1609 inicia o carregamento de seus componentes DLL 1615 através de chamadas da função de código de sistema operacional LoadLibraryExW 1618. O monitoramento gMg setup uma função Intercept Kernel LoadLibraryExW para garantir que todas instâncias de um DLL alvo sejam detectadas. Os DLLs contêm funções Export 1620 ou typelibraries COM 1624. Para Monitoramento de Método COM, logo antes da criação de objetos COM alvo, o monitoramento gMg intercepta diferentes estágios da criação COM 1627 com propósito de localizar VTable 1633 de COM, que inclui apontadores para seus métodos de interface 1636. Os estágios de execução são: gMg intercepta Métodos DIIGetClassObject, o DIIGetClassObject sendo chamado pela Aplicação Hospedeira e retorna IClassFactory; gMg intercepta IClassFactory::Create Instance; o IClassFactory::CreateInstance sendo chamado pela Aplicação Hospedeira e retorna Objeto COM recém-criado; e inspeção e operação com objeto COM recém-criado provê a localização de sua VTable.

Em cenários onde o objeto COM já foi instanciado e ligado atrasado (onde os Métodos são instanciados e registrados em estruturas VTable durante execução, em oposição ao tempo de compilação) a seu Método, o Sistema gMg cria seu próprio objeto COM temporário 1630 que provê o endereço para a mesma VTable usado por outros Métodos alvo

(já ligados) dentro do componente. Uma vez a Vtable localizada, o Sistema gMg aplica seu Monitoramento de Assinatura de Método 1639 (mostrado expandido em 1657) para monitorar de modo transparente e não obstrutivo os métodos alvo. Para monitorar as funções Export 1618, o Monitoramento de Assinatura de Método gMg também é usado para interceptar chamadas de função.

Referindo-se à figura 16A, 1657, uma vez localizados Métodos, o monitoramento gMg setup interceptações de Método de Assinatura para chamadas 1642 e retornos 1645 do Método alvo feitas direta ou indiretamente, pelo Chamador de Método 1606 da aplicação. O código de interceptação gMg mantém a integridade da função original e com base na chamada e/ou retorno da Aplicação Hospedeira, emite Eventos gMg apropriados para o interpretador gMg 1660 e passa informações relevantes solicitadas pelo script.

O retorno do monitoramento do Método é classificado quer como Genérico ou Específico (não no sentido objeto GUI). Como mostrado na figura 16A, são dadas duas técnicas para captura de retorno genérico Cópia de Stack 1648 e Alocação de Stack 1651 (ou stack *per* thread) que são abordadas nas seções seguintes. O Monitoramento Específico 1654 se refere a Métodos cujos parâmetros são encontrados em local publicado, tal como TypeLibrary. No entanto, um Método de interceptação genérico trabalha com qualquer função sem pré-conhecimento dos valores de retorno e parâmetros da função. Neste caso, os Eventos serão somente gerados sem informação específica, quando de uma chamada ou retorno. No Método de interceptação genérico, thunks de chamada e retorno e seus códigos são executados.

Das três estruturas importantes (Object Tree GUI, figuras 7 e 8, Árvore de Processo/Thread) no sistema de rastreamento e monitoramento gMg, o terceiro, nomeadamente Árvore de Módulo de Código, contém módulos, monitores de função, e instâncias de objeto COM,

dinamicamente detectados para suportar Monitoramento de Método.

Referindo-se à figura 17, onde é ilustrada uma Árvore de Módulo usada pelo Sensor de Monitoramento de Método, há uma Árvore de Módulo separada com uma lista de módulo 1705 na raiz. Um item árvore de monitor de função 1710, 1725, 1730, pode estar em qualquer lugar na árvore. Uma função de chamar Event a partir do Monitor de interface 1715, um objeto COM 1720 pode ser "filho" de uma função chamar Event 1715, se um dos argumentos (da função chamar Event (P1, Ptr_to_COM,...)) apontar para um objeto COM. Determinado pelo script, somente as funções selecionadas serão monitoradas. A Árvore de Módulo seletivamente contêm apenas o que é necessário para satisfazer as solicitações de script. Ademais, o Interpretador gMg, dirigido por comandos no script, atualiza Árvore de Objeto GUI e a Árvore de Módulo para modelar a aplicação GUI real e suportar função Eventos na aplicação.

Na iniciação, a Árvore de Módulo é preenchida com módulos de programa 1705. Objetos de função e Interface na árvore serão criados apenas por solicitação de um script. Uma vez em operação, a Árvore de Módulo é mantida usando uma interceptação de chamada e retorno LoadLibraryExW.

Nas funções Export, os pedidos de script a serem monitorados são adicionados aos Objetos de Chamada de Função "filho" de Objetos de Módulo na Árvore de Módulo. Objetos de Interface dos Objetos COM alvo também são adicionados como objetos "filho" de Objetos de Módulo para referência, e para monitorar a criação dos citados tipos de objetos. Objetos COM dos tipos desejados são adicionados como "filho" de Objetos de Interface quando criados. Objetos de Chamada de Função de Método são adicionados como "filho" de Objetos de Interface ou Objetos COM, quando o script solicita o monitoramento destes métodos de uma Interface, ou Objeto COM. Objetos COM descobertos a partir de chamadas de método

interceptadas são criados temporariamente para referência de script e nem sempre estão na Árvore de módulo. Eles são recolhidos pelo engine após uso.

O Monitoramento de Assinatura de Módulo provê uma técnica para monitorar chamadas de função, e é aplicado a diferentes situações para criar Eventos de Método gMg e recuperar propriedades. O Monitoramento de Assinatura de Método identifica os padrões opcode de byte que representam Assinaturas de Módulo reconhecidas de funções de Método. Uma vez localizadas, as funções detectadas podem prover interfaces a itens, tais como propriedades e Eventos GUI, assim como outros eventos ou processos internos que ocorrem em componentes de aplicação. Assinaturas de Método podem ser aplicadas universalmente a qualquer ambiente de Componente para qualquer tipo de chamada e capturar parâmetros de função. Uma vez determinado o endereço do procedimento, gMg vincula o mesmo usando uma instrução jump com uma rotina de interceptar chamada e uma instância de estrutura INFO, que rastreia toda chamada para a função alvo. A estrutura INFO contém um apontador de byte para o código de função, um apontador "vazio" para seu procedimento de interceptação, e uma estrutura INFO chamada InterceptMultEdxInfo que suporta a instância. Como parte do suporte Multi-Instância, há várias técnicas para suportar e transmitir informações relativas a respeito da interceptação de chamada de função, tal como registradores CPU (i.e. Registradores de Instância) ou outros mecanismos.

Referindo-se à figura 18, onde é ilustrada a criação de um Monitor de Método. Iniciada pelo comando de monitor de script gMg 1836 contido no script gMg 1833, primeiro o endereço da função é localizado pelo mecanismo MakeFunctionMonitor. Dado um Módulo N 1806 contendo N Métodos 1827 como preparação para localizar o endereço de função e executar um pedido de interceptação, a lista de módulo apropriada já encontrada e mantida

automaticamente durante chamadas interceptadas LoadLibraryExW usando enumeração de módulo API's, tal como CreateToolHelp32Snapshot 1815, Mod32First 1809. Como deve ser prontamente entendido por aqueles habilitados na técnica, estes API's podem ser substituídos por funções disponíveis atualizadas do Sistema Operacional desde que estas funções sejam promulgadas. A lista de Módulo 1803 foi preparada previamente no instante de um pedido de interceptação. MakeFunctionMonitor 1851 recebe o nome da função e um Objeto de Módulo, Objeto de Interface, ou Objeto COM e chama MakeFunction, uma vez localizado um nome de função Export ou o endereço de código de módulo COM. Então um Objeto Monitor de Função 1860 é criado como um derivado da classe de Objeto gMg, onde o objeto contém uma estrutura de interceptação de função. Propriedades implementadas contêm uma instância de objeto gMg que retorna um apontador para o procedimento original e o coloca em verdadeiro, se bem sucedido.

MakeFunctionMonitor implementa um tipo "Objeto Monitor de Função" que diferentemente de uma janela, HTML, DOC ou timer é um tipo de objeto de função que pode conter várias propriedades, tal como quantas vezes uma função alvo é chamada. Todas instâncias e alocações da função sendo rastreadas globalmente. No instante do pedido de interceptação, o módulo contendo a função desejada é conhecido, quer por ter sido determinado durante o pedido de interceptação ou pré-determinado. O script gMg passa o Módulo, e o GetProcAddress 1824 é usado para encontrar o endereço de início do corpo da função (i.e., MakeFunctionMonitor para Objeto de Módulo usa a função GetProcAddress 1824 para o nome de função na chamada para InterceptCallMultEDX 1818).

Chamado indiretamente como resultado do comando de script gMg 1839, CallIntercept 1848 realiza a interceptação e acessa a estrutura INFO 1863 localizada no objeto de função 1860. Para evitar uma recursão, se um comando de

script gMg inadvertidamente chamar uma função interceptada durante processamento de interceptação, será executado um lockD no gMglock. Antes de interceptar, um mecanismo de bloqueio verifica 1842 que a função CallIntercept 1848 não está sendo bloqueada pelo mesmo thread, em qual caso a interceptação de chamada será ignorada 1845. Assumindo a interceptação bem sucedida, então o código de interceptação é passado para um apontador global para esta estrutura INFO, que então é adicionada ao Objeto de Monitor de Função.

A função FunctionIntercept 1866 tem o objeto de função gMg 1860. O Evento gMg 1869, emitido quando a interceptação é executada com sucesso então é passado para a função chamar Do_gMgEvent 1872 que processa o Evento e todos scripts 1875 que estão sendo executados. Do_gMgEvent recebe a função chamar Evento, o processa, e terminado, libera o apontador global e o bloqueio global. Recebido pelo interpretador gMg, o Evento de Método usado pelas funções GetString e GetNum é chamado Método EventProperty 1869. As funções GetString e GetNum são aplicadas a cada tipo de Objeto gMg que se chamam se não reconhecerem seu próprio tipo. O script gMg pede uma propriedade StringEvent por índice, e se a propriedade não for localizada, o código de interpretador de script chama um dos outros tipos em uma cadeia de funções para cada tipo de objeto gMg.

Se dentro do script, tal como event.s1, a interpretação resulta em uma série de chamadas buscando propriedades Evento GetString, o índice que representa s1 é passado. Uma verificação é feita para um apontador nulo e um valor de seleção de Evento global determina se a função chamar Evento está sendo processada. Preferivelmente, se um diferente tipo de Evento gMg for recebido, tal como criar Evento que está sendo processado, a prioridade de escanear a Árvore de Objeto é aumentada, que resulta em escanear a árvore de Aplicação Hospedeira. Se o Evento estiver processando uma chamada de função, então é

formado um apontador, que é um desvio no stack onde estão os parâmetros da chamada original. Cada parâmetro pode ser recuperado a partir desta localização de stack.

Uma implementação provê uma função para manipular os parâmetros de extensão de string desconhecida. Esta função diz se o string tem sua localização desconhecida, e retorna valores negativo ou positivo que determinam se os valores encontrados são caracteres de um ou dois bits, respectivamente. A função retorna a zero se não houver string. Pode se determinar se a propriedade do string examinado é aplicável ao Evento de interceptação de chamada 1869 que está sendo processado no script. Se verdade, haverá um InterceptArgument de chamada 1878 no meio do processamento da função chamar gMgEvent 1872. Este é o único instante em que a propriedade de string variável é diferente de zero e aponta para o endereço da estrutura INFO a partir do stack (no endereço de stack).

Fisicamente dentro da estrutura de objeto, há um apontador para si mesmo, e também para a classe hospedeira (o único lugar onde o mesmo é usado), e contém informações a respeito da interceptação, e um apontador para o objeto gMg, que é a Objeto de Função 1860. Todos estas classes são localizadas no Sensor de Interceptação de Método e todas autocontidas.

Referindo-se à figura 19 onde são ilustradas Assinaturas de Método para implementar o Sensor de Método gMg, o Sistema de Monitoramento de Método gMg que circuita através de um arranjo de padrões de byte capturados 1920 do Método alvo chama Assinatura de Método 1930, e o compara com padrões opcopde de byte de função alvo 1910 descobertos por ferramentas de inspeção para detectar uma localização 1905 do Método. Para definições da Assinatura, são criados macros, que são strings de bytes que representam diferentes opcodes. As definições são usadas para documentação de opcodes típicos nas entradas de função. Assinaturas também incluem opcodes especialmente definidos, tal como códigos "não-interessa"

e "fim de assinatura". Por exemplo, bibliotecas específicas, tendo uma versão específica de código compilado, têm uma seqüência fixa de certas instruções em funções determinadas por versões de DLL, que muda em diferentes versões. O código da função é analisado (parsed) para determinar se o código de entrada 1910 é reconhecível, se for, volta um apontador de Assinatura. Esta avaliação também sugere quanto do código precisa ser copiado em algum lugar antes de inserir a instrução jump, e para onde retornar para executar a duração do código, exceto se for encontrada a instrução jump, como parte da função Intercept, e se o gMg detectado já interceptou a chamada de função de aplicação, em quais casos o processo aborta, em virtude de o processo de monitoramento gMg não poder ser recursivo.

A técnica de análise (parse) de opcode também trabalha em classes de diferentes CPUs derivativos. Em aplicações de thread único, uma técnica alternativa pode ser aplicada; restaurar o código alvo original antes de inserir o código gMg, efetivamente desabilitando a interceptação, e depois de retornar do código gMg (manípulo de interceptação), reabilitar a interceptação, colocando a instrução jump de volta no começo do código de função alvo. Para cada Assinatura documentada, o processamento passa através de cada código de byte até localizar um Terminador de Assinatura 1935. Se os valores não coincidirem, o processamento passa para a próxima Assinatura, mas se coincidirem ao longo de todo o valor do terminador de assinatura, então o valor localização de início será armazenado, e sua extensão retransmitida.

Uma vez encontradas as Assinaturas, para uma interceptação bem sucedida é importante localizar e colocar uma instrução jump de código gMg em um limite opcode livre 1955. Uma instrução jump não pode ficar no meio de um opcode, portanto precisa ser conhecida a localização de limites de opcode apropriados. Em uma camada de cinco bytes, os primeiros quatro bytes

compreendem dois opcodes 1960, 1965, e o quinto byte contém o endereço relativo 1970. O endereço relativo impede relocar a sobrecamada e executá-la, pelo fato de o endereço ser relativo a onde o mesmo está sendo executado a partir de sua função original. Esta restrição é superada pela conversão de opcodes relativos a localizações absolutas.

Na função alvo 1940 para uma sobrecamada alvo são necessários cinco bytes 1955 no mínimo (e um fim de assinatura) para aplicar uma instrução jump em um CPU de 32 bit. A sobre-camada também pode ser sete ou dez bytes, não mais que dezesseis bytes. Uma vez encontrado o opcode de cinco bytes, se insere um valor especial de fim de assinatura, que mostra para onde voltar quando se substitui o código de função. As instruções originais são copiadas e as séries inteiras de opcodes são sobrepostas 1945 e substituídas com uma instrução jump para a rotina de interceptação 1975. Os opcodes copiados são valores de byte armazenados em um arranjo de valores curtos 1980. Com o retorno a partir da rotina de interceptação 1975, o fluxo do programa volta para este ponto de interceptação/ continuação de código 1950.

Uma vez que as funções passam um ou mais parâmetros, também há funções gMg que buscam os parâmetros do Método monitorado (i.e. GetString functioncall()), onde os opcodes podem ser de ambos tipos - String e Numérico. Por exemplo, se um dado save event.s1 (s1 é uma propriedade de string) há uma função GetStringEvent Property() que tem um parâmetro que é um texto constante obtido da lista de Propriedades N-Event (NEP). A lista NEP define uma lista de textos possíveis todos eles sendo propriedades de Evento de string. GetStringEventProps() é chamado e passando o número da propriedade de Evento. As células de propriedade de Evento são propriedades de Evento de string, tal como cabeçalhos de estrutura URL no navegador URL S1 a Sn. As propriedades de Evento são enumeradas como argumentos e propriedades de Evento

Boolean, e usadas como argumento no opcode. As propriedades de Evento Numérico são encontradas da mesma maneira e enumeradas como argumentos e nas propriedades de Evento Boolean. Eventos String começam com "1" como propriedade numérica, e as propriedades de Evento Boolean começam com "1000" durante execução.

Para identificar se os parâmetros são ASCII ou Unicode, quer um API (ISTextUnicode) é chamado ou feita uma contagem dos caracteres unicode ASCII com os parâmetros strlengh de unicode. Uma contagem também é feita de caracteres ASCII, circuitando através do ASCII alvo e encontrando a extensão do ASCII total e o número de caracteres no string. Se uma significativa proporção dos caracteres for caracteres de código ASCII, e o string não for longo demais, então poderia ser um string unicode válido. Caso contrário, os caracteres serão considerados provavelmente não sendo um string unicode e zerados. Um string de proporção maior é retornado, onde uma contagem negativa indica um string ASCII, e se apontar um string nulo, um zero é retornado. Se nenhum string for localizado, um string vazio será retornado. A rotina também retorna um apontador para o stack contendo um conjunto de caracteres grandes.

Interceptações podem ocorrer no início ou no meio de um código de função alvo. O monitoramento gMg usa um desvio da base de função de zero e tem a provisão de um desvio de base de função não-zero, onde o ponto de interceptação ocorre no meio das funções. Interceptações de múltiplas instâncias da mesma função também podem ocorrer e ser detectadas.

O código é examinado quanto a instruções jump para um Thunk gMg. Se encontrada, a função já foi interceptada e o processo de instalar a interceptação é abortado; caso contrário, a função é setup para interceptação. Para modificar o código de Método alvo, a funcionalidade é necessária para controlar o bloco de memória contendo o código. A função ChangeMemory é usada para alterar

permissões no bloco de memória contendo a função a ser interceptada. A rotina ChangeMemory gMg muda os atributos do bloco de memória, removendo a operação de execução e instalando a operação de leitura/escrita. A interceptação da função depende de alterar as propriedades do bloco de memória, e falha sem modificá-los com sucesso no código da Aplicação Hospedeira. Um manípulo é usado para retornar a proteção original e restaurar os atributos. Como a mudança de memória pode ocorrer freqüentemente, o processo é dividido para ambos, mudar e restaurar a memória.

O uso de Thunks é uma técnica usada em uma gama de mecanismos de suporte a partir da última ligação de identificadores com objetos reais durante execução, ou rastreamento de Multi-Instâncias durante chamadas de Método. Aplicado em diferentes situações, um thunk pode ser um apontador para a função real como um apontador para um código, ou pode estar no espaço Thunk que retorna para a função real, ou ser o endereço da função real alvo. Apontadores são estabelecidos em um código monitorado para apontar uma interceptação Thunk, que é uma seção de memória com atributos reescritos e executáveis. Thunks são usados para rastrear múltiplas instâncias de chamada ou retorno. Para facilitar suportar a interceptação de um grande número de funções, um código de interceptação pode ser compartilhado, exceto para Thunks de interceptar ou Thunks de reiniciar. Cada código de entrada de função é substituído com um jump para um separado Thunk de interceptar separado *per* função. Entrementes, o Thunk de interceptar supre um apontador para uma estrutura de dados de engine (INFO) que representa a interceptação específico e então chama uma Função Handler comum compartilhada com um grande número de Funções interceptadas. A função Thunks e sua aplicação serão discutidas mais adiante em tabelas virtuais e monitoramento COM.

No caso de uma interceptação de chamada de função

genérica, a Função Handler retorna após processamento de script e Thunk de interceptar passa para o Thunk de reiniciar que tem uma cópia do código de entrada sobreposta da função interceptada original rastreado por um jump para o restante da função interceptada. Para funções específicas interceptadas (onde formatos e parâmetros de função são conhecidos), a Função Handler recebe os parâmetros passados pela aplicação com exatamente o mesmo formato daquele que da função real, e adicionalmente a Função Handler recebe INFOPointer para Função interceptada suprida pelo Thunk de interceptar. O parâmetro INFOpointer adicional pode ser passado pelo Thunk de interceptar, adicionando o mesmo ao stack ou colocando o mesmo em um registro de CPU não usado. A Função Handler chama a função original através do Thunk de reiniciar (que pode ser localizado via INFOstructure) e pode então retornar os resultados da função original para a aplicação que chama. A função Handler notifica o script da chamada junto com uma referência para INFO descrevendo o que foi chamado. A Função Handler também notifica a localização dos parâmetros no stack, de modo que o script possa referenciar os parâmetros como números ou strings. Quando os parâmetros da função interceptada forem conhecidos (uma função publicada, métodos COM em uma biblioteca Tipo COM, etc.) os parâmetros também podem ser referenciados em um script como Objetos COM (ex: "este" apontador) ou Objetos Windows (de handles windows). Para casos especiais, (ex LoadLibraryExW, DIIGetClassObject, e IClassFactory:: CreateInstance), a Função Handler também faz um trabalho adicional antes ou depois de chamar a função original (adiciona um novo Objeto de Módulo, intercepta IClassFactory::CreateInstance, e examina um valor de retorno, respectivamente).

Com thunks, se suportam múltiplos N threads, onde a alocação de thunk precisa ser bloqueada via cada thread. Sendo requerido bloquear com thread porque, por exemplo, dois threads podem chamar diferentes funções ou podem ter

a mesma função, onde cada thread requer seu próprio thunk de retorno. Em alguns casos, cada instância de chamada pode estar chamando a mesma função recursivamente. Ademais, a criação de stacks paralelos para cada thread é mais fácil, se o processamento liberar dinamicamente o que foi criado durante operação, e durante cada chamada usando Thunks, a despeito de a função usar quer uma chamada estilo C ou estilo Pascal. A aplicação de Thunks no Monitoramento de Método gMg é usada para gerar um amplo espectro de Eventos a partir de chamadas e retornos de Método de componente da Aplicação Hospedeira.

Referindo-se à figura 20 onde são mostradas Alocações de Thunks para Monitoramento de Método, quando uma Assinatura de Método for válida será encontrado um Comprimento ≥ 5. Se um valor válido for encontrado, a função VirtualAllocation 2006 aloca um bloco de memória 2009 com atributos de rescrita e aloca dinamicamente 24 bytes por vez. Para alocação de Thunk é usado um bloco reutilizável de memória buffer. Há um buffer fixo no programa para todos strings de expressão, e durante seu cálculo são usados strings temporários livres que precisam ser limpos após cada instrução gMg. Esta verificação precisa ser feita periodicamente. Um Heap gMg privada também é usada ao invés de usar um Heap de aplicação para maior isolamento e não afetar a Aplicação Hospedeira se houver algum erro. Uma função Heap (pilha) privada cria um Heap e então provê um manípulo para todas as alocações, de modo que todos novos operadores têm que ser usados com um novo Heap privado. InterceptCallMultEDX 2003 aloca dois blocos de memória - um para o código interceptar a chamada 2012 e outro para armazenar o código original 2015 para garantir a continuidade de processamento para função alvo. Dentro da seção Intercept gMg, séries de códigos são inseridas - uma instrução PUSHAD 2018, que é uma instrução de avançar todos registros de dados; rastreado de uma ação de um valor longo que é o endereço da INFOStructure 2121 que precisa

ser associado à instância Intercept.

A função InterceptMultEDX() se baseia na premissa que um registro de instância (por exemplo, registro EDX) é disponível para ser usado por um função chamada sem primeiro ter que salvar o registro, para melhorar a eficiência. Uma vez que deve se evitar modificar o stack CPU, deve ser inserido um registro EDX de mover para infodata. Então move o registro EDX para passar a informação de instância para a próxima parte de código. Esta abordagem pode ser usada quando se sabe qual função está sendo interceptada e também para criar um modelo (template) idêntico com exatamente os mesmos parâmetros, tal como a função kernel similar à LoadLibrary.

Informações armazenadas em RAM (i.e., HEAP) não podem ser usadas para o mesmo propósito EDX, porque conflitam com outros threads, em virtude de um acesso compartilhado poder ocorrer, e, portanto, não sendo a prova de thread (thread safe). O uso de registros ou stacks é o único modo a prova de thread de passar informações de instância sem precisar sincronizar ou fazer outras operações mais caras. O RAM pode ser usado para passar informações de instância, mas de modo menos eficiente. Para usar RAM em cada chamada, o interceptador teria que fazer busca com base no endereço da função interceptada. Ademais, a manutenção da tabelas (mapa de endereços de função para instâncias info) teria que ser sincronizada para ser a prova de thread, que pode ser uma operação cara.

Em alguns casos, o stack pode ser usado mas como RAM, esta solução também tem desvantagens com respeito a manter uma transparência. A função interceptação deve substituir completamente a função original e provavelmente chamá-la no processo. Isto pode ser difícil do ponto de vista de controle. Nesta situação, o processo de interceptação deve limpar o stack, similarmente à função original e ter exatamente os mesmos parâmetros, e em algum instante, adicionar um ou mais parâmetros, i.e. Info de Instância ou Contexto para suportar a

interceptação do sistema de monitoramento. Algumas funções alvo são solicitadas por instrução jump, e o stack não pode ser modificado por um botão de apertar (onde o botão de apertar resulta na informação extra colocada depois do endereço de retorno dos parâmetros e endereço de retorno da função alvo). Nesta situação, o código de rotina de interceptação pode não prover um acesso fácil aos parâmetros originais, e não pode limpar o stack como na função original, por exemplo.

```
Stack
...
dwFlags
hFile
IpLibFileName
Return-adress
//um botão de apertar não adiciona um parâmetro. Ele egue o endereço de retorno.
dwFlags
hfile
IpLibFileName
Return-adress
Info// para rotina de interceptação
```

Quanto a uma alternativa EDX, o uso de "PUSHInfoAddr" em vez de "MOVEEDX", "InfoAdrr" pode ser uma opção viável também a prova de thread, mas esta alternativa é mais lenta que usar o registro CPU.

Há vantagens em usar um registro CPU. Por exemplo, passar o endereço de instância pelo registro é ótimo para passar informações de contexto de instância. Esta abordagem somente funciona desde que nenhum parâmetro seja passado para o método/função monitorado pelo mesmo registro de passar INFO. Descobriu-se que passar parâmetros pelo registro EDX nunca é usado para funções Export. No entanto, é possível escrever códigos que causem uma colisão durante o Monitoramento gMg usando EDX, mas provavelmente deve requerer um assembler, e cujo uso improvável com compiladores disponíveis. Esta colisão é

extremamente improvável para funções COM e Export. Ademais, o uso de EDX não funcionaria para interceptação de chamadas de nível kernel que são feitas através de uma instrução "sysenter" ou outras chamadas internas que freqüentemente usam EAX e EDX como parâmetros. No entanto, nestas condições, um outro registro poderia ser substituído. O uso de EDX também reduz o tamanho do código. A invenção, enfim, não é tão limitada que o uso de EDX seja um requisito de interceptação genérica, quando passar no stack é razoável. No entanto, passar um material info de contexto adicional nos registros CPU é muito melhor que no stack ou memória, quando se substitui uma nova função Handler por múltiplas instâncias das originais.

```
Múltipla Instância:
HRESULTSTDMETHODCALLTYPEilClassFactory_CreateInstance
      (IClassFactory*This,
      /*[unique][in]*/Iunknow       *pUnkOuter,
      [in]*/REFIID      riid,
      /*[iid_is][out]*/void   **ppvObject)
      HRESULT_stdcalliDIIGetClassObject(REFCLSID rclsid,
      REFIIF      riid,
      LPVOID*ppv)
Instância Única
      LoadLibraryExW
```

Referindo-se novamente à figura 20, uma vez conhecidos os parâmetros da função alvo 2036, a função é interceptada, e então substituída com uma função "proxy" (ou interceptação) gMg 2045. A função original que chama 2024 permanece bloqueada aguardando os resultados da função. A função "proxy" 2045 chama a função original 2048, 2042, recebe o valor de retorno, e então retorna para a função que chama 2024 com os resultados 2051. O gMg cria uma função "proxy" 2045 exatamente com os mesmos parâmetros da função original 2030, mas adiciona pelo menos um parâmetro extra 2033 para informação de instância. Isto é feito em casos onde

é necessário enganchar (hook) múltiplas funções em diferentes DLLs 2060, 2063, 2066, todas com o mesmo nome de Método 2072, 2075, 2078, e com a mesma Assinatura de função. Para encaminhar informações para as rotinas de interceptação, o registro EDX também pode ser usado para passar um parâmetro gMg INFO extra que evita modificação do stack da aplicação original durante a interceptação de chamada.

