(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 267 534**

(51) Int. Cl.:
G06Q 40/00 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Número de solicitud europea: **00928844 .0**
(86) Fecha de presentación : **04.05.2000**
(87) Número de publicación de la solicitud: **1250667**
(87) Fecha de publicación de la solicitud: **23.10.2002**

(54) Título: **Producto para el tratamiento de tareas basado en componentes durante el procesamiento de demandas.**

(30) Prioridad: **04.05.1999 US 305234**

(45) Fecha de publicación de la mención BOPI:
16.03.2007

(45) Fecha de la publicación del folleto de la patente:
16.03.2007

(73) Titular/es: **Accenture LLP**
1661 Page Mill Road
Palo Alto, California 94304, US

(72) Inventor/es: **Guyan, George, V.;**
Pish, Robert, H. y
Muntada, Carles

(74) Agente: **Carvajal y Urquijo, Isabel**

Aviso: En el plazo de nueve meses a contar desde la fecha de publicación en el Boletín europeo de patentes, de la mención de concesión de la patente europea, cualquier persona podrá oponerse ante la Oficina Europea de Patentes a la patente concedida. La oposición deberá formularse por escrito y estar motivada; sólo se considerará como formulada una vez que se haya realizado el pago de la tasa de oposición (art. 99.1 del Convenio sobre concesión de Patentes Europeas).

Venta de fascículos: Oficina Española de Patentes y Marcas. Pº de la Castellana, 75 – 28071 Madrid

DESCRIPCIÓN

Producto para el tratamiento de tareas basado en componentes durante el procesamiento de demandas.

Campo de la invención

La presente invención se refiere a un aparato para hacer funcionar un software basado en componentes capaz de seleccionar tareas. Más particularmente, se refiere a la gestión de tareas para tratar una tarea durante un procesamiento de demanda de seguro utilizando un sistema informático.

Antecedentes de la invención

Los ordenadores se han convertido actualmente en una necesidad. Están presentes en casi todas las oficinas y hogares del mundo. Un entorno de hardware representativo se ilustra en la figura 1 perteneciente a la técnica anterior, que ilustra una configuración de hardware típica de un equipo de trabajo con una unidad 110 de procesamiento central, tal como un microprocesador, y una serie de otras unidades interconectadas a través de un sistema de bus 112. El equipo de trabajo mostrado en la figura 1 incluye una memoria 114 de acceso aleatorio (RAM, "random acces memory"), una memoria 116 de sólo lectura (ROM, "read only memory"), un adaptador 118 de E/S para conectar dispositivos periféricos tales como unidades 120 de almacenamiento de disco al bus 112, un adaptador 122 de interfaz de usuario para conectar un teclado 124, un ratón 126, un altavoz 128, un micrófono 132 y/u otros dispositivos de interfaz de usuario como una pantalla táctil (no mostrada) al bus 112, un adaptador 134 de comunicación para conectar el equipo de trabajo a una red de comunicación (por ejemplo, una red de procesamiento de datos) y un adaptador 136 de pantalla para conectar el bus 112 a un dispositivo 138 de pantalla. El equipo de trabajo normalmente tiene residiendo en el mismo un sistema operativo tal como el sistema operativo (SO) de Microsoft Windows NT o Windows/95, el sistema operativo de IBM OS/2, el MAC OS, o el sistema operativo UNIX.

La programación orientada a objetos (POO) se usa cada vez más para desarrollar aplicaciones complejas. A medida que la POO avanza hacia la corriente dominante del diseño y desarrollo de software, diversas soluciones de software requieren una adaptación para aprovechar los beneficios de la POO. Existe la necesidad de que estos principios de la POO se apliquen a una interfaz de transmisión de mensajes de un sistema de mensajes electrónicos de tal forma que pueda proporcionarse un conjunto de clases y objetos POO para la interfaz de transmisión de mensajes.

La POO es un proceso de desarrollo de software informático que utiliza objetos, incluyendo las etapas de analizar el problema, diseñar el sistema y construir el programa. Un objeto es un paquete de software que contiene tanto datos como un grupo de estructuras y procedimientos relacionados. Dado que contiene tanto datos como un grupo de estructuras y procedimientos, puede visualizase como un componente autónomo que no requiere otras estructuras, procedimientos o datos adicionales para llevar a cabo su tarea específica. Por lo tanto, la POO ve un programa informático como un grupo de componentes autónomos en su mayor parte, denominados objetos, cada uno de los cuales es responsable de una tarea específica. Este concepto de empaquetar datos, estructuras y procedimientos juntos en un componente o módulo se denomina encapsulación.

En general, los componentes POO son módulos de software reutilizables que presentan una interfaz que se ajusta a un modelo de objeto y a los que puede accederse en el tiempo de ejecución a través de una arquitectura de integración de componentes. Una arquitectura de integración de componentes es un conjunto de mecanismos de arquitectura que permiten a los módulos de software en diferentes espacios de proceso utilizar las capacidades o funciones de los demás. Esto se hace generalmente asumiendo un modelo de objeto de componente común sobre el que construir la arquitectura. A este respecto, resulta útil diferenciar entre un objeto y una clase de objetos. Un objeto es un ejemplo individual de la clase de objetos, que a menudo se denomina simplemente clase. Una clase de objetos puede considerarse como un proyecto a partir del cual pueden formarse numerosos objetos.

La POO permite al programador crear un objeto que forma parte de otro objeto. Por ejemplo, se dice que el objeto que representa un instrumento de pistón tiene una relación de composición con el objeto que representa un pistón. En realidad, un instrumento de pistón comprende un pistón, válvulas y muchos otros componentes; el hecho de que un pistón sea un elemento de un instrumento de pistón puede representarse lógica y semánticamente en POO por dos objetos.

La POO también permite la creación de un objeto que "depende de" otro objeto. Si hay dos objetos, uno que representa un instrumento de pistón y otro que representa un instrumento de pistón en el que el pistón está hecho de cerámica, entonces la relación entre los dos objetos no es la de composición. Un instrumento de pistón de cerámica no forma un instrumento de pistón. Más bien es meramente un tipo de instrumento de pistón que tiene una limitación más que el instrumento de pistón; su pistón está hecho de cerámica. En este caso, el objeto que representa el instrumento de pistón de cerámica se denomina objeto derivado y hereda todos los aspectos del objeto que representa el instrumento de pistón y añade una limitación o detalle adicional al mismo. El objeto que representa el instrumento de pistón de cerámica "depende de" el objeto que representa el instrumento de pistón. La relación entre estos objetos se denomina herencia.

Cuando el objeto o clase que representa el instrumento de pistón de cerámica hereda todos los aspectos de los objetos que representan el instrumento de pistón, hereda las características térmicas de un pistón estándar definido en la

clase de instrumentos de pistón. Sin embargo, el objeto del instrumento de pistón de cerámica redefine estas características térmicas específicas de la cerámica, que normalmente son diferentes de las asociadas a un pistón de metal. Salta por encima del original y utiliza nuevas funciones relativas a los pistones de cerámica. Diferentes tipos de instrumentos de pistón tienen características diferentes, pero pueden tener las mismas funciones subyacentes asociadas al mismo (por ejemplo, número de pistones en el instrumento, secuencias de encendido, lubricación, etc.). Para acceder a cada una de estas funciones en cualquier objeto de instrumento de pistón, un programador llamaría a las funciones iguales con el mismo nombre, pero cada tipo de instrumento de pistón puede tener implementaciones diferentes/redefinidas de las funciones bajo el mismo nombre. Esta habilidad para ocultar diferentes implementaciones de una función bajo el mismo nombre se denomina polimorfismo y simplifica enormemente la comunicación entre objetos.

Con los conceptos de relación de composición, encapsulación, herencia y polimorfismo, un objeto puede representar casi cualquier cosa en el mundo real. De hecho, la percepción lógica de la realidad es el único límite a la hora de determinar los tipos de cosas que pueden convertirse en objetos en un software orientado a objetos. Algunas categorías típicas son las siguientes:

- objetos pueden representar objetos físicos, tales como automóviles en una simulación de tráfico, componentes eléctricos en un programa de diseño de circuitos, países en un modelo económico, o un avión en un sistema de control del tráfico aéreo.

- Objetos pueden representar elementos del entorno ordenador-usuario tales como ventanas, menús u objetos gráficos.

- Un objeto puede representar un inventario, tal como un archivo o tabla personal de latitudes y longitudes de ciudades.

- Un objeto puede representar tipos de datos definidos por el usuario, tales como horas, ángulos y número complejos, o puntos en un plano.

Con esta enorme capacidad de un objeto para representar casi cualquier cosa separable lógicamente, la POO permite al desarrollador de software diseñar e implementar un programa informático que sea un modelo de algunos aspectos de la realidad, ya sea esta realidad una entidad física, un proceso, un sistema o una composición de material. Puesto que el objeto puede representar cualquier cosa, el desarrollador de software puede crear un objeto que pueda usarse como un componente en un proyecto de software mayor en el futuro.

Si el 90% de un nuevo programa de software POO consiste en componentes existentes ya probados, hechos a partir de objetos reutilizables preexistentes, entonces sólo debe escribirse y evaluarse desde cero el 10% restante del nuevo proyecto de software. Puesto que el 90% procede ya de un inventario de objetos reutilizables evaluados de forma exhaustiva, el campo potencial del que podría originarse un error es el 10% del programa. En consecuencia, la POO permite a los desarrolladores de software construir objetos a partir de otros objetos previamente construidos.

Este proceso se asemeja mucho a la compleja maquinaria que se construye a partir de montajes y submontajes. La tecnología POO hace por tanto la ingeniaría de programas (software) más similar a la ingeniería de sistemas (hardware) porque el software se construye a partir de componentes existentes, que están disponibles en forma de objetos para los desarrolladores. Todo ello se suma para mejorar la calidad del software así como aumentar la velocidad de su desarrollo.

El documento WO 91/08453 A describe un sistema informático para un entorno de desarrollo de códigos de programas. El sistema recibe normas de lenguaje de alto nivel de un lenguaje de programación almacenado en un almacén de datos junto con datos de modelación almacenados en un almacén central. El sistema introduce los datos almacenados y las normas de lenguaje en un generador de sistemas para generar un código fuente de programa original en un lenguaje que soportan varias entidades de hardware, y entonces junta y compila el código. El sistema incluye Módulos de Trabajo de Desarrollo para permitir que el programador especifique la entidad y las relaciones e introduzca normas, componentes, vistas, paneles de archivos y ventanas, que se almacenan en un almacén de datos y se incorporan al programa ejecutable final.

POTEL M: "MVP: Model-View-Presenter - The Taligent Programming Model for C++ and Java" Taligent Inc. 1996 describe un modelo de programación de clase abierta para marcos orientados a objetos que informatiza la programación en dos conceptos fundamentales: gestión de datos e interfaz de usuario. La gestión de datos incluye modelar resúmenes de datos, seleccionar subconjuntos de datos e identificar operaciones que pueden efectuarse sobre los datos. La interfaz de usuario incluye la visualización de los datos, acciones iniciadas por el usuario que actúan sobre los datos y la presentación de las interacciones asociadas unívocamente ("mapped") a los datos. Se describe además cómo estas acciones en la gestión de datos y la interfaz de usuario pueden situarse de manera diversa dependiendo de la partición especificada en entornos cliente-servidor.

Sumario de la invención

Un objeto de la presente invención es proporcionar un aparato para hacer funcionar un software basado en componentes capaz de seleccionar tareas, que evite un proceso costoso que requiera mucho tiempo antes escribir un código.

Según la invención, este objetivo se consigue mediante un aparato que presenta las características de la reivindicación 1.

Los desarrollos adicionales ventajosos se exponen en las reivindicaciones dependientes.

Según la invención, el aparato incluye un programa informático para hacer funcionar un software basado en componentes capaz de tratar tareas relacionadas con los seguros. El programa incluye un componente de datos que almacena, recupera y manipula datos utilizando una pluralidad de funciones. También se proporciona un componente de cliente que incluye un adaptador del componente de cliente que transmite y recibe datos hacia/desde el componente de datos. El componente de cliente también incluye un objeto de negocio que sirve como memoria caché de datos e incluye una lógica para manipular los datos. Se incluye también un controlador de interfaz de usuario que se adapta para tratar eventos generados por un usuario utilizando el objeto de negocio para memorizar datos en caché y el adaptador del componente de cliente para mantener los datos en última instancia en un almacén de datos. En uso, el componente de cliente permite a un usuario definir tareas que al completarse logran una finalidad relacionada con los seguros. Además, el usuario puede introducir normas que dictan qué tareas deberían seleccionarse basándose en un conjunto de eventos predeterminados. Los eventos se reciben entonces desde cualquier fuente, tal como una cola de eventos común. Finalmente, las tareas se seleccionan y se extraen basándose en los eventos recibidos.

Descripción de los dibujos

El objetivo anterior, así como los aspectos y ventajas, se entenderán mejor a partir de la siguiente descripción detallada de una realización preferida de la invención con referencia a los dibujos, en los que:

la figura 1 de la técnica anterior es un diagrama esquemático de la presente invención, y

la figura 2A es un diagrama de bloques de una realización de la presente invención,

la figura 2B es un diagrama de flujo que muestra cómo funcionan de manera general los componentes según una realización de la presente invención;

la figura 2C es un diagrama de flujo que muestra cómo funciona el controlador UI según una realización de la presente invención;

la figura 2D es un diagrama de flujo que muestra las interacciones entre el CCA, el CCI y el componente del servidor según una realización de la presente invención;

la figura 3 muestra el ciclo de vida de una interfaz de usuario típica y los métodos estándar que forman parte del marco de procesamiento de ventana;

la figura 4 es una ilustración que muestra cómo se vuelven a dibujar y se recopilan diferentes lenguajes;

la figura 5 es un diagrama de bloques de un objeto de arquitectura;

la figura 6 es una ilustración que muestra la disposición física de tablas de codificación-decodificación según una realización de la presente invención;

la figura 7 es un diagrama lógico según una realización de la presente invención;

la figura 8 es un diagrama de bloques del marco de seguridad y sus componentes;

la figura 9 es una ilustración que muestra las relaciones entre el elemento de seguridad y otros elementos;

la figura 10 es una ilustración del componente de negociación de una realización de la presente invención;

la figura 11 es un diagrama de flujo de las operaciones llevadas a cabo por el componente de organización según una realización de la presente invención;

la figura 12 es una ilustración del componente de participante según una realización de la presente invención;

la figura 13 es un diagrama de flujo de las operaciones llevadas a cabo por el componente de asistente de tareas según una realización de la presente invención;

la figura 14 es una ilustración del procesador de eventos en combinación con otros componentes del sistema según una realización de la presente invención; y

la figura 15 es una ilustración del motor de tareas según una realización de la presente invención.

Descripción de la invención

Los lenguajes de programación están empezando a soportar completamente los principios de POO, tales como encapsulación, herencia, polimorfismo y relación de composición. Con la llegada del lenguaje C++, muchos desarrolladores de software comercial se han unido a la POO. El C++ es un lenguaje de POO que ofrece un código rápido y ejecutable automáticamente. Además, el C++ es adecuado para proyectos tanto de aplicación comercial como de programación de sistemas. Por ahora, el C++ parece ser la elección más popular entre muchos programadores de POO, pero existe una gran cantidad de otros lenguajes de POO, tales como Smalltalk, Common Lisp Object System (CLOS), y Eiffel. Además, las capacidades de POO se están añadiendo a lenguajes de programación informática popular más tradicionales tales como Pascal.

Los beneficios de las clases de objetos pueden resumirse como sigue:

- Los objetos y sus clases correspondientes descomponen los complejos problemas de programación en problemas más pequeños y simples.

- La encapsulación hace efectiva la abstracción de datos mediante la organización de los datos en objetos pequeños e independientes que pueden comunicarse entre sí. La encapsulación protege los datos en un objeto de un daño accidental, pero permite que otros objetos interactúen con los datos llamando a las funciones y estructuras que forman parte del objeto.

- La subclasificación y la herencia hacen posible extender y modificar objetos mediante la derivación de nuevos tipos de objetos a partir de las clases estándar disponibles en el sistema. Así se crean nuevas capacidades sin tener que empezar de cero.

- El polimorfismo y la herencia múltiple hacen posible que diferentes programadores mezclen y hagan coincidir características de muchas clases diferentes y creen objetos especializados que aún pueden funcionar con objetos relacionados de formas predecibles.

- Las jerarquías de clases y las jerarquías de contención proporcionan un mecanismo flexible para modelar objetos del mundo real y las relaciones entre los mismos.

- Las bibliotecas de clases reutilizables son útiles en muchas situaciones pero también tienen algunas limitaciones. Por ejemplo:

- Complejidad. En un sistema complejo, las jerarquías de clases para clases relacionadas pueden volverse extremadamente confusas, con muchas docenas o incluso cientos de clases.

- Flujo de control. Un programa escrito con la ayuda de bibliotecas de clases sigue siendo responsable del flujo de control (es decir, debe controlar las interacciones entre todos los objetos creados a partir de una biblioteca particular). El programador tiene que decidir a qué funciones llamar en qué momentos para qué tipo de objetos.

- Duplicación del esfuerzo. Aunque las bibliotecas de clases permiten a los programadores utilizar y reutilizar muchas piezas pequeñas de código, cada programador une esas piezas de forma diferente. Dos programadores diferentes pueden usar el mismo conjunto de bibliotecas de clases para escribir dos programas que hagan exactamente lo mismo pero cuya estructura interna (es decir, el diseño) pueda ser bastante diferente, dependiendo de cientos de pequeñas decisiones que cada programador hace durante el proceso. Inevitablemente, piezas similares de código acaban realizando cosas similares de formas ligeramente diferentes y no funcionan juntas tan bien como deberían.

Las bibliotecas de clases son muy flexibles. A medida que los programas se hacen más complejos, más programadores se ven obligados a reinventar soluciones básicas a problemas básicos una y otra vez. Una extensión relativamente nueva del concepto de biblioteca de clases es tener un marco de bibliotecas de clases. Este marco es más complejo y consiste en colecciones importantes de clases cooperantes que capturan tanto los patrones a pequeña escala como los mecanismos principales que implementan los requisitos comunes y diseñan en un campo de aplicación específico. Se desarrollaron en primer lugar para programadores de aplicaciones libres a partir de tareas implicadas en menús desplegables, ventanas, cuadros de diálogo y otros elementos de interfaz de usuario estándar para ordenadores personales.

Los marcos también representan un cambio en la forma en que los programadores piensan acerca de la interacción entre el código que escriben y el código escrito por otros. En las inicializaciones de la programación de procedimientos, el programador llamaba a bibliotecas que proporcionaba el sistema operativo para realizar ciertas tareas, pero básicamente el programa ejecutaba la página de principio a fin, y el programador sólo era responsable del flujo de control. Esto era apropiado para imprimir nóminas, calcular una tabla matemática o solucionar otros problemas con un programa que se ejecutaba sólo de una forma.

El desarrollo de interfaces de usuario gráficas empezó a dar la vuelta a esta disposición de programación de procedimientos. Estas interfaces permiten al usuario, más que a la lógica de programa, dirigir el programa y decidir cuándo deben realizarse ciertas acciones. Hoy día, la mayoría de software de ordenador personal cumple esto mediante un bucle de eventos que controla el ratón, el teclado y otras fuentes de eventos externos y llama a las partes apropiadas

del código de programador de acuerdo con las acciones que el usuario realiza. El programador ya no determina el orden en que ocurren los eventos. En lugar de ello, un programa se divide en piezas separadas que se llaman en momentos impredecibles y en un orden impredecible. Al ceder el control a los usuarios de este modo, el desarrollador crea un programa que es mucho más fácil de usar. No obstante, las piezas individuales del programa escritas por el desarrollador todavía llaman a las bibliotecas proporcionadas por el sistema operativo para llevar a cabo ciertas tareas, y el programador todavía debe determinar el flujo de control en cada pieza después de haber sido llamada por el bucle de eventos. El código de aplicación todavía "se dispone sobre" del sistema.

Incluso los programas de bucle de eventos requieren que los programadores escriban una gran cantidad del código que no necesitaría escribirse por separado para cada aplicación. El concepto de marco de aplicación lleva más lejos el concepto de bucle de eventos. En lugar de tratar con todos los elementos básicos de la construcción de menús básicos, ventanas y cuadros de diálogo y después hacer que todos funcionen conjuntamente, los programadores que usan marcos de aplicación empiezan con código de aplicación de trabajo y elementos de interfaz de usuario básicos en su lugar. Posteriormente, construyen a partir de ahí sustituyendo algunas de las capacidades genéricas del marco por las capacidades específicas de la aplicación pretendida.

Los marcos de aplicación reducen la cantidad total de código que un programador debe escribir desde cero. Sin embargo, puesto que el marco es en realidad una aplicación genérica que despliega ventanas, soporta copiar y pegar, etcétera, el programador también puede ceder el control en mayor medida de lo que permiten los programas de bucle de eventos. El código del marco se encarga de casi todo el tratamiento de eventos y flujo de control, y el código del programador se llama sólo cuando el marco lo precisa (por ejemplo, para crear o manipular una estructura de datos protegida por patente).

Un programador que escribe un programa marco no sólo cede el control al usuario (tal como también sucede en los programas de bucle de eventos), sino que también cede el flujo de datos detallado en el programa al marco. Este enfoque permite la creación de sistemas más complejos que funcionan conjuntamente de formas interesantes, en contraposición a programas aislados, que tienen código cliente, que se crean una y otra vez para problemas similares.

Por tanto, tal como se ha explicado anteriormente, un marco es básicamente un grupo de clases cooperantes que constituyen una solución de diseño reutilizable para un campo problemático dado. Normalmente incluye objetos que proporcionan un comportamiento por defecto (por ejemplo, para menús y ventanas), y los programadores lo usan heredando parte de ese comportamiento por defecto y redefiniendo otro comportamiento de modo que el marco llama al código de aplicación en los momentos adecuados.

Hay tres diferencias principales entre marcos y bibliotecas de clases:

- Comportamiento versus protocolo. Las bibliotecas de clases son esencialmente grupos de comportamientos que pueden llamarse cuando se quieren esos comportamientos individuales en un programa. Un marco, por otro lado, proporciona no sólo el comportamiento sino también el protocolo o conjunto de normas que gobiernan las formas en que pueden combinarse los comportamientos, incluyendo normas para lo que se supone que un programador proporciona versus lo que proporciona el marco.

- Llamar versus redefinir. Con una biblioteca de clases, el programador instancia objetos y llama a sus funciones integrantes. Es posible instanciar y llamar a objetos del mismo modo con un marco (es decir, tratar el marco como una biblioteca de clases), pero para aprovechar completamente el diseño reutilizable de un marco, un programador normalmente escribe el código que redefine y es llamado por el marco. El marco gestiona el flujo de control entre sus objetos. Escribir un programa implica dividir las responsabilidades entre las diversas piezas de software a las que se llama por el marco más que especificar cómo deberían cooperar las diferentes piezas.

- Implementación versus diseño. Con las bibliotecas de clases, los programadores reutilizan sólo implementaciones, mientras que con marcos, reutilizan diseños. Un marco plasma la manera en que funciona una familia de programas o piezas de software relacionadas. Representa una solución de diseño genérica que puede adaptarse a una variedad de problemas específicos en un campo dado. Por ejemplo, un único marco puede plasmar la forma en que funciona una interfaz de usuario, a pesar de ello dos interfaces de usuario diferentes creadas con el mismo marco pueden solucionar problemas de interfaz muy diferentes.

Por tanto, mediante el desarrollo de marcos para soluciones a varios problemas y tareas de programación, pueden lograrse reducciones significativas en el esfuerzo de diseño y desarrollo de software. Una realización preferida de la invención utiliza un lenguaje de marcación de hipertexto "HyperText Markup Language" (HTML) para implementar documentos en Internet junto con un protocolo de comunicación segura de propósito general para un medio de transporte entre el cliente y el Newco. El HTTP u otros protocolos podrían sustituirse fácilmente por HTML sin una excesiva experimentación. Hay disponible información sobre estos productos en T. Berners-Lee, D. Connoly, "RFC 1866: Hypertext Markup Language - 2.0" (noviembre de 1995); y en R. Fielding, H. Frystyk, T. Berners-Lee, J. Gettys y J.C. Mogul, "Hypertext Transfer Protocol - HTTP/1.1: HTTP Working Group Internet Draft" (2 de mayo de 1996). HTML es un formato de datos simple empleado para crear documentos de hipertexto que pueden transferirse de una plataforma a otra. Los documentos HTML son documentos SGML con semánticas genéricas que son adecuadas para representar información de una amplia gama de campos. HTML se ha usado por la iniciativa de información global World-Wide Web (red mundial) desde 1990. HTML es una aplicación de la norma ISO 8879; 1986 Tratamiento de la

información - Sistemas de texto y oficina; lenguaje de marcación generalizado estándar (SGML, “Standard Generalizad Markup Language”).