A função Intercept gMg monta um botão de endereço da instância da estruturaINFO e constrói um procedimento de interceptar chamada que é especificado para ser chamado durante a interceptação 2045, 4048. Cópias do processo gMg do código original da função sendo substituídas e salvas 2048. No retorno, em seguida, todos registros são acionados, que provoca um jump para o código no Thunk (estrutura que contém um endereço, ou apontador) 2054 para retornar à rotina original de chamar. O código para a rotina original pode ser acessado usando prealproc que aponta para a rotina real.

Durante a chamada a partir do código de Aplicação Hospedeira, uma instrução jump 2039 é montada, que converte (thunks) do ponto de interceptação no código de função alvo original para a rotina de interceptação gMg associada. 2045. O código original é copiado 2048, e então montada outra instrução jump, que converte (thunks) de volta para o ponto de continuação 2042, (no código alvo) quando retorna a rotina de interceptação.

Isto completa a construção dos dois Thunks - um para interceptar e outro para retornar. Em uma implementação alternativa, os dois Thunks são agrupados em um Thunk maior para fazer uma chamada direta da rotina original a partir da rotina de interceptação gMg, ou uma chamada para o endereço prealproc que passa para a rotina original, daí desviando (bypassando) do Intercept.

Há um número de questões que surgem no setup ou no processo de monitoramento de Método e uso de Thunks: conflitos de threads entre aplicações multi-threads e

processamento de Evento em relação à atualização de Árvore de Objeto gMg. Um problema que pode ocorrer quando se modifica ou se sobre-escreve o código Thunk original, é chamar o Thunk e executá-lo, permitindo a possibilidade de inadvertidamente executar o código substituído. Para evitar este problema, o Sistema gMg usa uma técnica que pode ser usada dinamicamente para diferentes situações, nomeadamente suspensão de thread ou dormência (sleep).

Para evitar conflitos de thread durante o monitoramento de Método, a técnica de dormência emite uma função de dormência de N milisegundos para tentar evitar um conflito, se múltiplos threads tentarem executar o mesmo código ao mesmo tempo em que o gMg tenta modificar o código. A dormência suspende o thread e reinicia a fatia de tempo para este thread. Isto denota que o thread tem uma fatia de tempo completa para completar sua operação de reduzir a probabilidade de ocorrer um erro. Como mencionado, a instrução jump é montada como parte da modificação do código original que passa para a rotina de interceptação gMg. Isto é implementado como função macro ou em linha, que coloca no opcode jump e calcula o endereço do desvio.

Uma outra técnica é suspender todos demais threads no processo momentaneamente com uma chamada API. Se o thread provocar uma exceção, esta exceção pode ser interceptada em resposta a alguma mensagem com um hook e permitir que o thread provoque uma exceção, mas forçando o thread reiniciar de modo elegante. Para adicionalmente diminuir a probabilidade de ocorrer problemas, em adição à dormência, a prioridade de thread pode ser elevada. Um outro thread pode ser iniciado, mas tendo uma menor chance de conflito. Sozinha, a técnica de dormência pode ser suficiente, exceto quando se monitora ambientes com múltiplos CPUs. Com múltiplos CPUs, o OS tipicamente aloca o processamento em diferentes processos, durante a interceptação de método/ função.

Com respeito a multi-threading ou funções chamadas pela

tabela Import, um mecanismo de interceptação extra usando tabelas Import basicamente bloqueia a chamada, faz uma mudança de código para interceptação, e então desbloqueia a chamada, mudando os valores da tabela Import. A função GetProcAddress provê o endereço da tabela Import.

Adicionalmente, o Monitoramento de Método é intercalado com o escaneamento da Árvore de Objeto GUI para coordenar com e manter a árvore sincronizada com a Aplicação Hospedeira. Há um mecanismo de controle de escaneamento de árvore e Evento que chama uma cadeia de funções que começa com ObjectInterfaceEPT. GetStringEventProperty () é chamado primeiro e passa o índice, mas se a propriedade não estiver presente, então chama GetStringEvents Property() e em seguida StringGetWebEventProperty(). Antes de a função monitorada ser chamada, antes de a aplicação chamar a função interceptada, faz-se um teste para determinar se o gMg está potencialmente monitorando o Evento corrente. No caso de outros eventos estiverem sendo processados neste instante, e onde houver Eventos detectados, a árvore GUI da Aplicação Hospedeira será reescaneada para atualizar a prioridade de escaneamento da árvore de aplicação. Caso contrário, o único instante em que a árvore de aplicação escaneada é normalmente em Event. Se a aplicação estiver ocupada, muitos eventos poderão ser gerados que estarão sendo interceptados. Ademais, OS ws Timer Event é controlado, que mantém as mudanças da árvore de aplicação em verificação. Em alguns casos, se não tiver sido feito recentemente, quando se processa um Evento gMg, a árvore é reescaneada que faz ocorrer um Evento cria ou destrói em um processamento de outro tipo de Evento. Por esta razão, faz-se uma verificação.

Diferentes tipos de pacotes de componente de aplicação apresentam diferentes estruturas de Método, por conseguinte requerem diferentes métodos para localizar funções/método alvo e suas interfaces. Conquanto COM seja uma tecnologia de componente importante que é usada para

construir uma Aplicação Hospedeira empresarial Microsoft Windows, a detecção de Métodos COM através de VTables para recuperar propriedades ou detectar eventos é uma técnica gMg requerida para monitorar componentes.

Dentro de uma instância de Objeto COM - uma instância particular de uma classe - o primeiro objetivo gMg principal é localizar a VTable do objeto. Se o objeto não existir, a VTable pode ser recuperada durante a fase de criação, mas se o objeto existir, o gMg localiza a VTable através da criação de seu próprio objeto temporário. Quando qualquer novo objeto é criado, o objeto acessa a mesma VTable como objetos existentes já instanciados. O gMg cria um objeto temporário usado para localizar uma VTable existente, compartilhada com um conjunto de Métodos pertencente ao componente. Uma vez localizada VTable, as Rotinas de Interceptação de Assinatura de Método gMg podem ser instaladas para executar o processo de monitoramento.

Classes são usadas para instanciar objetos na memória, e são identificadas por ambos, GUID (um identificador único comumente usado para identificar qualquer componente de programação em Microsoft Windows) e o nome da interface. Encontrado no TypeLibrary, um escritor de script gMg provê o id via nome da interface usando ferramentas de inspeção gMg. Publicado pelo DLL, o id pode ser encontrado em uma utilidade, como gMg Tool Vtree ou OLE View da Microsoft. O DLL contém um tipo biblioteca tendo a descrição do objeto COM. Uma vez encontrado id, o GUID da interface associada à classe no objeto pode ser encontrado. Durante execução, o id da interface do objeto então resulta do script.

Uma classe pode implementar múltiplas interfaces (i.e., IUnknow, IDispatch, etc.). Todos as outras interfaces da classe implementada no objeto são usualmente publicadas no registro. Um id da classe é usado para criar uma instância do objeto para criar uma interface para o objeto alvo. Em caso contrário, uma vez conhecido o id

da interface, será possível obter o id da classe que implementa aquela interface.

Dada a classe id e a interface id uma instância do objeto pode ser criada. A instância é criada por um breve espaço de tempo antes de o objeto ser destruído, o endereço do VTable é armazenado, quando for descoberto na memória. A interface é monitorada apenas depois de o módulo que implementa a interface ser carregado. Um cria Evento gMg no objeto de módulo é recebido - por conseguinte o módulo existe. A interface pode ser monitorada, enquanto a VTable agora é instanciada na memória. O compilador foi previamente criado e no tempo de compilação gera todas as possíveis VTables que estão no código fonte durante execução.

Uma VTable, que é compilada como tabela fixa contendo apontadores para o código, é compartilhada por todas instâncias. Quando uma instância do objeto é criada, um bloco de memória é alocado com o primeiro DWord apontando para VTable. Por exemplo, se houver dez objetos do mesmo tipo, todos estes objetos compartilharão a mesma VTable, onde a VTable atua como interface.

Com respeito a compilações em diferentes linguagens de programação, por exemplo, enquanto as interfaces de programação C usam estruturas de dados para implementar seus Métodos, os componentes em C++ usam funções virtuais. Como discutido acima, Funções Virtuais são implementadas como VTables e contêm apontadores para os correspondentes Métodos. Ademais, a Vtable inclui um número de interfaces como parte da definição COM, (i.e. IUnknow, IDispatch, e outros conjuntos customizados opcionais). Dado um objeto COM que pode ter múltiplas instâncias, todos conjuntos de interfaces são derivados de IUnknow. Ademais, a derivação determina um conteúdo da VTable. Por exemplo, se uma interface é derivada de Invoke, então a VTable contém um apontador para um apontador Invoke para um Método. Mas se uma interface não deriva de IDispatch, e se for executada QueryInterface,

uma diferente VTable será construída.

Como parte de IUnKnow, que depende de diversos fatores, tais como classes hereditárias múltiplas, as primeiras três entradas da VTable - figura 21, 2106, sempre aponta para os métodos da interface IUnknow- QueryInterface rastreada de AddRef e Release. O próximo conjunto de interfaces derivado de IUnknow é IDispatch contendo Invoke, que é um método genérico para acessar um conjunto de Módulos dentro do Componente.

VTables nem sempre são únicas, conquanto pode haver mais que uma VTable apontando para o mesmo Método, tal como no caso de hereditariedade múltipla. Ademais, poderiam haver outras Vtables, localizadas em algum lugar, conectadas ao mesmo conjunto de Métodos. Por exemplo, poderia haver duas VTables que precisem ser retornadas (i.e., Método X). O Método X pode ser disponível com índice N em uma VTable com uma particular interface, ou também poderia ser disponível como um Dispatch id Q solicitado para uma diferente tabela.

Há um número de questões quando se usam VTables. Interceptar VTables contendo apontadores para funções é um método de monitoramento de chamadas de Método. No entanto, uma interceptação VTable de apontadores VTable pode ser inadequada em algumas situações, porque podem haver chamadas internas em uma chamada de função que pode desviar a VTable. Outros aspectos incluem que a Aplicação Hospedeira pode chamar a função diretamente sem usar VTables para localizar a função, também pode haver múltiplas Vtables todas apontando para a mesma função. Isto implica que o sistema de monitoramento tenha que interceptar todas VTables relevantes para dada função alvo, que é proibitivo, ou acessar todas VTables, que pode ser difícil. Com uma interface conhecida (iKnown), QueryInterface pode ser usada para localizar múltiplas Vtables, mas implica em um considerável sobre-custo de processamento. No entanto, a QueryInterface é um método alternativo, quando puder ser usada uma técnica de

interceptação de função de assinatura digital.

IDispatch, uma parte de "Automation" (da Microsoft Corp.) provê interfaces para componentes de aplicação onde "Automation" é usada para executar uma ligação atrasada. (i.e., as funções a serem chamadas são determinadas durante execução e não compilam o tempo). IDispatch é um derivado de IUnknown. Contido na VTable, o IDispatch terá uma interface dupla se Invoke e interface direta de Método, forem ambos disponíveis.

O Método é especificado, em "Automation" (tecnologia de componente da Microsoft) que invoca o Método indexado usando o Método Invoke genérico - parte do IDispatch. O Invoke executa um Método Específico, e tudo é mapeado para um Método para recuperar ou estabelecer uma propriedade. Quando não for uma implementação de interface dupla, o Invoke pode ser mais lento devido a fatores tais como ter que preparar todos os parâmetros, ou a necessidade de setup uma lista de variantes antes da chamada. No entanto, se for disponível uma interface dupla, há uma Função Virtual e um apontador para a função localizada na Vtable, que aponta para um Método localizado no modo real.

O primeiro objetivo é interceptar a criação de novas instâncias de objeto COM, localizar a interface IKnown e determinar se estão presentes funções Virtuais, mesmo se não derivadas de IDispatch. O cliente chama a função exportada DLLGetClassObject() que pede uma interface de classe fábrica. Uma vez obtida uma interface de estrutura de classe, o cliente chama o Método CreateInstance(), que requer o parâmetro para acrescentar um id do GUID da interface ao objeto a ser instanciado. A chamada para CreateInstance também é interceptada como parte do Monitoramento de Método gMg. O método requer como parâmetro, o GUID da interface que a instância a ser criada implementa. Quando o Objeto DLLGetClass completa o processamento, retorna o IKnow, o GUI da interface, e também ThisPointer a partir do qual pode ser encontrado

o apontador para a VTable.

Se a interface não derivar do Método IDispatch, implica que a aplicação está usando um ligação anterior, ou seja as funções a serem chamadas são conhecidas e vinculadas no instante da compilação, todas elas suportadas pelo Sistema gMg. Freqüentemente, o apontador para IDispatch tem a conotação de instância do objeto. Se uma interface deriva de IDispatch, ela cria sua própria Vtable, mas, ao contrário, se a interface não deriva de IDispatch, será criada VTable uma completamente separada.

Depois de encontrada a criação do novo COM, o próximo objetivo será localizar e acessar o código de Método real que implementa a funcionalidade em um componente COM. O Método COM real realiza uma ação específica e pode ser acessado quer diretamente ou via Invoke, onde tudo é acessível por um comando comum id. Algumas vezes, Métodos também podem ser expostos e chamados diretamente. Se em um TypeLibrary, um objeto for declarado interface dual em uma tabela virtual, e onde houver um apontador para uma tabela virtual, a tabela poderá ser verificada para localizar o endereço das funções alvo.

O método é realmente único no contexto de uma particular interface, tal como em um exemplo de controle de calendário de componente. Se houver um GUID da interface, a VTable e sua interface serão conhecidos. IKnow provê a VTable do Método, se este estiver disponível. Ademais, deve ser feita uma QueryInterface para obter IDispatch, que pode retornar uma diferente VTable.

A figura 21 adicionalmente ilustra objetos COM e suas interfaces de automação internas e VTables. Conquanto possam existir muitas instâncias de um objeto COM 2109, 2112, 2115, as instâncias compartilharão a mesma Vtable, que por sua vez compartilha o mesmo código 2135. Dentro do DLL 2115 há um apontador 2106 apontando para a VTable do DLL. Uma Função Virtual localizada na VTable 2136 é um apontador que faz parte da estrutura que indiretamente encontra o método (i.e., 2139, 2142, 2145). Por serem

virtuais, as novas classes podem derivar de diferentes classes para formar novas classes. Por exemplo, uma nova classe pode derivar da interface IDispatch 2124. Durante a compilação, a VTable original pode ser copiada, exceto pelo fato de um novo apontador apontar para uma classe Invoke recém derivada. Classes fixas derivadas de classes declaradas e com métodos virtuais puros devem ser providas com a implementação destes Métodos definidos durante a compilação e substituir suas respectivas definições de interface. Se os Métodos não forem providos, podem ocorrer erros de compilação. A porção da figura "Detalhes de Vtable" provê uma vista em close-up 2118 da VTable 2136. IUnknown 2121 e IDispatch 2124 de COM contendo QueryInterface 2127, AddRef 2130, Release 2133, e Invoke 2134 são específicos para a classe que está sendo criada e deve ter sua respectiva criação vinculada durante a compilação do componente.

A figura 21 representa uma rotina de interceptação específica para o Método Invoke, como parte do monitoramento de uma Função Virtual. Exatamente os mesmos parâmetros da função original são copiados e usados durante a chamada de gMg para a função monitorada, assim como o parâmetro gMg implicado. Pode haver muitos diferentes Invokes derivados contidos no componente, dependendo do projeto de implementação. Por exemplo, pode haver qualquer número de Métodos Invoke derivados de IDispatch (Invoke 2148 classe A, Invoke 2154 classe B, Invoke 2157 classe C, etc.), assim como tantas quanto chamadas para qualquer Método de classe Invoke. Diferentes chamadas para os respectivos Invoke (A, B, C) fazem a rotina de interceptação gMg alocar cada um dos Thunks 2169, 2172 contextual separado a qualquer número de chamadas 2175.

Considerando o Invoke classe A de exemplo, a interceptação de chamada gMg faz o programa se mover da redireção em 2163 para alcançar o Thunk de classe A 2169 apropriado para a chamada contendo o dado de contexto

para as estruturas de dado de instância de chamada específica. Um diferente Invoke derivado de IDispatch teria um diferente Thunk, por exemplo, com informação de Invoke B 2172. No entanto, a rotina Handling Intercept gMg 2166 é idêntica para todas instâncias e atende todos os Métodos VTable 2135. Ademais, o processamento gMg que rastreia da rotina de interceptação 2166 para a apropriada destinação de Método 2148, recupera a informação de contexto de chamada apropriada e o código 2178. Este será o apontador de destinação apropriada contida em uma estrutura de dado contextual. A informação de contexto é mantida em um Objeto gMg, que contém um apontador para um Thunk, localização de continuação de função de monitor 2160 (depois de inserida a instrução de jump de entrada pelo processamento de interceptação) e outras informações.

O fluxo de interceptação gMg é o seguinte: o setup do código Intercept ocorre através da técnica de usar um objeto gMg temporário. Então, quando o chamador 2103 invoca a função alvo, o processo de setup de interceptação se sobrepõe a uma parte do código original no ponto de entrada da função monitorada 2163 em 2148, mas mantendo a cópia do código sobreposto armazenado em 2178. Para executar a função real, a chamada de interceptação recupera a porção de código de entrada de 2178, e primeiro a executa, antes de passar para a localização da continuação 2160. Para o gMg executar sua própria chamada para realizar uma tarefa, tal como recuperar algumas propriedades da chamada de Método, a execução da chamada ocorre em 2178 pela interface interceptada estabelecida. Os Eventos gMg serão gerados, se o Método monitorado for chamado pela Aplicação Hospedeira. No entanto, o gMg tem um mecanismo de proteção, que impede a interceptação de uma chamada, se uma chamada for iniciada pelo interpretador gMg, verificando um bloqueio gMg global (figura 6). Somente a Aplicação Hospedeira origina chamadas que se tornam

Eventos de Método gMg.

Os parâmetros originais detectados na entrada da função monitorada são capturados e transferidos para o código original da função pela rotina de interceptação gMg. Resulta que a rotina de interceptação gMg recupera os parâmetros reais do stack. Com o retorno a partir do Método, os parâmetros são recuperados primariamente de registros ou apontadores indiretos nos registros.

Embora as discussões anteriores na seção de Monitoramento de Método tenham focalizado a pré-criação ou pós-criação de objetos de memória para localizar e monitorar chamadas de Método, o monitoramento de retorno de Método terá igual significância no sistema de movimento gMg. Além de capturar chamadas, as técnicas de capturar valores de retorno de Método são úteis para capturar e transmitir os resultados de Método para o Sistema de gMg para um processamento posterior. O monitoramento de retornos de Método produz os resultados das chamadas de Método, que são disponibilizadas para o script gMg para manipulação em muitos ambientes. Os princípios de interceptação de retorno de função podem visar muitos diferentes tipos de alvos em pacotes de componente - havendo dois métodos de captura da função retorno: a) método Cópia de Stack (Stack Copy) que compreende fazer uma cópia do stack; e b) método Alocação de Stack (Stack Allocation) que aloca e desloca dinamicamente Thunks para preservar contextos de chamada e retorno.

Para monitoramento de um retorno de Método, a primeira técnica Cópia de Stack se baseia em copiar uma seção do stack (usando a estrutura de stack do chamador) e usa isto para afetar o fluxo do programa da função alvo de/para as rotinas de interceptação de chamada e retorno gMg. O método Cópia de Stack intercepta o stack sem modificar o stack original. Fazendo uma cópia de uma seção do stack e interceptando pontos de execução do retorno de uma função, a execução de programa se move para uma rotina de interceptação gMg que detecta um

movimento de stack e executa operações de stack dentro da versão do código de gMg do stack. Dado que o Sistema gMg é transparente e não requer nenhuma modificação no código de fonte da aplicação, e que o componente é privado, se presume que não há nenhuma documentação disponível para a função monitorada alvo, e se desconhece quantos parâmetros estão sendo passados para a função. Por conseguinte, quando copiada, esta seção de stack capturada (grande suficiente para capturar N parâmetros) contém ambos parâmetros necessários de função e algum dado adicional que muito provavelmente não é relevantes para o monitoramento.

A técnica "Cópia de Stack" contém o endereço de retorno que aponta para o código de interceptação gMg que reconhece quanto o apontador de stack foi deslocado. Por exemplo, se o código gMg adiciona vinte palavras (como na configuração pré-estabelecida) então no retorno para o código gMg adiciona vinte ao apontador de stack, que posiciona o apontador de stack no lugar apropriado no stack. Uma cópia é feita de todos os parâmetros de função no stack, e a função retorna a chamada com um endereço de retorno. Finalmente, quando completada a chamada para o método, a cópia dos parâmetros no stack é removida antes de retornar para o chamador original. A abordagem deste exemplo é sempre copiar vinte palavras de parâmetros em cada retorno. No lugar de onde os parâmetros estão sendo passados e chamados do destruidor, UnInterceptCallEDX() desengancha a função, libera qualquer memória usada, e restaura qualquer memória salva para o estado original, se necessário, e quando a contagem de referência do objeto gMg se torna zero, ele destrói o objeto e libera o Intercept. Questões de tempo devem ser consideradas durante o estabelecimento da rotina de interceptação.

Referindo-se à figura 22 onde é ilustrado o Retorno de Monitoramento com a técnica "Cópia de Stack", um chamador 2205 chama o Método que está sendo monitorado. Quando

chega, o stack 2210 pode incluir os parâmetros 2215 do Método junto com o endereço de retorno 2220 do chamador, através do que o fluxo de programa continua depois de completada a chamada de Método.

No entanto, quando o método é monitorado a partir do estágio setup de monitor gMg (com respeito à figura 18), o endereço de entrada da chamada de função monitorada já foi localizado, e a entrada foi modificada com uma instrução jump 2230 durante a fase de chamada da interceptação. Antes de sobrepor o código original, os opcodes da função sobrepostos são mantidos e disponibilizados para a chamada interceptada para a chamada original de código de função 2245. Assim, quando a aplicação chama a função monitorada 2225, o fluxo de programa insere o endereço do código da função. Por causa de a entrada ter sido modificada, a instrução jump gMg 2230 é executada, que passa o fluxo de programa para a rotina de setup de interceptação gMg 2240. Primeiro, a rotina de setup de interceptação faz uma cópia com respeito de uma seção do stack 2250. O Sistema faz uma premissa com respeito ao número máximo de parâmetros em uma chamada de função monitorada, que determina o número de bytes do stack a ser capturado. Com base nisto, o gMg usa um tamanho limite configurado para capturar uma porção do stack com a entrada para função. Esta porção de stack contém os valores de parâmetro de função 2255, o endereço de retorno original 2260, mais algum dado adicional. Outra etapa neste estágio de setup de retorno é modificar o endereço de retorno 2260 na cópia de stack, que resulta em um apontador para execução de captura de retorno 2280 que ocorre antes de o fluxo de programa retornar para o chamador 2205. Neste ponto, é feito o setup da captura de retorno, e a captura dos valores de retorno de Método fica pronta para executar e rastrear.

Para executar a chamada para a função monitorada, são recuperados e executados os opcodes sobrepostos pela instrução jump 2230 durante o setup da rotina de

interceptação gMg antes de passar para a ponto de continuação do código 2235 do Método. O código remanescente da função monitorada agora é executado, incluindo os opcodes de entrada inicial (sobrepostos pela instrução jump da rotina original), assim como as operações necessárias de stack para passar parâmetros a partir do chamador original 2205. As operações de stack são realizadas na cópia da rotina de interceptação do stack 2255, em vez do stack original da aplicação 2215. Quando completo, o fluxo de programa chega no fim do Método e executa a instrução de retorno localizada na cópia de stack da rotina de interceptação gMg 2260. Esta instrução leva o fluxo de execução para o Código de Captura da de Retorno 2280 da Rotina de Interceptação. Neste momento, os valores de retorno são capturados quer a partir dos registros ou apontadores encontrados a partir da cópia de stack 2250 e armazenados 2265 em um dispositivo de armazenamento acessível ao chamador, e então um Evento gMg é emitido para o interpretador 2270. No nível de script gMg, se o script se refere ao objeto de Evento corrente da interceptação de chamada de função, o mesmo pode ser designado a uma variável de script, então a variável pode ser referenciada. Provido que a variável designada faça parte de uma condição (para uma detecção de chamada ou retorno), se for recebido o Evento de chamada (ou retorno) de Método, a lógica do script encaminha a execução do script para a ação apropriada. A etapa final é executar um retorno para o chamador original 2275 pulando (jumping) o stack original que contém o endereço do chamador 2205. A execução do código de retorno aparece de modo normal, i.e. ajustando o registro de apontador de stack com a quantidade copiada no início do processo de interceptação e o fluxo de programa é reiniciado dentro da Aplicação Hospedeira.

Alternativamente, o Método de Alocação de Stack provê um outro método para capturar valores de retorno. Diferentemente do Método de Cópia de Stack, no Método de

Alocação de Stack não é feita nenhuma cópia de stack do chamador. Em vez disso, provê-se uma alocação dinâmica de pequenos Thunks para rastrear os contextos de chamada e os respectivos retornos. Para rastrear valores *de Retorno*, um Thunk contém o endereço de retorno modificado no Stack que aponta para outro Thunk que examina os valores de retorno. Diferentemente de Thunks que suportam monitoramento de chamada, o monitoramento de retorno usa Thunks que sejam dinamicamente gerenciados, ou seja, a cada chamada e retorno eles são alocados e deslocados. Neste Método de Alocação de Stack há vários estágios: primeiro, a entrada para função monitorada a partir do chamador; em saguida, o setup da rotina Intercept de retorno que ocorre durante a chamada para a função monitorada; a chamada para a função monitorada original; o retorno e a captura dos valores de retorno; e por fim, a limpeza e o retorno para o chamador.

O Método de Monitoramento de Retorno de Alocação de Stack mostrado na figura 23 aloca dinamicamente a memória Thunk, onde o Thunk é usado para redirecionar o código e preservar os contextos de chamada ou retorno para preservar a continuidade da Aplicação Hospedeira. Quando uma função é interceptada, o código original sobreposto é salvo em um espaço Thunk, rastreado de uma instrução jump, que reinicia a execução da função monitorada. Diversos Thunks são setup, um para fazer a interceptação e outro para reiniciar a execução, depois de entrar na parte inicial da interceptação, e outro ainda para manipular o retorno.

Por causa do aspecto compartilhado de um dado Método, são necessários contextos de interceptação para manipular o roteamento apropriado de chamadas e retornos de Aplicação Hospedeira, usando dados individuais (i.e. Thunks de chamada 2230, 2227, 2354, 2357, 2342, 2351) que descrevem os detalhes das diferentes instâncias de chamada. Por exemplo, se dadas duas funções de Aplicação Hospedeira X e Y, ambas chamando o Método Q, então

a rotina de interceptação gMg P rastreará dois endereços de retorno diferentes - um para a função X e outro para a função Y. Mantendo o rastreamento dos diferentes contextos para garantir preservar o fluxo de programa e continuidade para diferentes funções Aplicação Hospedeira é uma função chave dos Thunks gMg.