Hasta la fecha, las herramientas de desarrollo Web se han limitado en su capacidad para crear aplicaciones Web dinámicas que abarcan de cliente a servidor e interactúan con recursos informáticos existentes. Hasta hace bien poco, HTML era la tecnología dominante usada en el desarrollo de soluciones basadas en Web. Sin embargo, se ha demostrado que el HTML es inadecuado en las siguientes áreas:

- Rendimiento reducido;
- Capacidades de interfaz de usuario restringidas;
- Sólo puede producir páginas Web estáticas,
- Carece de interfuncionalidad con aplicaciones y datos existentes; e
- Incapacidad para escalarse.

El lenguaje Java de Sun Microsystem soluciona muchos de los problemas del lado del cliente al:

- mejorar el rendimiento en el lado del cliente;
- habilitar la creación de aplicaciones Web dinámicas y en tiempo real; y
- proporcionar la capacidad de crear una amplia variedad de componentes de interfaz de usuario.

Con Java, los desarrolladores pueden crear componentes de interfaz de usuario (IU) robustos. Pueden crearse “widgets” (pequeñas aplicaciones o programas) personalizados (por ejemplo, teleimpresores de cotización en tiempo real (“stock tickers”), iconos animados, etc.), y se mejora el rendimiento en el lado del cliente. A diferencia del HTML, Java soporta la noción de validación en el lado del cliente, descargando el procesamiento adecuado al cliente para un rendimiento mejorado. Pueden crearse páginas Web dinámicas en tiempo real. Utilizando los componentes UI personalizados anteriormente mencionados también pueden crearse páginas Web dinámicas.

El lenguaje Java de Sun ha emergido como un lenguaje reconocido por la industria para “programación de Internet”. Sun define Java como: “un lenguaje de programación sencillo, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portátil, de alto rendimiento, multiensartado, dinámico, de moda, de finalidad general. Java soporta la programación para Internet en forma de applets de Java independientes de la plataforma.” Los “applets” de Java son aplicaciones pequeñas y especializadas que se ajustan a la interfaz de programación de aplicaciones (API) de Java Sun que permiten a los desarrolladores añadir “contenido interactivo” a documentos Web (por ejemplo, animaciones sencillas, adornos de página, juegos básicos, etc.). Los applets se ejecutan en un navegador compatible con Java (por ejemplo, el navegador Netscape) copiando el código del servidor al cliente. Desde un punto de vista del lenguaje, el conjunto característico del núcleo de Java se basa en C++. La literatura de Java de Sun afirma que Java es básicamente “C++ con extensiones desde el objetivo C para una resolución de método más dinámica”.

Otra tecnología que proporciona una función similar a Java la proporciona Microsoft y las Technologies ActiveX, para facilitar a los desarrolladores y diseñadores Web los medios para construir contenido dinámico para Internet y los ordenadores personales. ActiveX incluye herramientas para desarrollar animación, realidad virtual en 3-D, video y otro contenido multimedia. Las herramientas usan estándar de Internet, funcionan en múltiples plataformas y las soportan unas 100 empresas. Los bloques de construcción del grupo se llaman Controles ActiveX, pequeños y rápidos componentes que permiten a los desarrolladores introducir partes de software en páginas de lenguaje de marcación de hipertexto (HTML). Los controles ActiveX funcionan con una variedad de lenguajes de programación, incluyendo Visual C++ de Microsoft, Borland Delphi, el sistema de programación Visual Basic de Microsoft y, en el futuro, la herramienta de desarrollo de Microsoft para código Java, denominada “Jakarta”. Las tecnologías ActiveX también incluyen un marco de servidor ActiveX, que permite a los desarrolladores crear aplicaciones de servidor. Un experto común en la técnica reconoce que ActiveX podría sustituirse por Java sin una excesiva experimentación para poner en práctica la invención.

Descripción detallada

Una realización de la presente invención es un marco basado en servidor que utiliza una arquitectura basada en componentes. En referencia a la figura 2A, una realización de la presente invención incluye un objeto 200 de arquitectura, un objeto 202 de aplicación, un formulario 204 de interfaz de usuario, un controlador 206 de interfaz de usuario, un adaptador 208 de componente de cliente, una interfaz 210 de componente COM y un componente 222 de servidor.

En general, los componentes de la presente invención funcionan tal como se muestra en la figura 2B. En la etapa 230, se almacenan datos en un objeto del componente. En la etapa 232, las funciones que manipulan el objeto se encapsulan con los datos del objeto. Posteriormente, en la etapa 234, los datos de objeto almacenados pueden manipularse por otros componentes utilizando las funciones de la etapa 232.

Objeto de arquitectura

El objeto 200 de arquitectura proporciona un modelo de objeto fácil de usar que oculta la complejidad de la arquitectura al cliente. El objeto 200 de arquitectura proporciona servicios puramente técnicos y no contiene ninguna lógica de negocio o código funcional. Se usa en el cliente como el único punto de acceso a todos los servicios de arquitectura.

En el lado del servidor, el objeto 200 de arquitectura se complementa por un conjunto de funciones globales contenidas en módulos VB estándar.

El objeto 200 de arquitectura es responsable de proporcionar todos los servicios de arquitectura de cliente (es decir, acceso a tabla de códigos, fichero de errores, etc.), y un único punto de entrada a los servicios de arquitectura. El objeto 200 de arquitectura también es responsable de permitir que la arquitectura exista como una unidad autónoma, permitiendo así que se realicen cambios internos de la arquitectura con un mínimo impacto en la aplicación.

El objeto 200 de arquitectura proporciona un gestor de código, un perfil de cliente, un gestor de texto, un gestor de ID, un gestor de registro, un gestor de ficheros, un gestor de errores y un gestor de seguridad. El gestor de código lee códigos de una base de datos local en el cliente, clasifica los códigos en objetos, y los hace disponibles para la aplicación. El perfil de cliente proporciona información sobre el usuario actualmente conectado. El gestor de texto proporciona varios servicios de manipulación de texto tales como buscar y reemplazar. El gestor de ID genera IDs únicas y marcas de fecha y hora. El gestor de registro encapsula el acceso al registro del sistema. El gestor de ficheros escribe mensajes de error o informativos en el fichero de mensajes. El gestor de errores proporciona una forma sencilla de salvar y volver a elevar un error. Y el gestor de seguridad determina si el usuario actual está autorizado o no a realizar ciertas acciones.

Objeto de aplicación

El objeto 202 de aplicación tiene un método para iniciar cada operación de negocio en la aplicación. Utiliza una conexión tardía para instanciar los controladores IU objetivo para proporcionar autonomía entre ventanas. Esto permite que diferentes controladores utilicen el objeto 202 de aplicación sin unirse estáticamente a todos y cada uno de los controladores IU en la aplicación.

Al abrir un controlador IU, el objeto 202 de aplicación llama a la inicialización de la arquitectura, a la inicialización de clase y forma funciones del elemento de inicialización.

El objeto 202 de aplicación mantiene una lista de cada ventana activa, de modo que puede cerrar la aplicación en caso de error. Cuando una ventana se cierra, informa al objeto 202 de aplicación, y se elimina de la lista de ventanas activas del objeto 202 de aplicación.

El objeto 202 de aplicación es responsable de instanciar cada controlador 206 IU, pasando el contexto de datos/negocio al controlador 206 IU objetivo, e invocando servicios estándar tales como el controlador de inicialización, el formulario de inicialización y la arquitectura de inicialización. El objeto 202 de aplicación también mantiene un seguimiento de qué ventanas están activas de modo que puede coordinar el proceso de cierre.

Formulario IU

La responsabilidad principal del formulario 204 IU es reenviar los eventos importantes a su controlador 206. Permanece en gran medida no inteligente y contiene la menor lógica posible. La mayoría de elementos de tratamiento de eventos en el formulario sencillamente delegan el trabajo llamando a métodos en el controlador 206 del formulario.

El formulario 204 IU nunca habilita o deshabilita sus propios controles, sino que pide al controlador 206 que lo haga en su lugar. La lógica se incluye en el formulario 204 IU sólo cuando implica la ocultación de campos muy simples o pequeños detalles visuales.

El formulario 204 IU presenta una interfaz gráfica fácil de usar por el usuario e informa a su controlador 206 sobre acciones del usuario importantes. El formulario 204 IU también puede proporcionar una validación de datos básicos (por ejemplo, validación de tipos de datos) mediante el ocultamiento de entrada. Además, el formulario IU es responsable de adaptar su tamaño de forma inteligente, lanzando una ayuda sensible al contexto y descargándose.

Controlador de Interfaz de Usuario

Cada controlador 206 IU incluye un conjunto de métodos estándar para inicializar, habilitar y deshabilitar controles en su formulario 204 IU, validar datos en el formulario, tomar datos del formulario 204 IU y descargar el formulario 204 IU.

Los controladores 206 IU contienen la mayor parte de la lógica para manipular objetos 207 de negocio y gestionar la apariencia de su formulario 204 IU. Si su formulario no es de sólo lectura, el controlador 206 IU también hace un seguimiento de si los datos en el formulario 204 IU han cambiado o no, para así evitar escrituras innecesarias de la

base de datos cuando el usuario decide guardar. Además, los controladores de ventanas auxiliares (como el cuadro de diálogo "guardar archivo" en Microsoft Word), hacen un seguimiento de su controlador 206 IU llamante de modo que pueden notificarle cuándo están listos para cerrarse.

La figura 2C es un diagrama de flujo que muestra cómo funciona el controlador IU en una realización de la presente invención. En la etapa 236, un usuario introduce datos en un formulario IU. En la etapa 238, el controlador IU interpreta los datos introducidos en el formulario IU. En la etapa 240, el controlador IU coloca los datos apropiados en un objeto de negocio para utilizarlo y recuperarlo más tarde.

Un controlador 206 IU define una unidad lógica de trabajo (LUW, "Logical Unit of Work"). Si una LUW abarca más de un controlador 206 IU, la LUW se implementa como un objeto separado.

El controlador 206 IU es responsable de tratar eventos generados por el usuario que interactúa con el formulario 204 IU y proporcionar una validación de campos complejos y una validación de campos cruzados en una unidad lógica de trabajo. El controlador 206 IU también contiene la lógica para interactuar con objetos 207 de negocio, y crea nuevos objetos 207 de negocio cuando es necesario. Finalmente, el controlador 206 IU interactúa con adaptadores 208 de componentes de cliente para añadir, recuperar, modificar o borrar objetos 207 de negocio, y trata todos los errores del lado del cliente.

Objetos de negocio

La funcionalidad principal de los objetos 207 de negocio (ON) es actuar como un soporte de datos, permitiendo compartir datos a través de los controladores 206 de interfaz de usuario empleando un modelo de programación basado en objetos.

Los ON 207 realizan la validación de sus atributos cuando se ajustan, para mantener la integridad de la información que contienen. Los ON 207 también exponen métodos aparte de los descriptores de accesos para manipular sus datos, tales como métodos para cambiar el estado del ciclo de vida de un ON 207 o para derivar el valor de un atributo calculado.

En muchos casos, un ON 207 tendrá su propia tabla en la base de datos y su propia ventana para ver o editar las operaciones.

Los objetos 207 de negocio contienen información sobre una única entidad de negocio y mantienen la integridad de esa información. El ON 207 encapsula normas de negocio que pertenecen a esa única entidad de negocio y mantiene relaciones con otros objetos de negocio (por ejemplo, una reclamación contiene una serie de suplementos). Finalmente, el ON 207 proporciona propiedades adicionales relativas al estado de la información que contiene (tal como si la información ha cambiado o no), proporciona la validación de nuevos datos cuando sea necesario, y calcula atributos que se derivan de otros atributos (tal como el nombre completo, que se deriva del nombre, el segundo nombre y el apellido).

Adaptadores de Componentes de Cliente

Los adaptadores 208 de componentes de cliente (ACC) son responsables de recuperar, añadir, actualizar y borrar objetos de negocio en la base de datos. Los ACC 208 ocultan el formato de almacenamiento y la localización de los datos al controlador 206 IU. Al controlador 206 IU no le importa dónde o cómo se almacenan objetos, ya que esto lo controla el ACC 208.

El ACC 208 clasifica los datos contenidos en conjuntos de registros devueltos por el servidor en objetos 207 de negocio. Los ACC 208 enmascaran todas las peticiones remotas desde el controlador 206 IU a un componente específico, y actúan como un "gancho" para servicios tales como compresión de datos y encriptación de datos.

Interfaz de Componente COM

Una interfaz 210 de componente COM (ICC) es un "contrato" para los servicios proporcionados por un componente. Al "implementar" una interfaz 210 (ICC), un componente promete proporcionar todos los servicios definidos por la ICC 210.

La ICC 210 no es una entidad física (por lo que se ilustra con una línea de puntos). Su única razón para existir es definir la forma en que aparece un componente ante los otros objetos. Incluye las firmas o cabeceras de todas las propiedades o métodos públicos que proporcionará un componente.

Para implementar una ICC 210, un componente de servidor expone un conjunto de métodos denominados de manera especial, uno para cada método definido en la interfaz. Estos métodos no deberían hacer nada salvo delegar la petición a un método privado en el componente que hará el trabajo real.

La ICC 210 define un conjunto de servicios relacionados proporcionados por un componente. La ICC permite que cualquier componente "se oculte" tras la interfaz para realizar los servicios definidos por la interfaz "implementando" la interfaz.

Componente de Servidor

Los componentes 222 de servidor son de grano grueso y orientados a las transacciones. Están diseñados para una máxima eficacia.

Los componentes 222 de servidor encapsulan todos los accesos a la base de datos, y definen límites de transacción de negocios. Además, los componentes 222 de servidor son responsables de garantizar que las normas de negocio se cumplen durante las operaciones de acceso a datos.

Un componente 222 de servidor realiza operaciones de acceso a datos en nombre de los ACC 208 o de otros componentes y participantes en transacciones que abarcan componentes 222 de servidor comunicándose con otros componentes 222 de servidor. El componente 222 de servidor es accesible por múltiples personalidades de extremo frontal (por ejemplo, páginas activas del servidor), y contiene lógica de negocio diseñada para mantener la integridad de los datos en la base de datos.

La figura 2D es un diagrama de flujo que muestra las interacciones entre el ACC, la ICC y el componente de servidor según una realización de la presente invención. En la etapa 242, se hace una petición para colocar los datos creados por el cliente en la base de datos del servidor. En la etapa 244, los datos se transfieren al componente 222 de servidor utilizando una ICC 210. En la etapa 246, el componente 222 de servidor almacena los datos en la base de datos del servidor.

Disposición de normas de negocio

Visión general

La distribución de las normas de negocio a través de terceros de la aplicación afecta directamente a la robustez y al rendimiento del sistema en conjunto. Las normas de negocio pueden categorizarse en las siguientes secciones: relaciones, cálculos, y eventos de negocio.

Relaciones entre objetos de negocio

Los objetos 207 de negocio son responsables de conocer otros objetos 207 de negocio con los que están asociados.

Las relaciones entre ON 207 se construyen por el ACC 208 durante el proceso de clasificación. Por ejemplo, cuando un ACC 208 construye un ON 207 de reclamación, también construirá la serie de suplementos en caso necesario.

Datos de negocio calculados

Las normas de negocio que implican cálculos basados en los atributos del objeto 207 de negocio se codifican en los propios objetos 207 de negocio. El nombre completo de un participante es un buen ejemplo de un atributo calculado. En lugar de forzar a los controladores para concatenar el nombre, el segundo nombre y el apellido cada vez que se quiere mostrar el nombre completo, se expone un atributo calculado que lleva a cabo esta lógica en el objeto de negocio. De este modo, el código para componer el nombre completo sólo tiene que escribirse una vez y puede usarse por muchos controladores 206.

Otro ejemplo de un atributo calculado es la fecha de visualización de una tarea repetida. Cuando se completa una tarea con una norma de repetición, debe determinarse una nueva fecha de visualización. Esta fecha de visualización se calcula basándose en la fecha en que se completó la tarea y en la frecuencia de repetición definida por la norma de repetición. Al hacer que la lógica calcule la nueva fecha de visualización en el ON 207 de tarea garantiza que sólo se codifica una vez.

Respuestas a eventos de negocio

Las normas de negocio que se refieren a eventos de sistema y no implican la interacción de un usuario se cumplen en los componentes de servidor.

La finalización de una tarea es un evento principal en el sistema. Cuando se completa una tarea, el sistema primero garantiza que el realizador que completa la tarea se añada a la reclamación. Entonces, una vez marcada la tarea como completada en la base de datos, se comprueba para ver si la tarea tiene una norma de repetición. Si es así, se crea otra tarea y se añade a la base de datos. Finalmente, se notifica al componente de evento, porque el motor de tareas puede tener que reaccionar ante la finalización de la tarea.

Considérese el escenario en el que la lógica para cumplir esta norma estuviese situada en el controlador 206 IU.

El controlador 206 llama al componente de realizador para ver si el realizador que completa la tarea se ha añadido a la reclamación. Si el realizador no se ha añadido a la reclamación, el controlador 206 vuelve a llamar al componente de realizador para añadirlo.

A continuación, el controlador 206 llama al componente de tarea para marcar la tarea como completada en la base de datos. Si la tarea tiene una norma de repetición, el controlador 206 calcula la fecha en que la tarea debe volver a visualizarse y vuelve a llamar al componente de tarea para añadir una nueva tarea. Por último, el controlador 206 llama al componente de evento para notificar al motor de tareas de la finalización de la tarea.

La implementación anterior requiere cinco recorridos de ida y vuelta en la red en el peor caso. Además, cualquier otro controlador 206 o componente 222 de servidor que quiera completar una tarea debe codificar esta lógica una y otra vez. Cumplir esta norma en el componente 222 del servidor de tareas reduce el número de recorridos de ida y vuelta en la red y elimina la necesidad de codificar la lógica más de una vez.

Respuestas a los eventos de usuario

Todas las respuestas a eventos de usuario se coordinan por el controlador 206. El controlador 206 es responsable de acciones tales como habilitar o deshabilitar controles en su formulario, pedir la autorización del componente de seguridad, o hacer llamadas al ACC 208.

Autorización

Toda la lógica para conceder autorizaciones se encapsula dentro del componente de seguridad. Los controladores 206 y los componentes 222 deben preguntar al componente de seguridad si el usuario actual está autorizado para ejecutar ciertas operaciones de negocio en el sistema. El componente de seguridad responderá sí o no, según una lógica de seguridad predefinida:

Resumen

Tipo de norma de negocio	Ejemplo	Responsabilidad
Mantener relaciones entre ON	La reclamación incluye una serie de suplementos	Objetos de negocio
Construir relaciones entre ON	El ACC construye la serie de suplementos de la reclamación	ACC
Datos de negocio calculados	El participante calcula su nombre completo	Objetos de negocio
Respuestas a eventos de negocio	El componente de tarea colabora con otros componentes	Componentes
Pedir autorización	La biblioteca de tareas pregunta al componente de seguridad si el usuario actual está autorizado a acceder a la biblioteca de tareas	Controladores y componentes
Conceder autorización	El componente de seguridad determina si el usuario actual puede o no acceder a la biblioteca de tareas	Componente de seguridad

Marco de procesamiento de ventanas

El marco de ventanas por defecto proporciona un procesamiento de ventanas por defecto para cada ventana contenida en el sistema. Este procesamiento por defecto ayuda al desarrollador a desarrollar IU robustas y que pueden mantenerse, estandariza procesos comunes (tales como la inicialización de formularios) y facilita una integración suave con los servicios de arquitectura.

La figura 3 muestra el ciclo de vida de una interfaz de usuario típica y los métodos estándar que forman parte del marco 300 de tratamiento de ventanas.

El marco 300 de procesamiento de ventanas comprende lo siguiente:

Inicialización 302 de ventana;

Procesamiento 304 de guardar ventana;

Gestión 306 del estado de los controles de ventana;

Validación 308 de datos de ventana;

Procesamiento 310 del cierre de ventana.

Procesamiento 302 de inicialización de ventana: después de crear un controlador 206 para la ventana deseada, el objeto 202 de aplicación llama a un conjunto de funciones de inicialización estándar en el controlador 206 antes de que se muestre el formulario 204 al usuario. La estandarización de estas funciones hace las IU más homogéneas en toda la aplicación, al tiempo que favorecen una buena descomposición funcional.

Procesamiento 304 de guardar ventana: siempre que un usuario actualice cualquier texto del formulario o añada un elemento a un cuadro de lista, el controlador 206 IU marca el formulario como "sucio". Esto permite que el controlador 206 IU determine si los datos han cambiado cuando se cierra el formulario y obliga al usuario a confirmar o perder sus cambios.

Gestión 306 del estado de los controles de ventana: el habilitar o deshabilitar controles y opciones de menú es una parte muy compleja de la construcción de una IU. La lógica que modifica el estado de los controles está encapsulada en un único lugar para su mantenibilidad.

Validación 308 de datos de ventana: siempre que se cambian datos en un formulario, pueden romperse las normas de validación. El controlador es capaz de detectar estos cambios, validar los datos, y obligar al usuario a corregir entradas incorrectas.

Procesamiento 310 de cierre de ventana: el marco de cierre de ventana proporciona una ruta de finalización clara para cada IU en caso de error. Esto reduce la posibilidad de fugas de memoria, y fallos de protección generales.

Beneficios

Procesamiento estandarizado: la estandarización del procesamiento de ventanas aumenta la homogeneidad de la aplicación. Esto garantiza que todas las ventanas en la aplicación se comporten de una manera consistente para los usuarios finales, haciendo que la aplicación sea más fácil de usar. También acorta la curva de aprendizaje de los desarrolladores y aumenta la mantenibilidad, puesto que todas las ventanas se codifican de una manera consistente.

Desarrollo simplificado: los desarrolladores pueden aprovechar las mejores prácticas documentadas en el marco de procesamiento de ventanas para hacer más efectivas las decisiones de diseño y codificación. Además, una estructura proporciona un código "enlatado" que ofrece a los desarrolladores una inicialización durante el esfuerzo de codificación.

Arquitectura por capas: dado que varios módulos de arquitectura proporcionan un procesamiento estandarizado a cada ventana de aplicación, la lógica central puede cambiarse para cada ventana de sistema realizando simplemente modificaciones a un proceso individual.

Inicialización 302 de ventana

Para abrir una nueva ventana, el objeto 202 de aplicación crea el controlador 206 de ventana objetivo y llama a una serie de métodos en el controlador 206 para iniciarlo. La llamada de estos métodos, ArchlnitClass (clase de inicialización de arquitectura), InitClass (clase de inicialización), InitForm (formulario de inicio) y ShowForm (formulario de visualización), se ilustra más adelante.