O chamador 2303 chama a função monitorada 2315, e encontra a instrução jump 2318 que foi sobreposta na entrada do Método monitorado. A instrução jump alcança o código que inicia o processo setup da rotina de interceptação de retorno para manipular informações dinâmicas necessárias para realizar com sucesso a interceptação. O endereço de retorno 2312 é modificado no stack 2306 que também contém parâmetros 2309 para apontar para um Thunk que provê uma conexão jump 2345 para ambos deslocamento de Thunk compartilhado gMg e rotina de Captura de Retorno 2366 que se conecta à porção de retorno 2348 do contexto de chamada de Método, onde o contexto rastreia diferentes instâncias de chamada. Este Thunk alocado dinamicamente contém o endereço de retorno original 2348. Outra ação do setup de rotina de interceptação de retorno durante a chamada é a alocação de outro Thunk 2330, que provê uma informação de contexto para a instância de chamada corrente e também faz parte do setup da fase de retorno (o retorno será redirecionado *per* instância). O processamento então alcança a rotina de alocação compartilhada 2336, que aloca o Thunk 2342 que armazena o endereço de retorno original do chamador para a instância corrente e setup um vínculo para o processo de captura de retorno 2360.

A chamada da rotina Handling 2333 para o Método monitorado é executada 2339 chamando o Thunk 2354 que armazenou os opcodes do Método monitorado que foram sobrepostos pela instrução jump no ponto de entrada para o Método 2318. A execução do Método reinicia no ponto de continuação 2321 e continua até o processamento alcançar a instrução de retorno 2324, onde é completada.

Relembrando que o endereço de retorno original foi modificado 2312 durante a fase de setup de interceptação de retorno, a execução do programa agora alcança a rotina gMg 2345 que contém um vínculo para o código de captura de retorno, em vez de retornar para o chamador 2303. O fluxo agora chega ao fim do processo de captura 2366, onde Thunks são deslocados 2363, os valores de retorno são capturados 2360, e os Eventos gMg são emitidos para o interpretador gMg. No nível script gMg, se o script se refere ao objeto Evento corrente da interceptação de chamada de função, este pode ser designado a uma variável de script, então a variável poderá ser referenciada. Uma vez que a variável designada é parte de uma condição (quer para detecção de chamada ou retorno), se o Evento de chamada (ou retorno) de Método for recebido, a lógica do script encaminhará a execução do script para a apropriada. Com a saída capturada, o programa rastreia para o endereço de retorno original do chamador 2348 e um jump indireto manipula o retorno para o chamador 2303.

No Método de Alocação de Stack, Thunks gMg atuam como stack *per*-thread que aloca/desloca Thunks por ordem de stack, que precisam se reconhecerem, uma vez que a informação precisa ser pareada com o endereço de retorno. Um bloco de memória é pré-alocado para ser usado para alocação de memória incremental para uma operação eficiente, e suas propriedades são ambas - reescritas e executadas simultaneamente.

Em resumo, o monitoramento de Método gMg provê acesso a um amplo espectro de componentes a partir do sistema operacional para aplicações, e provê acesso a funcionalidade que pode ser quer publicada/documentada (para ferramentas não-gMg) ou não-publicadas/ não-documentadas. O Método de monitoramento é implementado como um Sensor adicional para o Sistema gMg que, junto com o Sensor GUI, se comunica com o interpretador gMg para prover integração, expansão, ou correlação com contextos GUI únicos e fontes de informação adicional que

são usadas para descrever comportamentos de Aplicação Hospedeira. Conquanto COM e funções Export sejam descritas nesta, os princípios aplicados ao sistema de monitoramento de Método gMg se aplicam a muitas outras tecnologias, e podem ser facilmente adaptadas a modelos de abstração de gMg.

Neste estágio, todas fontes de monitoramento gMg são geradas em forma de Eventos gMg, que produzem uma combinação de diferentes tipos de dados, especialmente todos dentro de um contexto GUI. Eventos de baixo nível, i.e. eventos detectados no ponto de uma fonte gMg, podem por sua vez ser processados por máquinas de estado em diferentes estágios de processamento nas localizações mais aplicáveis e eficientes. Em princípio, tão logo alguma coisa seja conhecida, deve ser considerada imediatamente para processamento de estado, dado o requisitos. Com este propósito, dentro do Sistema gMg, a execução de máquina de estado é disponível para ambos processamentos - Localizado e Remoto.

A figura 24 ilustra a hierarquia de máquina de Estado gMg. Dados ambos, os mecanismos de Máquina de Estado Remoto e Local, o Sistema gMg tem a capacidade de construir hierarquias de Máquina de Estado distribuídas para prover máxima flexibilidade e oportunamente suprir respostas em tempo real pela rede. Aplicações Hospedeiras alvo monitoradas 2418, 2421, 2424 são representadas nesta como tendo Sensores gMg injetados neles para detectar e coletar Eventos. Uma vez detectados, os mesmos são enviados como mensagens gMg para os respectivos interpretadores 2412, 2413, 2414, onde cada um deles executa um script (por exemplo, 2409). Por exemplo, o script pode definir uma Máquina de Estado "localizada" 2403 pequena e rápida que pode converter Eventos de disparo de baixo nível (detectados com Sensores gMg) em Eventos lógicos. Uma vez processados, estes Eventos lógicos são arranjados em pacotes como uma ou mais mensagens de rastreamento 2415, recebidas por um Máquina

de Estado remota (agora, um segundo nível de processamento), em geral no servidor de coleta gMg 2406. Estes Eventos lógicos, por sua vez, são adicionalmente processados por outro(s) conjunto(s) de Máquinas de Estado para produzir um Evento lógico multi-nível, com base nos sucessivos conjuntos de estados que podem ser passados adiante como resposta de detecção 2427.

Este processo é adicionalmente ilustrado com a Máquina de estado 2432 dentro da hierarquia 2430. Esta Máquina de Estado única (em close-up) tem uma série de Eventos de disparo de entrada 2445, 2448, 2451 e uma saída 2454. Esta saída por sua vez pode ser suprida a outras Máquinas de Estado que fiquem no mesmo script localmente 2409 ou remotamente 2406. A árvore hierárquica de Máquina de Estado 2430 representa uma coleção de Máquinas de Estado (i.e., 2430, 2433, 2436, 2439) em qualquer combinação de máquinas de estado local e remoto com saídas compatíveis com as entradas para formar um mecanismo de detecção de estado flexível com base em Eventos gMg. A capacidade de vincular as Máquinas de Estado de forma distribuída permite uma rápida detecção e roteamento de Eventos em um servidor centralizado, uma vez que a maior parte do processamento de estado pode executar localmente e imediatamente com uma detecção, em conseqüência de uma ação de usuário ou Aplicação Hospedeira local.

Dependendo dos requisitos, as Máquinas de Estado podem ser executadas no cliente (Local) para reduzir significativamente os requisitos de recurso de servidor em grandes bases de usuário. Comandos que preservam o estado entre Eventos facilitam este processamento otimizado para alertas em tempo real e outras operações sensíveis a tempo, assim como reduzem o tráfego de rede. Para conseguir isto, a linguagem de script gMg tem um comando Sequence que manipula o estado no computador cliente ou no espaço de cliente virtual como em serviços de terminal. O comando Sequence gMg detecta diferentes estados da Aplicação Hospedeira alvo como Eventos

descritos em expressões lógicas contidas na sintaxe de comando Sequence.

A seguinte Amostra de código ilustra o uso do comando Sequence para rastrear um grupo de estados que resulta em uma ação única, nomeadamente comando Track (Rastrear).

Amostra de Código:

```
init
{
função filesavemenuitem =
função savedialogmydocument =
função savedialogsavebutton =
função savedialogfilename =
var filename= nostring
{;
if(sequence(
wlbutdown&&filesavemenuitem,        //(1)
wlbutdown&&savedialogmydocuments:filename=      // continua
      findsibling(savedialogfilename).title//(2)
wlbutdown&&savedialogsavebutton))//(3)
track(saveend, topwindow, "File Save to My_Document Completed",
file = filename);
```

No exemplo de código, "left botton down" é apertado em diferentes itens de menu de diálogo aberto. O comando state busca uma seqüência de Eventos "left button down" no item "file menu open" (1), rastreado de um diálogo "open file save" (2) e por fim um "save command executed" (3) com disparo de Eventos SaveEnd e SavedMyDocuments Completed. A expressão IF deste exemplo contém uma série de condições, cada condição representando objetos GUI detectados como série de Eventos. O interpretador recebe cada Evento e mantém o rastreamento do estado entre Eventos, onde a seqüência agrega estados em uma ordem determinada pelos comandos. Com todas condições satisfeitas, a condição se revela verdadeira, e então a execução cai no corpo do comando Sequence e executa seu conteúdo. Em geral, o conteúdo é uma única mensagem de rastreamento, que reflete os estados e as condições

atendidas pelo comando Sequence e ou outras ações.
Outros componentes do comando Sequence permitem operações complexas, como OR's ou Operações Compostas, que são Seqüências Ordenadas ou Seqüências em qualquer ordem. Para All-Of Sequence, a condição do grupo Sequence para os Eventos A, B, C, se torna verdadeiro, então o grupo Sequence de condição será verdadeiro. Para Any_Off Sequence, se qualquer dos Eventos A, B, C ocorrer uma vez, então a condição de grupo Sequence de grupo se torna verdadeiro. No nível de seqüência, a operação permite a combinação de comandos Sequence com operadores Boolean. Ademais, OU/ E/ NÃO podem ser usados em qualquer combinação. Comandos Sequence também podem ser acomodados, i.e. um comando Sequence pode conter um outro comando Sequence, como no fragmento de código abaixo:

```
If(Sequence(S1a, Sequence(S2a,S2b, S2c),S1b))
{
Ação...
SendTrackMsg(Q)
}
```

Neste exemplo, o segundo comando Sequence está contido na condição clausula do primeiro comando Sequence. Durante o processamento, o interpretador encontra o primeiro Sequence 1 e setup o rastreamento de estado de suas condições. Uma de suas condições é outro comando Sequence 2, que por sua vez setup seu próprio rastreamento de estado separado. O comando Sequence 2 aguarda Eventos até completar sua seqüência lógica, através do que se torna verdadeiro. Quando Sequence 1 for avaliado de novo, sua expressão completa sua lógica, e o processamento continua. Em geral, a operação inteira é recursiva N vezes em qualquer nível de recurso aceitável (memória, CPU, etc.) permitido pelo ambiente de computação.

Ademais, um gerador de código, tal como um gerador de código Java com script gMg de entrada, pode ser usado para gerar e iniciar máquinas de estado, quer em um

componente cliente em várias configurações como script gMg, ou como um componente de Máquina de Estado de servidor de rastreamento e monitoramento gMg. O beneficio global deste mecanismo de Máquina de Estado Localizado e configuração flexível é a dramática redução de requisitos de recurso de servidor e tráfego de rede.

Referindo-se à figura 4, que ilustra uma visão geral do componente aberto, é mostrado que a aplicação gMg gera mensagens de rastreamento ou Eventos gMg coletados e recebidos pelo servidor de coleta gMg e banco de dados de rastreamento. Neste estágio, o processamento de análise gMg é implementado com base em Máquinas de Estado Remotas projetadas para um processamento flexível e eficiente. Na entrada do mecanismo de processamento, os Eventos são salvos no banco de dados e processados imediatamente por máquinas de estado na memória ou carregados para execução posterior, com base em Eventos de Máquina de Estado de programa (em tempo real ou em carga (batch) em outro momento). Ademais, na análise de Máquina de Estado há um número de mecanismos para detectar múltiplas instâncias de jogos de objeto GUI, quer venham de scripts gMg separados ou de conjuntos de Objeto GUI a partir de um único script.

A configuração inicial das Máquinas de Estado é definida em um arquivo XML que usa o paradigma de um diagrama de Estrada de Ferro. O diagrama Estrada de Ferro descreve processos ou condições paralelas de forma sucinta e normalmente bem conhecida na técnica. A rotina iniciação é chamada, quando o XML primeiro é carregado a partir de um arquivo, e um conjunto de Máquina de Estado inicial é criado. A rotina Parse verifica cada elemento, Propriedades, elementos "filho", etc.. O arquivo XML descreve as propriedades de uma solução de Máquina de Estado servidor com blocos que definem o conteúdo e transição de Máquina de Estado. O bloco incorpora o conceito de container que é uma estrutura no XML que ajuda a combinar múltiplos elementos para uma relação

lógica. Um container é uma estrutura Boolean, tal como OU, onde OU é um container para transições. O container é único para todos seus "filhos".

Termos de programação em bloco descrevem manipulação ou ações lógicas de Processo de Evento, como link, block, goto, transition. "Transitions" são guardados por condições que consistem de expressões Boolean ou expiração de ausência.

Geralmente, os estados são definidos na Máquina de Estado mantendo o rastreamento de onde uma dada mensagem ou Evento de rastreamento está sendo correntemente processado em uma dada Máquina de Estado. Dado um estado, a próxima destinação de transição depende do conteúdo de um subseqüente Evento definido nos campos de um registro de mensagem de rastreamento. Um Evento recebido tem uma localização de processamento que se torna Estado. Se um Evento corresponde ao critério de transição, uma localização de estado será alcançada e se torna o novo estado. Por conseguinte, um estado passa a ser o resultado líquido de uma execução de transação. O estado corrente depende do Evento de estado previamente encontrado, e muda seu critério de transição para corresponder a qualquer dado Evento corrente.

Dentro da sintaxe gMg para suas Máquinas de Estado, os critérios de transição são vinculados juntos com comandos link e goto. Um goto executa uma mudança de estado, enquanto o link não. Um link é uma ramificação adicional, enquanto um goto muda o estado. Um link atua como uma trajetória divisora paralela (comandos IF paralelo) para avaliar e rotear Eventos que entram no Sistema. Links criam possíveis trajetórias ou transversais para a próxima transição, mas não causam nenhuma mudança de estado. Labels podem ser usados como destinação a partir de um comando goto (estado) ou uma destinação a partir de um comando link (não-estado).

Outro comando, a declaração AnyTime é similar a um listener que aparece no fim do bloco, e executado cada

vez que sua condição corresponde a um evento que chega, e o estado corrente está dentro do bloco. O comando AnyTime pode ser comparado a um OR para todas as transições (i.e. dado um bloco com um AnyTime no fim), quando o estado está em algum lugar no bloco, o processamento verifica todos os Anytime executados em adição às transições. AnyTime é usado na situação de alguma coisa precisar ser feita independentemente do estado preciso em um dado bloco. O AnyTime atua como um listener. Se o processamento alcançar um bloco e um Evento for recebido, pode ser interessante no Evento, mas o estado não é necessariamente afetado pela execução de AnyTime dependendo das condições e dos critérios. Em um dado estado, quando outro Evento for recebido, são verificados todos AnyTimes em um dado bloco e todos blocos "pai". Então uma chamada é feita a todos AnyTimes que correspondam ao Evento e se verificam todas as potenciais transições que se rastreiam. A primeira que corresponder será executada. Em cada novo Evento, há um potencial para realizar uma transição de estado e executar muitos blocos AnyTime. Um comando AnyTime é avaliado uma vez para cada condição na lista de blocos, e uma ação resultante será executada uma vez a cada correspondência.

Por exemplo, se diversas paginas serão monitoradas e medidas quanto ao tempo de resposta, o comando AnyTime permitirá a programação de medições com poucos comandos. As páginas alvo são listadas em um comando de bloco e um comando AnyTime é adicionado. O comando AnyTime então aplica seu processo de medição a todas as paginas alvo por Eventos definidos e enumerados na lista no comando de bloco, independente da ordem em que se passam as paginas. A seqüência é dada por página - topic = página 1, topic = página 2, topic = página 3, etc. e as paginas representam uma seqüência de transições, independentemente da página em o usuário se encontra, o comando AnyTime resulta uma medição de tempo de carregamento de cada página. Outro exemplo de aplicação com AnyTime, é a contagem de um dado

processo. AnyTime é executado uma vez *per* Evento no bloco e a Máquina de Estado avalia uma Evento por vez. AnyTimes são associados somente com um bloco, e em um dado bloco.

A detecção da existência de múltiplos scripts requer uma combinação de identificadores de script e Agente. Uma sessão de Agente id é única, e na sessão de Agente id também está uma sessão de script unicamente identificada por uma sessão de script id. Os identificadores provêem uma detecção de instância. Em Máquinas de Estado usadas, por exemplo, para detectar um registro ou erro, as sessões de Agente ids são usadas para organizar as Máquinas de Estado identificáveis e únicas. Normalmente, Máquinas de Estado globais não são criadas com visibilidade global, mas eventualmente pode ser possível, se necessário. Usualmente não há um estado ao longo de todos usuários, ao invés, cada usuário ou processo requer seu próprio estado, que é rastreado pelas instâncias da Máquina de Estado.

Além da limpeza de identificadores, outro importante mecanismo estrutural de processamento para prover um eficiente processamento de Eventos é a definição de Escopo que origina da mensagem de rastreamento enviada pelo gerador de fonte de dados gMg. Escopos identificam e adicionalmente suportam uma detecção Multi-Instância de jogos de objeto GUI ou de outros processos que acontecem na fonte de dados de gMg.

Eventos de diferentes sessões são coletados em uma fila. Com base nos valores de campo, ao objeto Evento é designado um escopo. Usualmente o escopo também é a sessão, mas alternativamente o escopo pode ser definido usando alguns outros campos de mensagem de rastreamento, tal como o campo Topic. Em cada escopo, há uma ou mais instâncias de Máquinas de Estado. Por exemplo, se a tarefa for detectar erros, então uma Máquina de Estado para aquela sessão pode ser criada para determinar a mensagem sem se requerer múltiplas Máquinas de Estado.

Para evitar a proliferação de Máquinas de Estado

(e precisar mais recursos do servidor), se um novo Evento corresponde à transição inicial, não é necessário criar uma nova Máquina de Estado, se já houver uma Máquina de Estado naquele escopo. O novo Evento pode ser passado para processamento para a Máquina de Estado existente. Por exemplo, se aparecer uma página em uma aplicação, a página Evento será usada para iniciar a transição, o que acontece somente na primeira vez, quando a página for criada e mostrada. Somente neste momento a nova Máquina de Estado é criada. A Máquina de Estado criada é única, onde única significa uma única instância em um único Escopo.

A instância de Máquina de Estado é designada a um escopo com base no Evento que motivou sua criação. O escopo usualmente é identificado pela sessão id, onde Escopos são definidos com as definições de nível de mensagem de rastreamento L1 a L5. Há escopos de nível script e sessão. Se um escopo do nível script for incluído, há uma Máquina de Estado para determinar o id de script, que é único com respeito à sessão de id de Agente, e o cliente é avisado pelo servidor da criação de id de Agente, o que é repetido para cada Agente em cada máquina cliente.

As duas implementações seguintes ilustram a aplicação do componente escopo. Na primeira configuração, um procedimento login para uma aplicação requer uma transição inicial simples. Para um procedimento login seguro, o usuário supre a informação id e então submete sua entrada. No surgimento inicial da tela login (normalmente, uma vez a cada sessão) é gerado um Evento. O sensor recebe o Evento, e a máquina de estado na transição inicial cria uma nova Máquina de Estado. Em outra implementação, uma aplicação de janela com dois threads (um thread por janela), o monitoramento gMg rastreia o processamento em cada janela. Neste cenário, um usuário é identificado na sessão ou na Máquina de Estado. Também, podem ser providos múltiplos escopos na mesma sessão associados ao mesmo usuário. A análise de

servidor utiliza dois escopos, um para cada janela, mas ainda associados ao mesmo usuário. Ademais, dentro da coleção de Máquinas de Estado, pode haver uma Máquina de Estado diferente para cada janela.

Referindo-se à figura 25, dentro do conjunto de definições de Máquina de Estado contendo transições 2505, e dado um Evento serial 2550, cada condição estática no conjunto de definições de Máquina de Estado 2565 que se refere ao Evento (2510 a 2520) é processada por EvalStatic 2555, que deixa o conjunto de condições remanescente 2570, que se refere às variáveis de instância de Máquina de Estado 2525 a 2545 que devem ser avaliadas, no contexto de uma instância de Máquina de Estado. As condições de instância são inspecionadas com função EvalInstance 2560 para cada instância de Máquina de Estado, e a transição da primeira correspondência é executada, que é feita por Máquinas de Estado quer existentes ou recém-criadas. A transição inicial não é parcial, exceto quando se cria uma nova instância, mas então ela processa o Evento similarmente a qualquer outra transição. Se a transição correspondente for aceitável a partir do estado corrente, haverá vínculos para trajetórias abertas, e se a condição de transição for verdadeira, então também pode haver condições no nível de instância para o Evento.

Por exemplo, se uma mensagem de rastreamento chegar, e seu campo de status for igual a 5, este valor será armazenado na instância, mas não pode ser verificado neste ponto, porque a instância ainda não é acessível. Apenas quando o processamento chegar no acesso para instância, que estes valores serão avaliados, e feita uma verificação para verificar se o status corresponde ao Evento recebido. Este é um exemplo de condição de instância oposta à estática. Se a condição for aceitável e a condição de transição verdadeira, então uma avaliação de condição estática (falsa) leva a uma avaliação das condições de instância. O compilador da Máquina de Estado

do servidor distingue entre condições de instância e estática, e chama as rotinas de manipulação apropriadas. Se verdadeiras estas condições de Evento, a ação de código associada àquela transição será selecionada e executada. O processamento de transição então rastreia para o estágio pós-transição, ou se houver um goto, para um processamento de estado na destinação goto.

Referindo-se à figura 26, a maior parte do processamento está no processo de avaliação e no roteamento ou envio de milhares de Eventos recebidos 2603 em altas taxas. Dentro deste grande conjunto, para evitar ambigüidade de estados, todos os Eventos de sessão são serializados 2609, e então inseridos no conjunto de Máquinas de Estado 2621. O engine de estado 2627 verifica os Eventos recebidos com as Máquinas de Estado (2624). A distinção entre as definições de Máquina de Estado e suas instâncias, como discutido na seção anterior, também é aplicável aqui. A Máquina de Estado é a definição de um mecanismo de detecção de estado, e todas Máquinas de Estado definidas processam cada Evento inserido. Se uma condição de Evento (estático) for correspondida 2612, então o engine da Máquina de Estado verifica o Evento 2633 contra o escopo, e somente processa os Eventos em instâncias no escopo correspondente 2639. A condição de um certo estado da instância não pode ser decidida no nível de decisão de Máquina de Estado mais alto. Se uma Máquina de Estado estiver interessada em um Evento recebido, o Evento deve ser capturado pela instância da Máquina de Estado, dependendo do estado de cada instância. Dado o conjunto de todas transições, o objetivo global é primeiro testar as condições estáticas 2615, e em seguida testar todas condições de instância 2618. Verificar as condições estáticas é uma operação de nível mais alto feita primariamente para melhorar a eficiência.

As Máquinas de Estado são agrupadas por escopo, e para um dado Evento, as instâncias de Máquina de Estado são

buscadas em um mapa usando o escopo de Evento como chave. Dentro do conjunto de estruturas estáticas, as transições e todos AnyTimes que correspondem ao Evento 2612 são recuperadas. Uma avaliação estática pode ser usada como teste rápido para excluir transições que não correspondem com o Evento. Em vez de repetir o Processamento de Evento uma vez a cada instância, o processamento pode ser feito em nível mais alto (i.e., no nível de definição de Máquina de Estado 2615), e transições não acionadas com o Evento de entrada corrente na condição, não constarão da lista de transição. No entanto, se corresponder a uma transição, ou a um comando AnyTime, cada tipo de correspondência será adicionado a sua respectiva transição correspondente ou às listas AnyTime 2612.

Se certas condições forem atendidas, a próxima etapa será criar instâncias de Máquina de Estado. Se a lista de condições contiver uma transição inicial (i.e. a transição que mostrou ser correspondente) e se a Máquina de Estado não for única, serão criadas tantas instâncias quanto necessárias dentro do escopo 2636, ou se não houver instâncias criadas no escopo, uma nova instância precisará ser criada. Em outras palavras, quer a transição tenha sido correspondida ou a Máquina de Estado não for única, ou ainda se não houver instâncias dentro do escopo, deverá ser criada uma instância dentro do escopo. Se verdadeiro, uma nova instância de Máquina de Estado será criada.

Métodos utilitários também são mantidos pelo estado, ausência, etc.. Se uma transição for correspondida 2642 se verifica que a transição é acessível 2645 (a função IsAnyTimeValid verifica que este AnyTime é acessível a partir do estado corrente). A função Execução (transição) ocorre dentro da transição e é chamada dentro de uma transição válida 2648. Uma verificação é feita para determinar que agora a transição está sendo feita 2651. Como estabelecido anteriormente, instâncias gMg comuns podem ser verificadas eficientemente no nível mais

alto (definições de Máquina de Estado), mas se as condições se referirem à instância existente de Máquina de Estado, cada Máquina de Estado pode ter diferentes valores para aquelas instâncias, e portanto deve ser verificado na instância precisa.

Depois de completado o processo de transição, pode haver (ou não) outra transição. Tal condição ocorre se o processo que controla a fonte de dados for interrompido, ou se a Máquina de Estado tiver alcançado sua última transição. Se nenhuma transição corresponder ao estado corrente, depois de um período de tempo pré-determinado, então uma transição de ausência para impedir o bloqueio do sistema gMg.

Se uma transição for realizada, a ausência será verificada 2654. A ausência pode ser registrada em um buffer de ausência, que mais tarde poderá vir a ser acionado. Em cada estado, uma ausência registra a posição de ausência mais breve aceitável. Por exemplo, se houver duas ausências, (por exemplo 5 e 10 minutos) o menor deles será registrado. A ausência de 10 minutos será descartada uma vez que a ausência de 5 minutos expira primeiro, e as subseqüentes mudanças de estado são executadas primeiro. Em qualquer ponto há apenas uma ausência possível para uma Máquina de Estado - a ausência mais breve. Portanto, se uma transição for realizada, a ausência deve ser atualizada (refreshed), o novo estado é movido, todas ausências possíveis a partir do novo estado serão buscados e o novo pedido de ausência registrado. As ausências são verificadas quando um Evento chega. Havendo um buffer de pedidos de ausência contendo os pedidos de ausência registrados. Quando os Eventos chegam, com base no tempo, o buffer é inspecionado quanto a qualquer ausência, um Método de registro é chamado 2654, e sendo atualizada a informação a respeito da Máquina de Estado. Por exemplo, se o sistema estiver em um certo estado e houver uma ausência de 30 minutos, antes de transcorridos os 30 minutos, e se passar para um

estado diferente, haverá acesso a outro conjunto de ausências. No entanto, se não houver ausências, o registro de ausência será deletado. Se o sistema estiver aguardando um Evento, mas sem saber ao certo quando tal evento deve chegar, pode ser criada uma ausência paralela de um certo tempo (por exemplo, um minuto). Então, se o Evento chegar em um minuto, o processamento de estado prossegue, mas, se o Evento não chegar para impedir o bloqueio de processamento, o sistema pode prosrastrear com o registro de ausência de um minuto. Depois de realizada a transição, sendo removido o registro de ausência corrente 2654. Apenas novas ausências serão registradas, se aceitáveis, ou seja, havendo trajetórias de transição apenas a partir da localização corrente inspecionada. De outro lado, se na localização corrente um Evento não for recebido em um minuto, o buffer de ausência será verificado, a ausência usada, e a transição executada, e se passando para o próximo estágio. A partir deste próximo estado será feita uma busca de novas transições de ausência, que serão registradas.

Preferivelmente, as ausências são manipuladas depois de o Evento ter sido processado. A ausência é processada depois do processamento regular, porque a execução da transição pode remover a ausência. A ausência é registrada em buffer e tratada como transição, com a condição de a ausência ter uma ação de código. Uma ausência é uma transição especial sem Evento, porque na verdade é acionado pela falta de um Evento, onde a transição é disparada depois de um intervalo de tempo quando não há transição. A ação do código em geral tem um parâmetro para um Evento externo, mas na ausência não há nenhum. Há um buffer de ausência 2630 usado no nível de engine de estado. As entradas de buffer correspondem a minutos, e para cada entrada há uma lista de transições programada para a ausência naquele minuto. Depois de completado o tempo, o Evento corrente é usado para buscar o correspondente minuto no buffer de ausência. Todas

transições localizadas em entradas até e incluindo este minuto podem ser ausentados, onde houver uma ausência, as Máquinas de Estado serão deletadas. Não há loops em ausências pendentes, e o processamento é muito rápido para fazer chamadas para transições para todas Máquinas de Estado. Depois de uma transição ser realizada, o engine chama IfEndState() para a Máquina de Estado correntemente processada, e se no estado final, a solução termina.

Quando fecha, o sistema de monitoramento gMg com sua ampla gama de possíveis alvos de monitoramento provida por sua arquitetura sensora para interfaces de Método de componentes públicos e privados, cria fontes de dados que podem ser analisadas em um contexto de usuário GUI rico para refletir perfis de desempenho para sistemas de aplicações empresariais subjacentes. A capacidade do Sistema gMg em suportar aplicações Multi-Instância com detecção Genérica ou Específica de objetos GUI provê uma adaptabilidade e flexibilidade máxima em processos internos de aplicação dinâmica em ambientes operacionais. Adicionalmente, máquinas de estado distribuídas Locais e Remotas provêem uma análise adequada para tipos de Evento em uma gama de aplicações, tal como detectar tempos baixos de resposta, demasiadas condições de erro, análise de processo empresarial, etc..

Referindo-se à figura 27, onde é dado um fluxograma ilustrando processo 2700 para analisar Eventos de Método. Na etapa 2710, um programa de monitoramento independente 436 (incluindo Agente 414, Engine 416, e Solução 426) é operado para modelar objetos, na forma de diferentes estruturas de árvore, incluindo Árvore de Objeto GUI hierárquica 660 de uma Aplicação Hospedeira (para identificar objetos GUI), uma Árvore de Módulo de Código (figura 17) (para identificar módulos de código, e monitores de método, Instâncias COM) e uma árvore de Processo/ Thread (para identificar processos e threads da Aplicação Hospedeira). Uma indicação de um espectro

ambiental (i.e., condições de uma aplicação alvo manifestadas em objetos, objetos GUI, mensagens, chamadas e retornos de método) é produzida, etapa 2715, para janelas de aplicação. Eventos de Máquina de Estado que podem ocorrer em dois ou mais níveis para suprir Eventos significativos (por exemplo, um computador de usuário local de primeiro nível 2403 e um servidor remoto de segundo nível 2406) são processados, etapa 2720. Um ou mais Eventos gerados de Máquina de Estado pelos sensores do Sistema gMg ou processamento de Máquina de Estado em resposta a indicações de espectro ambiental dinâmico de Aplicação Hospedeira 515 (por exemplo, usuário, aplicação, OS, e eventos de método) são correlacionados, etapa 2725. Uma indicação de uma experiência de usuário é deduzida, etapa 2730, com base nos resultados correlacionados de Máquina de Estado. O dado de Evento lógico, sinérgico com as indicações de espectro ambiental, é coletado no servidor 557, etapa 2735, a partir dos eventos de máquina de estado. Padrões opcode de byte 1920 são identificados, etapa 2740, e usados para localizar e detectar pontos de entrada de Método das funções componentes. Na etapa 2745, é identificada uma função de interface em meio aos padrões opcode de byte 1910, onde a função de interface provê interfaces para Eventos e propriedades. O endereço da função interface é determinado, etapa 2750. A técnica usada depende do tipo da função interceptada (por exemplo, Funções Export 1620 ou COM TypeLibrary 1624) e quer o processo inicie antes (1627) ou depois (1630) da criação de objeto. Os padrões opcode permitem a execução de uma interceptação de chamada 1642 e retorno 1645 para identificar objetos, incluindo chamadas e retornos de método, e objetos GUI. O processo 2700 se vincula, etapa 2755, à função interface usando uma rotina Intercept (figura 18) 1848 para rastrear chamadas de função.

A figura 28 ilustra o fluxograma para o processo 2800 que monitora a funcionalidade da Aplicação Hospedeira.

Padrões opcode de byte representando funções de método dentro da Aplicação Hospedeira são identificados, etapa 2810, a partir de um arranjo de padrões de byte capturados do Método alvo - i.e. uma Assinatura de Método 1930. Na etapa 2815, o processo 2800, compara durante execução os padrões opcode de byte de função 1010 contra uma lista de Assinaturas de Método para determinar a localização do Método 1905. Vínculos são estabelecidos, etapa 2820, entre a função de método alvo, uma rotina Intercept de chamada 2012, e uma instância de estrutura de informação 2045 usando uma instrução de rotina jump 2039. O processo 2800 monitora, etapa 2825, retornos das chamadas para função de método usando alternativamente duas técnicas de manipulação de Stack: Cópia de Stack de captura de retorno 1648 e Alocação de Stack de retorno 1651, resultando na captura de valores de retorno que podem ser usados para recuperar informações adicionais de processos internos da aplicação.

Referindo-se às figuras 29A e 29B, onde é ilustrado um fluxograma de processo 2900, que monitora Eventos a partir de uma camada de apresentação de aplicação alvo de computador. Na etapa 2904, provê-se um script na forma de "Soluções" 4126 no computador de cliente 436, local na aplicação alvo. O script 426 é capaz de operar dentro da aplicação alvo. Instanciamentos de objetos 509, durante execução, a partir de múltiplos instanciamentos da aplicação alvo 545 são escaneados na etapa 2906. O escaneamento de objetos resulta em um conjunto de Eventos 515 em tempo real, que são eventos a partir do usuário, Aplicação, eventos OS, e valores particulares de objeto GUI e propriedades de Método a partir de chamadas e retornos de métodos da aplicação alvo. Um Object Tree GUI 832 é alocado, etapa 2908, para organizar estruturas de dados, que então são adaptadas para refletir, etapa 2910, mudanças dinâmicas em tempo real 854 para GUI da aplicação real. Um fluxo multi-tipo de Eventos a partir da adaptação reflete estados reais da aplicação real

1115, 1103. Esta estrutura GUI reflete um tipo de Evento que provê um contexto para todos os outros tipos de dados para intercalar e referenciar. Na etapa 2912, os instanciamentos de objeto são detectados a partir dos instanciamento da aplicação alvo, que correspondem a uma pré-determinada estrutura de objeto usando expressões capazes de detectar categorias de objetos relacionados (com referência a amostras de código). A figura 15 mostra expressões de script que podem ser genéricas 1510, para detectar categorias inteiras de objetos GUI relacionados (ou mais especificamente 1505 para detectar objetos GUI únicos). As expressões de detecção podem detectar objetos GUI em qualquer lugar neste espectro conceitual para refletir as várias variações encontradas em estruturas de aplicação, durante execução. Na etapa 2914, pelo menos uma porção de um espectro ambiental do objeto detectado é capturado do conteúdo dos objetos detectados a partir de qualquer instanciamento de aplicação alvo.

Na etapa 2916, Eventos Lógicos 521 são produzidos por expressões de seqüência lógica de script (máquina de estado local) ou processamento de Máquina de Estado (máquina de estado remoto) a partir de Eventos a partir de Sensores gMg 438, 439, 440 recebidos do espectro ambiental (i.e. Método, OS, Usuário, e Eventos de Aplicação). O conteúdo processado é remontado (i.e., os dados são coletados em um servidor 446), etapa 2918 para refletir estruturas de multi-instanciamento detectadas por Sensores gMg e refletidos pelo Object Tree GUI. O processamento de máquina de estado hierárquico do Sistema gMg 2430 e 2442 detecta seqüências de eventos que recriam o estado da aplicação, como em contextos experimentados por um usuário. As estruturas remontadas são correlacionadas, etapa 2920, em conjuntos de dados detectados usando Escopos 2639 definidos para máquinas de estado único que separam instâncias das aplicações e instâncias de conjuntos de objeto GUI de sessões de usuário para refletir estados anteriores da aplicação.

A reflexão da estrutura de aplicação alvo (i.e. scripts múltiplos usam o mesmo Object Tree GUI 660) é compartilhada, etapa 2922, em meio a uma pluralidade de scripts 945, 948, 951, onde cada script da pluralidade de scripts é associado ao respectivo instanciamento da aplicação alvo 901, 902, 903. Um processamento intermediário usa eventos capturados, em forma de mensagem de rastreamento (entre 436 e 446), que são recebidos e processados para detecção e visualização em tempo real, ou coletados em um dispositivo de armazenamento de dados para análise posterior. Recursos adicionais permitem uma análise de tendência complexa e histórica, etapa 2924, do conjunto de dados de rastreamento que usa uma definição semântica comumente entendida que sincroniza a fonte (no cliente) com a destinação (no servidor) para recriar e dar uma idéia das condições da experiência do usuário.

Referindo-se particularmente à figura 29B, o processo 2900 continua na etapa 2932, onde uma entidade organizadora hierárquica usando Contextos gMg (figura 12) que representam uma porção da aplicação alvo e contém informação contextual (i.e. Níveis de Contexto 1205, 1210, 1215) é declarada. O espectro ambiental capturado suportado pela entidade organizadora hierárquica (i.e. Object Tree 1015 e seu componente de Variável de Objeto 1035 ou conjunto de Variáveis de Objeto 1039) é analisado, etapa 2934. Na etapa 2936, o processo 2900 sensoreia, usando Hooks (figura 14), uma interação de aplicação alvo com usuário ou uma interação com um evento gerado por aplicação alvo que aparece em um processo 1406 ou thread 1409, 1412, 1418, 1421. Usando variáveis de expressão de script lógico, as etapas 2938 (se referindo à amostra de código anteriormente descrita - "Amostra de Código de Ligação Específica e Genérica" e Variáveis de Objeto 1005) para gerar eventos com base na execução de script em resposta aos resultados da etapa de sensoreamento (i.e. geração de Eventos gMg com um

mecanismo, tal como WhenObjects 1151).

Um ou mais Eventos gerados são passados, etapa 2940, para um interpretador 745. Os Eventos seletivamente gerados são analisados, etapa 2942. A seleção usa os vários mecanismos do Sistema gMg, tal como expressões lógicas no script para descrever as condições da detecção, variáveis e Variáveis de Objeto, WhenObjeto, e dentro da estrutura Object Tree GUI. Na etapa 2946, a análise é correlacionada com parâmetros referentes a fontes externas, em particular, dados a partir de servidor ou infra-estrutura ambiental. O script é associado, etapa 2948, a um único processo 1315 de uma janela na aplicação alvo. Uma única mensagem é detectada, etapa 2950, que é associada ao thread-id. A mensagem é interceptada, etapa 2952, por uma rotina hook ativa 1433 a 1442. O conteúdo da mensagem é avaliado com operações e expressões de script (com referência à amostra de código "ManyInstances), etapa 2954, para produzir eventos e propriedades, de modo a obter informações com respeito à aplicação alvo. Na etapa 2956, o processo 290 gera reportes 454, alertas 450, correlações 452, e ações e respostas 456 com os resultados das etapas de análise e correlação. Na etapa 2958, o escaneamento repetido da aplicação continuamente acresce a Árvore de Objeto GUI gMg 660 de estruturas que refletem a aplicação GUI com as etapas de alocar, adaptar, detectar dinamicamente (i.e. operações de busca intermitentes pelo mecanismo WhenObject figura 11, Hooks figura 14) e capturar em um pré-determinado intervalo 615 (i.e. controlado por relógio OS, ou por certas condições) solicitadas pelos Eventos detectados.

Conquanto a invenção tenha sido descrita em detalhes, vários melhoramentos, alternativas, e equivalentes serão aparentes àqueles habilitados na técnica, mas deve ser entendido que a descrição detalhada foi provida de forma exemplar e de modo nenhum limitante. Numerosas mudanças quanto à construção, combinação, e arranjo de componentes

poderão ser consideradas sem sair do escopo e espírito da presente invenção, como reivindicada nas reivindicações anexas. A substituição de elementos de uma configuração para outra também é contemplada nesta especificação. A presente invenção sendo definida somente em vista das reivindicações anexas e de equivalentes das reivindicações citadas nesta.

REIVINDICAÇÕES

Método para monitorar eventos derivados de uma camada de apresentação, e métodos de software de uma aplicação alvo, caracterizado pelo fato de compreender as etapas de:

- prover, independentemente de recompilar um código fonte da aplicação alvo, um script operável de modo a executar em um nível dentro da aplicação alvo, sendo que a aplicação alvo tem uma estrutura;
- escanear instanciamentos durante execução de objetos incluindo métodos de chamada, retornos de métodos, e objetos GUI da aplicação alvo;
- alocar estruturas em tempo real aos instanciamentos de objeto;
- adaptar as estruturas alocadas para criar uma reflexão da estrutura de aplicação alvo;
- detectar um ou mais instanciamentos de objeto que correspondam a uma pré-determinada estrutura de aplicação alvo; e
- capturar pelo menos uma porção de um espectro ambiental do objeto detectado.

2- Método, de acordo com a reivindicação 1, caracterizado pelo fato de adicionalmente compreender as etapas de:

- processar o espectro ambiental capturado do objeto detectado para os eventos lógicos;
- remontar os eventos lógicos para refletir estruturas multi-instanciamento em contextos distintos, sendo que os contextos distintos são estados anteriores da aplicação alvo; e
- correlacionar as estruturas multi-instanciamento remontadas para analisar o estado anterior da aplicação alvo.

3- Método, de acordo com a reivindicação 1, caracterizado pelo fato de o espectro ambiental capturado incluir categorias de informação com respeito ao objeto detectado, sendo que as informações provêem um vislumbre

com respeito ao ambiente da aplicação alvo.

4- Método, de acordo com a reivindicação 1, caracterizado pelo fato de a etapa de escanear adicionalmente compreender a etapa de selecionar instanciamentos de objeto com base em pelo menos um critério de monitoramento pré-definido.

5- Método, de acordo com a reivindicação 1, caracterizado pelo fato de a etapa de prover prover o script de um computador de cliente.

6- Método, de acordo com a reivindicação 1, caracterizado pelo fato de adicionalmente compreender a etapa de:

- compartilhar a reflexão da estrutura de aplicação alvo dentre uma pluralidade de scripts, sendo que cada script é associado ao respectivo instanciamento da aplicação alvo;

sendo que a reflexão da estrutura de aplicação alvo provê um contexto GUI para os dados de rastreamento coletados.

7- Método, de acordo com a reivindicação 1, caracterizado pelo fato de adicionalmente compreender as etapas de:

- detectar instanciamentos genéricos de objetos usando expressões de script capazes de detectar categorias de objetos relacionados; e
- processar os conteúdos e as propriedades do objeto genérico detectado para determinar a ocorrência de eventos de aplicação alvo no contexto.

8- Método, de acordo com a reivindicação 7, caracterizado pelo fato de adicionalmente compreender a etapa de construir contextos de usuário usando um nível de mensagem de rastreamento, e convenções e semântica pré-definidas.

9- Método, de acordo com a reivindicação 1, caracterizado pelo fato de adicionalmente compreender as etapas de:

- declarar uma entidade de organização hierárquica que representa uma porção da aplicação alvo e contém

informações contextuais a partir de pelo menos o espectro ambiental capturado; e

- analisar o espectro ambiental capturado com base na entidade de organização hierárquica.

10- Método, de acordo com a reivindicação 9, caracterizado pelo fato de incluir a etapa adicional de avaliar um valor de nível de contexto distingüido dentre múltiplos instanciamentos de objeto durante execução da aplicação alvo.

11- Método, de acordo com a reivindicação 2, caracterizado pelo fato de quando houver múltiplos instanciamentos da aplicação alvo, a etapa de escanear escanear cada instanciamento de aplicação alvo, e a etapa de detectar detectar instanciamentos de objeto com respeito a mais que um instanciamento da aplicação alvo.

12- Método, de acordo com a reivindicação 11, caracterizado pelo fato de a etapa de capturar capturar os conteúdos dos objetos detectados a partir de qualquer instanciamento de aplicação alvo.

13- Método, de acordo com a reivindicação 4, caracterizado pelo fato de o script do computador de cliente se localizar na aplicação alvo.

14- Método, de acordo com a reivindicação 1, caracterizado pelo fato de adicionalmente compreender as etapas de:

- sensorear pelo menos uma interação de aplicação alvo com um usuário e um evento gerado por uma aplicação alvo;

seletivamente gerar eventos com base na execução de script em resposta aos resultados da etapa de sensorear;

- enviar um ou mais dos eventos gerados a um interpretador;

- analisar os eventos gerados seletivamente; e

correlacionar a análise através de um mapeamento funcional para fontes externas.

15- Método, de acordo com a reivindicação 14, caracterizado pelo fato de a etapa de gerar pelo menos um de informe, alerta, correlação, e resposta contendo os

resultados das etapas de analisar e correlacionar.

16- Método, de acordo com a reivindicação 1, caracterizado pelo fato de as etapas de escanear, alocar, adaptar, detectar, e capturar serem repetidas em pré-determinados intervalos para criar uma reflexão atualizada da aplicação alvo.

17- Método, de acordo com a reivindicação 1, caracterizado pelo fato de as etapas de escanear, alocar, adaptar, detectar, e capturar serem solicitadas pelos eventos detectados para criar uma reflexão atualizada da aplicação alvo.

18- Método, de acordo com a reivindicação 1, caracterizado pelo fato de adicionalmente compreender as etapas de:

- associar o script a um único processo de uma janela na aplicação alvo;
- detectar a mensagem associada ao thread-id;
- interceptar a mensagem com uma rotina hook ativa no script;
- avaliar o conteúdo da mensagem; e
- derivar eventos e propriedades a partir da mensagem para obter informações na aplicação alvo.

19- Método para analisar eventos de método, caracterizado pelo fato de compreender as etapas de:

- prover um programa de monitoramento independente, o citado programa de monitoramento sendo operável de modo a modelar objetos de uma janela de aplicação e identificar os objetos a serem monitorados com base em um critério de monitoramento pré-determinado;
- derivar uma indicação de espectro ambiental para a janela de aplicação;
- processar eventos de estado máquina que ocorram pelo menos em um servidor e em uma máquina de cliente;
- correlacionar os eventos de máquina com as indicações de espectro ambiental obtidas;
- deduzir uma indicação de uma experiência de usuário com base nos resultados da etapa de correlacionar; e

- reportar as indicações deduzidas ao usuário do sistema.

20- Método, de acordo com a reivindicação 19, caracterizado pelo fato de a etapa de deduzir se basear em indicadores recebidos pelo lado servidor.

21- Método, de acordo com a reivindicação 20, caracterizado pelo fato de a etapa de correlacionar, adicionalmente compreender colher dados de evento lógicos a partir de eventos de estado máquina que são sinérgicos com indicações de espectro ambiental.

22- Método, de acordo com a reivindicação 19, caracterizado pelo fato de o programa de monitoramento independente incluir instruções operáveis para realizar as etapas de:

- identificar padrões opcode de byte;
- detectar pelo menos uma função de interface dentre os padrões opcode de byte identificados que provêem interfaces aos eventos e propriedades de um processo de aplicação, da janela de aplicação, ou de objetos GUI de janela;
- determinar um endereço da função de interface detectada; e
- vincular a função de interface detectada usando uma rotina de intersecção;

sendo que deve se rastrear a chamada de função de interface.

23- Método, de acordo com a reivindicação 22, caracterizado pelo fato de a etapa de vincular empregar uma função de intersecção de retorno, sendo que o retorno é rastreado.

24- Método, de acordo com a reivindicação 22, caracterizado pelo fato de o programa de monitoramento independente incluir instruções operáveis de modo a realizar a etapa de operar com a intersecção de função genérica de maneira diferente da maneira como se opera uma intersecção de função especifica;

- sendo que a intersecção de função genérica

não requer um pré-conhecimento dos parâmetros e formatos da função de interface detectada.

25 Método, de acordo com a reivindicação 22, caracterizado pelo fato de o programa de monitoramento adicionalmente incluir instruções operáveis para realizar a etapa de operar uma intersecção de função genérica de maneira diferente da maneira que se opera uma intersecção de função especifica;

- sendo que uma intersecção de função especifica contém uma indicação dos parâmetros publicados e um formato da função de interface detectada e a intersecção de função especifica permite uso direto dos parâmetros de função de interface detectada em um script.

26- Método, de acordo com a reivindicação 22, caracterizado pelo fato de adicionalmente compreender as etapas de:

- enviar uma informação de instância a um método de chamada usando um registro de informação de instância;

sendo que uma pilha em uma aplicação hospedeira não é modificado, e o registro de informação de instância contém um apontador para uma informação de instância de uma função de chamada para preservar os contextos da chamada de método.

27- Sistema para monitorar eventos em uma camada de apresentação de uma aplicação alvo de computador, tendo uma pré-determinada estrutura, caracterizado pelo fato de compreender:

- um servidor e computador de cliente interconectados em uma rede de comunicação;
- um programa de monitoramento sendo executado no servidor incluindo um script operável de modo a operar em um nível abaixo da camada de apresentação;

sendo que o programa de monitoramento é operável de modo a realizar as etapas de:

- escanear durante execução os instanciamentos de objetos da aplicação alvo;
- alocar estruturas em tempo real nos instanciamentos

de objeto;

- adaptar as estruturas alocadas para criar uma reflexão da estrutura de aplicação alvo;

- detectar um ou mais instanciamentos de objeto que correspondam a uma pré-determinada estrutura de objeto; e

- capturar pelo menos o conteúdo dos objetos detectados.

28- Sistema, de acordo com a reivindicação 27, caracterizado pelo fato de o script se encontrar no local do computador da aplicação alvo.

29 Sistema, de acordo com a reivindicação 28, caracterizado pelo fato de o computador cliente ser um computador pessoal, uma estação de trabalho, um assistente pessoal, ou uma plataforma capaz de executar instruções de computador.

30- Método para monitorar uma funcionalidade de aplicação hospedeira, caracterizado pelo fato de compreender as etapas de:

- identificar padrões opcode de byte representando funções de método na aplicação hospedeira;

- determinar um endereço para pelo menos uma das funções de método;

- vincular, usando o endereço, a função de método a uma rotina de intersecção de chamada e a uma instância de estrutura de informação; e

- monitorar retornos a partir das chamadas para a função de método com manipulação de pilha.

31- Método, de acordo com a reivindicação 30, caracterizado pelo fato de a etapa de vincular incluir uso de uma instrução de rotina "jump".

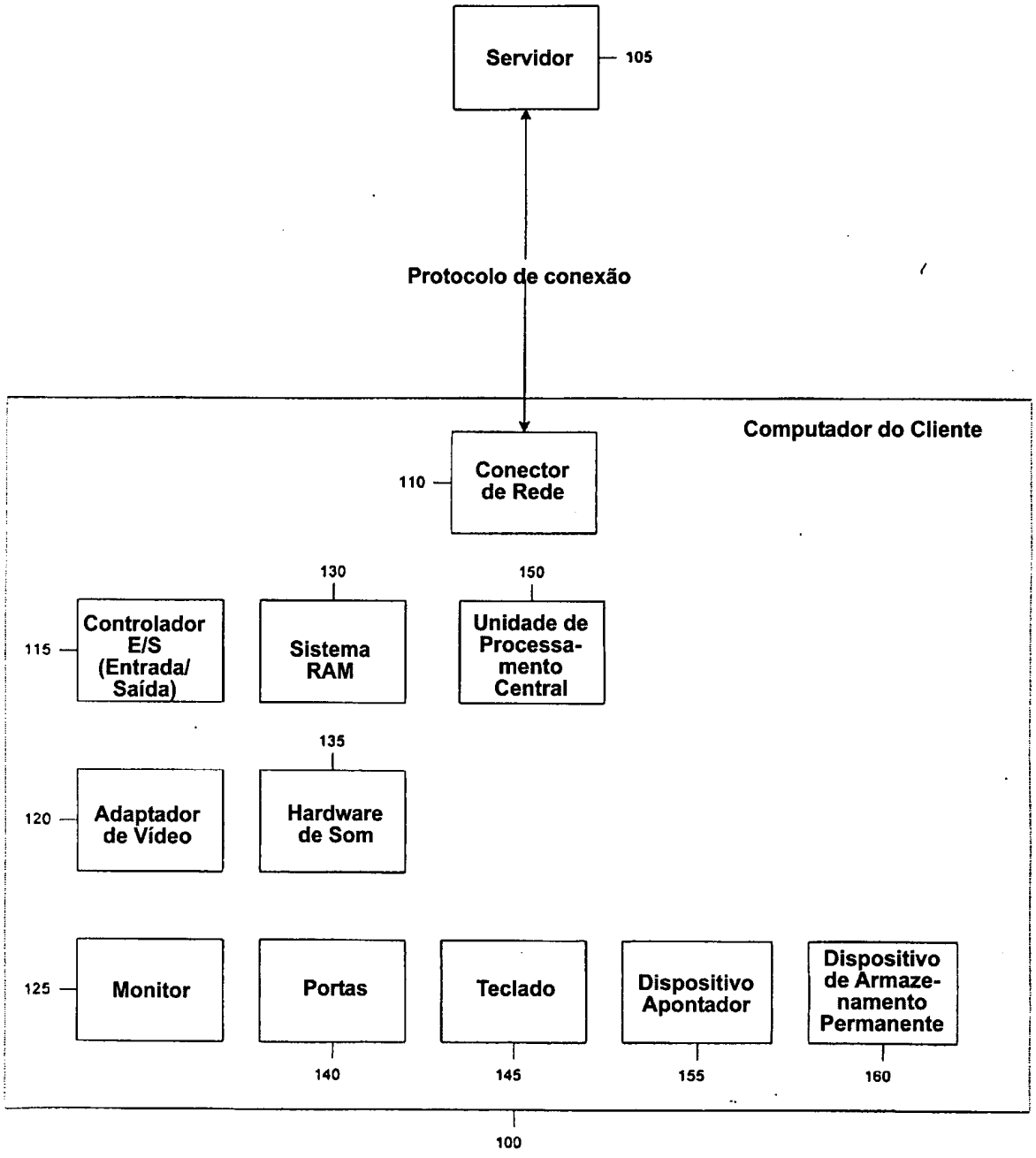

Servidor
105
Protocolo de conexão
Computador do Cliente
110
Conector de Rede
130
150
115
Controlador E/S (Entrada/ Saída)
Sistema RAM
Unidade de Processa-mento Central
135
120
Adaptador de Vídeo
Hardware de Som
125
Monitor
Portas
Teclado
Dispositivo Apontador
Dispositivo de Armaze-namento Permanente
140
145
155
160
100