ArchlnitClass

La finalidad principal de la función ArchlnitClass es informar al controlador 206 objetivo de quién lo está llamando. El objeto 202 de aplicación "realiza las instrucciones" pasando al controlador 206 objetivo una referencia de sí mismo y una referencia al controlador 206 objetivo. Además, sirve como gancho en el controlador 206 para añadir funcionalidad de arquitectura en el futuro.

```
Public Sub ArchInitClass(objApp As Object, objCallingCTLR As Object)
    ' remember who called me
    Set m_objApp = objApp
    Set m_objCallingCTLR = objCallingCTLR
End Sub
```

InitClass

Esta función proporciona una nueva forma para que el objeto 202 de aplicación facilite al controlador 206 objetivo cualquier dato que necesite para realizar su procesamiento. En este punto es cuando el controlador 206 objetivo puede determinar en qué "modo" está. Los modos de formulario típicos incluyen, modo adición, modo edición, y modo visualización. Si la ventana está en el modo adición, crea un nuevo ON 207 del tipo adecuado en este método.

```
Public Sub InitClass(colPrevSelection As CArchCollection)
    If colPrevSelection Is Nothing Then
        ' no accounts were previously selected
        Set m_colPrevSelection = New CArchCollection
        Set m_colNewSelection = New CArchCollection
    Else
        ' some accounts may have already been selected
        Set m_colPrevSelection = colPrevSelection
        Set m_colNewSelection = colPrevSelection.Clone()
    End If

    Set m_colResults = New CArchCollection

    DetermineFormMode()
End Sub
```

InitForm

El procedimiento InitForm de cada controlador 206 coordina cualquier inicialización del formulario 204 antes de que se muestre. Dado que la inicialización es a menudo un proceso de múltiples etapas, InitForm crea la ventana y entonces delega la mayor parte de la lógica de inicialización a métodos auxiliares que tienen, cada uno, una única finalidad, con el fin de seguir las normas de buena descomposición funcional. Por ejemplo, la lógica para determinar un estado de formulario 204 basándose en las acciones del usuario y las restricciones de seguridad relevantes y pasar a ese estado se encapsula en el método DetermineFormState (determinar estado de formulario).

```
Public Sub InitForm()
    ' create my form
    Set m_frmCurrentForm = New frmAccountSearch


    ' figure out the state of my form based on arguments I received in
InitClass and
    ' enable/disable the appropriate controls
    DetermineFormState()
    ' fill my form with data
    PopulateForm()
End Sub
```

PopulateForm (formulario de ocupación)

PopulateForm es un método privado responsable de rellenar el formulario con datos durante la inicialización. Se llama exactamente una vez por el método InitForm. PopulateForm se usa para rellenar conjuntos de cuadros en un formulario 204, tomar los detalles de un objeto para una ventana de edición o mostrar objetos que ya se han seleccionado por el usuario, como en el siguiente ejemplo:

```
Private Sub PopulateForm()
    Dim acct As CAccount
    Dim item As GTListItem

    ' display any accounts already selected by the user
    ' create and add a ListItem for every Account in the previous
selection collection
    With frmCurrentForm.lvwResults.ListItems
        .Clear
        For Each acct In m_colPrevSelection
            Set item = .Add(, acct.Number, acct.Number)
            item.SubItems(1) = acct.Name
        Next
    End With
End Sub
```

Showform (formulario de visualización)

El método ShowForm simplemente centra y muestra el formulario 204 que acaba de inicializarse.

```
Public Sub ShowForm()
    ' center my form
    frmCurrentForm.Move(Screen.Width - frmCurrentForm.Width) / 2,_
                        (Screen.Height - frmCurrentForm.Height) / 2
    ' display my form
    frmCurrentForm.Show vbModal
End Sub
```

Gestión 306 del estado de los controles de ventana

A menudo es necesario habilitar o deshabilitar controles en un formulario 204 en respuesta a las acciones del usuario. Esta sección describe los patrones empleados por la arquitectura basada en componentes para MTS (CBAM) para gestionar este proceso de manera efectiva.

Modo de formulario

Resulta útil distinguir entre modo de formulario y estado de formulario. El modo de formulario indica la razón por la se ha invocado el formulario 204. A menudo, los formularios 204 se usan para más de una finalidad. Un ejemplo común es el uso del mismo formulario para ver, añadir y editar un tipo de objeto particular, tal como una tarea o reclamación. En este caso, los modos de formulario incluirían Ver, Añadir y Actualizar.

Los modos de un formulario 204 también se usan para cumplir con las restricciones de seguridad basadas en el nivel de acceso actual del usuario. Por ejemplo, la biblioteca de tareas es una ventana que limita el acceso a plantillas de tarea basándose en el papel actual del usuario. Podría tener un modo Bibliotecario y un modo No Bibliotecario para reflejar el hecho de que no puede permitirse a un usuario no bibliotecario la edición de plantillas de tarea. De esta forma, los modos ayudan a cumplir el requisito de que ciertos controles en el formulario 204 permanecen deshabilitados a menos que el usuario tenga un cierto nivel de acceso.

No siempre es necesario que un formulario 204 tenga un modo; un formulario puede ser tan sencillo que sólo tenga un modo, el modo por defecto. En este caso, aunque no sea inmediatamente necesario, puede ser beneficioso hacer que el formulario sea “consciente del modo” de manera que pueda extenderse fácilmente en caso necesario.

Estado del formulario

Un formulario 204 tendrá un número de diferentes estados para cada modo, siendo un estado una combinación única de controles habilitados/deshabilitados, visibles/invisibles. Cuando un formulario 204 pasa a un estado diferente, al menos un control se habilita o deshabilita o se modifica de alguna manera.

Una diferencia clave entre modo de formulario y estado de formulario es que el modo se determina cuando se inicia el controlador 206 y permanece constante hasta que se finaliza el controlador 206. El estado se determina cuando se inicia la ventana, pero se reevalúa constantemente en respuesta a las acciones del usuario.

Tratamiento de eventos IU

Cuando cambia el valor de un control en el formulario 204, es necesario volver a evaluar el estado de los controles en el formulario (si están habilitados o deshabilitados, visibles o invisibles, etc.). Si el cambio de valor de un control pudiera provocar el cambio de estado de un segundo control, se escribe un elemento de tratamiento para el evento apropiado del primer control.

La siguiente tabla enumera los controles comunes y los eventos que se desencadenan cuando cambia su valor.

Control	Evento
Cuadro de texto (TextBox)	Cambio
Conjunto de cuadros (ComboBox)	Cambio
Cuadro de lista (ListBox)	Clic
Cuadro de comprobación (CheckBox)	Clic
Botón de opción (Option Button)	Clic

El elemento de tratamiento de eventos llama al método DetermineFormState en el controlador 206.

Ajuste del estado de los controles

Es esencial para la mantenibilidad que el proceso de ajuste del estado de los controles esté separado del proceso de ajuste de los valores de estos controles. El método DetermineFormState en el controlador 206 fuerza esta separación entre el ajuste del estado de los controles y el ajuste de sus valores.

DetermineFormState es el único método que modifica el estado de cualquiera de los controles del formulario 204. Debido a que los requisitos de estado del control son tan complejos y varían en tan gran medida, esta es la única restricción que impone el marco de arquitectura.

Si es necesario, los parámetros se pasan a la función DetermineFormState para actuar como “indicios” o “claves” para determinar el nuevo estado del formulario 204. Para formularios complejos, resulta útil descomponer la función DetermineFormState en varias funciones auxiliares, que tratan cada una un grupo de controles relacionados en el formulario o mueven el formulario 204 a un estado diferente.

Ejemplo

La ventana de tarea editar/añadir/ver tiene tres modos: Editar, Añadir y Ver. En el modo Añadir, todo el formulario puede editarse. Algunos detalles permanecerán deshabilitados en el modo Editar ya que sólo deberían ajustarse una vez añadida la tarea. Tanto en el modo Añadir como en el modo Editar, la norma de repetición puede editarse. El hecho de habilitar la edición de la norma de repetición siempre deshabilita la edición manual de los datos obligatorios y de visualización de la tarea. En el modo Ver, sólo están habilitados el conjunto de cuadros de categoría y el cuadro de comprobación privado.

(Programa pasa a página siguiente)

```
' Edit/Add/View Task Form
Private Sub txtName_Change()
    myController.DetermineFormState
End Sub


' Edit/Add/View Task Controller
Public Sub DetermineFormState()
    On Error Goto ErrorHandler
    Select Case m_nFormMode
        ' In Edit Mode, enable only "editable" details and Repeat Rule
editing if necessary
        Case cmFormModeEdit
            EnableAddDetails False
            EnableEditDetails True
            EnableViewDetails True
            If m_frmCurrentForm.chkRepetetiveTask.Checked Then
                EnableEditRepeatRule True
                EnableEditDisplayDueDates False
            Else
                EnableEditRepeatRule False
                EnableEditDisplayDueDates True
            End If
            If m_nFormDirty Then EnableSave True Else EnableSave False
        ' In Add Mode, enable all details and Repeat Rule editing if
necessary
        Case cmFormModeAdd
            EnableAddDetails True
            EnableEditDetails True
            EnableViewDetails True
            If m_frmCurrentForm.chkRepetetiveTask.Checked Then
                EnableEditRepeatRule True
                EnableEditDisplayDueDates False
            Else
                EnableEditRepeatRule False
                EnableEditDisplayDueDates True
            End If
            If m_nFormDirty Then EnableSave True Else EnableSave False
        ' In View Mode, disable everything except a few details
        Case cmFormModeView
            EnableAddDetails False
            EnableEditDetails False
            EnableViewDetails True
            EnableEditRepeatRule False
            EnableEditDisplayDueDates False
            EnableSave False
        Case Else
    End Select
    Exit Sub
ErrorHandler:
    ' error handling
End Sub


' Edit/Add/View Task Controller
Private Sub EnableAddDetails(bYesNo As Boolean)
    On Error Goto ErrorHandler

    ' Enable or disable controls that should be available only when the
task is being added.
    With frmCurrentForm
        .Name.Enabled = bYesNo
        .Description.Enabled = bYesNo
        .Type.Enabled = bYesNo
        .Level.Enabled = bYesNo
        .Source.Enabled = bYesNo
    End With
    Exit Sub
ErrorHandler:
    ' error handling logic
End Sub
```

Validación 308 de datos de la ventana

La validación de los datos de la ventana es el proceso por el que los datos en la ventana se examinan en busca de errores, inconsistencias, y formateado adecuado. Es importante para la consistencia que este proceso se implemente de forma similar o idéntica en todas las ventanas de la aplicación.

Tipos de validación

Enmascaramiento de entrada

El enmascaramiento de entrada es la primera línea de defensa. Implica el filtrado de los datos (normalmente carácter por carácter) cuando se introducen, para evitar siquiera que el usuario introduzca datos incorrectos. El enmascaramiento de entrada puede hacerse de manera programática o a través de un cuadro de texto enmascarado especial, aunque la lógica siempre se sitúa en el formulario, y se invoca siempre que cambia un campo de enmascaramiento.

Comprobación del intervalo del campo individual

La comprobación en el intervalo del campo individual determina la validez del valor de un campo en el formulario comparándolo con un conjunto de valores válidos. La comprobación en el intervalo del campo individual puede hacerse a través de un conjunto de cuadros, un selector cíclico, o de manera programática en el formulario, y se invoca siempre que cambia el intervalo de campo comprobado.

Validación de campos cruzados

La validación de campos cruzados compara los valores de dos o más campos para determinar si una norma de validación se cumple o se infringe, y se produce justo antes de guardar (o buscar). La validación de campos cruzados puede hacerse en el controlador 206 o en el objeto 207 de negocio, sin embargo, es preferible disponer la lógica en el objeto 207 de negocio cuando la lógica de validación puede compartirse por múltiples controladores 206.

Los datos incorrectos se atrapan y se eliminan lo antes posible durante el proceso de entrada. El enmascaramiento de entrada y la comprobación del intervalo proporcionan la primera línea de defensa, seguida de la validación de campos cruzados cuando la ventana guarda (o busca).

Validación de campo individual

La validación de todos los campos individuales se realiza a través de algún tipo de enmascaramiento de entrada. Las máscaras que se adjuntan a los cuadros de texto se usan para validar el tipo o el formato de los datos que se introducen. Los conjuntos de cuadros y los selectores cíclicos también pueden usarse para limitar al usuario para validar opciones. Si nada de esto es suficiente, una pequeña cantidad de la lógica puede disponerse en el elemento de tratamiento de eventos del formulario para realizar la funcionalidad de enmascaramiento, tal como mantener un valor por debajo de un cierto umbral o excluir apóstrofes de un cuadro de texto.

Validación de campos cruzados

Cuando un usuario clica OK o Guardar, el formulario llama a IsFormDataValid (son válidos los datos del formulario) en el controlador para realizar la validación de campos cruzados (por ejemplo, verificando que una fecha inicial es inferior a una fecha final). Si el objeto 207 de negocio contiene normas de validación, el controlador 206 puede llamar a un método en el objeto 207 de negocio para asegurar que no se infringen esas normas.

Si el controlador 206 detecta datos incorrectos, notificará al usuario con un cuadro de mensaje y, si es posible, para indicar qué campo o campos tienen un error. En ninguna circunstancia la ventana realizará la validación cuando el usuario está intentando cancelar.

Ejemplo

```
' Generic Edit Form
Private Sub cmdOK_Click()
    On Error Goto ErrorHandler
    ' shut down if my data is valid.
    ' saving/canceling will occur in my controller's QueryUnload function
    If IsFormDataValid Then Unload Me
    Exit Sub
ErrorHandler:
    Err.Raise Err.Number
End Sub

Public Function IsFormDataValid() As Boolean
    On Error Goto ErrorHandler

    ' assume success
    IsFormDataValid = True

    ' evaluate all validation rules
    With frmCurrentForm
                                    '
                ' make sure start date is earlier than end date
                If .txtStartDate.Text > .txtEndDate.Text Then
                    IsFormDataValid = False
                    MsgBox cmMsgInvalidEndDate
                    .txtEndDate.SetFocus
                ElseIf ...
                    ' more validation rules
                End If
            End With

            Exit Function
        ErrorHandler:
            ' error handling logic
        End Function
```

Procesamiento 304 de guardar ventana

El "procesamiento de guardar" ventana implica hacer un seguimiento de los cambios de los datos en un formulario 204 y responder guardando y cancelando eventos iniciados por el usuario.

Seguimiento de los cambios en los datos del formulario

Cada ventana en la aplicación CBAM contiene un campo dentro de su objeto de control correspondiente conocido como indicador temporal ("dirty flag"). El indicador temporal se ajusta a Verdadero siempre que un usuario final modifique datos en la ventana. El controlador 206 IU interroga este campo para determinar cuándo debería obligarse a un usuario a Cancelar o si debería invocarse un procedimiento remoto al cerrar la ventana.

La cubierta de la aplicación proporciona un procesamiento estándar para cada ventana que contiene un botón de OK o Guardar.

Guardar

El procesamiento de Guardar por defecto está implementado en el controlador 206 IU como sigue:

El controlador IU recibe Notificación de que se ha hecho clic sobre el botón OK. Entonces el controlador 206 comprueba su indicador temporal. Si el indicador marca temporal, el controlador 206 llama al método InterrogateForm (interrogar formulario) para recuperar los datos del formulario 204 y llama a un componente 222 de servidor para almacenar el objeto 207 de negocio en la base de datos. Si el indicador temporal no está ajustado, no es necesario guardar. Entonces se cierra la ventana.

Cancelar

Cuando el usuario cancela una ventana, el controlador 206 IU examina inmediatamente el indicador temporal. Si el indicador se ajusta como Verdadero, se da lugar a que el usuario pierda sus cambios si decide cerrar la ventana.

Cuando se le obliga, el usuario puede elegir continuar cerrando la ventana y perdiendo sus cambios o decidir no cerrar y continuar trabajando.

Procesamiento 310 de cierre de ventana

En caso de error, a veces es necesario cerrar una ventana o terminar la aplicación completa. Es crítico que todas las ventanas sigan el proceso de cierre para evitar los GPF asociados normalmente a un acabado incorrecto. A continuación se indica cómo se cierra la ventana/aplicación.

Alcance del cierre

El alcance del cierre es lo más pequeño posible. Si se produce un error en un controlador 206 que no afecta al resto de la aplicación, sólo se cerrará esa ventana. Si se produce un error que amenaza a toda la aplicación, hay una forma de cerrar rápidamente todas las ventanas abiertas en la aplicación. La estrategia de cierre de ventanas es capaz de albergar ambos tipos de cierre.

Cierre

Para saber qué ventanas deben cerrarse, la arquitectura hace un seguimiento de qué ventanas están abiertas. Siempre que un objeto 202 de aplicación crea un controlador 206, llama a su función RegCTLR (registrar control) para añadir el controlador 206 a una serie de controladores abiertos. Igualmente, siempre que se cierra una ventana, informa al objeto 202 de aplicación de que se está cerrando llamando a la función UnRegCTLR (deshacer registro del control) del objeto 202 de aplicación, y el objeto 202 de aplicación eliminará el controlador 206 cerrado de su serie. En caso de error, el objeto 202 de aplicación pasa a través de su serie de controladores abiertos, diciendo a cada controlador que “se desactive” o se cierre inmediatamente.

Elemento de tratamiento de errores general

El GeneralErrorHandler (elemento de tratamiento de errores general) es un método en MArch.bas que actúa como el punto de entrada en el mecanismo de tratamiento de errores de la arquitectura. Un componente o un controlador llamará al GeneralErrorHandler cuando encuentren cualquier tipo de error desconocido o inesperado. El elemento de tratamiento de errores general devolverá un valor indicando qué debería hacer el componente o el controlador: (1) reanudarse en la línea que desencadenó el error (2) reanudarse en la instrucción después de la línea que desencadenó el error (3) salir de la función (4) desactivarse (5) cerrar toda la aplicación.

```
ErrorHandler:
    Select Case CStr(Err.Number)
        ' handle a search with no result error
        Case cmErrNoClaimTreeData
            MsgBox cmMsgNoResultsQuery, vbInformation
            frmCurrentForm.StatusBar.Panels(1) = cmNoResultsQuery
            'Sets mouse pointer back to default
            frmCurrentForm.MousePointer = vbDefault
        Case Else
            Dim nResumeCode As Integer
            nResumeCode = GeneralErrorHandler(objApp.objArch.AsMsgStruct,
cmController, _
                                              cmClassName, cmMethodName)
            Select Case CStr(nResumeCode)
                Case cmErrorResume
                    Resume
                Case cmErrorResumeNext
                    Resume Next
                Case cmErrorExit
                    Exit Sub
                Case cmErrorQuiesce
                    Quiesce
                Case Else
                    objApp.Shutdown
            End Select
    End Select
End Sub
```

Para evitar llamada recursivas, el GeneralErrorHandler mantiene una serie de controladores que están en proceso de cerrarse. Si se llama dos veces en una fila por el mismo controlador 206, es capaz de detectar y cortocircuitar el bucle. Cuando el controlador 206 finalmente acaba, llama a la función UnRegisterError (deshacer el registro de error) para que el GeneralErrorHandler sepa que se ha cerrado y eliminado de la serie de controladores.

Proceso de cierre

Después de que el GeneralErrorHandler le diga qué hacer, el controlador 206 con un error puede intentar ejecutar la instrucción que provocó el error, proceder como si nada hubiese pasado, salir de la función actual, llamar a su función de Desactivación para cerrarse, o llamar al método de Cierre en el objeto 202 de aplicación parar cerrar toda la aplicación.

Métodos estándar adicionales

Buscar

Los controladores 206 que gestionan las ventanas de búsqueda tienen un método público denominado Find<Noun>s, en el que <Noun> es el tipo de objeto que se busca. A este método se le llama en el elemento de tratamiento de evento errores por el botón Find Now (buscar ahora).

Guardar

Cualquier controlador 206 que gestione una ventana de edición tiene un método público denominado Save (Guardar) que guarda los cambios que hace el usuario en los datos del formulario 204. A este método se le llama por los elementos de tratamiento de evento por ambos botones Guardar y OK (cuando/si el botón OK necesita guardar cambios antes del cierre).

Cerrar

Una ventana VB se cierra por el usuario de varias formas: a través del cuadro de control en la esquina izquierda superior, el botón X en la esquina derecha superior, o el botón Cerrar. Cuando se cierra el formulario, el único método al que siempre se llamará, independientemente de la forma en que se inició el cierre, es el elemento de tratamiento de evento QueryUnload (descargar pregunta) del formulario 204.

Debido a esto, no puede haber un método de cierre estándar. Cualquier procesamiento que deba producirse cuando se cierra una ventana debe hacerse en el método QueryUnload en el controlador 206 (al que se llama por el elemento de tratamiento de evento QueryUnload del formulario).

La instrucción VB, Unload Me (descárgame), aparece en el elemento de tratamiento de evento del botón de cierre para iniciar manualmente el proceso de descarga. De esta forma, el botón de cierre imita la funcionalidad del cuadro de control y del botón X, de manera que el proceso de cierre se trata del mismo modo cada vez, independientemente de cómo se ha desencadenado el cierre por el usuario. El elemento de tratamiento de evento del botón OK también ejecuta la instrucción Unload Me, pero primero llama al método Guardar en el controlador para guardar cualquier cambio pendiente.

Objetos de negocio

Los objetos 207 de negocio son responsables de contener datos, mantener la integridad de esos datos, y exponer funciones que hagan los datos fáciles de manipular. Siempre que una lógica pertenezca a un único NO 207, es candidata a disponerse en ese NO. Esto garantiza que no se codificará una vez para cada controlador 206 que lo necesite. A continuación se dan algunos ejemplos estándar de lógica de objeto de negocio.

Lógica de negocio: Gestión del estado del ciclo de vida

Visión general

El "estado" de un objeto 207 de negocio es el conjunto de todos sus atributos. El estado del ciclo de vida se refiere únicamente a un único atributo (o a un pequeño grupo de atributos) que determinan en qué punto de su ciclo de vida se encuentra el NO 207. Por ejemplo, los estados del ciclo de vida de una tarea son Abierta, Completada, Eliminada, o Error. Los objetos de negocio normalmente implican pasar un NO hacia su estado final (es decir, Completado para una tarea, Cerrado para un suplemento, etc.).

A menudo, existen restricciones para el paso de un ON a través de su ciclo de vida. Por ejemplo, una tarea sólo puede pasar al estado Error después de haberse Completado o Eliminado primero. Los NO proporcionan un mecanismo para garantizar que no infringen las restricciones del ciclo de vida cuando pasan de un estado a otro.

Enfoque

Un NO 207 tiene un método para pasar a cada uno de sus diferentes estados del ciclo de vida. En lugar de simplemente exponer una variable pública que contenga el estado del ciclo de vida de la tarea, el NO expone métodos, tales como Task.Clear(), Task.Complete(), y Task.MarkInError(), que hacen pasar la tarea a un nuevo estado. Este enfoque evita que la tarea contenga un valor incorrecto para el estado del ciclo de vida, y pone en evidencia qué son los estados del ciclo de vida de una tarea.

Ejemplo

```
' CTask Business Object
Public Sub MarkInError()
    On Error Goto ErrorHandler

    Select Case m_nLifeCycleState
        ' move to error only if I've already been completed or cleared
        Case cmTaskCompleted, cmTaskCleared
            m_nLifeCycleState = cmTaskInError
        ' otherwise, raise an error
        Case Else
            Err.Raise cmErrInvalidLifeCycleState
    End Select
    Exit Sub
ErrorHandler:
    Err.Raise Err.Number
End Sub
```

Lógica de negocio: funcionamiento en grupos de objetos de negocio

Visión general

En ocasiones, un NO 207 actúa como un recipiente para un grupo de otros NO. Esto ocurre cuando se llevan a cabo operaciones que implican múltiples NO. Por ejemplo, para cerrar, una reclamación asegura que no hay abiertos suplementos o tareas. Puede haber un método en el NO de reclamación -CanClose()- que evalúe las normas de negocio que restringen el cierre de una reclamación y las ponga en verdadero o falso. Otra situación podría implicar recuperar las tareas abiertas para una reclamación. La reclamación puede pasar a través de su serie de tareas, preguntando a cada tarea si está abierta y, si lo está, añadirla a una serie temporal que se devuelve al elemento llamante.

Ejemplo

```
' Claim Business Object
' Error handling omitted for clarity
Public Function CanClose() As Boolean
    CanClose = HasOpenTasks() And HasOpenSupplements()
End Function

Public Function HasOpenTasks() As Boolean
    ' assume that I have open tasks
    HasOpenTasks = True
    ' loop through all my tasks and exit if I find one that is open
    Dim task As CTask
    For Each task In m_colTasks
        If task.IsOpen() Then Exit Function
    Next task
    ' I must not have any open tasks
    HasOpenTasks = False
End Function


Public Function HasOpenSupplements() As Boolean
    ' assume that I have open supplements
    HasOpenSupplements = True
    ' loop through all my supplements and exit if I find one that is open
    Dim supp As CSupplement
    For Each supp In m_colSupplements
        If supp.IsOpen() Then Exit Function
    Next supp
    HasOpenSupplements = False
End Function

Public Function GetOpenTasks() As Collection
    Dim task As CTask
    Dim colOpenTasks As Collection
    For Each task In m_colTasks
        If task.IsOpen() Then colOpenTasks.Add task, task.Id
    Next task
    Set GetOpenTasks = colOpenTasks
End Function
```

Estructuras de objetos de negocio

Visión general

Cuando un NO 207 se añade o se actualiza, envía todos sus atributos a un componente 222 de servidor para escribirse en la base de datos. En lugar de referirse explícitamente a cada atributo en la lista de parámetros de las funciones en el ACC 208 y el componente 222 de servidor, todos los atributos se envían en una única matriz variante. Esta matriz también se conoce como una estructura.

Enfoque

Cada NO 207 editable tiene un método denominado AsStruct (como estructura) que toma las variables miembro del objeto y las pone en una matriz variante. El ACC 208 llama a este método en un NO 207 antes de enviar el NO 207 al componente 222 de servidor para añadirse o actualizarse. La razón por la que esto es necesario es que, aunque pueden pasarse referencias de objetos por valor a través de la red, los propios objetos no. Sólo los tipos de datos básicos como Número entero y Cadena pueden enviarse por valor a un componente 222 de servidor. Una enumeración VB se

usa para denominar las ranuras de la estructura, de manera que el componente 222 de servidor pueda usar un nombre simbólico para acceder a elementos en la matriz en lugar de a un índice. Obsérvese que esto se usa generalmente sólo cuando se realizan adiciones o actualizaciones completas de un objeto 207 de negocio.

En algunos casos, hay una razón para reinstanciar el NO 207 en el lado del servidor. El método FromStruct (desde la estructura) hace exactamente lo contrario que el método AsStruct e inicia el NO 207 a partir de una matriz variante. El tamaño de la estructura que se ha pasado como un parámetro al FromStruct se comprueba para aumentar la certeza de que es una estructura válida.

Cuando un NO 207 contiene una referencia a otro NO 207, el método AsStruct almacena la clave principal del NO 207 al que se hace referencia. Por ejemplo, la estructura de Tarea contiene un PerformerId (identificación de realizador), no el NO 207 de realizador al que hace referencia la tarea. Cuando el método FromStruct se encuentra con el PerformerId en la estructura de tarea, instancia un nuevo NO de realizador y lo llena en la ID, dejando el resto del NO de realizador vacío.

Ejemplo

```
' CTask Business Object

' enumeration of all task attributes
Public Enum TaskAttributes
    cmTaskId
    cmTaskName
    :
    cmTaskDescription
End Enum

' all task attributes declarations here

' all setter and getter functions here

Public Function AsStruct() As CTask
    On Error Goto ErrorHandler

    ' create and fill structure
    Dim vStruct(cmTaskNumOfAttributes - 1) As Variant
    vStruct(cmTaskId) = m_vId
    vStruct(cmTaskName) = m_sName
    vStruct(cmTaskPerformerId) = m_vPerformerId
    :
    vStruct(cmTaskDescription) = m_sDescription

    AsStruct = vStruct
    Exit Function
ErrorHandler:
    Err.Raise Err.Number
End Function

Public Sub FromStruct(vStruct As Variant)
    On Error Goto ErrorHandler

    ' check size of vStruct
    If Ubound(vStruct) <> (cmTaskNumOfAttributes - 1) Then Err.Raise
cmErrInvalidParameters

    ' update my values from the structure
    m_vId = vStruct(cmTaskId)
    m_sName = vStruct(cmTaskName)


        m_vPerformer.Id = vStruct(cmTaskPerformerId)
        :
        m_sDescription = vStruct(cmTaskDescription)
        Exit Sub
    ErrorHandler:
        Err.Raise Err.Number
    End Sub
```

Clonación de objetos de negocio

Visión general

A menudo se hace una copia de un objeto 207 de negocio. La clonación es una forma de implementar esta clase de funcionalidad mediante encapsulación del proceso de copiado en el propio NO 207. Los controladores 206 que necesitan hacer cambios de prueba en un objeto 207 de negocio sencillamente piden al NO 207 original un clon y hacen los cambios en el clon. Si el usuario decide guardar los cambios, el controlador 206 pide al NO original que se actualice a partir de los cambios hechos en el clon.

Enfoque

Cada NO 207 tiene un método de clon para devolver una copia superficial de sí mismo. Una copia superficial es una copia que no incluye copias de los otros objetos a los que se refiere el NO 207, sino tan sólo una copia de una referencia a esos objetos. Por ejemplo, para clonar una tarea, no se da al clon un objeto de reclamación totalmente nuevo; se da al clon una nueva referencia del objeto de reclamación existente. Las series son la única excepción a esta norma, siempre se copian completamente ya que contienen referencias a otros NO.

Cada NO 207 tiene también un método UpdateFromClone (actualizar a partir del clon) para permitir "fundir" un clon de nuevo consigo mismo cambiando sus atributos para que coincidan con los cambios hechos en el colon.

Ejemplo

```
' CTask Business Object
Public Function Clone() As CTask
    On Error Goto ErrorHandler

    ' create clone object
    Dim tskClone As CTask
    Set tskClone = New CTask

    ' fill clone with my data
    With tskClone
        .Id = m_vId
        .Name = m_sName
        .PerformerId = m_vPerformerId
        Set .Performer = m_prfPerformer

        .Description = m_sDescription
    End With

    Set Clone = tskClone
    Exit Function
ErrorHandler:
    Err.Raise Err.Number
End Function

Public Sub UpdateFromClone(tskClone As CTask)
    On Error Goto ErrorHandler

    ' set my values equal to the clone's val
    With tskClone
        m_vId = .ID
        m_sName = .Name
        m_vPerformerId = .PerformerId
        Set m_prfPerformer = .Performer
        :
        m_sDescription = .Description
    End With

    Exit Sub
ErrorHandler:
    Err.Raise Err.Number
End Sub
```

Objetos de negocio sin finalizar

Visión general

Los NO 207 se llenan ocasionalmente sólo a la mitad por motivos de rendimiento. Esto se hace para preguntas que abarcan múltiples tablas que devuelven grandes conjuntos de datos. El uso de NO 207 sin finalizar puede ser un proceso propenso a errores, por lo que es esencial que la no finalización de los NO se gestione y se contenga de forma cuidadosa.

En la mayor parte de las aplicaciones, hay dos tipos de ventanas, ventanas de búsqueda y ventanas de edición/detalles. Las ventanas de búsqueda son las únicas ventanas que dejan sin finalizar los NO. Generalmente, la no finalización sólo es un problema cuando una ventana de detalles que espera un NO completamente finalizado recibe un NO sin finalizar desde una ventana de búsqueda.

Enfoque

Las ventanas de detalles regeneran ("refresh") los NO 207 que les pasan las ventanas de búsqueda, independientemente de si ya están totalmente finalizados o no. Esto aborda los problemas asociados con el traspaso de NO sin finalizar y también ayuda a garantizar que el NO 207 está actualizado.

Este enfoque requiere otro tipo de método (aparte de Obtener, Añadir, Actualizar y Borrar) en el ACC 208: un método de regeneración. Este método es muy similar al método de Obtener (de hecho, llama al mismo método en el componente de servidor) pero es único porque regenera los datos en los objetos que ya se han creado. El controlador 206 de la ventana de detalles llama al ACC 208 adecuado que pasa el NO 207 que debe regenerarse, y puede asumir que, cuando el control vuelve desde ACC 208, el NO 207 estará actualizado y totalmente finalizado.

Esto puede no ser necesario si dos ventanas están estrechamente relacionadas. Si la primera ventana es la única ventana que abre la segunda, es necesario que la segunda ventana regenere el NO 207 que le ha pasado la primera ventana si conoce que el NO 207 está totalmente finalizado para su uso.

ACC

Los ACC 208 son responsables de transmitir datos desde filas y columnas en un conjunto de registros a objetos 207 de negocio, y ejecutar llamadas a los componentes 222 de servidor en nombre de los controladores 206.

Recuperar objetos de negocio

Visión general

Después de pedir a un componente que recupere datos, el ACC 208 clasifica los datos devueltos por el componente en objetos 207 de negocio que se usan por el controlador 206 IU.

Enfoque

El proceso de clasificación es como sigue:

los ACC 208 llaman a GetRows (obtener filas) en el conjunto de registros para obtener una copia de sus datos en una matriz variante con el fin de liberar el conjunto de registros lo antes posible. Existe un método para coordinar la clasificación de cada conjunto de registros devuelto por el componente.

Sólo se coordina un conjunto de registros en el proceso de clasificación de un único método. Existe un método para construir un NO a partir de una única fila de un conjunto de registros. Este método se llama una vez para cada fila en el conjunto de registros por el método de coordinación de clasificación.

Ejemplo

```
' Task CCA
Public Function GetAllTasks() As Collection
    On Error Goto ErrorHandler

    ' call a helper method to retrieve tasks
    Dim vRows As Variant
    vRows = RetrieveAllTasks

    Dim i As Integer
    Dim task As CTask
    Dim colTasks As Collection
    Set colTasks = New Collection

    ' vRows is dimmed as column, row.  Loop til I run out of rows.
    For i = 0 To Ubound(vRows, 2)
        ' build BO using helper method
        Set task = BuildTaskFromRow(vRows, i)
        ' add to collection with ID as the key
        colTasks.Add task, task.Id
    Next i

    Set MarshalTasks = colTasks
    Exit Function
ErrorHandler:
    Err.Raise Err.Number
End Function

Private Function RetrieveAllTasks() As Variant
    On Error Goto ErrorHandler

    ' call my component and get a recordset full of all tasks
    Dim rs As ADOR.Recordset
    Set rs = tskComp.GetAllTasks()

    ' get data in variant array from the recordset
    GetAllTasks = rs.GetRows

    ' release the recordset ASAP
    rs.Close
    Set rs = Nothing

    Exit Function
ErrorHandler:
    Err.Raise Err.Number
End Function

Private Function BuildTaskFromRow(vRows As Variant, nCurrentRow As
Integer, _
                                  Optional task As CTask) As CTask
    On Error Goto ErrorHandler

    ' create task if it wasn't passed
     If task Is Nothing Then Set task = New CTask


    ' fill task with data
    With task
        .Id = vRows(0, nCurrentRow)
        .Name = vRows(1, nCurrentRow)
        .PerformerId = vRows(2, nCurrentRow)
        :
        .Description = vRows(32, nCurrentRow)
    End With

    Set BuildTaskFromRow = task
    Exit Function
ErrorHandler:
    Err.Raise Err.Number
End Function
```

Regenerar objetos de negocio

Visión general

La lógica para regenerar NO 207 es muy similar a la lógica para crearlos en primer lugar. Un método "regenerar" es muy similar a un método "obtener", pero debe usar ON 207 que ya existen al llevar a cabo un proceso de clasificación.

Ejemplo

```
' Task CCA
Public Sub RefreshTask(task As CTask)
    On Error Goto ErrorHandler

    ' call a helper method to retrieve tasks
    Dim vRow As Variant
    vRow = RetrieveTaskWithId(task.Id)

    BuildTaskFromRow vRow, i, task
    Exit Sub
ErrorHandler:
    Err.Raise Err.Number
End Sub

Private Function RetrieveTaskWithId(vId As Variant) As Variant
    On Error Goto ErrorHandler

    ' call my component and get a recordset full of all tasks
    Dim rs As ADOR.Recordset
    Set rs = tskComp GetTaskWithId(vId)

    ' get data in variant array from the recordset
    RetrieveTaskWithId = rs.GetRows

    ' release the recordset ASAP
    rs.Close
    Set rs = Nothing

    Exit Function
ErrorHandler:

        Err.Raise Err.Number
    End Function
```

Añadir objetos de negocio

Visión general

Los controladores 206 son responsables de crear y ocupar nuevos NO 207. Para añadir un NO 207 a la base de datos, el controlador 206 debe llamar al ACC 208 y pasarle el objeto 207 de negocio que debe añadirse. El ACC 208 llama al método AsStruct en el NO 207, y pasa la estructura NO al componente para guardarla. Entonces actualiza el NO 207 con la ID y la marca de fecha y hora generada por el servidor. Obsérvese que el método en el ACC 208 simplemente actualiza el NO 207.

Ejemplo

```
' Task CCA
Public Sub AddTask(task As CTask)
    On Error Goto ErrorHandler

    ' call component to add task passing a task structure
    Dim vIdAndTimestamp As Variant
    vIdAndTimestamp = tskComp.AddTask(task.AsStruct())

    ' update ID and Timestamp on task
    task.Id = vIdAndTimestamp(0)
    task.TimeStamp = vIdAndTimestamp(1)
    Exit Sub
ErrorHandler:
    Err.Raise Err.Number
End Sub
```

Actualización de objetos de negocio

Visión general

El proceso de actualización es muy similar al proceso de adición. La única diferencia es que el componente de servidor sólo devuelve una marca de fecha y hora, ya que el NO ya tiene una ID.

Ejemplo

```
' Task CCA
Public Sub UpdateTask(task As CTask)
    On Error Goto ErrorHandler

    ' call component to update task passing a task structure
    Dim lTimeStamp As Long
    lTimeStamp = tskComp.AddTask(task.AsStruct())

    ' update Timestamp on task
    task.TimeStamp = lTimeStamp


        Exit Sub
ErrorHandler:
    Err.Raise Err.Number
End Sub
```

Borrar objetos de negocio

Visión general de borrar

Como en los métodos de añadir y actualizar, los métodos de borrar toman un objeto 207 de negocio como parámetro y no tienen un valor devuelto. El método de borrar no modifica el objeto 207 que está borrando ya que el objeto se eliminará pronto.

Ejemplo

```
' Task CCA
Public Sub DeleteTask(task As CTask)
    On Error Goto ErrorHandler

    ' call component to update task passing a the ID and Timestamp
    tskComp.DeleteTask task.Id, task.TimeStamp
    Exit Sub
ErrorHandler:
    Err.Raise Err.Number
End Sub
```

Componente de servidor

Los componentes 222 de servidor tienen dos finalidades: cumplir las normas de negocio y llevar a cabo operaciones de acceso a datos. Están diseñados para evitar la duplicación de la lógica entre funciones.

Diseño para la reutilización

Cumplir la encapsulación

Cada componente 222 de servidor encapsula una única tabla de base de datos o un conjunto de tablas de base de datos estrechamente relacionadas. En la medida de lo posible, los componentes 222 de servidor seleccionan o modifican datos de una única tabla. Un componente selecciona ocasionalmente de una tabla que "pertenece" o está encapsulada por otro componente con el fin de usar una unión (por motivos de eficiencia). Un componente 222 de servidor colabora a menudo con otros componentes de servidor para completar una transacción de negocio.

Partición lógica entre múltiples clases

Si el componente se vuelve muy grande, se divide en más de una clase. Cuando esto ocurre, se divide en dos clases, una para normas de negocio y una para acceso a datos. La clase de normas de negocio implementa la interfaz del componente y utiliza la clase de acceso a datos para modificar los datos según se necesite.

Ejemplo

```
Private Function MarkTaskInError(vMsg As Variant, _
                                 vTaskId As Variant, _
                                 lTimestamp As Variant, _
                                 sReason As String) As Long
    On Error GoTo ErrorHandler
    Const cmMethodName = "MarkTaskInError"

    ' set the SQL statement
    Dim sSQL As String
    sSQL = cmSQLMarkTaskInError

    ' get a new timestamp
    Dim lNewTimeStamp As Long
    lNewTimeStamp = GetTimeStamp()

    ' create and fill a collection of arguments to be merged with
    ' the SQL by the ExecuteQuery method
    Dim colArgs As CCollection
    Set colArgs = New CCollection
    With colArgs
        .Add lNewTimeStamp
        .Add cmDBBooleanTrue
        .Add sReason
        .Add vTaskId
        .Add lTimestamp
    End With

    ' run the SQL and set my return value
    ExecuteQuery vMsg, cmUpdate, sSQL, colArguments:=colArgs
    MarkTaskInError = lNewTimeStamp

    ' tell MTS I'm done
    GetObjectContext.SetComplete
    Exit Function

ErrorHandler:
    ' do error handling here
End Function
```

Tratamiento de errores

Información general

Con la excepción de "Class_Initialize" (inicialización de clase), "Class_Terminate" (finalización de clase), y métodos llamados dentro de un elemento de tratamiento de errores, cada función o subrutina tiene una instrucción "On Error Go To" (en caso de error ir a) definida por el usuario. La primera línea de este procedimiento es: "On Error GoTo ErrorHandler" (en caso de error ir al elemento de tratamiento de errores). A una línea próxima al final del procedimiento se le da la etiqueta "ErrorHandler". (Obsérvese que dado que las etiquetas de línea en VB 5.0 tienen un alcance de procedimiento, cada procedimiento puede tener una línea etiquetada "ErrorHandler"). La etiqueta ErrorHandler va precedida de una instrucción de función Exit Sub o Exit para evitar ejecutar el código de tratamiento de errores cuando no hay ningún error.

Los errores se tratan de manera diferente basándose en el nivel del módulo en la aplicación (es decir, los módulos de interfaz de usuario son responsables de mostrar mensajes de error al usuario).

Todos los módulos se benefician de la arquitectura técnica para registrar cronológicamente los mensajes. Los módulos de cliente que ya tienen una referencia en la arquitectura llaman al objeto Log Manager (gestor de registro cronológico) directamente. Dado que los módulos de servidor normalmente no tienen una referencia en la arquitectura, utilizan la función global LogMessage() que se cumple en cada componente de servidor.

Cualquier error que surja en un componente 222 de servidor se trata por el controlador 206 IU llamante. Esto asegura que el usuario ha recibido una notificación adecuada del error y que los errores de negocio no se traducen en errores fatales no tratados.

Todos los errores inesperados se tratan por una función del elemento de tratamiento de errores general en el módulo de arquitectura global para cerrar siempre la aplicación con cortesía.

Errores del componente de servidor

El elemento de tratamiento de errores para cada módulo de servicio contiene una instrucción Case para comprobar todos los errores anticipados. Si el error no es un error recuperable, la lógica para el tratamiento es informar en primer lugar al MTS del error llamando al Get ObjectContext.SetAbort(). A continuación, la función global LogMessage() se llama para registrar cronológicamente la descripción corta para el personal de soporte de nivel uno. Entonces se llama a la función LogMessage() una segunda vez para registrar cronológicamente la descripción detallada del error para el personal de soporte de nivel superior. Finalmente, el error vuelve a elevarse, de manera que la función llamante conozca la operación que falló.

Una condición Case por defecto se codifica para tratar cualquier error inesperado. Esto registra cronológicamente el error VB generado y entonces lo eleva. A continuación se proporciona una muestra de código:

A continuación se da un ejemplo de cómo se implementa el tratamiento de errores en el componente de tarea cuando se hace un intento de reasignar una tarea a un realizador que no existe. Al ejecutar SQL para reasignar una tarea a un realizador que no existe se genera un error de infracción de integridad referencial que queda atrapado en este elemento de tratamiento de errores:

```
'Class Declarations
Private Const cmClassName = "CTaskComp"

Public Sub ReassignTask(…)
    On Error GoTo ErrorHandler
    Private Const cmMethodName = "ReassignTask"
    Private Const cmErrReassignTask = "Could not reassign task."
    :
    ' logic to reassign a task
    :
    GetObjectContext.SetComplete
    Exit Sub

ErrorHandler:
    Dim sShortDescr As String
    sShortDescr = cmErrReassignTask
' log short description as warning
LogMessage vMsg, Err.Number, cmSeverityWarning, cmClassName,
cmMethodName, sShortDescr

    Dim sLongDescr As String
    Select Case Err.Number
        Case cmErrRefIntegrityViolation
            GetObjectContext.SetAbort
            sLongDescr = "Referential integrity violation - tried " _
                & "to reassign task to a non-existant performer.  " _
                & "Association ID: " & sAssnId _
                & "Association Type: " & sAssnType _
                & "Old Performer Id: " & sOldPerformerId _
                & "New Performer Id: " & sNewPerformerId
            ' log long description as severe
            LogMessage vMsg, Err.Number, cmSeveritySevere, cmClassName,
cmMethodName, _
                       sLongDescr
            Err.Raise Err.Number
        :
        ' more error handling
        :
        Case Else
            ' let architecture handle unanticipated error
            Dim nResumeCode As Integer
            nResumeCode = GeneralErrorHandler(vMsg, cmServer, cmClassName,
cmMethodName)
            Select Case nResumeCode
                Case cmErrorResume
                    Resume
                Case cmErrorResumeNext
                    Resume Next
                Case cmErrorExit
                    Exit Sub
                Case Else
                    GetObjectContext.Abort
                    Err.Raise Err.Number
            End Select
    End Select
End Sub
```

ACC, ICC, Objetos de Negocio y Formularios

Todos los ICC, ACC, objetos de negocio y formularios elevan cualquier error generado. Una muestra de código se proporciona a continuación:

```
Sub SubName()
    On Error GoTo ErrorHandler
    <the procedure's code here>
    .........
    Exit Sub
ErrorHandler:
    Err.Raise Err.Number
End Sub
```

Errores del controlador de interfaz de usuario

Los controladores 206 de interfaz de usuario tratan cualquier error generado y pasado desde los niveles inferiores de la aplicación. Los módulos IU son responsables de tratar cualquier error que pueda elevarse por los componentes 222 de servidor mostrando un cuadro de mensaje al usuario.

Cualquier error generado en las IU también se muestra al usuario en un cuadro de diálogo. Cualquier error iniciado en el cliente se registra cronológicamente usando el procedimiento LogMessage(). Los errores iniciados en el servidor ya se habrán registrado cronológicamente y por tanto no necesitan volver a registrarse cronológicamente.

Todos los errores inesperados se atrapan por un método de errores general en el módulo de arquitectura global. Dependiendo del valor devuelto por esta función, el controlador puede reanudarse en la instrucción que desencadenó el error, reanudarse en la siguiente instrucción, llamar a su función de desactivación para cerrarse o llamar al método de cierre en el objeto de aplicación para cerrar toda la aplicación.

No se elevan errores desde este nivel de la aplicación, ya que los controladores tratan todos los errores. Una muestra de código de un elemento de tratamiento de errores de controlador se proporciona a continuación:

```
'Class Constants
Private Const cmClassName As String = "<ComponentName>"

Sub SubName()
    On Error GoTo ErrorHandler
    Const cmMethodName As String = "<MethodName>"
    :
    :
    <the procedure's code here>
    :
    :
    Exit Sub

ErrorHandler:

    Select Case CStr(Err.Number)
        Case …
            'display the error to the user
            ' perform any necessary logic
            Exit Sub (or Resume, or Resume Next)
        :
        :
        Case Else
            Dim nResumeCode As Integer


            nResumeCode = GeneralErrorHandler(vMsg, cmController,
cmClassName, cmMethodName)
            Select Case CStr(nResumeCode)
                Case cmErrorResume
                    Resume
                Case cmErrorResumeNext
                    Resume Next
                Case cmErrorExit
                    Exit Sub
                Case cmErrorQuiesce
                    Quiesce
                Case Else
                    objApp.SHUTDOWN
            End Select
    End Select
End Sub
```

Localización

La aplicación CBAM está construida de tal forma que puede localizarse para diferentes idiomas y países con un esfuerzo mínimo o conversión.

Requisitos y alcance

La arquitectura CBAM proporciona soporte para ciertas características de localización:

Almacén de recursos localizable;

diseño de interfaz de usuario flexible;

localización del formato de datos; y

exposición de características de localización del sistema operativo Windows.

Lista de comprobación del enfoque de localización

Característica de localización	Soportada por servicio de arquitectura	Soportada por la arquitectura del API	Mejores prácticas y suposiciones*
Código de idioma (identificador local)	√		
Husos horarios			√
Fecha/hora		√	
Nombre			√
Números de teléfono			√
Funciones a evitar			√
Pesos y medidas			√
Moneda			√
Direcciones/jerarquías de dirección			√
Menús, iconos, etiquetas/identificadores en Windows			√
Mensajes/diálogos	√		
Funciones de cadena, orden de clasificación y comparación de cadenas	√		
Tablas de código	√		
Listas desplegables	√		
Plantillas de formularios y correspondencia			√
Documentación en línea e impresa			√
Base de datos (DB2)			√
Controles por 3as partes			√
Miscelánea			√

Almacén de erratas localizables

La aplicación CBAM tiene una infraestructura para soportar múltiples idiomas. La arquitectura actúa como un almacén de erratas centralizado a través de su enfoque de tabla de códigos.