FIG.1

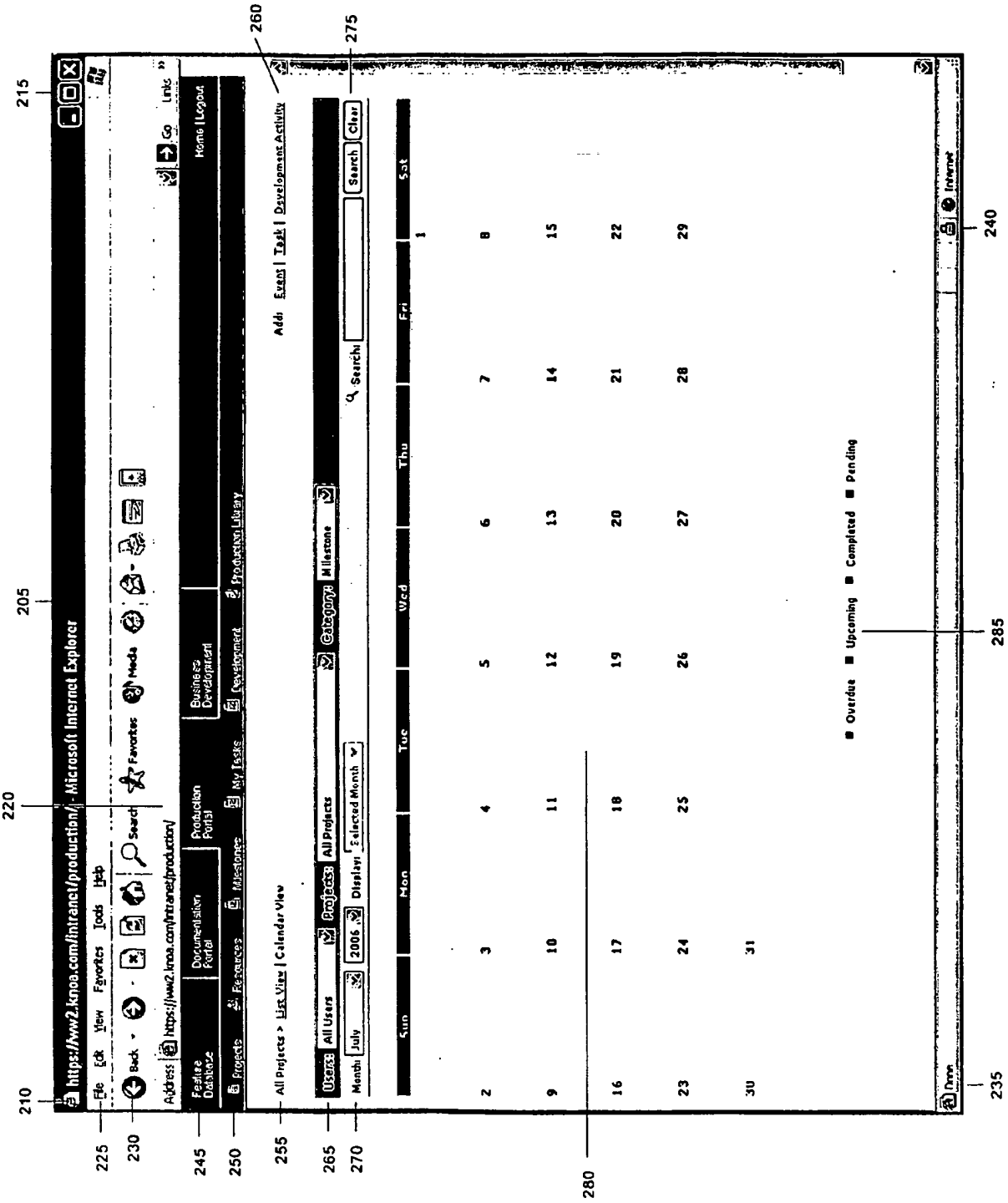

https://www2.knoa.com/intranet/production/ - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address https://www2.knoa.com/intranet/production/
Go Links
Feature Dashboard
Documentation Portal
Production Portal
Business Development
Home | Logout
Projects Resources Milestones My Tasks Development Production Library
All Projects > List View | Calendar View
Add Event | Task | Development Activity
Users All Users Projects All Projects Category Milestone
Month July 2006 Display Selected Month
Search Search Clear
Sun Mon Tue Wed Thu Fri Sat
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
Overdue Upcoming Completed Pending
Done
Internet
210
215
220
205
225
230
245
250
255
260
275
265
270
280
285
235
240

FIG.2

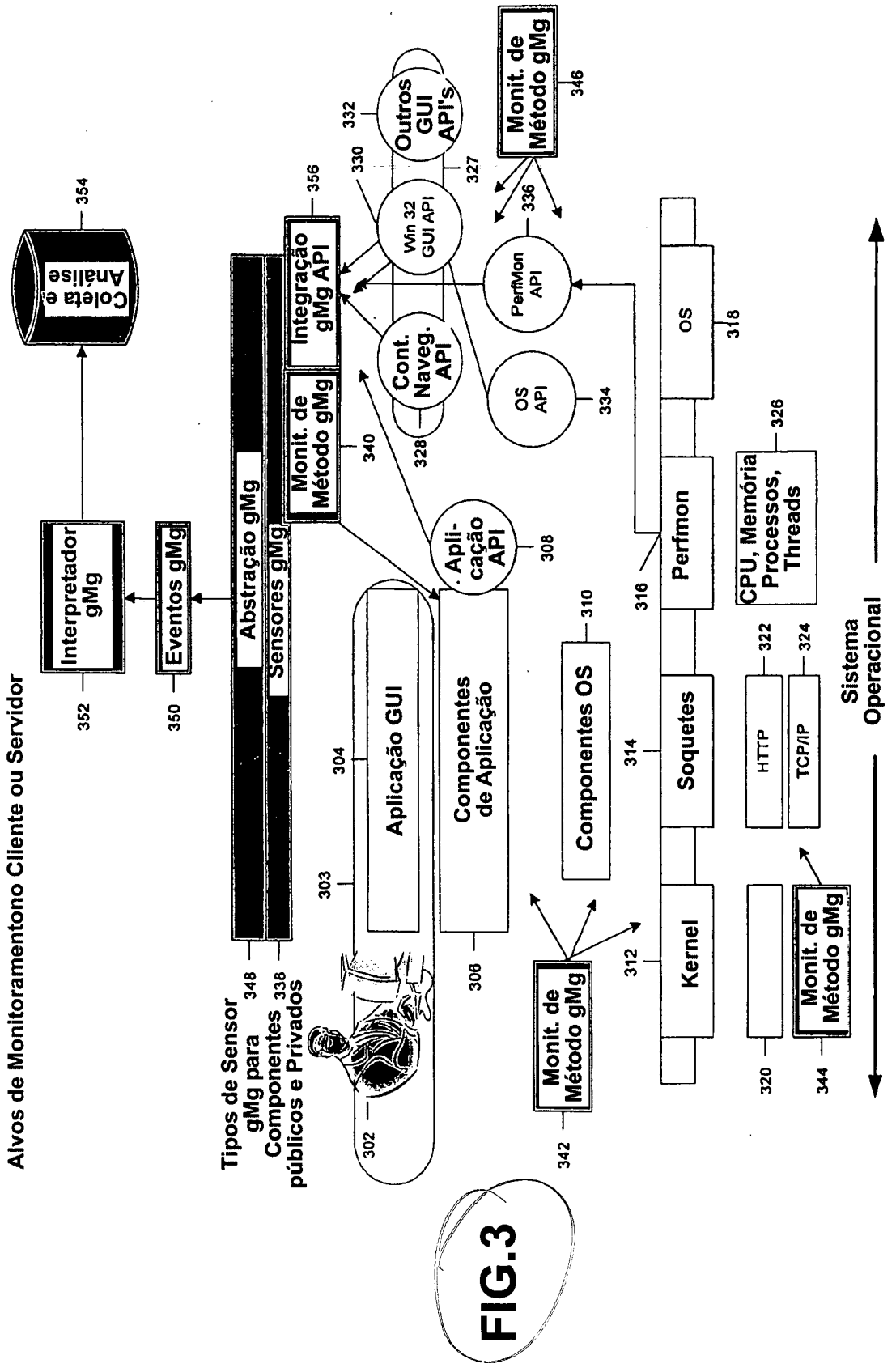

Alvos de Monitoramentono Cliente ou Servidor
Coleta e Análise
354
Interpretador gMg
352
Eventos gMg
350
Abstração gMg
Sensores gMg
Tipos de Sensor gMg para Componentes públicos e Privados
348
338
Integração gMg API
356
Monit. de Método gMg
340
Cont. Naveg. API
328
Win 32 GUI API
330
Outros GUI API's
332
327
Monit. de Método gMg
346
336
PerfMon API
OS API
334
Apli-cação API
308
Aplicação GUI
304
303
Componentes de Aplicação
306
302
Componentes OS
310
Monit. de Método gMg
342
316
Perfmon
OS
318
Soquetes
314
Kernel
312
CPU, Memória Processos, Threads
326
HTTP
322
TCP/IP
324
320
Monit. de Método gMg
344
Sistema Operacional

FIG.3

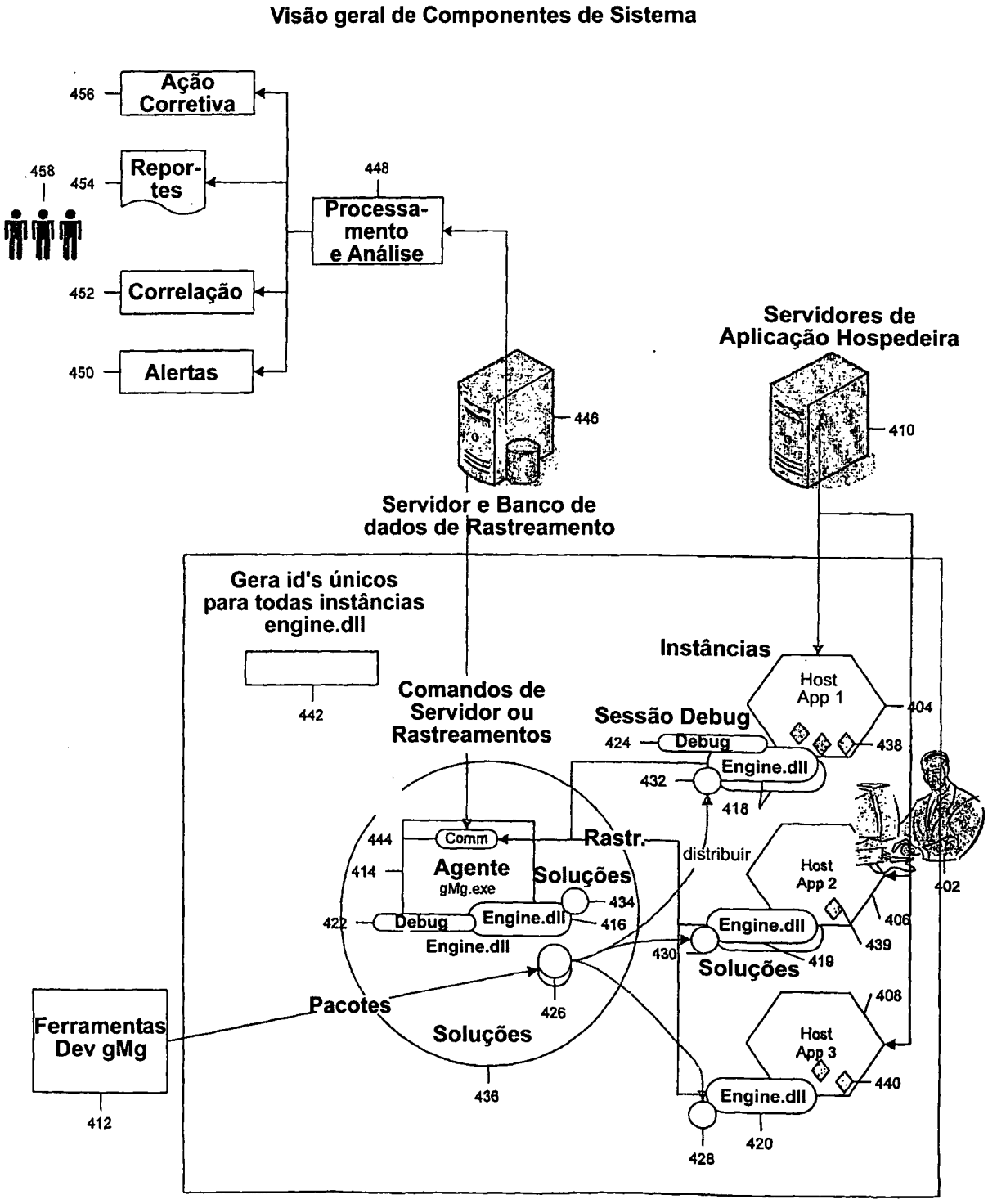

Visão geral de Componentes de Sistema
456
Ação Corretiva
458
454
Repor-tes
448
Processa-mento e Análise
452
Correlação
450
Alertas
Servidores de Aplicação Hospedeira
446
410
Servidor e Banco de dados de Rastreamento
Gera id's únicos para todas instâncias engine.dll
442
Instâncias
Comandos de Servidor ou Rastreamentos
Sessão Debug
Host App 1
404
424
Debug
438
Engine.dll
432
418
444
Comm
Rastr.
distribuir
414
Agente
gMg.exe
Soluções
434
422
Debug
Engine.dll
416
Engine.dll
Host App 2
406
439
Engine.dll
430
419
Soluções
402
Pacotes
426
Soluções
436
408
Host App 3
440
Engine.dll
420
428
Ferramentas Dev gMg
412

FIG.4

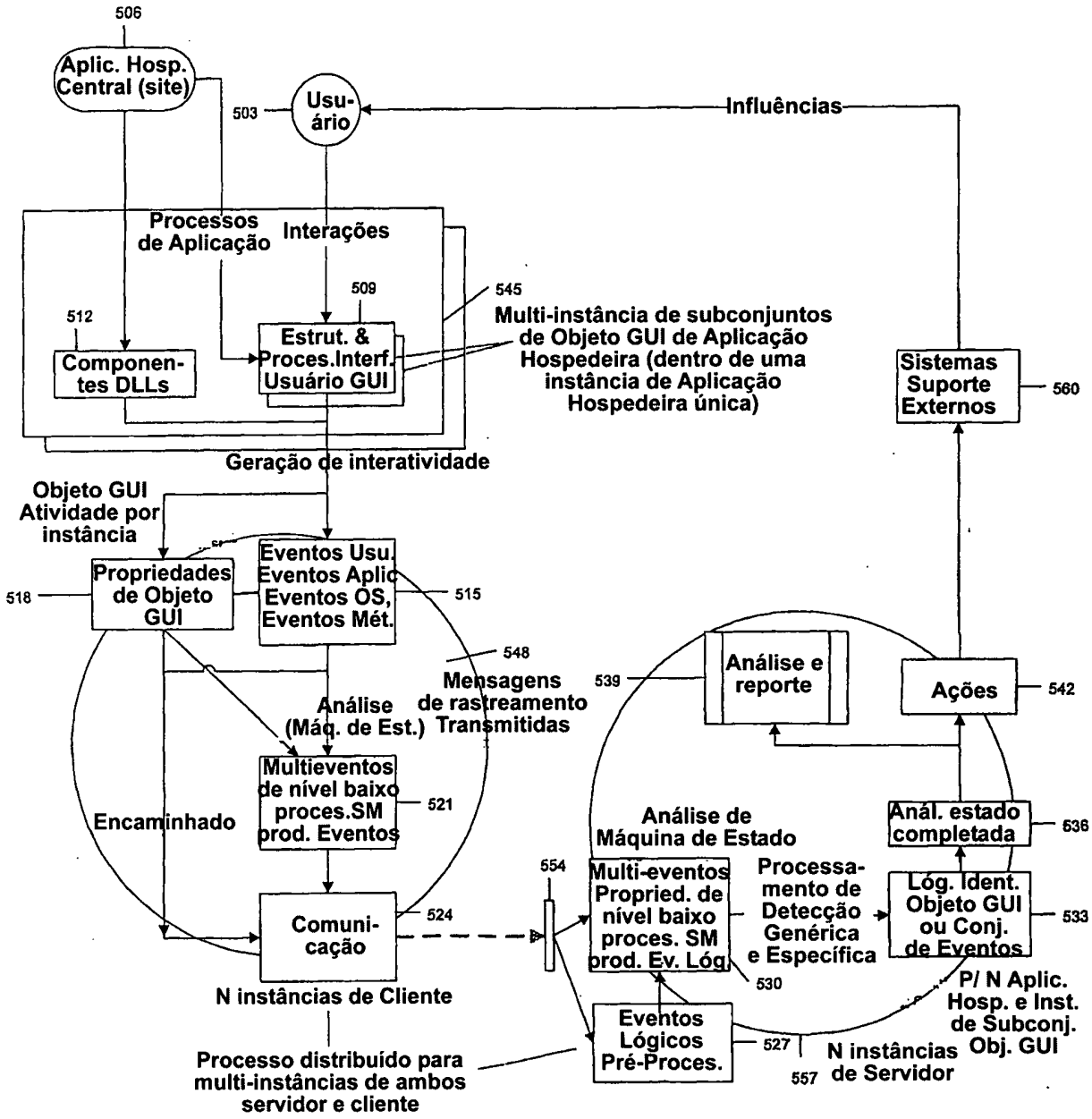

Visão Geral de Fluxo de Evento de Propriedade
506
Aplic. Hosp. Central (site)
503
Usu-ário
Influências
Processos de Aplicação
Interações
509
545
512
Componen-tes DLLs
Estrut. & Proces.Interf. Usuário GUI
Multi-instância de subconjuntos de Objeto GUI de Aplicação Hospedeira (dentro de uma instância de Aplicação Hospedeira única)
Sistemas Suporte Externos
560
Geração de interatividade
Objeto GUI Atividade por instância
Propriedades de Objeto GUI
518
Eventos Usu. Eventos Aplic Eventos OS, Eventos Mét.
515
548
Mensagens de rastreamento Transmitidas
Análise (Máq. de Est.)
Multieventos de nível baixo proces.SM prod. Eventos
521
Encaminhado
Comuni-cação
524
N instâncias de Cliente
Processo distribuído para multi-instâncias de ambos servidor e cliente
Análise e reporte
539
Ações
542
Análise de Máquina de Estado
Anál. estado completada
536
554
Multi-eventos Propried. de nível baixo proces. SM prod. Ev. Lóg
Processa-mento de Detecção Genérica e Específica
Lóg. Ident. Objeto GUI ou Conj. de Eventos
533
530
P/ N Aplic. Hosp. e Inst. de Subconj. Obj. GUI
Eventos Lógicos Pré-Proces.
527
N instâncias de Servidor
557

FIG.5

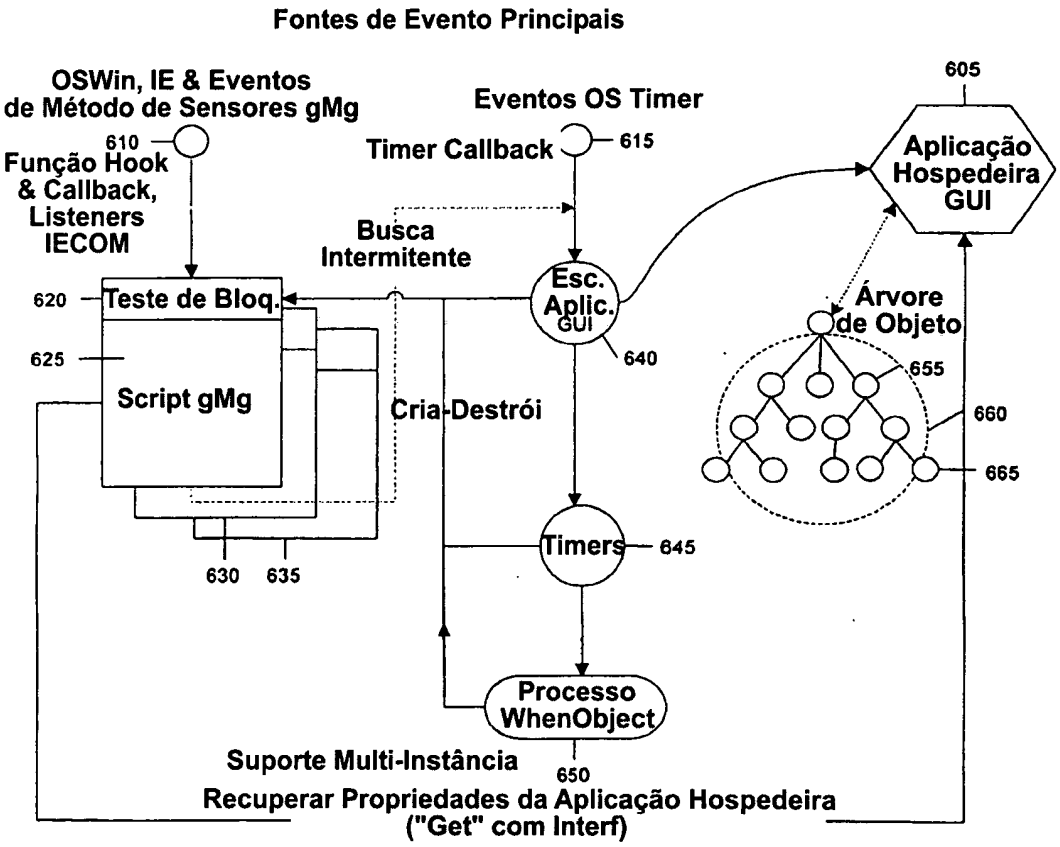

Fontes de Evento Principais
OSWin, IE & Eventos de Método de Sensores gMg
Eventos OS Timer
605
610
Função Hook & Callback, Listeners IECOM
Timer Callback
615
Aplicação Hospedeira GUI
Busca Intermitente
Esc. Aplic. GUI
640
620
Teste de Bloq.
Árvore de Objeto
625
Script gMg
655
660
665
Cria-Destrói
Timers
645
630
635
Processo WhenObject
Suporte Multi-Instância
650
Recuperar Propriedades da Aplicação Hospedeira ("Get" com Interf)

FIG.6

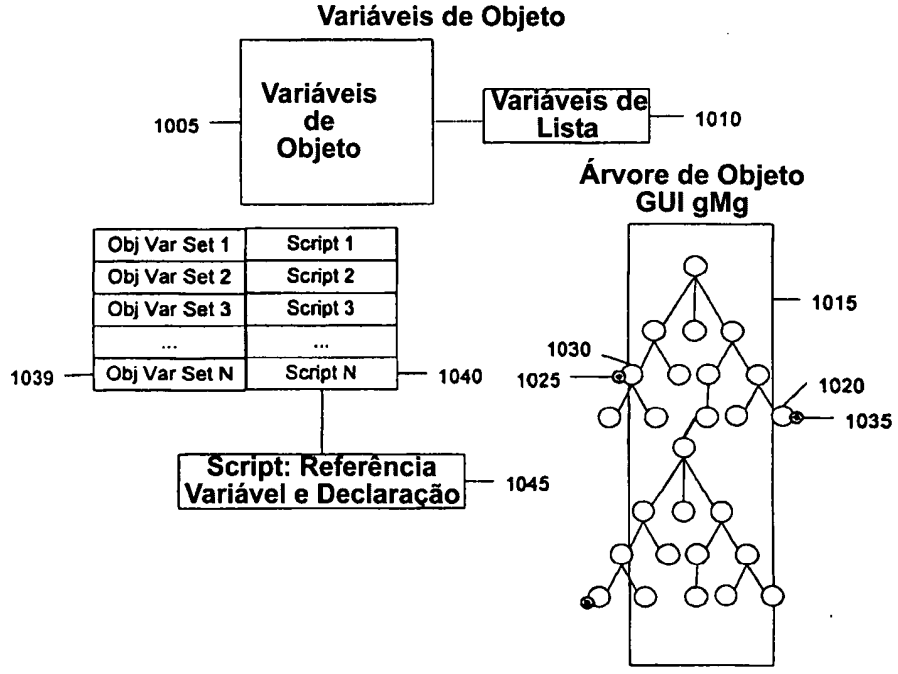

Variáveis de Objeto
1005
Variáveis de Objeto
Variáveis de Lista
1010
Árvore de Objeto GUI gMg
Obj Var Set 1
Script 1
Obj Var Set 2
Script 2
Obj Var Set 3
Script 3
...
...
1039
Obj Var Set N
Script N
1040
1015
1030
1025
1020
1035
Script: Referência Variável e Declaração
1045

FIG.10

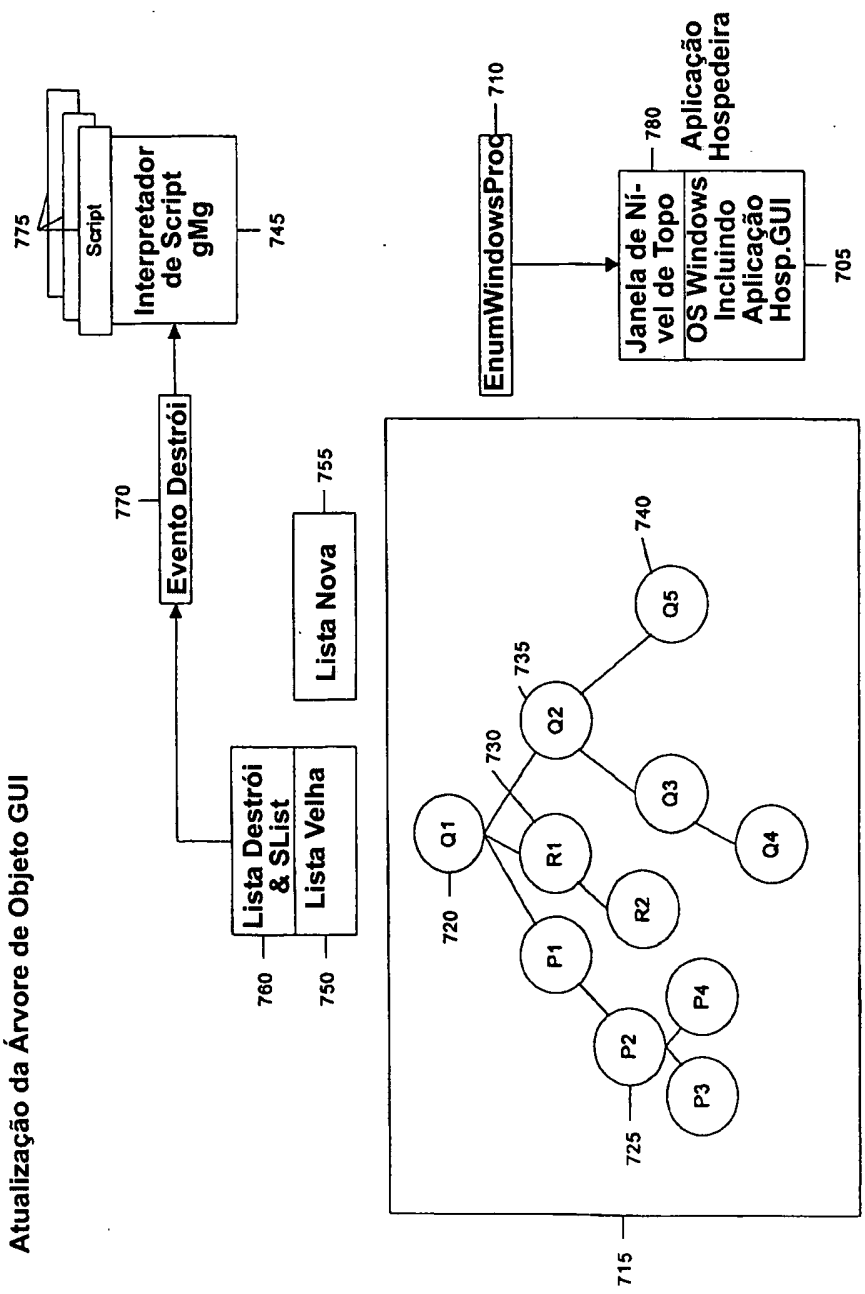
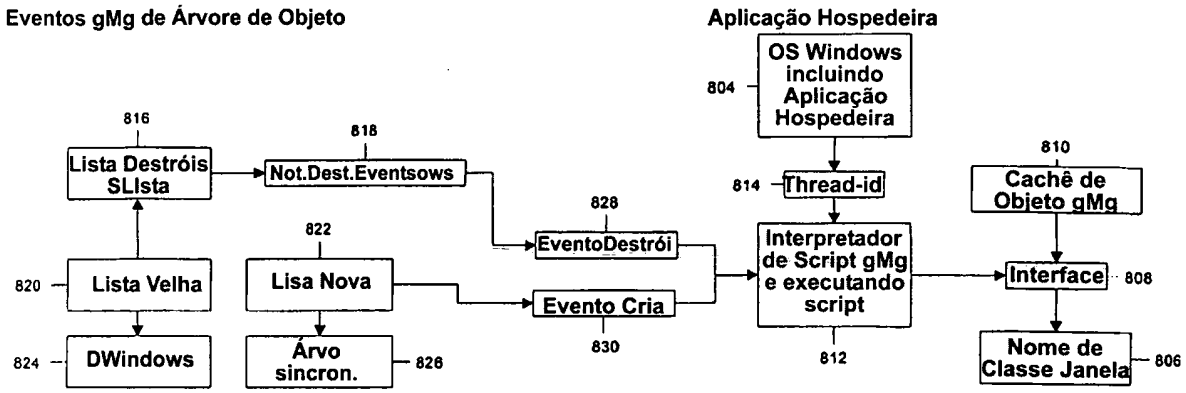
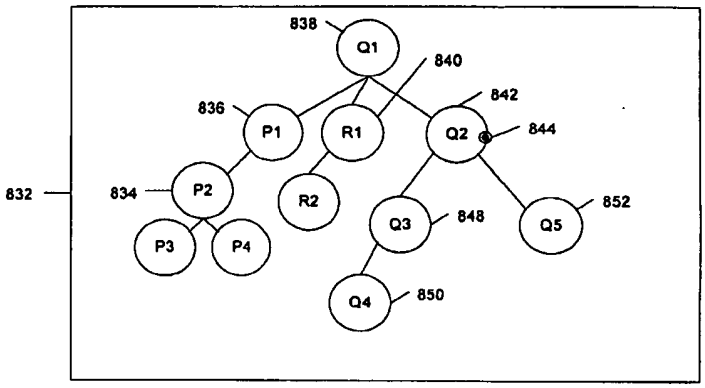
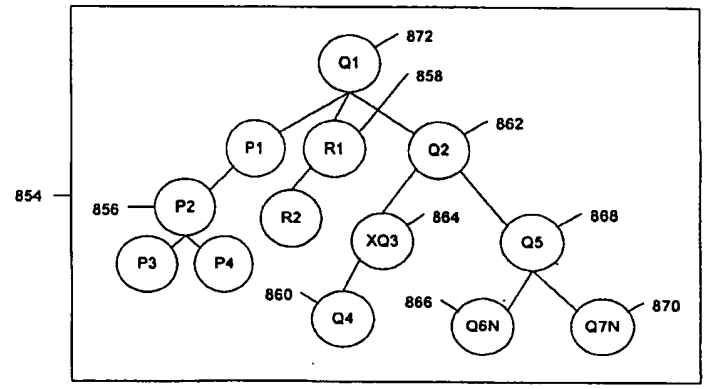

Atualização da Árvore de Objeto GUI
Script
Interpretador de Script gMg
745
775
Evento Destrói
770
Lista Destrói & SList
Lista Velha
760
750
Lista Nova
755
EnumWindowsProc
710
Janela de Nível de Topo
OS Windows Incluindo Aplicação Hosp.GUI
780
Aplicação Hospedeira
705
Q1
Q2
Q3
Q4
Q5
R1
R2
P1
P2
P3
P4
720
725
730
735
740
715

FIG.7





Estado Original



Deltas

FIG.8

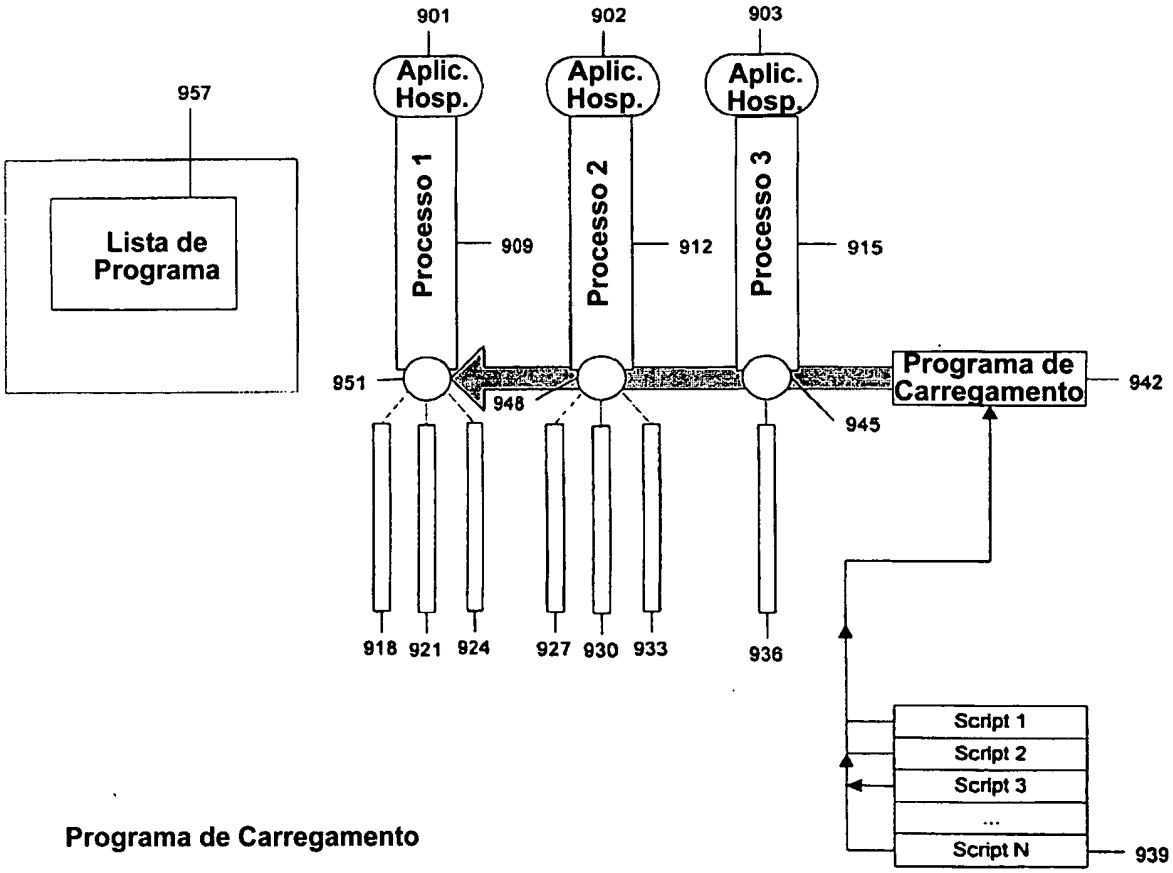

901
902
903
Aplic. Hosp.
Aplic. Hosp.
Aplic. Hosp.
957
Lista de Programa
Processo 1
Processo 2
Processo 3
909
912
915
951
948
945
Programa de Carregamento
942
918
921
924
927
930
933
936
Script 1
Script 2
Script 3
...
Script N
939
Programa de Carregamento

FIG.9

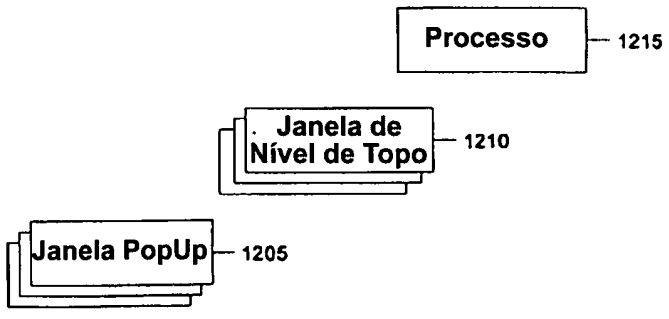

Processo
1215
Janela de Nível de Topo
1210
Janela PopUp
1205

FIG.12

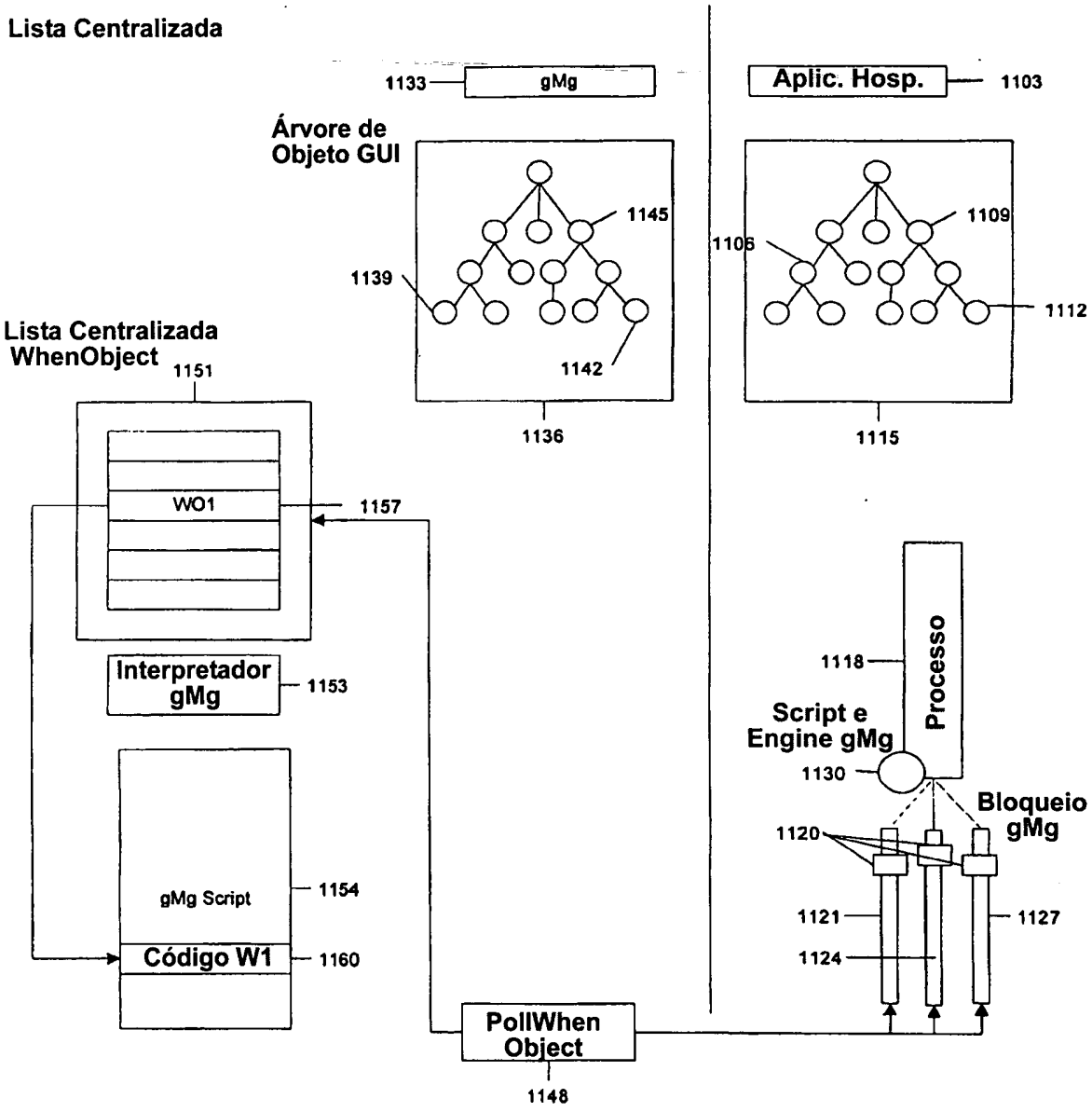

Lista Centralizada
1133
gMg
Aplic. Hosp.
1103
Árvore de Objeto GUI
1145
1139
1142
1136
1106
1109
1112
1115
Lista Centralizada WhenObject
1151
WO1
1157
Interpretador gMg
1153
gMg Script
1154
Código W1
1160
1118
Processo
Script e Engine gMg
1130
Bloqueio gMg
1120
1121
1127
1124
PollWhen Object
1148

FIG.11

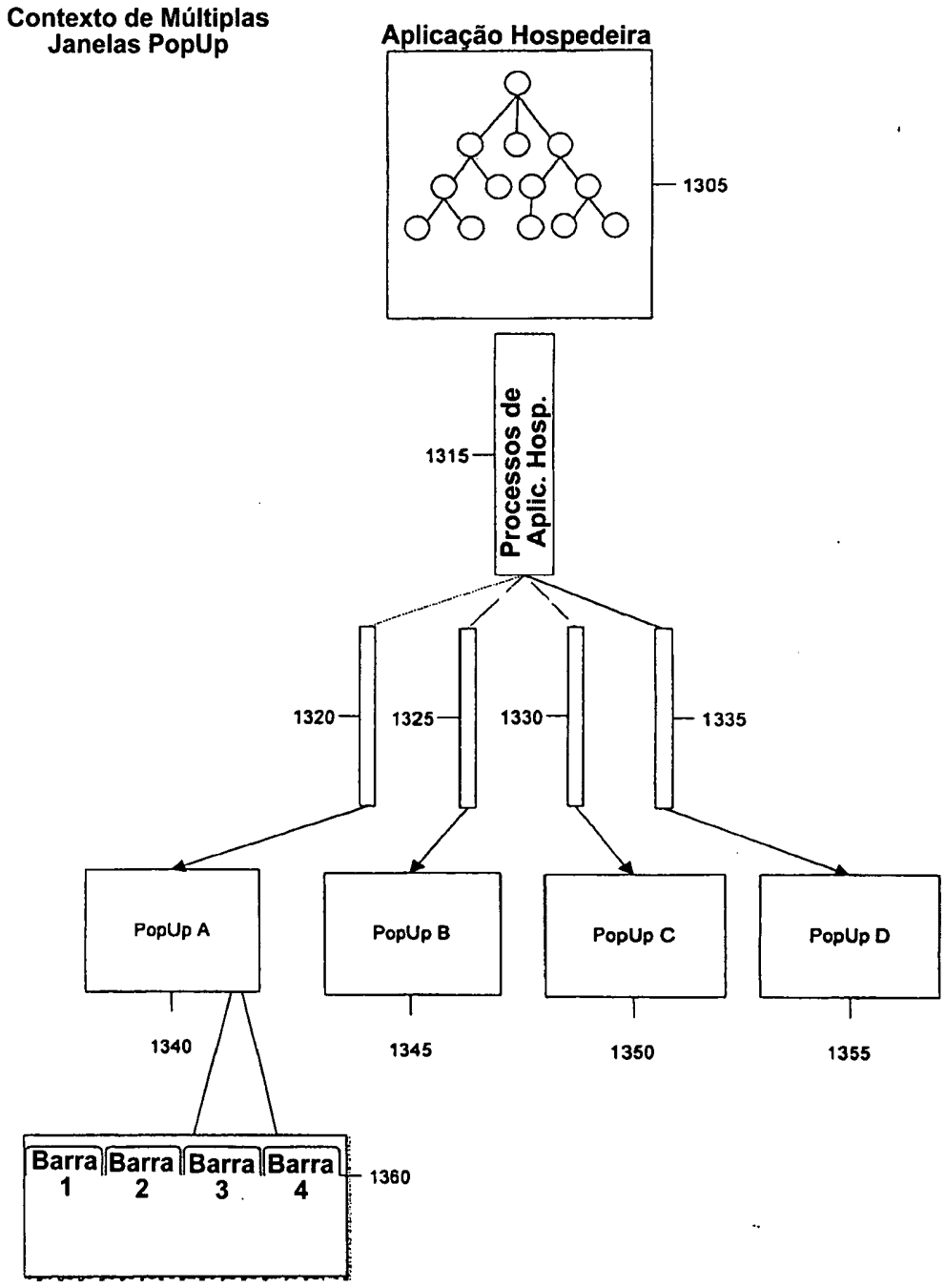

Contexto de Múltiplas Janelas PopUp
Aplicação Hospedeira
1305
1315
Processos de Aplic. Hosp.
1320
1325
1330
1335
PopUp A
PopUp B
PopUp C
PopUp D
1340
1345
1350
1355
Barra 1
Barra 2
Barra 3
Barra 4
1360

FIG.13

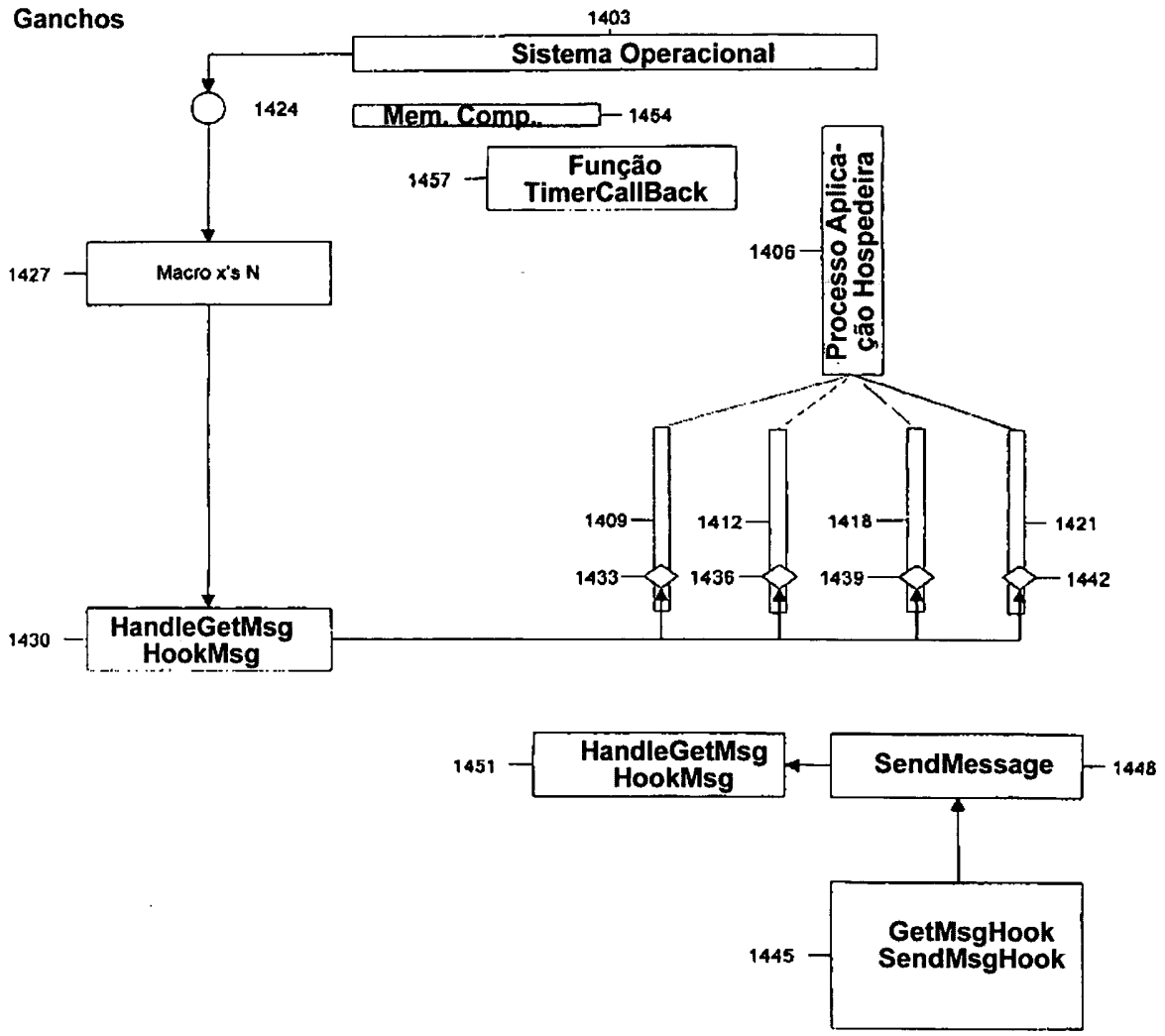

Ganchos
1403
Sistema Operacional
1424
Mem. Comp.
1454
1457
Função TimerCallBack
1406
Processo Aplica-ção Hospedeira
1427
Macro x's N
1409
1412
1418
1421
1433
1436
1439
1442
1430
HandleGetMsg HookMsg
1451
HandleGetMsg HookMsg
SendMessage
1448
1445
GetMsgHook SendMsgHook

FIG.14

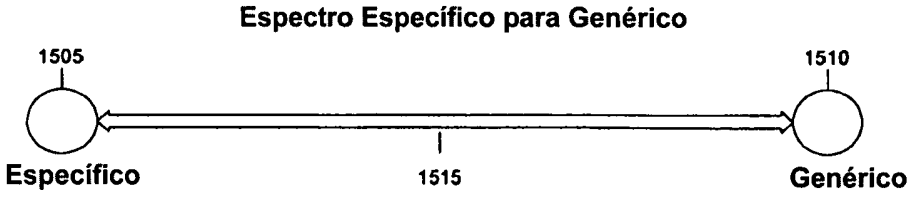

Espectro Específico para Genérico
1505
1510
Específico
1515
Genérico

FIG.15

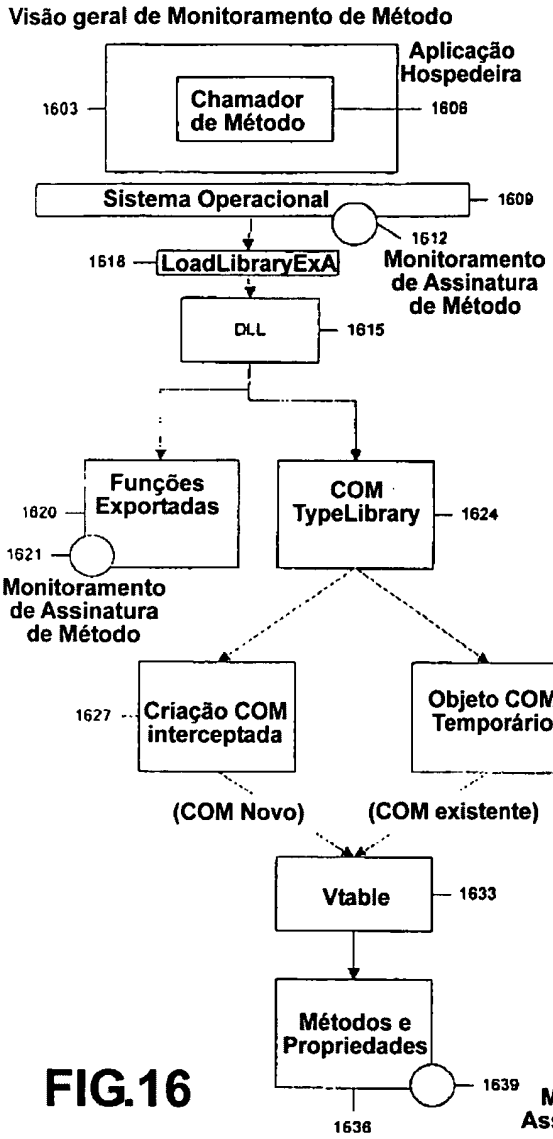


FIG.16

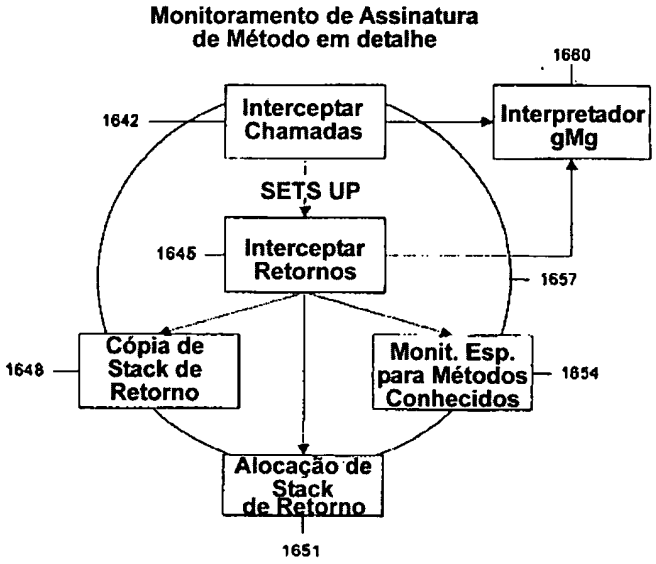


FIG.16A

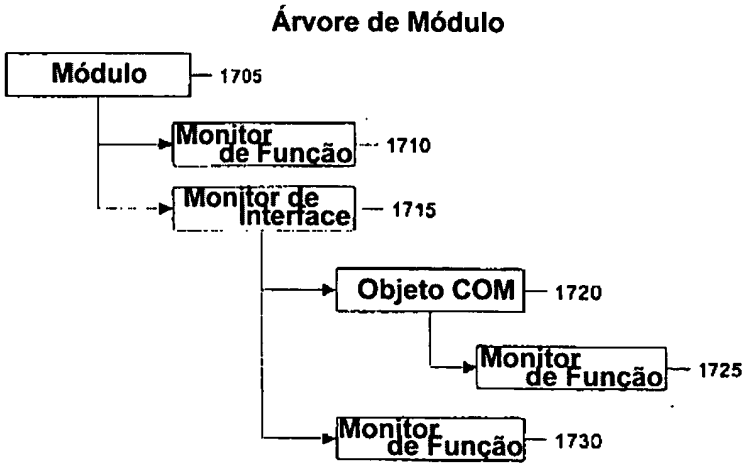


FIG.17

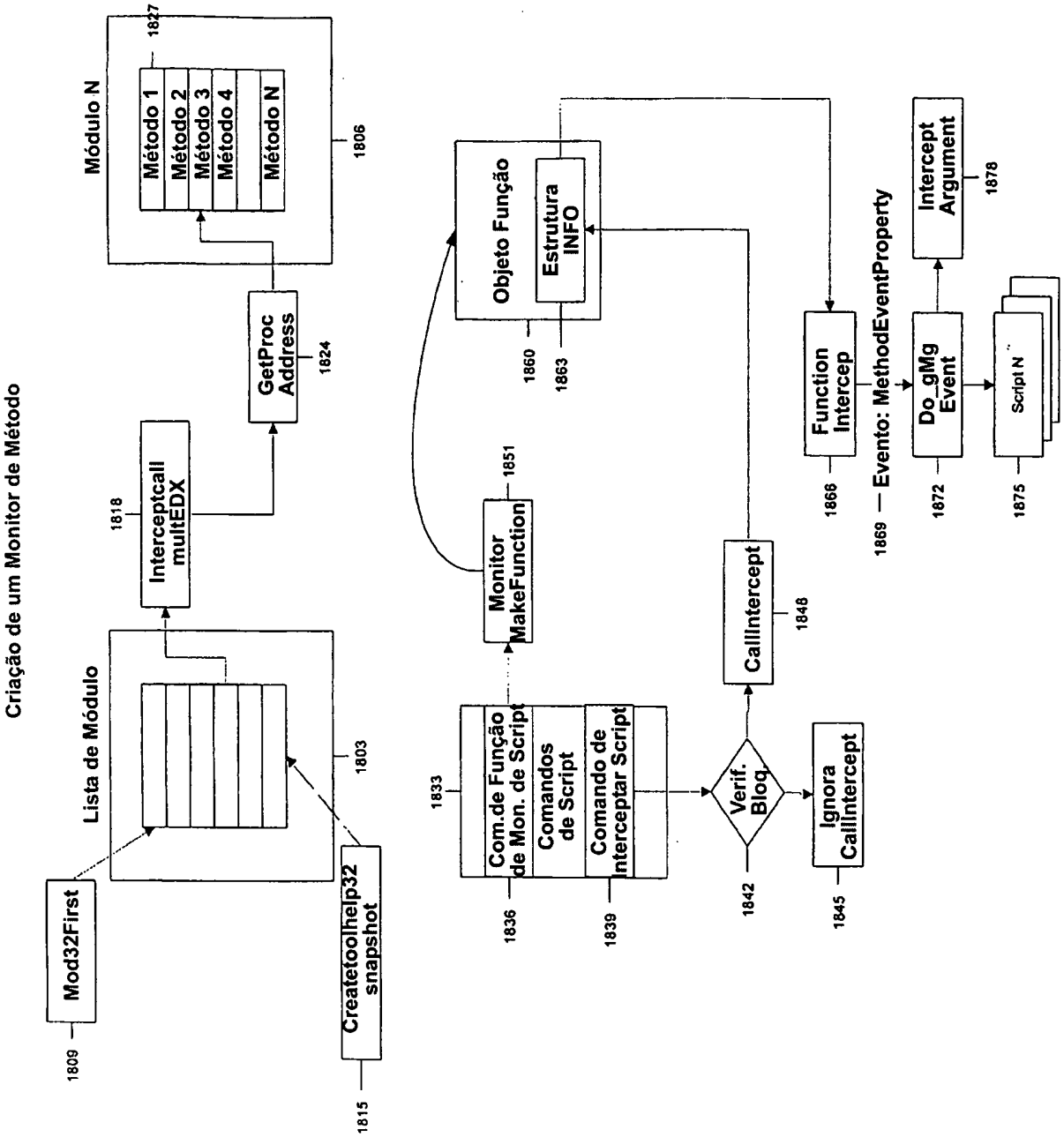

Criação de um Monitor de Método
Módulo N
1827
Método 1
Método 2
Método 3
Método 4
Método N
1806
GetProc Address
1824
Interceptcall multEDX
1818
Lista de Módulo
1803
Mod32First
1809
Createtoolhelp32 snapshot
1815
Objeto Função
Estrutura INFO
1860
1863
Monitor MakeFunction
1851
Com.de Função de Mon. de Script
Comandos de Script
Comando de Interceptar Script
1833
1836
1839
Verif. Bloq
1842
CallIntercept
1848
Ignora CallIntercept
1845
Function Intercep
1866
1869 — Evento: MethodEventProperty
Do_gMg Event
1872
Script N
1875
Intercept Argument
1878