Las tablas de códigos tienen la localización en mente. Cada fila en la tabla de códigos contiene un identificador de idioma asociado. A través del identificador de idioma, cualquier código dado puede soportar valores de cualquier idioma.

Interfaz 400 flexible

La interfaz 400 de usuario flexible y el código hacen que la personalización sea sencilla. La figura 4 ilustra cómo se vuelven a pintar y se recopilan diferentes idiomas. Por ejemplo, pueden albergarse fácilmente una IU 404 en inglés y una IU 406 en francés. Esto conlleva un mínimo esfuerzo porque ambas IU comparten el mismo código 402 base central. Las actualizaciones de las IU son meramente un cambio superficial.

Se usan gráficas genéricas y se evita el congestionamiento para crear una interfaz de usuario que sea fácil de localizar.

Localización de datos

Los ajustes de localización del idioma afectan a la forma en que los datos se muestran en las IU (interfaces de usuario). El formato de visualización del sistema por defecto es diferente para cada idioma/país. Por ejemplo:

Inglés (Estos Unidos) muestra “mes/día/año” (p. ej. “05/16/98”)

Inglés (Reino Unido) muestra “día/mes/año” (p.ej. “16/05/98”)

La IU de la presente invención emplea una serie de controles de datos por terceras partes que incluyen *Sheridan Calendar Widgets* (de Sheridan Software) que permiten a los desarrolladores ajustar máscaras de entrada predefinidas para datos (a través de las páginas de propiedad de los controles; la propiedad en este caso es “máscara”).

Aunque la propiedad máscara puede manipularse, el ajuste por defecto se acepta preferiblemente (el ajuste por defecto para máscara es “0 - sistema por defecto”; se ajusta en el momento del diseño). Al aceptar los ajustes del sistema por defecto se elimina la necesidad de codificar múltiples locales (con alguna posible excepción), no se interfiere con las funciones intrínsecas de Visual Basic tales como DateAdd (añadir datos), y permite formatear datos como cadenas para su uso en SQL.

El programa de evaluación ilustrado a continuación muestra cómo un dato que usa el formato de datos del sistema por defecto en inglés (Reino Unido) vuelve a formatearse en un formato definido por el usuario (en este caso, una cadena constante para usar con instrucciones DB2 SQL):

```
                nResumeCode = GeneralErrorHandler(vMsg, cmController,
cmClassName, cmMethodName)
                Select Case CStr(nResumeCode)
                    Case cmErrorResume
                        Resume
                    Case cmErrorResumeNext
                        Resume Next
                    Case cmErrorExit
                        Exit Sub
                    Case cmErrorQuiesce
                        Quiesce
                    Case Else
                        objApp.SHUTDOWN
                End Select
    End Select
End Sub
```

Aprovechamiento del sistema operativo Windows

La arquitectura CBAM expone métodos de interfaz en el objeto de RegistryService (servicio de registro) para acceder a valores específicos locales que se ajustan desde el panel de control.

La arquitectura expone una API del objeto RegistryService que permite acceder a toda la información disponible en el panel de control. A continuación se muestra la firma del API:

GetRegionalInfo (Info As RegionalInfo) As String

en la que RegionalInfo puede ser cualquiera de los valores en la tabla siguiente:

Valores RegionalInfo			
CmLanguageId	CmDTDateSeparator	cmDayLongNameMonday	cmMonthLongNameJan
CmLanguageLocalized	CmDTTimeSeparator	cmDayLongNameTuesday	cmMonthLongNameFeb
CmLanguageEnglish	CmDTShortDateFormat	cmDayLongNameWednesday	cmMonthLongNameMar
CmLanguageAbbr	CmLDTongDateFormat	cmDayLongNameThursday	cmMonthLongNameApr
CmLanguageNative	CmDTTimeFormat	cmDayLongNameFriday	cmMonthLongNameMay
CmCountryId	CmDTDateFormatOrdering	cmDayLongNameSaturday	cmMonthLongNameJun
CmCountryLocalized	CmDTLongDateOrdering	cmDayLongNameSunday	cmMonthLongNameJul
CmCountryEnglish	CmDTTimeFormatSpecifier	cmDayAbbrNameMonday	cmMonthLongNameAug
CmCountryAbbr	CmDTCenturyFormatspecifier	cmDayAbbrNameTuesday	cmMonthLongNameSep
CmCountryNative	CmDTTimeWithLeadingZeros	cmDayAbbrNameWednesday	cmMonthLongNameOct
CmLanguageDefaultId	CmDTDayWithLeadingZeros	cmDayAbbrNameThursday	cmMonthLongNameNov
CmCountryDefaultId	CmDTMonthWithLeadingZeros	cmDayAbbrNameFriday	cmMonthLongNameDec
	CmDTDesignatorAM	cmDayAbbrNameSaturday	cmMonthAbbrNameJan
	CmDTDesignatorPM	cmDayAbbrNameSunday	cmMonthAbbrNameFeb
			cmMonthAbbrNameMar
			cmMonthAbbrNameApr
			cmMonthAbbrNameMay
			cniMonthAbbrNameJun
			cmMonthAbbrNameJul
			cmMonthAbbrNameAug
			cmMonthAbbrNameSep
			cmMonthAbbrNameOct
			cmMonthAbbrNameNov
			cmMonthAbbrNameDec

Ejemplo de obtener RegionalInfo

```
Private Sub Command1_Click()
    MsgBox "This is the language id for English: " & _
    GetRegionalInfo(cmLanguageId)
End Sub
```

Unidad lógica de trabajo

El patrón de unidad lógica de trabajo (LUW) permite la separación del asunto entre los controladores 206 IU y la lógica de negocio.

Visión general

Normalmente, cuando un usuario abre una ventana, hace cambios y clica en OK o guardar, se llama a un componente 222 de servidor para ejecutar una transacción que guardará los cambios del usuario en la base de datos. Debido a esto, puede decirse que la ventana define el límite de la transacción, ya que la transacción se confirma cuando se cierra la ventana.

El patrón LUW es útil cuando las transacciones de la base de datos abarcan ventanas. Por ejemplo, un usuario empieza a editar datos en una ventana y después, sin guardar, abre otra ventana y empieza a editar datos en esa ventana, el proceso de guardar implica múltiples ventanas. Ningún controlador 206 de ventana puede gestionar el proceso de guardar, ya que deben guardarse datos de las dos ventanas como una parte de una unidad de trabajo indivisible. En su lugar, se introduce un objeto LUW para gestionar el proceso de guardar.

La LUW actúa como una clase de “cesta de la compra”. Cuando un controlador 206 modifica un objeto 207 de negocio, lo pone en la cesta para abonarlo (guardarlo) más tarde. Puede pasar la cesta a otro controlador 206 para acabar la compra (modificar más objetos), y después a un tercer controlador que realiza el pago (pida a la LUW que inicie el guardado).

Enfoque

Los controladores 206 pueden tener diferentes niveles de “conciencia” LUW:

Requiere nueva: siempre crea una LUW nueva;

Requiere: requiere una LUW y crea una LUW nueva sólo si el controlador llamante no pasa una;

Requiere existente: requiere una LUW pero no crea una LUW nueva si el controlador llamante no pasa una. Eleva un error si no se pasa una LUW; y

No soportada: no es capaz de usar una LUW.

Los controladores 206 que siempre requieren una nueva LUW crean esa LUW en su función ArchInitClass durante la inicialización. Puede escoger si implicar o no a otras ventanas en su LUW. Si es deseable que otra ventana se implique en un LUW existente, el controlador 206 que posee la LUW pasa una referencia a esa LUW cuando llama al objeto 202 de aplicación para que abra la segunda ventana. Los controladores 206 que requieren una LUW o que requieren una LUW existente aceptan la LUW como un parámetro en la función ArchInitClass.

Las LUW contienen toda la lógica necesaria para persistir sus “contenidos” - los NOs 207 modificados. Tratan métodos de llamada en el ACC 208 y actualizan los NO 207 con nuevas ID y/o marcas de fecha y hora.

Jerarquía API de la arquitectura

A continuación se da una visión general del modelo de objeto de arquitectura, incluyendo una descripción de cada método y los parámetros que acepta. Secciones adicionales se refieren a los conceptos detrás de áreas específicas (memorizar código en caché, registro cronológico de los mensajes y acceso a datos) más detalladamente.

Objeto de arquitectura

La figura 5 ilustra las propiedades actuales en el objeto 200 de arquitectura.

Las siguientes son API situadas en el objeto 200 de arquitectura que devuelven una instancia recuperada o creada de un objeto que implementa las siguientes interfaces:

CodesMan() 500;

TextMan() 502;

IdMan() 504;

RegMan() 506;

LogMan() 508;

ErrMan() 510;

UserMan() 512; y

SecurityMan() 514.

AsMsgStruct()

Este método en el objeto de arquitectura devuelve una estructura variante para pasar junto a un mensaje remoto.

Sintaxis:

```
Public Function AsMsgStruct() As Variant
End Function
```

Ejemplo:

```
Dim vMsg As Variant
vMsg =  objArch.AsMsgStruct
```

CodesMan

Las siguientes son API situadas en la interfaz del objeto 200 de arquitectura llamado CodesMan 500:

CheckCacheFreshness();

FillControl(ctlControl, nCategory, nFillType, [nCodeStatus], [colAssignedCodes]);

FilterCodes(colAllCodes, nCodeStatus);

GetCategoryCodes(nCategory);

GetCodeObject(nCategory, sCode);

GetResourceString(IStringId);

GetServerDate();

RefreshCache();

RemoveValidDates(sCode, colPassedInAssignedCodes); y

SetServerDate(dtServerDate).

CheckCacheFreshnes()

Comprueba si la memoria caché ha expirado, si es así, regenera.

Sintaxis:

```
Private Sub CheckCacheFreshness()
End Sub
```

Ejemplo:

```
CheckCacheFreshness
```

FillControl()

Esta API se usa para rellenar cuadros desplegables o conjuntos de cuadros con valores de una lista de CodeDecodes. Devuelve una serie para posteriores búsquedas a los objetos Code empleados para rellenar los controles.

Sintaxis:

```
Public Function FillControl(ctlControl As Object, nCategory As CodeDecodeCats, nFillType As
CodeDecodeLengths, Optional nCodeStatus As CodeDecodeFilters = cmValidCodes, Optional
colAssignedCodes As CCollection) As CCollection
End Function
```

Parámetros:

ctlControl: Una referencia a una lista desplegable o conjunto de cuadros.

nCategory: La constante basada en números enteros que clasificó estos CodeDecodes a partir de otros. Varias de las constantes válidas incluyen:

cmCatTaskType = 1

cmCatSource

cmCatTaskStatus

nFillType: El atributo de CodeDecode que desea rellenar. Varios de los valores válidos incluye:

cmCode

cmShortDecode

cmLongDecode

nCodeStatus: Valor opcional que filtra los Code Decodes según sus fechas Ejectivas y de Expiración. Varias de las constantes válidas incluyen:

cmAllCodes	Códigos pendientes + válidos + expirados
cmPendingCodes	Códigos cuya fecha efectiva es superior a la fecha actual
cnValidCodes	Códigos no pendientes + expirados

colAssignedCodes: se usan al rellenar un control que debe rellenar e incluir valores asignados.

Ejemplo:

```
'Declare an instance variable for States collection on object
Private colStates As CCollection

'Call FillControll API, and set local collection inst var to collection of codes which were
used to fill the controll.  This collection will be used for subsequent lookups.
Set colStates = objArch.CodesMan.FillControl(frmCurrentForm.cboStates, cmCatStates,
cmLongDecode)
```

FilterCodes()

Devuelve una serie de code/decodes que se filtran usando sus fechas de efectividad y de expiración basándose en qué nCodeStatus es pasado desde el método fillcontrol.

Sintaxis:

```
Private Function FilterCodes(colAllCodes As CCollection, nCodeStatus As CodeDecodeFilters) As
CCollection
End Function
```

Parámetros:

```
colAllCodes:
nCodeStatus:
```

Ejemplo:

```
Set colFilteredCodes = FilterCodes(colCodes, nCodeStatus)
```

GetCategoryCodes()

Devuelve una serie de objetos CCode dada una categoría válida.

Sintaxis:

```
Public Function GetCategoryCodes(nCategory As CodeDecodeCats) As CCollection
End Function
```

Parámetros,

nCategory: La constante basada en números enteros que clasificó estos CodeDecodes a partir de otros.

Ejemplo:

```
Dim colMyStates As CCollection
Set colMyStates = objArch.CodesMan.GetCategoryCodes(cmCatStates)
```

```
'Below shows an example of looking up the Code value for the currently selected state.
With frmCurrentForm.cboStates
If .ListIndex > -1 Then
Dim objCode As CCode
    Set objCode =  colStates(.ItemData(.ListIndex))
    sStateCode = objCode.Code
End If
End With
```

GetCodeDbject()

Devuelve un objeto CCode válido dados una categoría específica y un código.

Sintaxis:

```
Public Function GetCodeObject(nCategory As CodeDecodeCats, sCode As String) As CCode
End Function
```

Parámetros:

nCategory: La constante basada en números enteros que clasificó estos CodeDecodes a partir de otros.

sCode: Una cadena que indica el atributo de Code del objeto CodeDecode.

Ejemplo:

```
frmCurrentForm.lblState = objArch.CodesMan.GetCodeObject(cmCatStates, "IL").LongDecode
```

GetResourceString()

Devuelve una cadena del archivo de recursos dada una cadena ID específica.

Sintaxis:

```
Private Function GetResourceString(lStringId As Long) As String
End Function
```

Parámetros:

lStringId: La id asociada con la cadena en el archivo de recursos.

Ejemplo:

```
sMsg = arch.CodesMan.GetResourceString(CLng(vMessage))
```

GetServerDate()

Devuelve la fecha desde el servidor.

Sintaxis:

```
Private Function GetServerDate() As Date
End Function
```

Ejemplo:

```
SetServerDate CCA.GetServerDate
```

Refresh Cache() (Regenerar memoria caché)

Regenera todos los objetos de código en la memoria caché.

Sintaxis:

```
Private Sub RefreshCache()
End Sub
```

Ejemplo:

```
m_Cache.RefreshCache
```

RemoveValidCodes() (eliminar códigos válidos)

Elimina todos los códigos válidos de los pasados en la serie de códigos asignados, lo cual se usa para ver qué códigos están asignados y no son válidos.

Sintaxis:

```
Private Sub RemoveValidCodes(sCode As String, colPassedInAssignedCodes As CCollection)
End Sub
```

Parámetros:

sCode: Nombre de código

colPassedInAssignedCodes: Códigos ya en uso

Ejemplo:

```
RemoveValidCodes codCode.Code, colPassedInAssignedCodes
```

SetServerDate()

Establece la fecha del servidor.

Sintaxis:

```
Private Sub SetServerDate(dtServerDate As Date)
End Sub
```

Parámetros:

dtServerDate: Fecha del servidor.

Ejemplo:

```
SetServerDate CCA.GetServerDate
```

TextMan

Las siguientes son API situadas en la interfaz del objeto 200 de arquitectura llamada TextMan 502.

PairUpAposts_();

PairUpAmps_(); y

MergeParms ().

PairUpAposts_()

Pares de apóstrofes superiores en la cadena pasada.

Sintaxis:

```
Public Function PairUpAposts(sOriginalString As String) As String
End Function
```

Parámetros:

sOriginalString: cadena pasada por el llamante

Ejemplo:

```
Dim sString As String
sString = objArch.TextMan.PairUpAposts("This is Monika's string")
'expected return: sString = "This is Monika''s string"
```

PairUpAmps()

Pares superiores de signos & en la cadena pasada.

Sintaxis:

```
Public Function PairUpAmps(sOriginalString As String) As String
End Function
```

Parámetros:

sOriginalString: cadena pasada por el llamante

Ejemplo:

```
Dim sString As String
sString = objArch.TextMan.PairUpAmps("Forms&Corr")
 'expected return: sString = "Forms&&Corr"
```

MergeParms()

Funde cadena con la serie de parámetros pasados.

Sintaxis:

```
Public Function MergeParms(sString As String, colParms As CCollection) As String
End Function
```

Parámetros:

sOriginalString: cadena pasada por el llamante

colParms As Collection: serie de parámetros pasados por el llamante

Ejemplo:

```
Dim sString As String
sString = objArch.TextMan.MergeParms(sString, colParms)
```

IdMan

Las siguientes son API situadas en la interfaz en el objeto 200 de arquitectura llamado IdMan 504:

GetGUID();

GetSequenceID();

GetTimeStamp();

GetTrackingNbr(); y

GetUniqueId().

GetGUID()

Sintaxis;

```
Public Function GetGUID()
End Function
```

Ejemplo:

```
Dim vNewGuid As Variant
vNewGuid = objArch.IdMan.GetGUID
```

GetSequenceID_()

Sintaxis:

```
Public Function GetSequenceId(sTemplateType As CounterName) As String
End Function
```

Parámetros:

sTemplateType: La cadena que especifica la plantilla que solicita una id de secuencia (es decir cmCountFC = Forms & Corr)

Ejemplo:

```
frmCurrentForm.txtTemplateNumber = objArch.IdMan.GetSequenceId(cmCountFC)
```

GetTimeStamp_()

Sintaxis:

```
Public Function GetTimeStamp()
End Function
```

Ejemplo:

```
Dim nNewTimeStamp As Long
nNewTimeStamp = objArch.IdMan.GetTimeStamp
```

GetTrackingNbr_()

Sintaxis:

```
Public Function GetTrackingNbr()
End Function
```

Ejemplo:

```
Set objTechArch = New CTechArch
sUniqueTrackNum = objTechArch.IdMan.GetTrackingNbr
```

GetUniqueId_()

Sintaxis:

```
Public Function GetUniqueId()
End Function
```

Ejemplo:

```
Dim vUid As Variant
vNewUid = objArch.IdMan.GetUniqueId
```

RegMan

Las siguientes son API situadas en la interfaz del objeto 200 de arquitectura llamada RegMan 506:

GetCacheLife();

GetClientDSN();

GetComputerName();

GetDefaultAndValidate();

GetFCArchiveDirectory();

GetFCDistributionDirectory();

GetFCMasterDirectory();

GetFCUserDirectory();

GetFCWorkingDirectory();

GetHelpPath();

GetLocalInfo();

GetLogLevel();

GetRegionalInfo();

GetRegValue();

GetServerDSN();

GetSetting();

GetTimerLogLevel();

GetTimerLogPath(); y

GetUseLocalCodes().

GetCacheLife()

Sintaxis:

```
Public Function GetCacheLife() As String
End Function
```

Ejemplo:

```
Dim s As String
s = objArch.RegMan.GetCacheLife
```

GetClientDSN()

Sintaxis:

```
Public Function GetClientDSN() As String
End Function
```

Ejemplo:

```
Dim s As String
s = objArch.RegMan.GetClientDSN
```

GetComputerName()

Sintaxis:

```
Public Function GetComputerName() As String
End Function
```

Ejemplo:

```
Dim s As String
s = objArch.RegMan.GetComputerName
```

GetDefaultAndValidate()

Sintaxis:

```
Private Function GetDefaultAndValidate(sKey As String) As String
End Function
```

Parámetros:

sKey: La clave dentro del registro que el usuario está solicitando (es decir: Help Path)

Ejemplo:

```
Dim sDefault As String
sDefault = objArch.RegMan.GetDefaultAndValidate(sKey)
```

GetFCArchiveDirectory()

Sintaxis:

```
Public Function GetFCArchiveDirectory() As String
End Function
```

Ejemplo:

```
Dim s As String
s = objArch.RegMan.GetFCArchiveDirectory
```

GetFCDistributionDirectory()

Sintaxis:

```
Public Function GetFCDistributionDirectory() As String
End Function
```

Ejemplo:

```
Dim s As String
s = objArch.RegMan.GetFCDistributionDirectory
```

GetFCMasterDirectory()

Sintaxis:

```
Public Function GetFCMasterDirectory() As String
End Function
```

Ejemplo:

```
Dim s As String
s = objArch.RegMan.GetFCMasterDirectory
```

GetFCUserDirectory()

Sintaxis:

```
Public Function GetFCUserDirectory() As String
End Function
```

Ejemplo:

```
Dim s As String
s = objArch.RegMan.GetFCUserDirectory
```

GetFCWorkingDirectory()

Sintaxis:

```
Public Function GetFCWorkingDirectory() As String
End Function
```

Ejemplo:

```
Dim s As String
s = objArch.RegMan.GetFCWorkingDirectory
```

GetHelpPath_()

Sintaxis:

```
Public Function GetHelpPath() As String
End Function
```

Ejemplo:

```
Dim s As String
s = objArch.RegMan.GetHelpPath
```

GetLocalInfo()

Sintaxis:

```
Public Function GetLocalInfo() As String
End Function
```

Ejemplo:

```
Dim s As String
s = objArch.RegMan.GetLocalInfo
```

GetLogLevel()

Sintaxis:

```
Public Function GetLogLevel() As String
End Function
```

Ejemplo:

```
Dim s As String


s = objArch.RegMan.GetLogLevel
```

GetRegionalInfo()

Permite acceder a todos los valores específicos locales que se establecen desde el panel de control.

Sintaxis:

```
Public Function GetRegionalInfo(Info As RegionalInfo) As String
End Function
```

Parámetros:

Info: cadena que contiene la información regional. Varias de las constantes válidas incluyen:

- cmLanguageId = &H1 'id de idioma
- cmLanguageLocalized = &H2 'nombre de idioma localizado
- cmLanguageEnglish = &H1001 'nombre de idioma inglés
- cmLanguageAbbr = &H3 'nombre de idioma abreviado
- cmLanguageNative = &H4 'nombre nativo de idioma

Ejemplo:

```
Dim s As String
s = objArch.RegMan.GetRegionalInfo
```

GetRegValue()

Sintaxis:

```
Public Function GetRegValue() As String
End Function
```

Ejemplo:

```
Dim s As String
s = objArch.RegMan.GetRegValue
```

GetServerDSN()

Sintaxis:

```
Public Function GetServerDSN() As String
End Function
```

Ejemplo:

```
Dim s As String
s = objArch.RegMan.GetServerDSN
```

GetSetting()

Obtener ajuste (setting) desde el registro.

Sintaxis:

```
Public Function GetSetting(sKey As String) As String
End Function
```

Parámetros:

sKey: La clave dentro del registro que el usuario está solicitando (es decir: Help Path)

Parámetros:

```
GetHelpPath = GetSetting(cmRegHelpPathKey)
```

GetTimerLogLevel()

Sintaxis:

```
Public Function GetTimerLogLevel() As String
End Function
```

Ejemplo:

```
Dim s As String
s = objArch.RegMan.GetTimerLogLevel
```

GetTimerLogPath()

Sintaxis:

```
Public Function GetTimerLogPath() As String
End Function
```

Ejemplo:

```
Dim s As String
s = objArch.RegMan.GetTimerLogPath
```

GetUseLocalCodes_()

Sintaxis:

```
Public Function GetUseLocalCodes() As String
End Function
```

Ejemplo:

```
Dim s As String
s = objArch.RegMan.GetUseLocalCodes
```

LPSRTRoVBString()

Extrae una cadena VB de una memoria intermedia que contiene una cadena acabada en cero.

Sintaxis:

```
Private Function LPSTRToVBString$(ByVal s$)
End Function
```

LogMan

Las siguientes son API situadas en la interfaz del objeto 200 de arquitectura llamado LogMan 508:

LogMessage ();

Write ToDatavase(); y

Write ToLocalLog().

LogMessage ()

Se usa para registrar cronológicamente el mensaje. Esta función determinará dónde debería registrarse cronológicamente el mensaje, dado el caso, basándose en su gravedad y en el nivel de registro cronológico del vMsg.