FIG.18

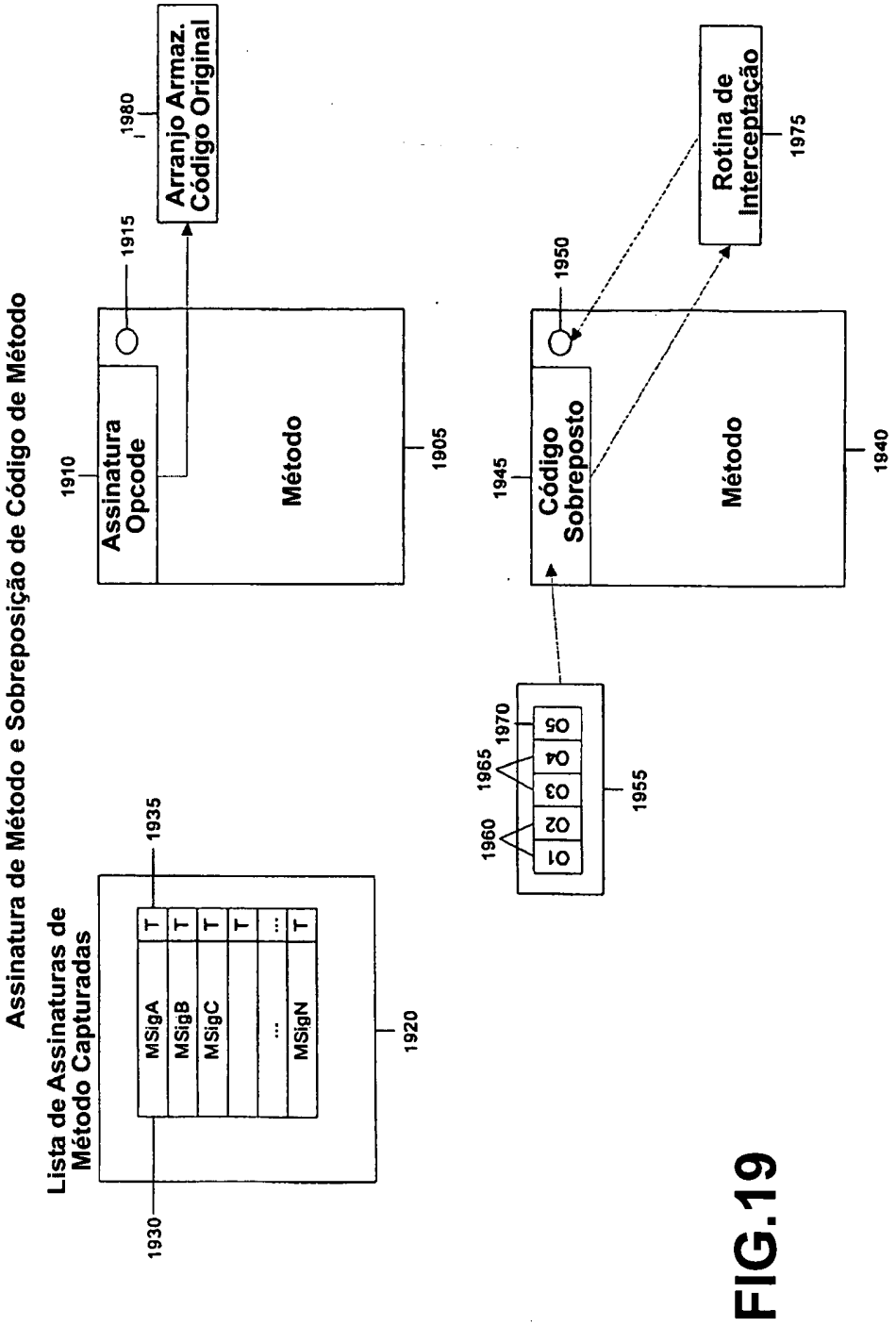

Assinatura de Método e Sobreposição de Código de Método
Lista de Assinaturas de Método Capturadas
MSigA
MSigB
MSigC
...
MSigN
T
1935
1920
1930
Assinatura Opcode
Método
1910
1915
1905
Arranjo Armaz. Código Original
1980
Código Sobreposto
Método
1945
1950
1940
Rotina de Interceptação
1975
01
02
03
04
05
1960
1965
1970
1955

FIG.19

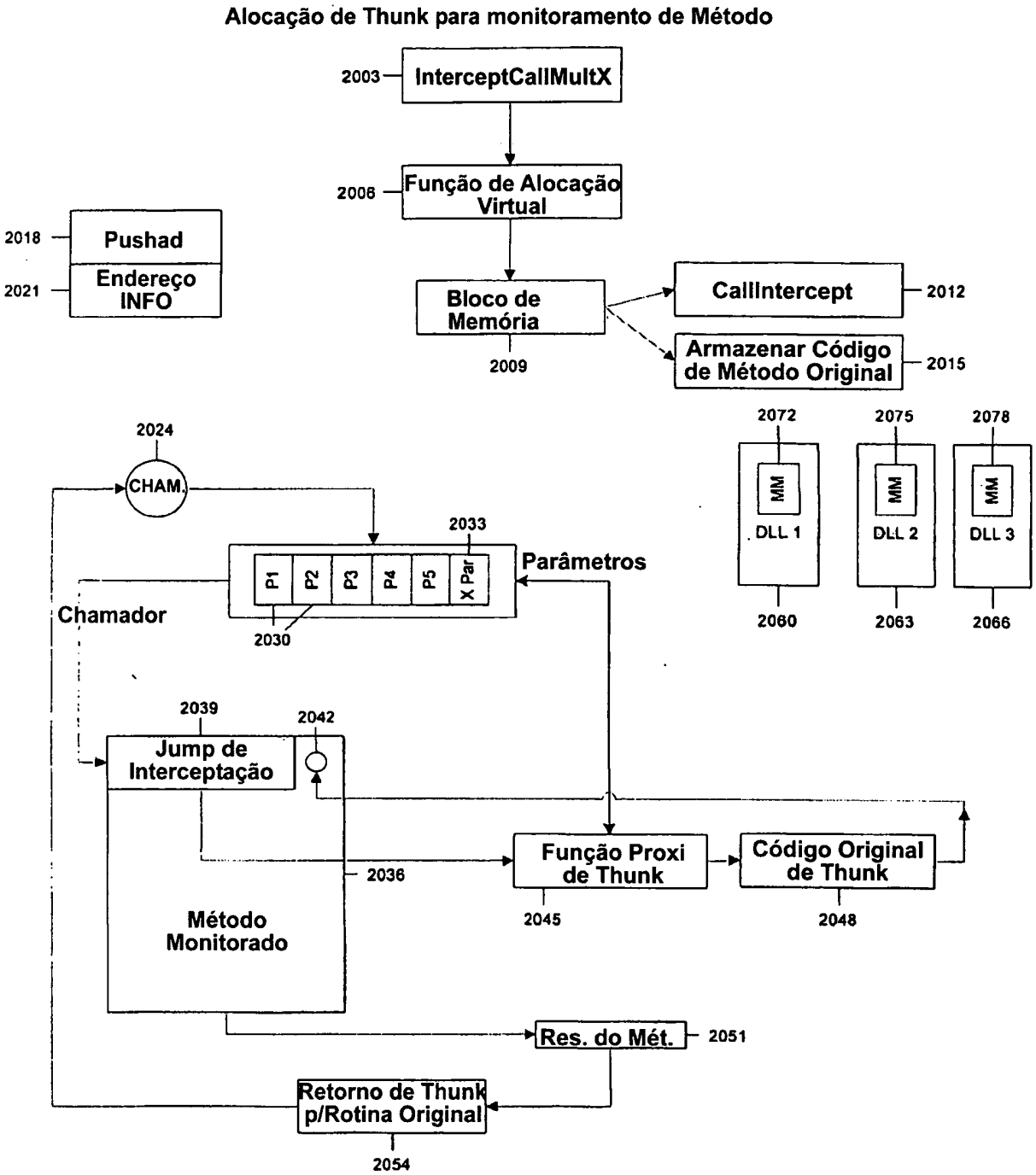

Alocação de Thunk para monitoramento de Método
2003
InterceptCallMultX
2006
Função de Alocação Virtual
2018
Pushad
2021
Endereço INFO
Bloco de Memória
2009
CallIntercept
2012
Armazenar Código de Método Original
2015
2024
CHAM.
2033
P1
P2
P3
P4
P5
X Par
Parâmetros
Chamador
2030
2072
2075
2078
MM
DLL 1
DLL 2
DLL 3
2060
2063
2066
2039
2042
Jump de Interceptação
2036
Método Monitorado
Função Proxi de Thunk
2045
Código Original de Thunk
2048
Res. do Mét.
2051
Retorno de Thunk p/Rotina Original
2054

FIG.20

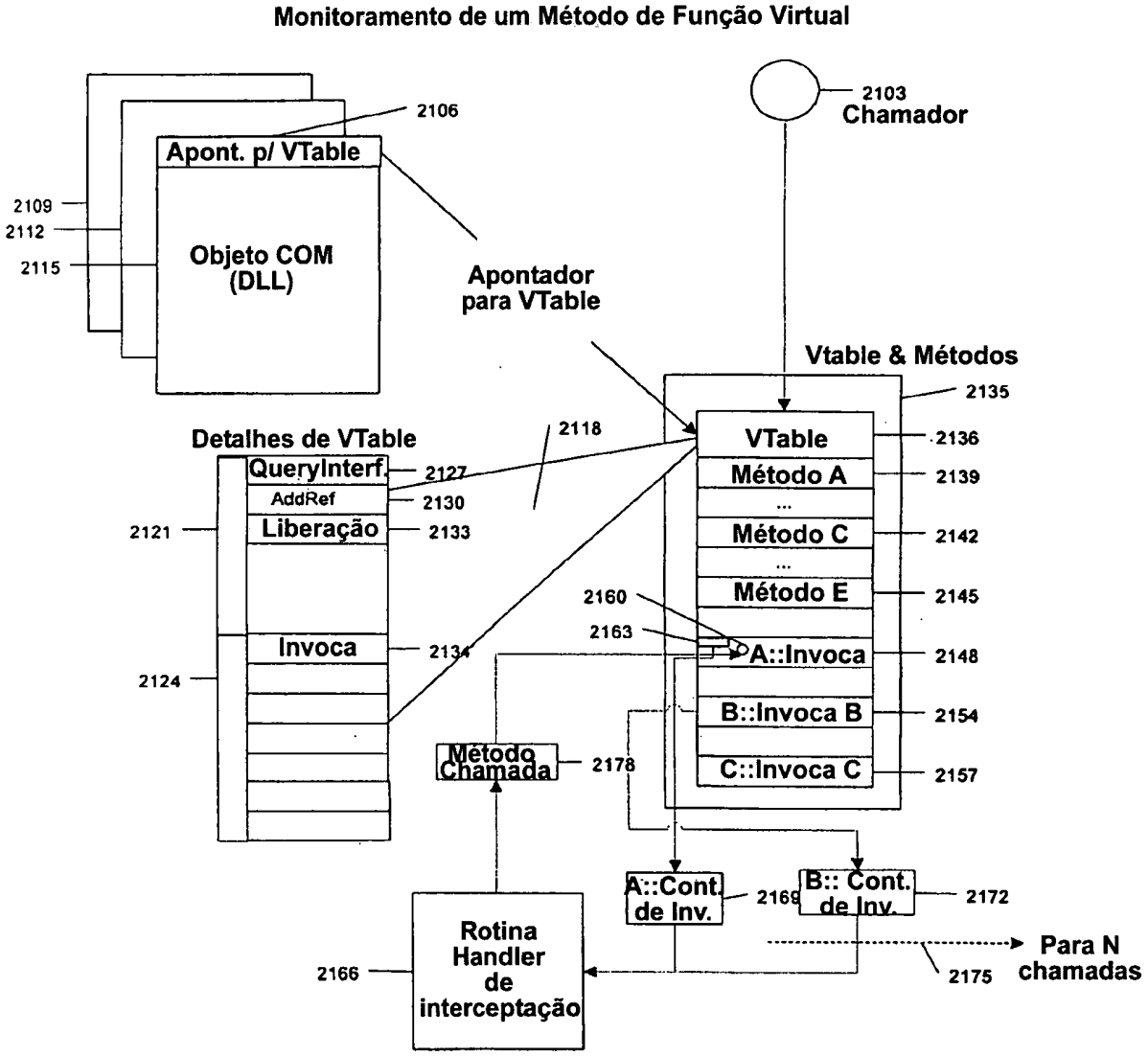

Monitoramento de um Método de Função Virtual
2103
Chamador
2106
Apont. p/ VTable
2109
2112
2115
Objeto COM
(DLL)
Apontador
para VTable
Vtable & Métodos
2135
Detalhes de VTable
2118
VTable
2136
QueryInterf.
2127
Método A
2139
AddRef
2130
...
2121
Liberação
2133
Método C
2142
...
Método E
2145
2160
2163
A::Invoca
2148
Invoca
2134
2124
B::Invoca B
2154
Método
Chamada
2178
C::Invoca C
2157
A::Cont.
de Inv.
2169
B:: Cont.
de Inv.
2172
Rotina
Handler
de
interceptação
2166
Para N
chamadas
2175

FIG.21

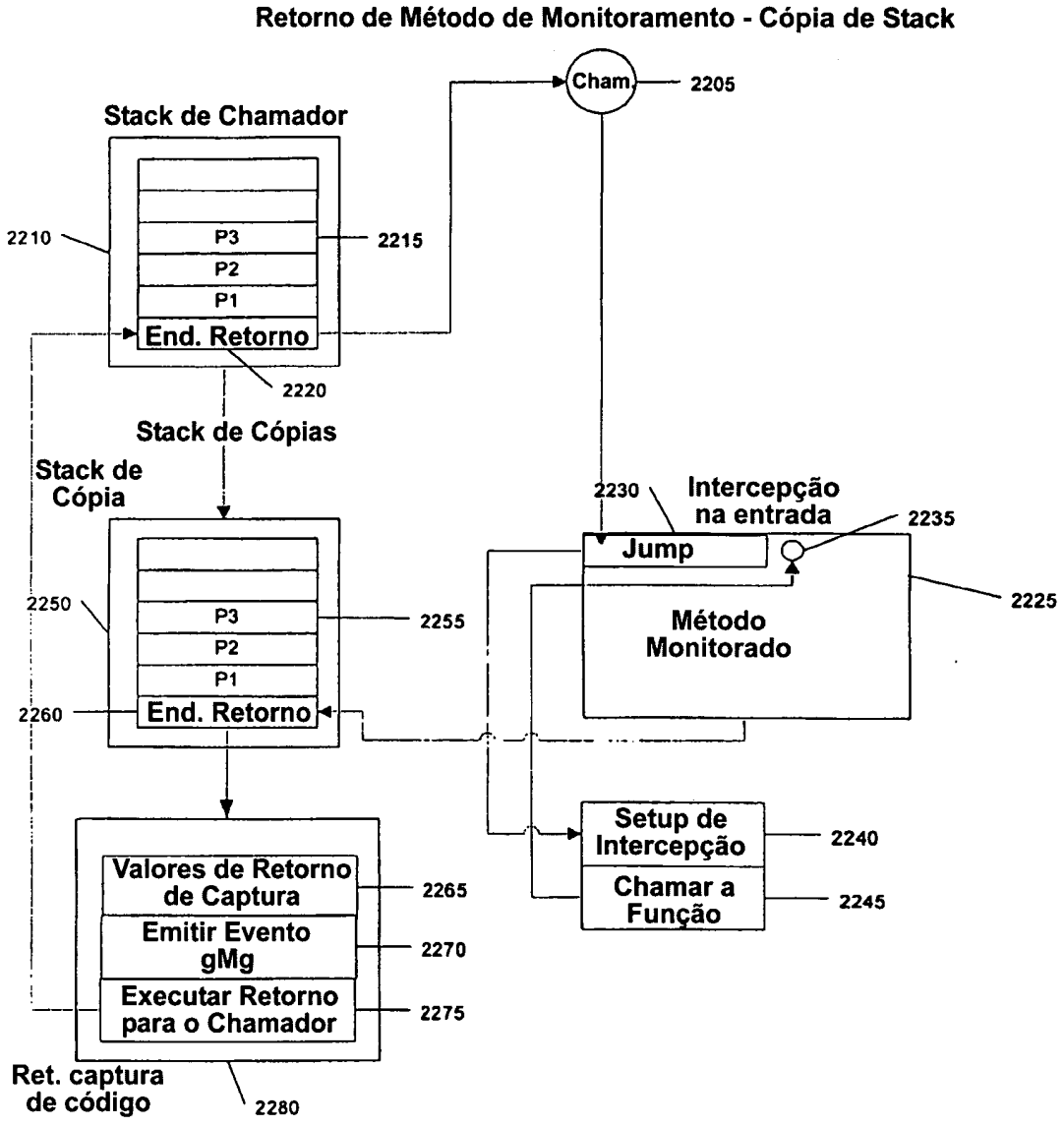

Retorno de Método de Monitoramento - Cópia de Stack
Cham.
2205
Stack de Chamador
2210
P3
P2
P1
End. Retorno
2215
2220
Stack de Cópias
Stack de Cópia
2230
Intercepção na entrada
2235
Jump
2225
Método Monitorado
2250
P3
P2
P1
End. Retorno
2255
2260
Setup de Intercepção
2240
Chamar a Função
2245
Valores de Retorno de Captura
2265
Emitir Evento gMg
2270
Executar Retorno para o Chamador
2275
Ret. captura de código
2280

FIG.22

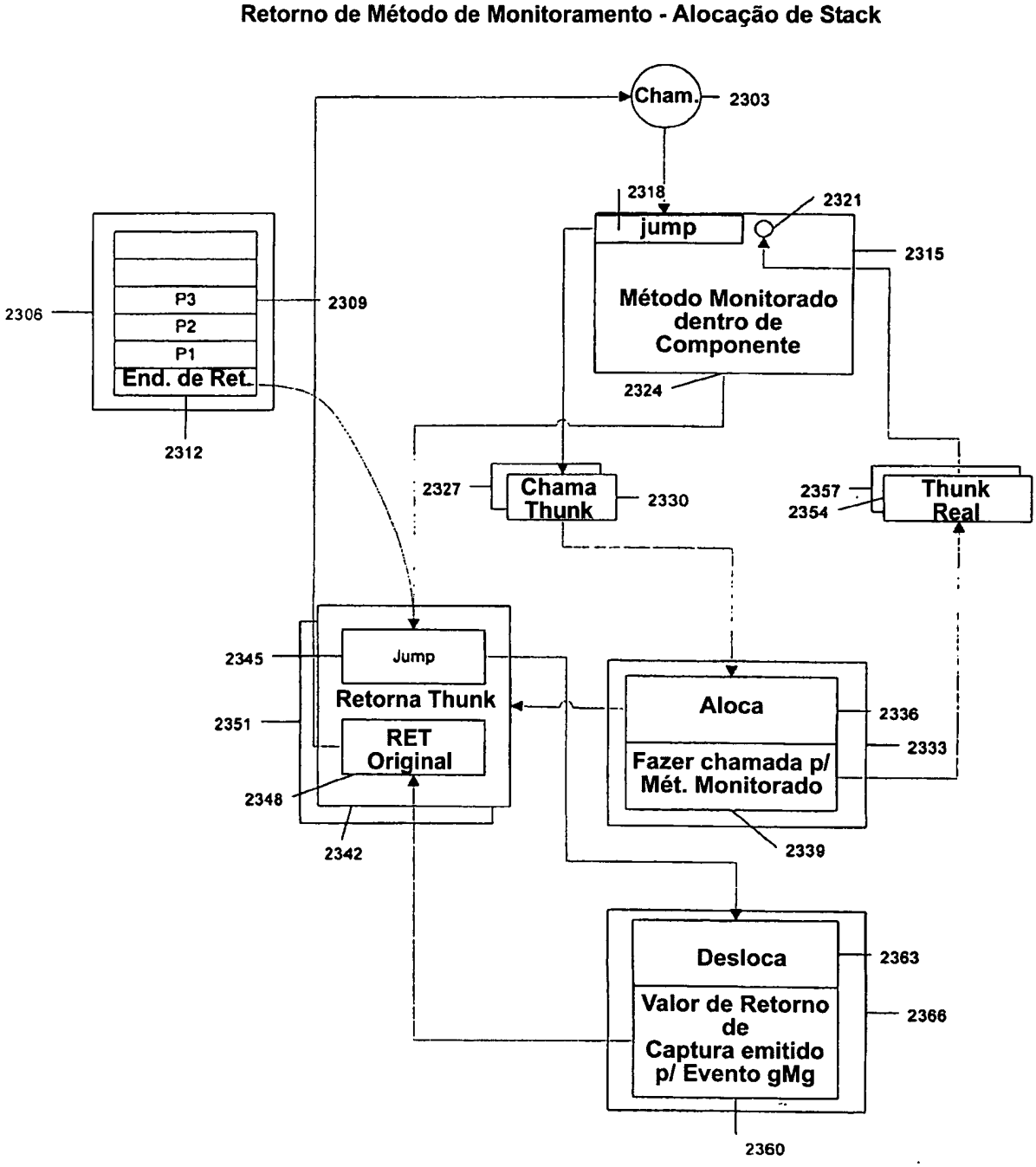

Retorno de Método de Monitoramento - Alocação de Stack
Cham.
2303
2318
2321
jump
2315
Método Monitorado dentro de Componente
2324
2306
P3
P2
P1
End. de Ret.
2309
2312
2327
Chama Thunk
2330
2357
2354
Thunk Real
2345
Jump
Retorna Thunk
2351
RET Original
2348
2342
Aloca
2336
2333
Fazer chamada p/ Mét. Monitorado
2339
Desloca
2363
Valor de Retorno de Captura emitido p/ Evento gMg
2366
2360

FIG.23

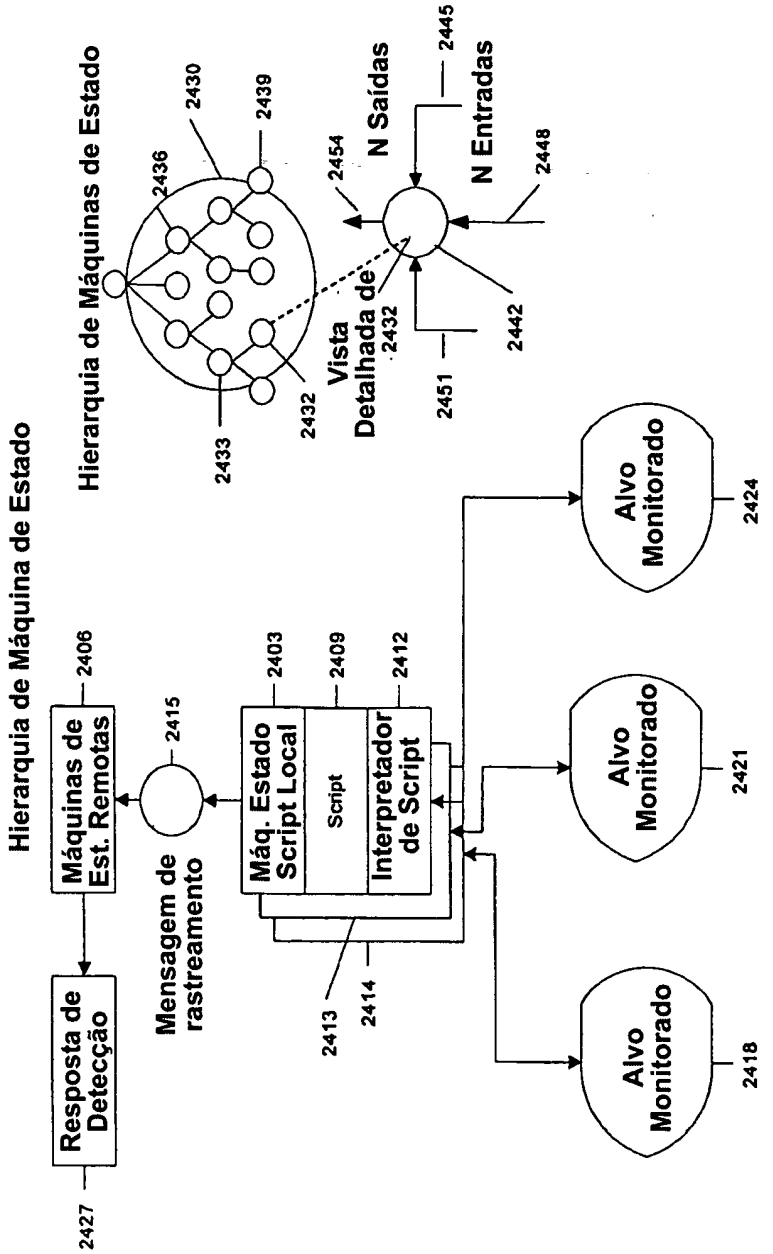

Hierarquia de Máquina de Estado
Máquinas de Est. Remotas
2406
Resposta de Detecção
2427
Mensagem de rastreamento
2415
Máq. Estado Script Local
2403
Script
2409
Interpretador de Script
2412
2413
2414
Hierarquia de Máquinas de Estado
2430
2439
2436
2433
2432
Vista Detalhada de 2432
2454
N Saídas
2445
N Entradas
2448
2451
2442
Alvo Monitorado
2424
Alvo Monitorado
2421
Alvo Monitorado
2418