Sintaxis:

```
Public Sub LogMessage(vMsg As Variant, _
                      lSeverity As Long, _
                      sClassName As String, _
                      sMethodName As String, _
                      sVersion As String, _
                      lErrorNum As Long, _
                      Optional sText As String = vbNullString)
End Sub
```

Parámetros:

vMsg: el mensaje de arquitectura estándar

lSeverity: la gravedad del mensaje

sClassName: el nombre de la clase que registra cronológicamente el mensaje

sMethodName: el nombre del método que registra cronológicamente el mensaje

sVersion: la versión del archivo binario (EXE o DLL) que contiene el método que registra cronológicamente el mensaje

lErrorNum: el número del error actual

sText: un parámetro opcional que contiene el texto del mensaje. Si se omite, se buscará el texto en un archivo de cadena o se empleará la descripción de error VB genérica

Ejemplo:

```
If Err.Number <> 0 Then
    ' log message
    Arch.LogMan.LogMessage(vMsg, cmSeverityFatal, "COrganizationCTLR", "InitForm",
                           GetVersion(), Err.Number, Err.Description)
    ' re-raise the error
    Err.Raise Err.Number
End If
```

Write ToDatabase()

Se usa para registrar cronológicamente el mensaje en la base de datos en el servidor usando CLoggingComp. Esta función devuelve el TrackingId (identificación de seguimiento) que se genera por el CLoggingObject.

Sintaxis:

```
Private Sub WriteToDatabase(vMsg As Variant, msgToLog As CMessage)
End Sub
```

Parámetros:

vMsg: el mensaje de arquitectura estándar

msgToLog: un parámetro que contiene el texto del mensaje.

Ejemplo:

```
If msgToLog.IsLoggableAtLevel(m_lLocalLogLevel) Then
    WriteToDatabase vMsg, msgToLog
End If
```

Write ToLocalLog()

Se usa para registrar cronológicamente el mensaje en un archivo plano, en caso de Windows 95, o en el registro cronológico de eventos NT, en caso de Windows NT.

Sintaxis:

```
Private Sub WriteToLocalLog(msgToLog As CMessage)
End Sub
```

Parámetros:

msgToLog: un parámetro que contiene el texto del mensaje.

Ejemplo:

```
ErrorHandler:
    WriteToLocalLog msgToLog
End Sub
```

ErrMann

Las siguientes son API situadas en la interfaz del objeto 200 de arquitectura llamado ErrMan 510:

HandleError();

RaiseOriginal();

ResetError(); y

Update().

HandleError()

Este método se pasa hasta el elemento de tratamiento de errores general en MArch.bas

Sintaxis:

```
Public Function HandleError(vMsg As Variant, nCompType As CompType, sClassName As String,
sMethodName As String) As ErrResumeCodes
End Sub
```

Parámetros.

vMsg: información de arquitectura general

nCompType: contiene información de capa (Cliente o Servidor)

nClassName: clase que elevó el error.

sMethodName: método que elevó el error.

RaiseOriginal()

Este método se usa para Reset (reiniciar) el objeto de error y elevarlo.

Sintaxis:

```
Public Sub RaiseOriginal()
End Sub
```

Ejemplo:

```
objArch.ErrMan.RaiseOriginal
```

ResetError()

Este método se usa para reiniciar atributos.

Sintaxis:

```
Public Sub ResetError()
End Sub
```

Ejemplo:

```
objArch.ErrMan.ResetError
```

Update()

Este método se usa para actualizar atributos en los valores de objeto de error global de VB.

Sintaxis:

```
Public Sub Update()
End Sub
```

Ejemplo:

```
objArch.ErrMan.Update
```

UserMan

Las siguientes son API situadas en la interfaz del objeto 200 de arquitectura llamado UserMan 512.

Userld:

Employeeld;

Employee Name;

EmployecFirstName;

EmployeeLastName;

EmployeeMiddielnitial;

GetAuthorizedEmployees;

IsSuperOf();

IsRelativeOf();

IsInRole().

Userld()

Sintaxis:

```
Public Property Get UserId() As String
End Property
```

Ejemplo:

```
Dim sNewUserId As String
sNewUserId = objArch.UserMan.UserId
```

Employeeld()

Sintaxis:

```
Public Property Get EmployeeId() As String
End Property
```

Ejemplo:

```
Dim sNewEmployeeId As String
sNewEmployeeId = objArch.UserMan.EmployeeId
```

Employee Name()

Sintaxis:

```
Public Property Get EmployeeName() As String
End Property
```

Ejemplo:

```
Dim sName As String
sName = objArch.UserMan.EmployeeName
```

EmployecFirstName()

Sintaxis:

```
Public Property Get EmployeeFirstName() As String
End Property
```

Ejemplo:

```
Dim sFName As String
sFName = objArch.UserMan.EmployeeFirstName
```

EmployeeLastName()

Sintaxis:

```
Public Property Get EmployeeLastName()As String
End Property
```

Ejemplo:

```
Dim sLName As String
sLName = objArch.UserMan.EmployeeLastName
```

EmployeeMiddielnitial()

Sintaxis:

```
Public Property Get EmployeeMiddleInitial() As String
End Property
```

Ejemplo:

```
Dim sMI As String
sMI = objArch.UserMan.EmployeeMiddleInitial
```

GetAuthorizedEmployees()

Crea una serie de supervisiones de usuario a partir del diccionario y devuelve la serie de GetAuthorizedEmployees de empleados autorizados

Sintaxis:

```
Public Function GetAuthorizedEmployees() As CCollection
End Function
```

Ejemplo:

```
Dim colAuth As Collection
colAuth = objArch.UserMan.GetAuthorizedEmployees
```

IsSuperOf_()

Comprueba si el usuario actual es supervisor del usuario que se ha pasado

Sintaxis:

```
Public Function IsSuperOf(sEmpId As String) As Boolean
End Function
```

Parámetros:

```
sEmpId: string containing Employee ID number
```

Ejemplo:

```
Dim bIsSuperOfMonika As Boolean
bIsSuperOfMonika = objArch.UserMan.IsSuperOf("TS012345")
```

IsRelativeOf_()

Comprueba si el usuario que se ha pasado está relacionado con el usuario actual

Sintaxis:

```
Public Function IsRelativeOf(sEmpId As String) As Boolean
End Function
```

Parámetros:

```
sEmpId: string containing Employee ID number
```

Ejemplo:

```
Dim bIsRelativeOfMonika As Boolean
bIsRelativeOfMonika = objArch.UserMan.IsRelativeOf("TS012345")
```

IsInRole_()

Comprueba para ver si el usuario actual está en un papel determinado

Sintaxis:

```
Public Function IsInRole(sRole As String) As Boolean
End Function
```

Parámetros:

sRole: cadena que contiene un papel

Ejemplo:

```
Dim bIsInRoleTaskLibrarian As Boolean
bIsInRoleTaskLibrarian = objArch.UserMan.IsInRole("TA")
```

SecurityMan

Las siguientes son API situadas en la interfaz del objeto 200 de arquitectura llamado SecurityMan 514.

EvalClaimRules;

EvalFileNoteRules;

EvalFormsCorrRules;

EvalOrgRules;

EvalRunApplicationRules;

EvalRunEventProcRules;

EvalTaskTemplateRules;

EvaIUserProfilesRules;

IsOperAuthorized;

GetUserId; y

OverrideUser.

EvalClaimRules()

Esta API hace referencia a normas de negocio para comprobación de seguridad de la reclamación y devuelve una regla booleana si se cumplen las normas.

Sintaxis:

```
Private Function EvalClaimRules(lBasicOp As cmBasicOperations, vContextData As Variant) As
Boolean
End Function
```

Parámetros:

lBasicOp: una operación básica que el usuario actual está deseando realizar (por ejemplo, Borrar)

vContextData: una matriz variante que contiene objetos de negocio relevantes u otra información.

Ejemplo:

```
Select Case lOperation
Case cmWorkOnClaim
            IsOperAuthorized = EvalClaimRules(cmView, vContextData) And _
                               EvalClaimRules(cmEdit, vContextData)
```

EvalFileNoteRules()

Esta API se refiere a normas de negocio para comprobación de seguridad de FileNote y devuelve una regla booleana si se cumplen las normas.

Sintaxis:

```
Private Function EvalFileNoteRules(lBasicOp As cmBasicOperations, vContextData As Variant) As
Boolean
End Function
```

Parámetros:

lBasicOp: una operación básica que el usuario actual está deseando realizar (por ejemplo, Borrar)

vContextData: una matriz variante que contiene objetos de negocio relevantes u otra información.

Ejemplo:

```
Select Case lOperation
Case cmDeleteFileNote
            IsOperAuthorized = EvalFileNoteRules(cmDelete, vContextData)
```

EvalFormsCorrRules()

Esta API se refiere a normas de negocio para comprobación de seguridad de formularios y Corr y devuelve una regla booleana si se cumplen las normas.

Sintaxis:

```
Private Function EvalFormsCorrRules(lBasicOp As cmBasicOperations) As Boolean
End Function
```

Parámetros:

lBasicOp: una operación básica que el usuario actual está deseando realizar (por ejemplo, Borrar)

Ejemplo:

```
Select Case lOperation
Case cmMaintainFormsCorr
            IsOperAuthorized = EvalFormsCorrRules(cmEdit) And _
                               EvalFormsCorrRules(cmDelete) And _
                               EvalFormsCorrRules(cmAdd)
```

EvalOrgRules()

Esta API se refiere a normas de negocio para comprobación de seguridad del procesador de eventos y devuelve una regla booleana si se cumplen las normas.

Sintaxis:

```
Private Function EvalOrgRules(lBasicOp As cmBasicOperations) As Boolean
End Function
```

Parámetros:

lBasicOp: una operación básica que el usuario actual está deseando realizar (por ejemplo, Borrar)

Ejemplo:

```
Select Case lOperation
Case cmMaintainOrg
             IsOperAuthorized = EvalOrgRules(cmAdd) And _
                                EvalOrgRules(cmEdit) And _
                                    --

                        EvalOrgRules(cmDelete)
```

EvalRunApplicationRules()

Esta API se refiere a normas de negocio para ejecutar la aplicación y devuelve una regla booleana si se cumplen las normas.

Sintaxis:

```
Private Function EvalRunApplicationRules(lBasicOp As cmBasicOperations) As Boolean
End Function
```

Parámetros:

lBasicOp: una operación básica que el usuario actual está deseando realizar (por ejemplo, Borrar)

Ejemplo:

```
Select Case lOperation
        Case cmRunApplication
            IsOperAuthorized = EvalRunApplicationRules(cmExecute)
```

EvalRunEventProcRules()

Esta API se refiere a normas de negocio para comprobación de seguridad del procesador de eventos y devuelve una regla booleana si se cumplen las normas.

Sintaxis:

```
Private Function EvalRunEventProcRules(lBasicOp As cmBasicOperations) As Boolean
End Function
```

Parámetros:

lBasicOp: una operación básica que el usuario actual está deseando realizar (por ejemplo, Borrar)

Ejemplo:

```
Select Case lOperation
Case cmRunEventProcessor
            IsOperAuthorized = EvalRunEventProcRules(cmExecute)
```

EvalTaskTemplateRules()

Esta API se refiere a normas de negocio para comprobación de seguridad plantillas de tareas y devuelve una regla booleana si se cumplen las normas.

Sintaxis:

```
Private Function EvalTaskTemplateRules(lBasicOp As cmBasicOperations) As Boolean
End Function
```

Parámetros:

lBasicOp: una operación básica que el usuario actual está deseando realizar (por ejemplo, Borrar)

Ejemplo:

```
Select Case lOperation
Case cmMaintainTaskLibrary
            IsOperAuthorized = EvalTaskTemplateRules(cmAdd) And _
                               EvalTaskTemplateRules(cmEdit) And _
                               EvalTaskTemplateRules(cmDelete)
```

EvalUserProfilesRules()

Esta API se refiere a normas de negocio para comprobación de seguridad de plantillas de tareas y devuelve una regla booleana si se cumplen las normas.

Sintaxis:

```
Private Function EvalUserProfileRules(lBasicOp As cmBasicOperations, vContextData As Variant)
As Boolean
End Function
```

Parámetros:

lBasicOp: una operación básica que el usuario actual está deseando realizar (por ejemplo, Borrar)

vContextData: una matriz variante que contiene objetos de negocio relevantes u otra información.

Ejemplo:

```
Select Case lOperation
Case cmIsRelativeOf
            IsOperAuthorized = EvalUserProfileRules(cmView, vContextData) And _
                               EvalUserProfileRules(cmAdd, vContextData) And _
                               EvalUserProfileRules(cmEdit, vContextData) And _
                               EvalUserProfileRules(cmDelete, vContextData)
```

GetUserId()

Devuelve el identificador del nombre/usuario de conexión del usuario actual.

Sintaxis:

```
Public Function GetUserId() As String
End Function
```

Ejemplo:

```
Dim sUserId as String
sUserId = GetUserId
```

IsOperAuthorized()

Esta API se refiere a normas de negocio y devuelve una regla booleana que determina si el usuario tiene privilegios de seguridad para realizar una determinada operación.

Sintaxis:

```
Public Function IsOperAuthorized(vMsg, as variant, nOperation as cmOperations, vContext As
Variant) As Boolean
End Function
```

Parámetros:

vMsg: mensaje de arquitectura estándar

nOperation: una enumeración que contiene el nombre de operación que ha de comprobarse.

vContext: una matriz variante que contiene objetos de negocio relevantes u otra información.

Ejemplo:

```
Dim bCanIDoThis As Boolean
bCanIDoThis = objArch.SecurityMan.IsOperAuthorized(vMsg,aOperationName, vContext)
TlbEditIcon.Enabled = bCanIDoThis
```

OverrideUser()

Reinicia para un usuario diferente.

Sintaxis:

```
Public Sub OverrideUser(Optional sUserId As String, Optional dictRoles As CDictionary,
Optional dictSubs As CDictionary)
End Function
```

Parámetros:

```
sUserId:
dictRoles:
dictSubs:
```

Ejemplo:

```
Dim x As New CTechArch
    x.SecurityMan.OverrideUser "Everyone", New CDictionary, New CDictionary
```

Marco de códigos

Requisitos básicos

Se crean tablas separadas (CodesDecodes) para almacenar los valores estáticos.

Sólo las referencias a codes/decodes se almacenan en las tablas de negocio (por ejemplo, tarea) que utilizan estos valores. Esto minimiza el tamaño de las tablas de negocio ya que almacenar un valor de código ocupa mucho menos espacio de almacenamiento que su valor Decode correspondiente (por ejemplo, para estado, “AL” se almacena en cada fila de la tabla en lugar de la cadena “Alabama”).

Los CodeDecodes se almacenan localmente en el equipo de trabajo del cliente en una DBMS local. Al arrancar la aplicación, se realiza un procedimiento para garantizar que las tablas locales están en sincronía con la DBMS central.

Enfoque de la infraestructura

El enfoque de la infraestructura 600 Code Decode de la presente invención indica el método de modelación física de tablas de códigos. El modelo permite que los códigos se extiendan sin impacto en el modelo y/o aplicación y arquitectura de datos físicos. La figura 6 muestra la disposición física de las tablas CodeDecode según una realización de la presente invención.

Infraestructura

El modelo físico de la infraestructura 600 CodeDecode hace lo siguiente:

Soporta la funcionalidad relacional entre objetos CodeDecode;

Soporta la extensibilidad sin modificación en la DBMS o en la arquitectura de aplicación;

Proporciona un enfoque consistente para acceder a todos los elementos CodeDecode; y

es fácil de mantener.

Estas tablas genéricas son capaces de tratar nuevas categorías, y modificar relaciones sin necesidad de cambiar la DBMS o la arquitectura de aplicación CodeDecode.

Los beneficios de este modelo son la extensibilidad y la mantenibilidad. Este modelo permite modificaciones de categorías de código sin impacto en la DBMS o en el código de la arquitectura de aplicación. Este modelo también requiere menos tablas que mantener. Además, sólo es necesario un método para acceder a CodeDecodes.

Relaciones de tabla y descripciones de campo:

(pk, primary key) indica una clave principal

Code_Category 602 (Categoría 602 de código)

- C_Category (pk): el número de categoría para un grupo de códigos
- C_Cache (actualmente no utilizado): puede indicar si la categoría debería almacenarse en caché en memoria en el equipo del cliente
- T_Category: una descripción de texto de la categoría (por ejemplo, tipos de tarea de aplicación, estado de la demanda, días de la semana).
- D_Last_Update: la fecha en que se actualizó por última vez cualquier dato en la categoría dada; este campo se usa para determinar si actualizar una categoría o categorías en la base de datos local

Relaciones

- Una relación uno-a-muchos con el Código de tabla (es decir, una categoría puede tener múltiples códigos)

Code 604 (Código 604)

- C_Category (pk): el número de categoría para un grupo de códigos
- C_Code (pk): un breve identificador de código (hasta diez caracteres; la longitud máxima actual usada es cinco caracteres)
- D_Effective: un campo de fecha que indica la fecha efectiva del código
- D_Expiration: un campo de fecha que indica la fecha de expiración del código (por defecto es 1 de enero de 2999)

Relaciones

- Una relación muchos-a-uno con la Code_Category 602 (descrita anteriormente)
- Una relación uno-a-muchos con las Code_Relations 606 (relaciones 606 de código) (una combinación categoría-y-código dada puede estar relacionada con múltiples otras combinaciones categoría-y-código)

Code_Relations 606 (Relaciones 606 de código)

- C_Category1 (pk): la primera categoría
- C_Code1 (pk): el primer código
- C_Category 2 (pk): la categoría relacionada
- C_Code 2 (pk): el código relacionado

Relaciones

- Una relación muchos-a-uno con la tabla de códigos (cada categoría y código en la tabla de códigos puede tener múltiples combinaciones categoría-código relacionadas)

Code_Decode 608

- C_Category (pk): el número de categoría para un grupo de códigos
- C_Code (pk): un breve identificador de código (hasta diez caracteres; la longitud máxima actual que se está empleando es de cinco caracteres)
- N_Lang_ID (pk): un valor que indica el ajuste de idioma local (tal como se define en un ajuste regional del equipo). Por ejemplo, el valor para inglés (Estados Unidos) se almacena como 0409. El uso de este ajuste permite el almacenamiento y selección de descripciones de código de texto basadas en el idioma escogido.
- T_Short_Desc: una descripción textual abreviada de C_Code
- T_Long_Desc: una descripción textual completa de C_Code, lo que el usuario realmente ve (por ejemplo, cerrar suplemento - recuperar, archivar nota, compensación de trabajadores).

Enfoque del soporte de localización

Habilitar localización

Los códigos soportan múltiples idiomas. La clave de esta característica es almacenar un identificador de idioma junto con cada valor CodeDecode. Este campo de idioma forma parte de la clave compuesta de la tabla Code_Decode. Cada búsqueda de API de código incluye una llamada de nivel de sistema para recuperar la variable de sistema de idioma. Este valor se usa como parte de la llamada para recuperar los valores con un idioma válido dado.

Mantener ajuste de localización de idioma

Un enlace a la variable de entorno de sistema Idioma a las claves de idioma se almacena en cada CodeDecode. Este valor se modifica en cualquier momento por el usuario simplemente editando la interfaz de usuario de ajustes regionales disponible en la carpeta Panel de Control de Microsoft Windows.

Enfoque de la expiración de códigos

Tratar códigos sensibles al tiempo es un aspecto importante al rellenar controles con una lista de valores. Un objetivo es permitir sólo al usuario ver y seleccionar entradas adecuadas. El desafío reside en ser capaz de expirar códigos sin afectar negativamente a la aplicación. Para lograr esto, se tiene en cuenta cómo cada IU decide qué valores es adecuado mostrar al usuario dado su modo actual.

Los tres modos IU más comunes que afectan a códigos sensibles al tiempo son el modo añadir, el modo ver y el modo editar.

Modo añadir

En el modo añadir, normalmente sólo se mostrarán al usuario códigos válidos como opciones de selección. Debe observarse que la constante, cmValidCodes, está por defecto y seguirá funcionando igual cuando se omita este parámetro opcional.

Set constates = objArch.codesMan.FillControl(ftmCurrentForm.cboStates, cmCatStates, cmLongDecode, cmValidCodes).

Modo ver

En el modo ver, normalmente el usuario está viendo los resultados de datos históricos sin capacidad directa de editarlos. La edición de datos históricos seleccionados lanza otra IU. Con esto, los controles se rellenan con códigos válidos y expirados, o en otras palabras, códigos no pendientes.

Set colStates = objArch.CodesMan.FillControl(frmCurrentForm.cboStates, cmCatStates, cmLongDecode, cmNonPendingCodes).

Modo editar

En el modo editar, se permiten cambios para códigos válidos pero también se muestran códigos expirados si ya están asignados a una entidad.

Dim colAssignedCodes As New cCollection

colAssignedCodes.Add HistoricalAddress.State

Set colStates = objArch.CodesMan.FillControl(frmCurrentForm.cboStates, cmCatStates, cmLongDecode, cmValidCodes,colAssignedCodes).

Actualizar CodeDecodes locales

Las tablas CodeDecode locales se mantienen en sincronía con el almacenamiento central de CodeDecodes. La arquitectura es responsable de realizar una comprobación para ver si hay algún Code decode nuevo o actualizado desde el servidor de manera regular.

La arquitectura, al detectar categorías CodeDecode nuevas o modificadas, también devuelve los datos asociados y realiza una actualización de la base de datos local. La figura 7 es un diagrama lógico de este proceso 700.

Tras una llamada API, se realiza una comprobación para determinar si la arquitectura se ha inicializado 702. Si es así, se hace una comprobación para determinar si el intervalo de renovación ha expirado 704. Si el intervalo de renovación no ha expirado, la llamada API está completa 706. Sin embargo, si la arquitectura no se ha inicializado o si el intervalo de renovación ha expirado, entonces los campos "LastUpdate" (última actualización) para cada categoría se leen desde el CodeDecode y se pasan al servidor 708. Entonces se leen nuevas categorías actualizadas desde la base 710 de datos. Finalmente, se actualiza la base 712 de datos local.

API de acceso a código

Las siguientes son API situadas en la interfaz del objeto 200 de arquitectura llamado CodesMan 500.

GetCodeObjetct(nCategory, sCode);

GetCategoryCodes(nCategory);

FillControl(ctlControl, nCategory, nFillType, [nCodeStatus], [colAssignedCodes]).

GetCodeObjetct: devuelve un objeto CCode dados una categoría específica y un código.

Sintaxis:

GetCodeObject(nCategory, sCode)

Parámetros:

nCategory: la constante basada en números enteros que clasificó estos CodeDecodes a partir de otros.

sCode: Una cadena que indica el atributo de código del objeto CodeDecode.

Ejemplo:

```
frmCurrentForm.lblState = objArch.CodesMan.GetCodeObject (cmCatStates, "IL").LongDecode
```

GetCategoryCodes: devuelve una serie de objetos CCode dada una categoría válida.

Sintaxis:

GetCategoryCodes(nCategory)

Parámetros:

nCategory: la constante basada en números enteros que clasificó estos CodeDecodes a partir de otros.

Ejemplo:

```
Dim colMyStates As CCollection
Set colMyStates = objArch.CodesMan.GetCategory(cmCatStates)
```

FillControl: Esta API se usa para rellenar cuadros desplegables o conjuntos de cuadros con valores de una lista de CodeDecodes. Devuelve una serie de búsquedas siguientes a los objetos de código para rellenar controles.

Sintaxis:

```
FillControl(ctlControl, nCategory, nFillType, [nCodeStatus], [colAssignedCodes])
```

Parámetros:

CtlControl: una referencia a un cuadro desplegable o conjunto de cuadros pasados.

nCategory: la constante basada en números enteros que clasificó estos CodeDecodes a partir de otros.

nFillType: el atributo del CodeDecode que se quiere rellenar. Los valores válidos incluyen:

```
cmCode
cmShortDecode
cmLongDecode
```

nCodeStatus: valor opcional que filtra los Code Decodes según sus fechas de efectividad y de expiración. Las constantes válidas incluyen las siguientes:

cmAllCodes	Códigos pendientes + válidos + expirados
cmPendingCodes	Códigos cuya fecha efectiva es superior a la fecha actual
cmValidCodes	Códigos no pendientes o expirados
cmExpiredCodes	Códigos cuya fecha expirada es superior a la fecha actual
cmNonPendingCodes	Códigos válidos + expirados
cmNonValidCodes	Códigos pendientes + expirados
cmNonExpiredCodes	Códigos pendientes + válidos
colAssignedCodes:	se usan al rellenar un control que debe rellenar e incluir valores asignados.

Ejemplo:

'Declare an instance variable for States collection on object Private colstates As CCollection

'Call FillControl API, and set local collection inst var to collection of codes which were used to fill the control.

Esta serie se usará para búsquedas posteriores.

```
Set colStates = objArch.CodesMan.FillControl(frmCurrentForm.cboStates, cmCatStates,
cmLongDecode)

'Below shows an example of looking up the Code value for the currently selected state.
With frmCurrentForm.cboStates
   If .ListIndex > -1 Then
      Dim objCode As CCode
      Set objCode =  colStates(.ItemData(.ListIndex))
      sStateCode = objCode.Code
   End If
End With
```

API de acceso a códigos relacionales

Los objetos de código devueltos a través de las API "GetCodeObject" o "GetCategoryCodes" pueden tener relaciones con otros objetos de código. Esto permite la funcionalidad en la que los códigos están asociados a otros objetos de código individuales.

Las API usadas para recuperar esos valores son similares a las de la interfaz CodesMan. La diferencia, sin embargo, es que los métodos se llaman en el objeto de código en lugar de en la interfaz CodesManager: a continuación se vuelven a enumerar las API.

GetCodeObject(nCategory, sCode);_

GetCategoryCodes(nCategory);

FillControl(ctlControl, nCategory, nFillType, [nCodeStatus], [colAssignedCodes]).

A continuación se dan algunas muestras de código para ilustrar cómo estas API también se llaman en objetos de código.

Ejemplo de GetCodeObject:

```
Dim objBondCode As CCode
Set objBondCode = objArch.CodesMan.GetCodeObject(cmCatLOB, "B")
Dim objSuretyCode As CCode
Set objSuretyCode = objBondCode.GetCodeObject(cmCatSupplement, "B01")
```

Ejemplo de GetCategory:

```
Dim objBondCode As CCode
Set objBondCode = objArch.CodesMan.GetCodeObject(cmCatLOB, "B")
Dim colSupplements As CCollection
Set colSupplements = objBondCode.GetCategory(cmCatSupplement)
```

Ejemplo de FillControl:

```
Dim objBondCode As CCode
Set objBondCode = objArch.CodesMan.GetCodeObject(cmCatLOB, "B")
Dim colSupplements As CCollection
Set colSupplements = objBondCode.FillControl(frmForm.cboSupplements, cmCatSupplements,
    cmLongDecode)
```

Registro cronológico de mensajes

La arquitectura de registro cronológico de mensajes permite el registro cronológico de mensajes de una manera segura y consistente. La interfaz para el componente de registro cronológico de mensajes es simple y consistente, permitiendo el registro cronológico de mensaje en cualquier capa de procesamiento. Se registran cronológicamente tanto mensajes de error como de información en un almacén centralizado.

Mejores prácticas

Los mensajes se registran cronológicamente siempre por la arquitectura cuando se produce un error irrecuperable (es decir, la red se cae) y no se trata de manera explícita. El registro cronológico de mensajes puede usarse según se necesite para facilitar el diagnóstico y el arreglo de SIR. Esta clase de registro cronológico es esencialmente útil en puntos de integración entre clases y componentes. Los mensajes registrados cronológicamente con la finalidad de depurar fallos (“debug”) tienen una gravedad informativa, para no ser confundidos con mensajes de error verdaderos.

Uso

Un mensaje se registra cronológicamente llamando a la función LogMessage() en la arquitectura.

Descripción de parámetros:

vMsg: el mensaje de arquitectura estándar

ISeverity: la gravedad del mensaje

sClassName: el nombre de la clase que registra cronológicamente el mensaje

sMethodName: el nombre del método que registra cronológicamente el mensaje

sVersion: la versión del archivo binario (EXE o DLL) que contiene el método que registra cronológicamente el mensaje

lErrorNum: el número de error actual

sText: un parámetro opcional que contiene el texto del mensaje. En caso de omitirse, se buscará el texto en un archivo de cadena o bien se usará la descripción de error VB genérica

sText: un parámetro opcional que contiene el texto del mensaje. En caso de omitirse, se buscará el texto en un archivo de cadena o bien se usará la descripción de error VB genérica

1LoggingOptions: un parámetro opcional que contiene una constante que especifica dónde registrar cronológicamente el mensaje (es decir, el paso de cmLogToDBAndEventViewer a LogMessage registrará cronológicamente el error en la base de datos y el visor de eventos).

Niveles de registro cronológico

Antes de registrar cronológicamente un mensaje, se compara su gravedad al nivel de registro cronológico del equipo actual. Si la gravedad del mensaje es menor a o igual que el nivel de registro cronológico, entonces se registra cronológico el mensaje.

Los valores válidos para el nivel de registro cronológico se definen como una enumeración en VB. Incluyen:

Valor	Nombre	Descripción	Ejemplo
0	CmFatal	Una condición crítica que cierra o amenaza todo el sistema	Rotura del servidor de aplicación
1	CmSevere	Una condición que cierra o amenaza un componente importante de todo el sistema	Fallo de red
2	CmWarning	Un aviso de que algo en el sistema va mal pero no cierra ni amenaza con cerrar el sistema	Error de bloqueo optimista
3	CmInformation al	Notificación de un suceso particular con fines de registro cronológico e informe	Información de depuración de fallos al desarrollador

Ejemplo

```
If Err.Number <> 0 Then
    ' log message
    Arch.LogMan.LogMessage(vMsg, cmSeverityFatal, "COrganizationCTLR", "InitForm",
                           GetVersion(), Err.Number, Err.Description)
    ' re-raise the error
    Err.Raise Err.Number
End If
```

Registro cronológico de base de datos

Nombre del campo	Descripción
N MSG ID	ID única del mensaje
D MSG	Fecha en que tuvo lugar el mensaje
C ERR SEV	Gravedad del error
N USER ID	Nombre del usuario cuando se produjo el error
N MACH ID	Nombre del equipo en el que se produjo el error
M CLASS	Nombre de la clase en la que se produjo el error
M METHOD	Nombre del método en el se produjo el error
N CMPNT VER	Versión del archivo binario en que se produjo el error
C ERR	Número del error
T MSG	Texto del mensaje

Registro cronológico local

Los mensajes se registran cronológicamente siempre en el Event Log (registro cronológico de eventos) del servidor de la aplicación; sin embargo, esto no es necesariamente cierto para la base de datos, tal como indica el parámetro opcional pasado al LogMessage, ILoggingOptions (opciones de registro cronológico). Un administrador con los derechos de acceso adecuados puede conectarse al servidor de la aplicación MTS a distancia y ver su Event Log. Sólo un paquete MTS contiene el componente de Event Log, de modo que los errores se escribirán todos en el mismo Event Log del servidor de aplicación.

Los eventos registrados cronológicamente a través de Visual Basic siempre tienen "VBRuntime" (tiempo de ejecución VB) como fuente. El campo Ordenador se ocupa automáticamente con el nombre del ordenador que está registrando cronológicamente el evento (es decir, el servidor de la aplicación MTS) en lugar de con el ordenador que generó el evento (normalmente un ordenador de cliente).

Los mismos detalles del evento que están escritos en la base de datos se formatean en una cadena legible y se escriben en el registro cronológico. Se añade automáticamente el texto "la aplicación VB identificada por ... registrada cronológicamente" por VB; el texto que sigue contiene los detalles del mensaje.

Acceso a datos

Todos salvo algunos casos excepcionales utilizan al API "ExecuteQuery" (ejecutar pregunta). Esta API cubre operaciones de bases de datos singulares en las que existe una única entrada y una única salida. Esencialmente sólo debería excluir ciertos grupos de tipos de operaciones.

El marco de acceso a datos sirve para la finalidad de rendimiento, consistencia y mantenibilidad.

Rendimiento

El método de "Execute Query" incorpora patrones de uso para usar ADO de una manera eficiente. Ejemplos de estos patrones incluyen el uso de conjuntos de registros desconectados, y la declaración explícita de parámetros opcionales que dan como resultado el mejor rendimiento.

Consistencia

Este método proporciona una interfaz común para el desarrollo de acceso a datos. Con una interfaz de acceso a datos sencilla y estable dada, pueden desarrollarse y difundirse las mejores prácticas.

Mantenibilidad

Puesto que el método está situado en una única localización, está muy modularizado y puede mantenerse con un pequeño impacto sobre sus llamantes.

Los servidores de aplicación a menudo utilizan la interfaz de acceso a datos de objetos ActiveX Data (ADO). Esto permite un modelo de programación simplificado así como la habilitación de las realizaciones para usar una variedad de fuentes de datos.

El método "ExecuteQuery"

Visión general

El método "ExecuteQuery" debería usarse para la mayoría de llamadas SQL de la aplicación. Este método encapsula la funcionalidad para usar ADO de una manera efectiva y eficiente. Esta API se aplica a situaciones en las que debe ejecutarse una única operación que devuelve un único objeto de conjunto de registros.

Sintaxis:

```
Set obj =  ExecuteQuery(vMsg, nTranType, sSQL, [nMaxRows], [adoTransConn], [args])
```

Parámetros:

vMsg

Este parámetro es la estructura TechArch. Se usa como una señal (token) para capturar información tal como la métrica de rendimiento, información de errores y seguridad.

nTranType

Una constante definida por la aplicación que indica qué tipo de operación se está llevando a cabo. Los valores para este parámetro pueden ser uno de las siguientes constantes:

cmselect

cmSelectLocal

cmUpdate

cmInsert

cmDelete

sSQL

Cadena que contiene el código SQL para realizarse contra la DBMS.

nMaxRows (opcional)

Valor de número entero que representa el número máximo de registros que devolverá el conjunto de registros de la pregunta actual.

adoTransConn (opcional)

Un objeto de conexión ADO. Se crea y se pasa en la pregunta de ejecución para operaciones que requieren un control de transacciones ADO (véase la sección de “Uso de transacciones”).

args (opcional)

Una lista de parámetros que deben insertarse de manera respetuosa en la instrucción SQL.

Implementación

En una realización de la presente invención el método “ExecuteQuery” reside en el archivo MservArch.bas. Este archivo debería incorporarse en todos los proyectos de tipo ServerComponente (Componente de Servidor). Esto permitirá que cada componente de servidor acceda a este método.

Obsérvese: puesto que este método es un método público en un módulo “bas”, está disponible globalmente desde cualquier punto en el proyecto.

```
Public Function ExecuteQuery(vMsg As Variant, _
                             nTranType As TranTypes, _
                             sSQL As String, _
                             Optional nMaxRows As Integer = 0, _
                             Optional adoTransConn As ADODB.Connection, _
                             Optional colArguments As CCollection) As Variant
    On Error GoTo ErrorHandler
```

```
Const cmMethodName As String = "ExecuteQuery"

StartTimeLogger vMsg, cmTimerIdDBTotal, cmClassName, cmMethodName

'find out if this call is an isolate operation or
'part of an ADO (not MTS) transaction
Dim isAtomicTrans As Boolean
isAtomicTrans = adoTransConn Is Nothing

Dim nRecordsAffected As Integer
Dim adoRS As New ADODB.Recordset
Dim adoConn As ADODB.Connection
Dim lAuxErrNumber As Long

'open a new connection or keep using the passed in connection
Set adoConn = IIf(isAtomicTrans, New ADODB.Connection, adoTransConn)

If isAtomicTrans Then
    adoConn.Open cmODBC_Connect

    'ADO will wait indefinitely until the execution is complete during performance
     testing
    #If IsPerfTest Then
        adoConn.CommandTimeout = 0
    #End If
End If
'Make sure date args are formatted for DB2 if appropriate
If Not colArguments Is Nothing Then _
    Set colArguments = FormatArgsForDB2(colArguments)

'merge the passed in arguments with the SQL string
sSQL = MergeSQL(sSQL, colArguments)

Debug.Print Time & ": " & sSQL

'execute the SQL statement depending on the transaction type
Select Case CStr(nTranType)
    Case cmSelect
        adoRS.MaxRecords = nMaxRows
        adoRS.CursorLocation = adUseClient
        adoRS.Open sSQL, adoConn, adOpenForwardOnly, adLockReadOnly, adCmdText
        Set adoRS.ActiveConnection = Nothing
        Set ExecuteQuery = adoRS
    Case cmSelectLocal
        adoRS.MaxRecords = nMaxRows
        adoRS.CursorLocation = adUseClient
        adoRS.Open sSQL, adoConn, adOpenStatic, adLockBatchOptimistic, adCmdText
        Set adoRS.ActiveConnection = Nothing
        Set ExecuteQuery = adoRS
    Case cmInsert
        Set adoRS = adoConn.Execute(sSQL, nRecordsAffected, adCmdText)
        If nRecordsAffected <= 0 Then Err.Raise cmErrQueryInsert
        Set adoRS = Nothing
        ExecuteQuery = nRecordsAffected
    Case cmUpdate, cmDelete
        Set adoRS = adoConn.Execute(sSQL, nRecordsAffected, adCmdText)
        If nRecordsAffected <= 0 Then Err.Raise cmErrOptimisticLock
```

```
                Set adoRS = Nothing
                ExecuteQuery = nRecordsAffected
            Case cmSpFileNote
                Set adoRS = adoConn.Execute(sSQL, nRecordsAffected, adCmdText)
                Set adoRS = Nothing
            Case Else
                Err.Raise cmErrInvalidParameters
        End Select

        StopTimeLogger vMsg, cmTimerIdDBTotal, cmClassName, cmMethodName
        Exit Function

ErrorHandler:
    Dim objArch As Object
    Set objArch = CreateObject("cmArch.CTechArch")

    Select Case CStr(Err)
        Case cmErrQueryInsert, cmErrOptimisticLock, cmErrInvalidParameters
            'Raise error
            Err.Raise Err

        Case cmErrDSNNotFound
            Dim sMsgText As String
            sMsgText = "Data Source Name not found." & vbCrLf & "( " & _
                       CStr(objArch.RegMan.GetServerDSN) & " )"

            ' Create a new message log and log the message
            objArch.LogMan.LogMessage vMsg, cmSeverityFatal, cmClassName, cmMethodName, _
                         GetVersion(), cmErrDSNNotFound, sMsgText, cmLogToEventViewerOnly


            lAuxErrNumber = adoConn.Errors(0).NativeError 'The error code is stored since
                                                          when closing the conection it will
                                                          be lost
            If adoConn.State <> adStateClosed Then adoConn.Close

            Err.Raise cmErrDSNNotFound, , sMsgText
        Case Else
            ' Create a new message log and log the message
            objArch.LogMan.LogMessage vMsg, cmSeverityFatal, cmClassName, cmMethodName, _
                          GetVersion(), Err.Number, Err.Description, cmLogToEventViewerOnly

            lAuxErrNumber = adoConn.Errors(0).NativeError 'The error code is stored since
                                                          when closing the conection it will
                                                          be lost
            If adoConn.State <> adStateClosed Then adoConn.Close
            Err.Raise lAuxErrNumber
    End Select
End Function
```

Selección de registros

ExecuteQuery utiliza conjuntos de registros desconectados para instrucciones de tipo “seleccionar”. Esto requiere que los clientes, particularmente los ACC, contengan una referencia del ADOR, conjunto de registros de objeto ActiveX. Esta DLL es un subconjunto del ADODB DLL. ADOR contiene sólo el objeto de conjunto de registros.

El uso de conjuntos de registros desconectados permite clasificar objetos de conjunto de registros desde el servidor al cliente. Esto tiene un rendimiento mucho más eficiente que la matriz variante que se asocia al uso de API “GetRows” (obtener filas) en el servidor. Este aumento del rendimiento es especialmente evidente cuando el servidor de la aplicación está cargado con un gran número de usuarios concurrentes.

Muestra de adaptador de componente de cliente (ACC)

```
Dim vAns as Variant
Dim adoRS As ADOR.Recordset
Set adoRS = objServer.PerformSelect(vMsg, nId)
If objRS.EOF Then
    Set objRS = Nothing
    Exit Function
End If

vAns = adoRS.GetRows
Set adoRS = Nothing

'Marshall vAns into objects
...
```

Muestra de componente de servidor

```
Private Const cmCustSQL = "Select * from Customer where id = ?"

Public Function PerformSelect(vMsg, nId) as Variant
    Dim colArgs as CCollection
    Set colArgs = New Ccollection
    colArgs.Add nId
    Set PerformSelect = ExecuteQuery(vMsg, cmSelect, sCustSQL, , ,colArgs)
End Function
```

Recorte de código de ExecuteQuery (sección de selección)

```
Case cmSelect
    adoRS.MaxRecords = nMaxRows
    adoRS.CursorLocation = adUseClient
    adoRS.Open sSQL, adoConn, adOpenForwardOnly, adLockReadOnly, adCmdText
    Set ExecuteQuery = adoRS
...
```

Inserción de registros

La inserción de registros requiere cierta información que pertenece al bloqueo optimista. En el servidor, se pide un valor único para indicar la vez que se modificó. Este valor único se devuelve al reclamante de modo que pueda usarse para posteriores operaciones en la base de datos.

Muestra de adaptador de componente de cliente (ACC)

```
Dim vNewTS as Variant
vNewTS = objServer.PerformInsert(vMsg, nId, sName)
'Set object's TimeStamp to vNewTS
```

Muestra del componente de servidor

```
Private Const cmCustInsertSQL = "Insert Customer (nId, Name, LastUpdated) Values(?, '?', ?)"

Public Function PerformInsert(vMsg, nId, sName) As Variant
Dim lCurrTS as Long
lCurrTS = GetTimeStamp
Dim colArgs as CCollection
Set colArgs = New Ccollection
colArgs.Add nId
colArgs.Add sName
colArgs.Add lCurrTS
ExecuteQuery(vMsg, cmInsert, sCustInsertSQL, , , colArgs)
PerformInsert = lCurrTS
```

Recorte de código de ExecuteQuery, (sección de inserción)

```
Case cmInsert
    Set adoRS = adoConn.Execute(sSQL, nRecordsAffected, adCmdText)
    If nRecordsAffected <= 0 Then Err.Raise cmErrQueryInsert
    Set adoRS = Nothing
    ExecuteQuery = nRecordsAffected
```

Actualización de registros

La actualización de registros requiere cierta información que pertenece al bloqueo optimista. En el servidor, se pide un valor único para indicar la última vez que se modificó. También se usa la última marca de fecha y hora para validar, durante la actualización, que el registro no se ha modificado desde la última vez que se leyó.

Muestra de adaptador de componente de cliente (ACC)

```
Dim vNewTS as Variant
vNewTS = objServer.PerformUpdate(vMsg, 1, 'Rick', 8907654)
'Set object's TimeStamp to vNewTS
```

Muestra del recorte de código del componente de servidor

```
Private Const cmCustUpdateSQL = _
"Update Customer Set Name = '?', LastUpdated = ? " & _
"Where Id = ? " & _
"And LastUpdated = ? "


Public Function PerformUpdate(vMsg, nId, sName, lLastTS) As Variant
    Dim lCurrTS as Long
    lCurrTS = GetTimeStamp
    Dim colArgs as CCollection
    Set colArgs = New Ccollection
    colArgs.Add sName
    colArgs.Add lCurrTS
    colArgs.Add nId
    colArgs.Add lLastTS
    PerformUpdate = ExecuteQuery(vMsg, cmUpdate, sCustUpdateSQL, , , colArgs)
    PerformUpdate = lCurrTS
End Function
```

Recorte de código de ExecuteQuery (sección de actualización)

```
Case cmUpdate
    Set adoRS = adoConn.Execute(sSQL, nRecordsAffected, adCmdText)
    If nRecordsAffected < 0 Then Err.Raise cmErrOptimisticLock
    ExecuteQuery = nRecordsAffected
```

Borrado de registros

En el borrado de registros se usa la última marca de fecha y hora para validar, durante el borrado, que el registro no se ha modificado desde la última vez que se leyó.

Muestra de adaptador de componente de cliente (ACC)

```
Dim vAns as Variant
vAns = objServer.PerformDelete(vMsg, nId ,lLastTS)
```

Muestra del componente de servidor

```
Private Const cmCustDeleteSQL = _
"Delete From Customer " & _
"Where Id = ? " & _
"And LastUpdated = ? "

Public Function PerformDelete(vMsg, nId lLastTS) As Variant
    Dim colArgs as CCollection
    Set colArgs = New Ccollection
    colArgs.Add nId
    colArgs.Add lLastTS
    PerformDelete = ExecuteQuery(vMsg, cmDelete, cmCustDeleteSQL)
Exit Function
```

Recorte de código de ExecuteQuery (sección de borrado)

```
Case cmDelete
    Set adoRS = adoConn.Execute(sSQL, nRecordsAffected, adCmdText)
    If nRecordsAffected < 0 Then Err.Raise cmErrOptimisticLock
    ExecuteQuery = nRecordsAffected
```

Marco de bloqueo de la base de datos

El bloqueo de la base de datos garantiza la integridad de la base de datos en un entorno multiusuario. El bloqueo evita el problema común de actualizaciones perdidas de múltiples usuarios que actualizan el mismo registro.

Opciones de solución

Bloqueo pesimista

Esta política de bloqueo permite al primer usuario a tener acceso total al registro mientras que a los siguientes usuarios se les deniega el acceso o tienen un acceso de sólo lectura hasta que el registro se desbloquea. Hay inconvenientes en este método de bloqueo. Es un método que es susceptible de estancarse en la base de datos así como a tener un rendimiento pobre cuando se encuentra con conflictos.

Bloqueo optimista

El enfoque optimista para bloquear un registro se basa en la suposición de que no es un procesamiento normal que múltiples usuarios lean y actualicen registros de forma concurrente. Esta situación se trata como un procesamiento excepcional en lugar de como un procesamiento normal. Los bloqueos no se sitúan en realidad en la base de datos en el momento de la lectura. Un mecanismo de marca de fecha y hora se usa en el momento de la actualización o el borrado para asegurar que otro usuario no ha modificado ni borrado el registro desde que se leyó por última vez el registro.

Una realización preferida de la presente invención utiliza un enfoque de bloqueo optimista para controlar la concurrencia. Esto garantiza la integridad de la base de datos así como la baja sobrecarga asociada con esta forma de bloqueo. Otros beneficios de este método son la mayor disponibilidad de los registros para múltiples usuarios y una minimización de los estancamientos en la base de datos.

Los candidatos de la tabla para el control de la concurrencia se identifican durante el “ejercicio de modelación de datos”. La única tabla que se actualiza de forma concurrente es el mecanismo de bloqueo optimista. Una vez identificados éstos, se añade lo siguiente a la aplicación.

Añadir el campo “N_Last_Updt” a la tabla en la base de datos;

Rutinas de tratamiento de errores en aquellas operaciones que modifican o borran desde esta tabla; y

Visualización/Notificación al usuario de que se ha producido el error.

Uso

El esquema siguiente describe los papeles de los dos tipos básicos de componentes para habilitar el bloqueo optimista.

Suposición: el campo de bloqueo optimista es de tipo Fecha y se llama "N_Last_Updt"

	Componentes de cliente	**Componentes de servidor**
Acceso a lectura	Almacenar el valor N_Last_Updt en el objeto de negocio para su uso en posibles actualizaciones o eliminaciones	Recuperar datos (siempre incluyendo el campo N_Last_Updt). Seleccionar Id, FirstName, N_Last_Updt DEL cliente siendo id = 10
Inserciones	Normal	Dim ICurrTS As Double ICurrTS = GetTimeStamp (obtener marca de fecha y hora) INSERTAR EN cliente (Id, FirstName, N_Last_Updt) VALORES (1, "Rick", ICurrTS); Devolver nueva marca de fecha y hora (ICurrTS) así como la nueva Id
Actualizaciones	Pasar marca de fecha y hora anteriormente leída para identificar si se ha modificado la fila. Esto es además de un identificador único y cualquier dato que deba actualizarse. Tratar la excepción si se ha modificado previamente el registro. Notificar al usuario el conflicto. Restaurar cualquier cambio.	Dim ICurrTS As Double 1CurrTS = GetTimeStamp ACTUALIZAR cliente ESTABLCER el nombre propio = "Richard" N_Last_Updt = ICurrTS siendo id = 1 Y LastUpdate = lastReadTimestamp (marcha de fecha y hora leída por última vez) Si no hay filas afectadas, tratar y propagar el error de vuelta al cliente. Devolver nueva marca de fecha y hora (ICurrTS)
Eliminaciones	Pasar marca de fecha y hora anteriormente leída para identificar si se ha modificado la fila. Esto es además de un	BORRAR cliente Siendo id = 1 Y N_Last_Updt = lastReadTimestamp; Si no hay filas afectadas, tratar y propagar el error de vuelta al cliente.

	identificador único Tratar la excepción si se ha modificado previamente el registro. Notificar al usuario el conflicto. Restaurar cualquier cambio.	