FIG.24

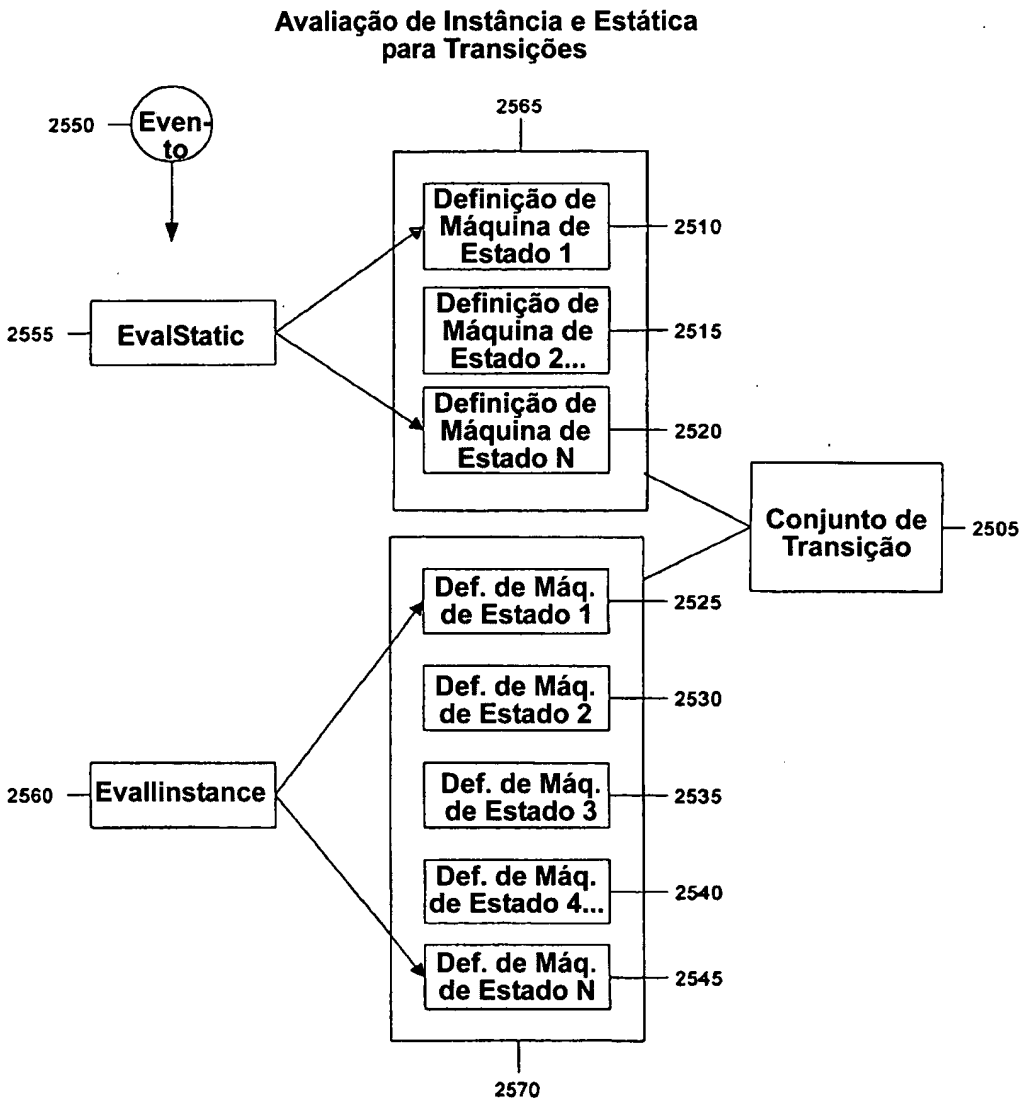

Avaliação de Instância e Estática para Transições
2550
Even-to
2565
Definição de Máquina de Estado 1
2510
Definição de Máquina de Estado 2...
2515
Definição de Máquina de Estado N
2520
2555
EvalStatic
Conjunto de Transição
2505
Def. de Máq. de Estado 1
2525
Def. de Máq. de Estado 2
2530
Def. de Máq. de Estado 3
2535
Def. de Máq. de Estado 4...
2540
Def. de Máq. de Estado N
2545
2560
EvalInstance
2570

FIG.25

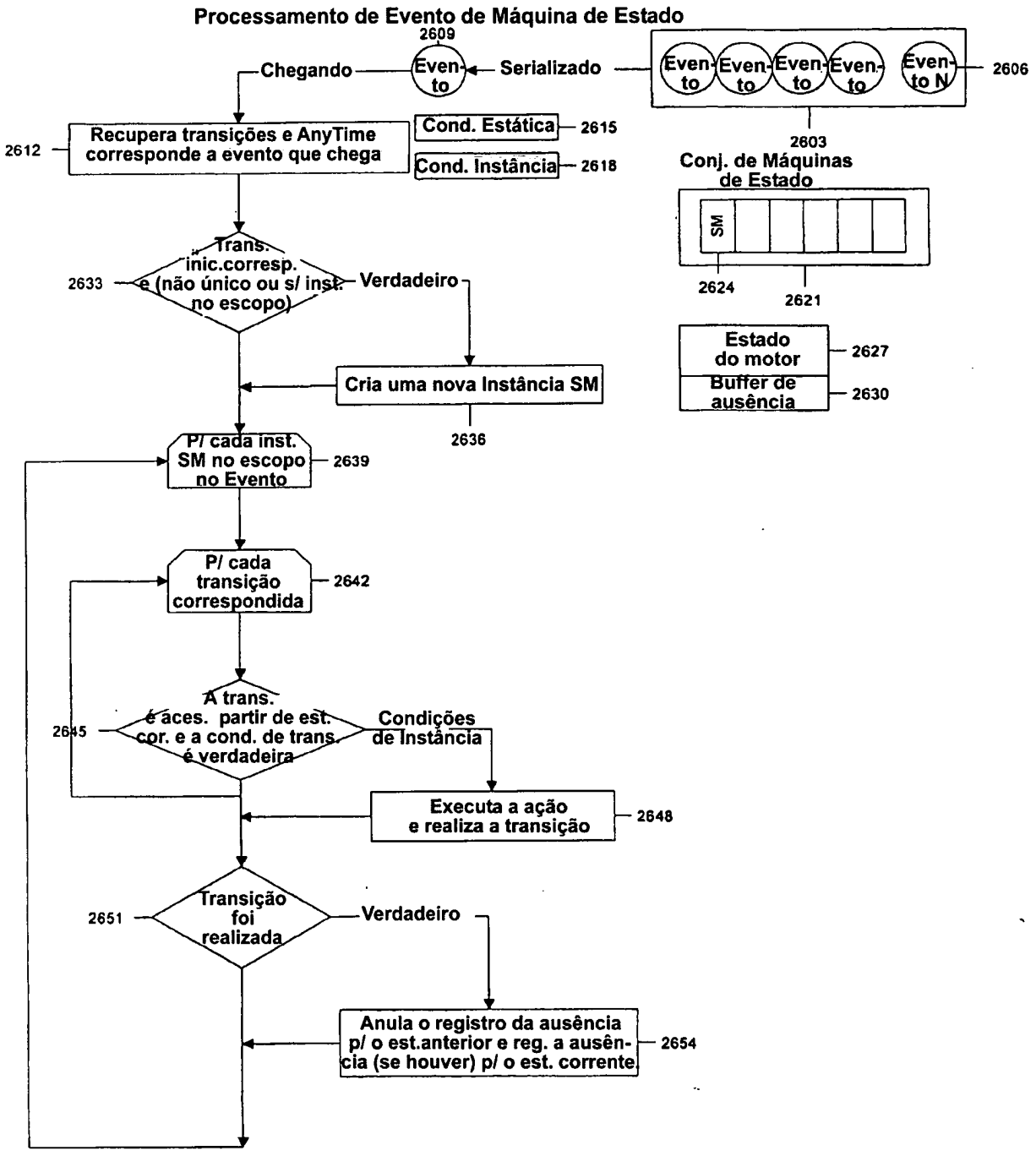

Processamento de Evento de Máquina de Estado
2609
Even-to
Chegando
Serializado
Even-to
Even-to
Even-to
Even-to
Even-to N
2606
2603
Conj. de Máquinas de Estado
SM
2624
2621
Recupera transições e AnyTime corresponde a evento que chega
2612
Cond. Estática
2615
Cond. Instância
2618
Trans. inic.corresp. e (não único ou s/ inst. no escopo)
2633
Verdadeiro
Cria uma nova Instância SM
2636
Estado do motor
2627
Buffer de ausência
2630
P/ cada inst. SM no escopo no Evento
2639
P/ cada transição correspondida
2642
A trans. é aces. partir de est. cor. e a cond. de trans. é verdadeira
2645
Condições de Instância
Executa a ação e realiza a transição
2648
Transição foi realizada
2651
Verdadeiro
Anula o registro da ausência p/ o est.anterior e reg. a ausência (se houver) p/ o est. corrente
2654

FIG.26

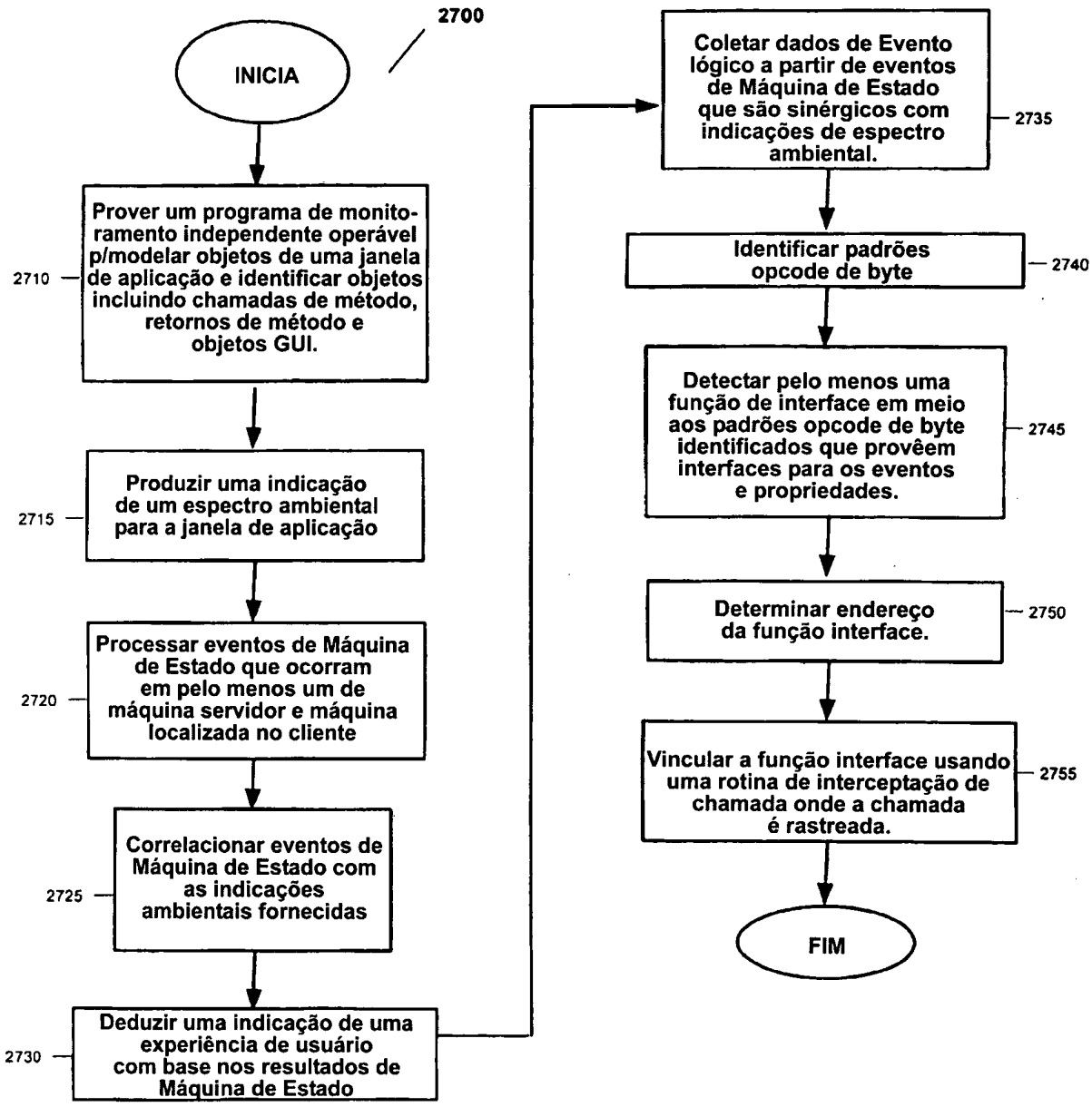

2700
INICIA
Prover um programa de monitoramento independente operável p/modelar objetos de uma janela de aplicação e identificar objetos incluindo chamadas de método, retornos de método e objetos GUI.
2710
Produzir uma indicação de um espectro ambiental para a janela de aplicação
2715
Processar eventos de Máquina de Estado que ocorram em pelo menos um de máquina servidor e máquina localizada no cliente
2720
Correlacionar eventos de Máquina de Estado com as indicações ambientais fornecidas
2725
Deduzir uma indicação de uma experiência de usuário com base nos resultados de Máquina de Estado
2730
Coletar dados de Evento lógico a partir de eventos de Máquina de Estado que são sinérgicos com indicações de espectro ambiental.
2735
Identificar padrões opcode de byte
2740
Detectar pelo menos uma função de interface em meio aos padrões opcode de byte identificados que provêem interfaces para os eventos e propriedades.
2745
Determinar endereço da função interface.
2750
Vincular a função interface usando uma rotina de interceptação de chamada onde a chamada é rastreada.
2755
FIM

FIG.27

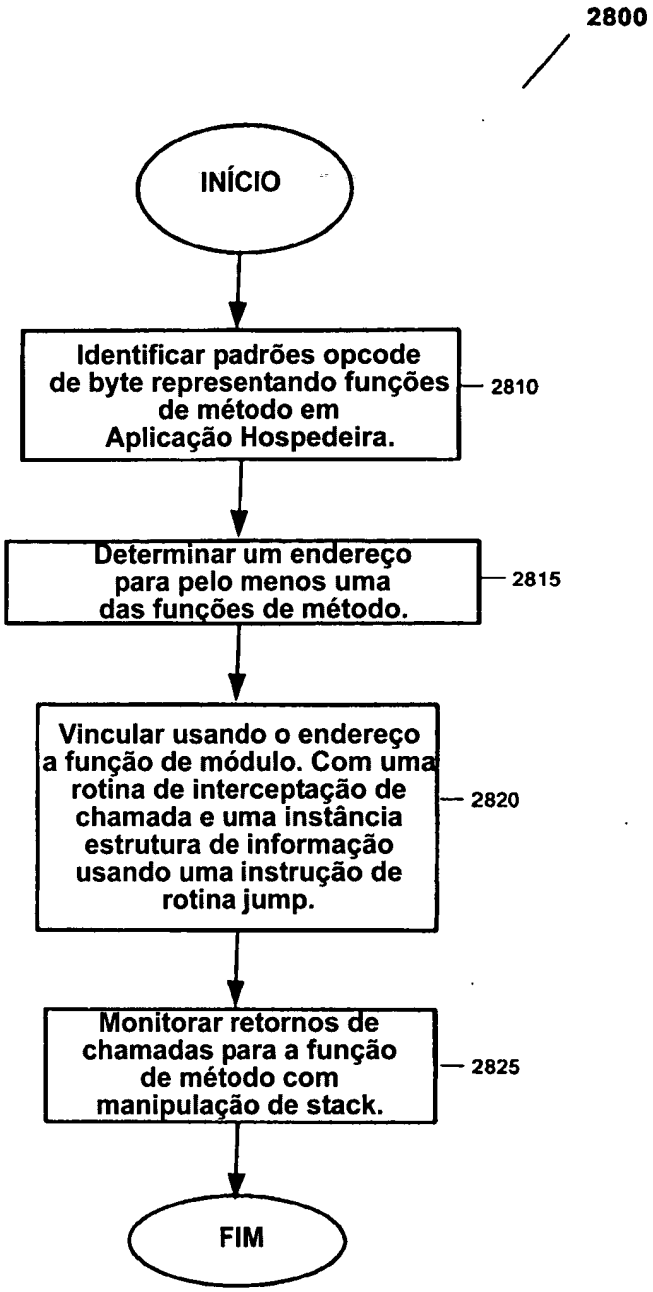

2800
INÍCIO
Identificar padrões opcode de byte representando funções de método em Aplicação Hospedeira.
2810
Determinar um endereço para pelo menos uma das funções de método.
2815
Vincular usando o endereço a função de módulo. Com uma rotina de interceptação de chamada e uma instância estrutura de informação usando uma instrução de rotina jump.
2820
Monitorar retornos de chamadas para a função de método com manipulação de stack.
2825
FIM

FIG.28

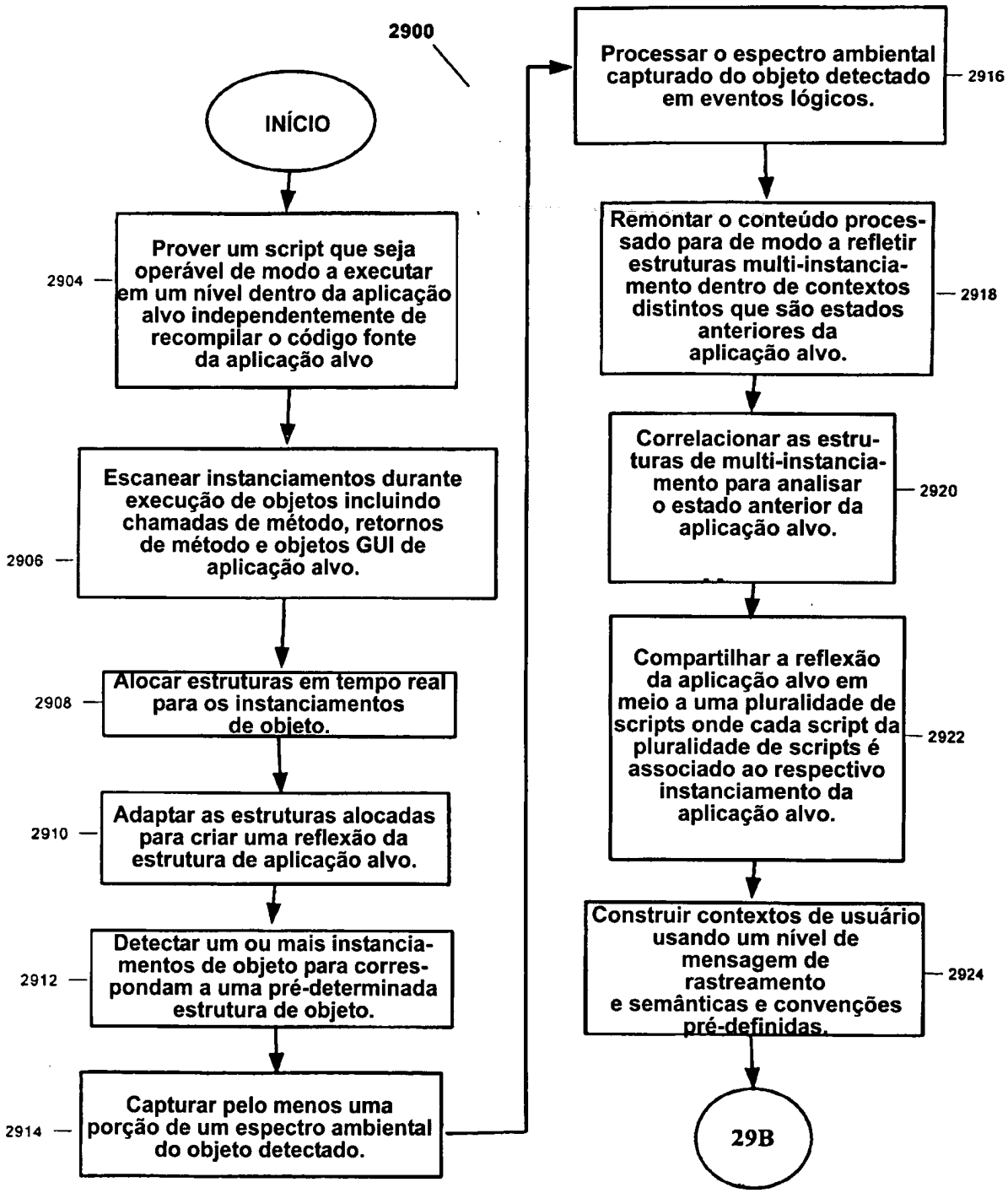

2900
INÍCIO
Prover um script que seja operável de modo a executar em um nível dentro da aplicação alvo independentemente de recompilar o código fonte da aplicação alvo
2904
Escanear instanciamentos durante execução de objetos incluindo chamadas de método, retornos de método e objetos GUI de aplicação alvo.
2906
Alocar estruturas em tempo real para os instanciamentos de objeto.
2908
Adaptar as estruturas alocadas para criar uma reflexão da estrutura de aplicação alvo.
2910
Detectar um ou mais instanciamentos de objeto para correspondam a uma pré-determinada estrutura de objeto.
2912
Capturar pelo menos uma porção de um espectro ambiental do objeto detectado.
2914
Processar o espectro ambiental capturado do objeto detectado em eventos lógicos.
2916
Remontar o conteúdo processado para de modo a refletir estruturas multi-instanciamento dentro de contextos distintos que são estados anteriores da aplicação alvo.
2918
Correlacionar as estruturas de multi-instanciamento para analisar o estado anterior da aplicação alvo.
2920
Compartilhar a reflexão da aplicação alvo em meio a uma pluralidade de scripts onde cada script da pluralidade de scripts é associado ao respectivo instanciamento da aplicação alvo.
2922
Construir contextos de usuário usando um nível de mensagem de rastreamento e semânticas e convenções pré-definidas.
2924
29B

FIG.29A

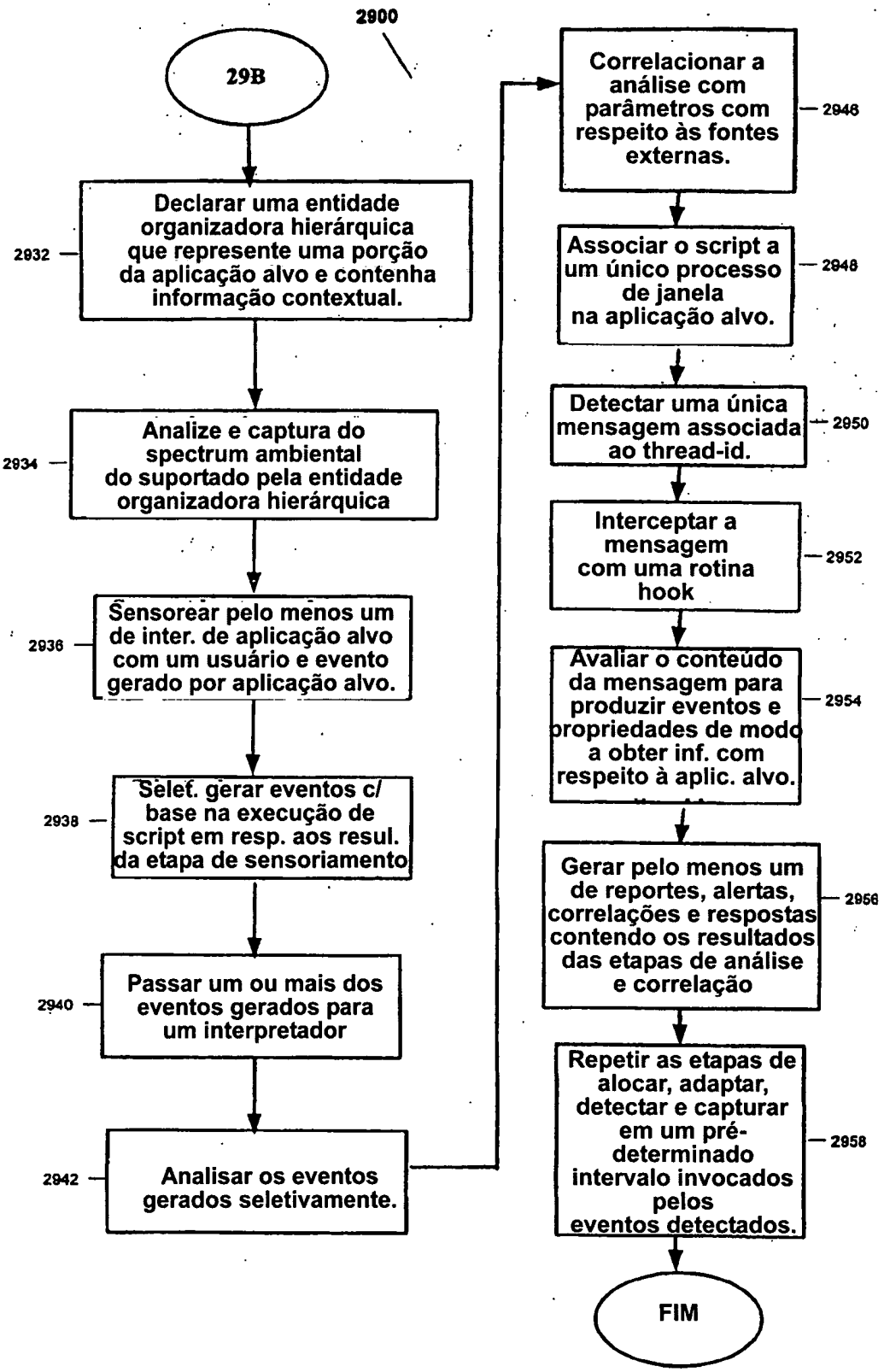

2900
29B
Declarar uma entidade organizadora hierárquica que represente uma porção da aplicação alvo e contenha informação contextual.
2932
Analize e captura do spectrum ambiental do suportado pela entidade organizadora hierárquica
2934
Sensorear pelo menos um de inter. de aplicação alvo com um usuário e evento gerado por aplicação alvo.
2936
Selet. gerar eventos c/ base na execução de script em resp. aos resul. da etapa de sensoriamento
2938
Passar um ou mais dos eventos gerados para um interpretador
2940
Analisar os eventos gerados seletivamente.
2942
Correlacionar a análise com parâmetros com respeito às fontes externas.
2946
Associar o script a um único processo de janela na aplicação alvo.
2948
Detectar uma única mensagem associada ao thread-id.
2950
Interceptar a mensagem com uma rotina hook
2952
Avaliar o conteúdo da mensagem para produzir eventos e propriedades de modo a obter inf. com respeito à aplic. alvo.
2954
Gerar pelo menos um de reportes, alertas, correlações e respostas contendo os resultados das etapas de análise e correlação
2956
Repetir as etapas de alocar, adaptar, detectar e capturar em um pré-determinado intervalo invocados pelos eventos detectados.
2958
FIM

FIG.29B

PI 0618412-0

RESUMO

"MÉTODO PARA MONITORAR EVENTOS DERIVADOS DE UMA CAMADA DE APRESENTAÇÃO, MÉTODO PARA ANALISAR EVENTOS DE MÉTODO, SISTEMA PARA MONITORAR EVENTOS EM UMA CAMADA DE APRESENTAÇÃO DE UMA APLICAÇÃO ALVO DE COMPUTADOR, E MÉTODO PARA MONITORAR UMA FUNCIONALIDADE DE APLICAÇÃO HOSPEDEIRA".

A presente invenção apresenta um sistema e método para monitorar eventos derivados de uma camada de apresentação de uma aplicação alvo de computador incluindo as etapas de prover, independentemente de recompilar o código de fonte da aplicação alvo, um script executado em um nível dentro da aplicação alvo. O script escanea instanciamentos durante a execução de objetos da aplicação alvo, e aloca estruturas em tempo real para instanciamentos de objeto. Estas estruturas alocadas são adaptadas para criar uma reflexão da estrutura de aplicação alvo, usada junto com instanciações do objeto detectado correspondentes a uma pré-determinada estrutura de objeto, para capturar uma porção de espectro ambiental do objeto detectado. Ademais, o sistema pode processar eventos de estado da máquina que ocorram pelo menos em um servidor e em uma máquina de cliente, correlacionar os eventos de estado da máquina com o espectro ambiental, e deduzir uma experiência de usuário com base nos eventos de estado de máquina correlacionados.