Gran conjunto de resultados

Al recuperar registros de una base de datos, si los criterios de búsqueda son demasiado amplios, la cantidad de datos que deben recuperarse de la base de datos y pasarse a través de la red afectará al rendimiento percibido por el usuario. Las ventanas que piden tales datos se pintarán con dificultad y las búsquedas serán lentas. La formación de las consultas a la base de datos se hace de tal forma que se recupera una cantidad manejable de datos. Hay algunas opciones para abordar los problemas que suceden por un gran conjunto de resultados. Las opciones se dan a continuación en orden de preferencia.

Volver a diseñar la interfaz/controlador para devolver conjuntos de resultados más pequeños. Al diseñar controladores que presenten las consultas a la base de datos de forma inteligente, las consultas que se presentan al servidor de la base de datos no devuelven un conjunto de resultados suficientemente grande para afectar el rendimiento percibido por el usuario. En esencia, el potencial para recuperar demasiados registros indica que las IU y los controladores se han diseñado de manera diferente. Un ejemplo de una IU de búsqueda bien diseñada es una en la que se pide al usuario que introduzca un criterio de búsqueda mínimo para evitar un conjunto de resultados excesivamente grande.

Tener conjuntos de resultados desplazables. La recuperación con desplazamiento de un gran conjunto de resultados es la recuperación incremental de un subconjunto de resultados repetida tantas veces como lo pida el usuario o hasta que se obtenga el conjunto de resultados total. Los resultados se recuperan mediante el enfoque Bounded Query (consulta limitada) en la que el primer registro se determina por una cláusula “where” (dónde) con valores calculados.

Requisitos de cliente para el conjunto de resultados desplazable

IU preferida

Las visualizaciones preferidas son como sigue:

Los resultados devueltos se visualizan en un cuadro desplegable de GreenTree;

Se proporciona un botón de acción con la etiqueta Más... para que el usuario obtenga los resultados restantes;

El botón Más se habilita cuando el usuario ha realizado una búsqueda inicial y todavía hay resultados que recuperar;

El botón Más se deshabilita cuando no quedan resultados que recuperar;

El cuadro desplegable y el botón de acción se incluyen en un grupo de cuadros para proporcionar una asociación visual entre el botón y el cuadro desplegable.

Consulta limitada

Las consultas que se implementan con el conjunto de resultados limitado se envían al servidor. El servidor implementa el método ExecuteQuery para recuperar los conjuntos de registros como de costumbre. Las consultas de resultados limitados tienen una orden por cláusula que incluye el orden de clasificación requerido por el negocio junto con un número suficiente de columnas para asegurar que todas las filas pueden identificarse de manera unívoca. El conjunto de registros está limitado por la variable nMaxRows (número máximo de filas) pasada desde el cliente incrementada para obtener la primera fila del siguiente conjunto de resultados. El resultado devuelto por el componente es un conjunto de registros al igual que con una consulta que no está limitada. El ACC 208 crea los objetos y los pasa de vuelta al controlador 206. El controlador 206 añade esta serie de resultados devueltos de objeto a su serie de objetos (una acumulación de los resultados anteriores) y al hacer esto realizará la comparación del último objeto con el primer objeto de la siguiente fila. Los valores necesarios para discriminar las dos filas se añaden a la matriz variante que es necesaria para pasar al componente para la siguiente consulta.

El controlador 206 en el cliente retiene los valores para nMaxRows, la instrucción SQL inicial, y la matriz de valores para discernir entre la última fila de la consulta anterior y la primera fila de la siguiente consulta. El mecanismo por el que el controlador 206 es consciente de que hay más registros que recuperar es comprobando que el número de resultados es uno mayor que el número máximo de filas. Para evitar la recuperación de registros más allá del final del archivo, el controlador 206 deshabilita estas funciones en la IU. Por ejemplo, un botón de control Más en la IU, usado para pedir los datos, se deshabilita cuando el número de objetos devueltos es menor que el nMaxRow +1.

Responsabilidad de la aplicación

Servidor

El componente de servidor es responsable de crear una serie de argumentos y adjuntar la instrucción SQL para añadir una cláusula dónde que podrá discriminar entre la última fila de la consulta anterior y al primera fila de la siguiente.

ACC

El ACC 208 procesa el conjunto de registros en objetos como en las consultas no limitadas. El ACC 208 reenvía la matriz variante pasada desde el controlador 206 para identificar los resultados limitados.

Controlador

El controlador 206 tiene la responsabilidad de deshabilitar el control Más cuando se haya llegado al final del archivo. El controlador 206 ocupa la matriz variante (vKeys) con los valores necesarios para determinar el inicio de la siguiente consulta.

Ejemplo

Un ACC 208 está codificado para una búsqueda definida por el usuario que tiene el potencial de devolver un conjunto de resultados considerable. El ejemplo de código siguiente implementa el enfoque de consulta limitada.

En el servidor, el desarrollador codifica la consulta como sigue:

```
Public Function RetrieveBusinessObjects(vMsg As Variant, ByVal sSql As String, ByVal nMaxRows As
Integer, Optional ByVal vKeys As Variant) As Recordset
    On Error GoTo ErrorHandler

    'Declare local constants
    Const cmMethodName As String = "RetrieveBusinessObjects"


    'Declare local variables
    Dim cmClassName As String
    Dim colArgs As New CCollection

    'initialize instance variables
    cmClassName = "CSRSTestComp"


    'fill argument collection
    Set colArgs = ArgumentsForBusinessObject(vKeys , sSQL)

    'increment nMaxRows to obtain row for comparison

    nMaxRows = nMaxRows + 1

    'ExecuteQuery
        Set RetrieveBusinessObjects = ExecuteQuery(vMsg, cmSelectLocal, sQuery, nMaxRows, ,
colArgs)


    'Tell MTS we're done
    GetObjectContext.SetComplete
    Exit Function

ErrorHandler:
    Select Case Err.Number

        Case Else
            Dim iResumeCode As Integer
            iResumeCode = GeneralErrorHandler(vMsg, cmServer, cmClassName, cmMethodName)
            Select Case iResumeCode
                Case cmErrorResume
                    Resume
                Case cmErrorResumeNext
                    Resume Next
                Case cmErrorExit
                    Exit Function
                Case Else



                                        GetObjectContext.SetAbort
                                        Err.Raise Err.Number
                                End Select
                        End Select
                End Function
```

Para determinar la cláusula dónde adicional necesaria para determinar el punto de inicio de la consulta, se añade siguiente método:

```
Private Function ArgumentsForBusinessObject(vKeys As Variant, sSql As string ) As CCollection
    Dim colArgs As Ccollection
    Const cmGreaterThanWhereString As String = " ? > ? "
    Const cmGreaterThanOrEqualWhereString As String = " ? >= ? AND "

    ' initialize local variables
    Set colArgs = New Ccollection
    sSql = sSql + "WHERE"


    With colArgs
        If vKeys(0) <> Empty Then
            .Add ("N_TASK_TEMPL_ID")
            .Add (vKeys(0))

        End If
        'If vKeys(1) <> Nothing Then
            '.Add value2 fieldName
            '.add vKeys(1)
            sSql = sSql + cmGreaterThanOrEqualWhereString
        'End If
        'If vKeys(2) <> Nothing Then
            '.Add value3 fieldName
            '.add vKeys(2)
            sSql = sSql + cmGreaterThanOrEqualWhereString
        'End If
    End With

    'finalize SQL statement
    sSql = sSql + cmGreaterThanWhereString

    Set ArgumentsForBusinessObject = colArgs
End Function
```

En el ACC 208, debe darse el permiso para el paso de las vKeys.

Función pública RetrieveBusinessObjects_(vMsg As Variant, sSql As String, nMaxRows As Integer, Optional ByVal vKeys As Variant) As CCollection

```
    Set percmpComponent = New CSRSTestComp
    Dim i As Integer

    Set adoRS = percmpComponent.RetrieveBusinessObjects(vMsg, sSql, nMaxRows, vKeys)


        'convert recordset to business objects
        adoRS.MoveFirst
        Do Until adoRS.EOF
            Call ConvertToBusinessObject
            adoRS.MoveNext
        Loop

        'return the collection of business objects
        Set RetrieveBusinessObjects = dictBusinessObject
      Set dictBusinessObject = New CCollection

    End Function
```

El controlador inicia la consulta y actualiza la matriz variante de claves y las propiedades del formulario 204 basándose en el resultado devuelto. Además del código mostrado para el ejemplo siguiente, se habilita el control Más si se vacía la búsqueda.

```
'declare instance variables
Private nMaxRows As Integer
Dim interimResults As CCollection
Dim vResults As CCollection
Dim vKeys(3) As Variant

'declare Constants
Private Const nDefaultAmount As Long = 50
Private Const cmRetrieveBusinessObjectSQL = "SELECT * FROM NODE_RULE ORDER BY _
N_TASK_TEMPL_ID"
```

Durante la inicialización de clase se realiza lo siguiente:

```
Public Sub Class_init()

    'obtain settings from registry
    nMaxRows = CInt(GetSetting(cmRegApp, cmRegArchSection, cmLimitedResultAmountKey,
lDefaultAmount))

    Call resetSearch
    Set objCCA = New {CCA class name}

End Sub
```

La funcionalidad de reiniciar la búsqueda se mantiene fuera de la inicialización de modo que esto puede llamarse desde otras partes de la aplicación.

```
Public Sub resetSearch()
    Dim I as Integer
        Set vResults = New Ccollection
        For I = 0 To 3
            Set vKeys(I)  = Empty
        Next
Set vKeys(0)  = Empty

        frmCurrentForm.cmdMore.Enabled = True

End Sub

Public Sub RetrieveBusinessObjects()
        Const cmMethodName As String = "retrieveBusinessObjects"

        Call RetainMouse

        ' get arch message
        Dim vMsg As Variant
        vMsg = objApp.objArch.AsMsgStruct()

        ' call the component
        Dim pair As CArchPair

       'Declare local variables
        Dim sSql As String
        Dim colArgs As CCollection
        Dim cmClassName As String

    Set interimResults = objCCA.RetrieveBusinessObjects(vMsg, cmRetrieveBusinessObjectSQL,
nMaxRows, vKeys)

    ctr = ProcessObjectCollection

    'stop if size of return is less than the maximum
        If ctr < nMaxRows + 1  Then frmCurrentForm.cmdMore.Enabled = False
        ' restore pointer
        Screen.MousePointer = lPrevPtr
    End Sub
```

Para retener los valores para discriminar entre la última fila del conjunto de resultados y la primera fila del siguiente se usa el siguiente método en el controlador:

```
Private Function ProcessObjectCollection() As Integer

' merge results with the instance variable for the collection
        Dim ctr As Integer
            ctr = 0
        For Each element In interimResults
            ctr = ctr + 1
            'retain Keys for subsequent Queries
            With element
            Select Case ctr
                Case nMaxRows
                     'store all values that may be used for row comparison
                     vKeys(0) = .NodeId

                      'add last object to collection
                     vResults.Add element

                Case nMaxRows + 1
                     'last object only used for comparison
                     'If the proceeding value can be used to uniquely
                     'identify row then delete value from array

                   ' THERE SHOULD BE N - 1 nested If statements where N = size of vKeys
                   'If .value2 <> vKeys(1) Then
                       'vKeys(2) = Empty
                      If .NodeId <> vKeys(0) Then vKeys(1) = Empty
                    'End If
               Case Else
                     vResults.Add element

            End Select
            End With
         Next
        ProcessObjectCollection = ctr
End Function
```

Operación de ejemplo con datos

Persona			
Nombre propio	Apellido	Estado	ID única
Joy	Andersen	Cerrado	22
Jay	Anderson	Abierto	12
John	Barleycorn	Cerrado	512
John	Barleycorn	Abierto	32
Esther	Davidson	Abierto	88
David	Dyson	Cerrado	98
Bobby	Halford	Abierto	234
Steven	Jackowski	Cerrado	4
Kyle	Johnsen	Abierto	65
Jeff	Johansen	Abierto	13
Mary	Johnson	Cerrado	24
Larry	Olsen	Abierto	21
William	O'Neil	Cerrado	29
Jane	Pick	Abierto	3285

Para este ejemplo nMaxRows = 3. La casilla de negocio llama al conjunto de resultados que se ordenen por apellido, y el desarrollador sabe que cualquier fila puede identificarse únicamente por los campos nombre propio, apellido e ID única de modo que la SQL inicial añadida como una constante en el controlador debería ser:

SELECCIONAR*DE persona ORDENADO POR Apellido, nombre propio, ID única.

Consulta inicial

La primera consulta se envía con una matriz vKeys vacía. Cuando el servidor recibe esta consulta, el método ArgumentsForBusinessObject (argumentos para objeto de negocio) identifica los elementos como vacíos y no ocupa el colArgs. La consulta se ejecuta con la SQL inicial sin variar. El conjunto de registros de tamaño nMaxRows+1 se devuelve al ACC 208 y se procesan igual que los resultados no limitados. El ACC 208 devuelve la serie de objetos al controlador 206. El controlador 206 procede a ocupar la serie vResults con los objetos devueltos. vResults es la serie exhaustiva de objetos devueltos. Cuando se llega al último objeto de la primera petición (en nMaxRows), los valores se almacenan en vKeys como tal;

vKeys(0) = apellido (Barleycorn)

vKeys(1) = nombre propio (John)

vKeys(2) = ID única (512)

Cuando se llega al primer objeto de la siguiente petición (en nMaxRows +1), se realiza una comparación de las variables de objeto frente a los valores vKeys. Dado que los apellidos coinciden, vKeys(2) no se borrará y no se realizarán más comprobaciones.

Siguiente consulta

La siguiente consulta pasará las vKyes junto con la misma. El servidor crea la serie de argumentos a partir de las vKeys y adjunta la cadena sSql en consecuencia. La instrucción sSql que se ha pasado para ejecutar la consulta es

SELECCIONAR*DE persona ORDENADO POR Apellido, nombre propio, ID única DONDE ¿ >=? Y ? >=? Y ? > ?

Esta sSql y serie se incluye en la llamada a ExecuteQuery que funde los argumentos con la cadena basándose en el método de arquitectura MergeSQL para completar la instrucción SQL. El punto de inicio del conjunto de registros se define por la cláusula DONDE y el límite se establece por el valor nMaxRows.

Criterios DONDE de consulta menos restrictivos

Después de la segunda consulta, la última fila de la consulta es David Dyson y la siguiente es Bobby Halford. Dado que el apellido es diferente, las vKeys estarán vacías salvo por vKeys(0) = Dyson. El ProcessObjectCollection (procesar serie de objetos) ocupará las vKeys como sigue al procesar el objeto nMaxRows:

vKeys(0) = apellido (Dyson)

vKeys(1) = nombre propio (David)

vKeys(2) = ID única (98)

Después de identificar las diferencias entre los valores vKeys y el objeto nMaxRows+1, la matriz de vKeys se actualiza como sigue:

vKeys(0) = apellido (Dyson)

vKeys(1) = vacío

vKeys(2) = vacío

La consulta que se devuelve desde ArgumentsForBusinessObject es

SELECCIONAR*DE persona ORDENADO POR Apellido, nombre propio, ID única DONDE ? > ?

y el colArgs que procesa el campo de nombre, nombre propio, y el valor (“David”) ExecuteQuery funde los argumentos con la instrucción sql como antes y devuelve el valor.

Final

Después de la quinta iteración el conjunto de resultados sólo tendrá 2 registros. Cuando el controlador 206 procesa la serie devuelta, la cuenta devuelta por ProcessObjectCollection es menor que nMaxRows + 1 lo que indica que se han recuperado todos los registros.

Marco de seguridad

Implementación

La figura 8 muestra una representación del marco 800 de seguridad y sus principales componentes.

Puede verse en la figura 8 que el objeto 802 de seguridad está presente en el cliente y que se proporciona una API de seguridad en el servidor. El objeto 802 de seguridad proporciona un método responsable de autorizar cualquier operación, con una estructura vMsg, una ID de la operación y un parámetro opcional que describe el contexto de la operación determinados.

Cliente

Autenticación de usuario

La autenticación de usuario se trata a través de un método situado en el objeto 802 de seguridad llamado IsOperAuthorized (está el operador autorizado). Cuando el objeto de aplicación se carga, llama al método IsOperAuthorized, con la operación de "conectarse" (login), antes de ejecutar el procesamiento posterior. Este método llama a continuación a un DLL de autenticación, que es responsable de identificar al usuario como un usuario autenticado en la seguridad de la empresa.

Controladores IU

Los controladores IU limitan el acceso a sus funciones restringiendo el acceso a widgets específicos habilitándolos y deshabilitándolos. La lógica para habilitar y deshabilitar widgets permanece en el controlador 206 IU, pero la lógica para determinar si un usuario tiene acceso a una funcionalidad específica se sitúa en el objeto 802 de seguridad en forma de normas de negocio. El controlador 206 IU llama al método IsOperAuthorized para ajustar el estado de sus widgets.

Servidor

La seguridad del servidor se implementa restringiendo el acceso a los datos de tres formas diferentes.

Método de seguridad del servidor

Los componentes 222 de servidor llaman a la API IsOperAuthorized en la arquitectura antes de ejecutar toda operación. En todos los casos, el objeto 802 de seguridad devuelve una regla booleana, de acuerdo con los derechos de acceso del usuario y las normas de negocio.

Filtrado SQL

Incluye atributos de seguridad, como gravedad de la reclamación o nota de archivo público / privado, en las instrucciones SQL al seleccionar o actualizar filas. Esto restringe de manera eficiente el conjunto de datos resultante, y evita que se devuelvan datos restringidos al cliente.

Descripción

Cualquier seguridad relacionada con la GUI se implementa en el cliente utilizando el objeto 802 de seguridad. La información está disponible tanto en el perfil de cliente como en los objetos 207 de negocio, lo que habilita las normas de seguridad para que se evalúen correctamente.

IsOperAuthorized se llama para ajustar widgets en el momento de la carga de una IU o si hay un cambio de estado en la IU.

La autenticación de usuario siempre se usa por los objetos 202 de aplicación para validar el privilegio de un usuario a lanzar la aplicación.

El filtrado SQL se usa en los casos en los que datos sensibles ni siquiera deben estar disponibles en el cliente, o cuando se obtiene una gran ventaja al reducir el tamaño del conjunto de datos devueltos al cliente.

El filtrado SQL se usa sólo en casos muy raros cuando el rendimiento supone un serio problema. Se usa con cuidado para evitar un aumento de la complejidad e impactos en el rendimiento ya que algunas consultas pueden ser engorrosas y agregar seguridad a las mismas puede aumentar la complejidad todavía más.

Marco de seguridad

Visión general

El objeto 802 de seguridad sirve para sostener normas de negocio fuertemente codificadas para conceder o denegar a un usuario el acceso a varias funciones de la aplicación. Esta información se devuelve a los controladores 206 IU que hacen las modificaciones necesarias en el estado IU. El objeto perfil de cliente sirve para memorizar en caché información específica del usuario y de seguridad (y estática) directamente en el cliente. Esta información es necesaria para evaluar las normas de negocio en el objeto 802 de seguridad.

Relaciones

La figura 9 muestra las relaciones entre el elemento de seguridad y otros elementos.

Objeto de arquitectura

El objeto TechArch (arquitectura técnica) es responsable de proporcionar el acceso y mantener el estado de los objetos 902 de perfil de cliente y 802 de seguridad. El objeto 902 perfil de cliente se instancia y se destruye en los métodos de inicialización y finalización de TechArch, respectivamente. Este objeto se mantiene a través de una variable de instancia en el objeto TechArch.

CInitcompACC

El objeto CInitCompACC 904 proporciona dos servicios al objeto 200 de arquitectura, sirve como un punto de acceso al servidor 906 CInitComp, y clasifica el conjunto de resultados de consulta en un objeto 902 de perfil de cliente.

CInitComp

El objeto 906 de servidor CInitComp proporciona acceso a datos a los datos que residen en las tablas 908 de organización. Estos datos son útiles en el cliente para determinar el nivel de acceso a datos basándose en las normas de negocio fuertemente codificadas.

Tablas de organización

Las tablas 908 de organización contienen información de usuario, empleado y unidad necesaria para formar la jerarquía de la información necesaria para determinar el nivel de acceso a información sensible.

Perfil de cliente

El objeto 902 perfil de cliente sirve para memorizar en caché información de seguridad específica de usuario, estática directamente en el cliente. Esta información es necesaria para determinar el nivel de acceso a datos de la información para el usuario, lo que se consigue pasando los valores necesarios al objeto 802 de seguridad.

Objeto de seguridad

El objeto 802 de seguridad contiene normas de negocio usadas para determinar los privilegios de acceso de un usuario en relación con funciones específicas. El objeto acepta ciertos parámetros pasados por los varios controladores 902 IU y los pasa a través de la lógica de normas de negocio que, a su vez, interroga al objeto 902 de perfil de cliente sobre información de usuario específica.

Perfil de cliente

Atributos

Los siguientes son atributos internos para el objeto 902 de perfil de cliente. Estos atributos no se exponen en la aplicación y sólo deberían usarse por el objeto 802 de seguridad:

• sProfile:

Este atributo se pasa por la aplicación antigua en el arranque y contiene las TSIds de usuario, indicador externo, la cuenta de elementos de grupo y los elementos de grupo. Se clasifica en estos atributos por petición de los objetos de aplicación.

• colSpecialUsers:

Este atributo memoriza en caché información de una tabla que contiene usuarios especiales que no se ajustan a ninguno de los papeles descritos, tal como Bibliotecario de organización (por ejemplo, Vicepresidente, o CEO de la sociedad).

• sTSId:

Se trata del TSId del usuario actual, y corresponde a su Id de Windows NT. Se usa para obtener información sobre el usuario actualmente conectado a partir de las tablas 908 de organización.

• sEmployeeId:

Corresponde al Id de empleado del usuario, tal como se almacena en las tablas 908 de organización. Se usa contra la Id de empleado pasada, para comprobar relaciones ente usuarios anteriores y el actual.

• SEmpoyeeName, sEmployeeFirst, sEmployeeMI y sEmployeeLast:

Todos estos atributos corresponden al nombre del usuario actual.

• dictClientPrivileges:

Este atributo contiene una serie de identificadores que indican qué papel desempeña o qué autoridad posee una persona. Este valor se usa para identificar el papel estático del usuario conectado.

Estos valores se usan por la lógica de negocio de seguridad que concede o deniega el acceso basándose en si el usuario es interno o externo, o en si el usuario tiene un cargo administrativo dado. Los valores existentes son los siguientes:

• SC - indica una autoridad de reclamación sensible

• CC- indica una autoridad de cambio del estado de una reclamación

• MT - indica una autoridad de mantener plantillas F&C

• MO - indica una autoridad de mantener organización

• MR - indica una autoridad de mantener papeles

Las siguientes son adiciones propuestas:

• TA - indica autoridad para ejecutar asistente de tareas

• FN - indica autoridad para ejecutar notas de archivo

• CH - indica autoridad para ejecutar historial de reclamaciones

• TL - indica autoridad para mantener plantillas de tareas

• dictProxyList

Este atributo contiene la jerarquía de cobertura de un empleado. Se usa para determinar si el usuario/empleado actual tiene permiso para realizar alguna acción basándose en su relación con otros usuario/empleados en su jerarquía. Un negocio ejemplo de esto es el caso de un supervisor, que tiene derecho a ver información a la que sus subordinados tienen acceso. La relación de las API hace uso de dictProxyList para determinar si el usuario asignado a la información es superior o subordinado del usuario actual.

• boolInternal

Este atributo indica si el usuario conectado es externo o interno. También se clasifica a partir del atributo sProfile, pasado por la aplicación antigua.

Métodos públicos

Las siguientes son las API expuestas por el objeto de perfil de cliente. Esas API se usan para la comprobación de seguridad por el objeto de seguridad y no deberían usarse por los desarrolladores en ninguna parte de la aplicación.

• · GetAuthorizedEmployees As Collection

Esta función devuelve una serie de Id de empleado de los empleados supervisados por el usuario actual.

- · IsSuperOf (sUserId) As Boolean

Esta API devuelve verdadero si el usuario conectado es un superior de la Id de usuario pasada. Busca el valor sUserId dentro del atributo dictProxyList.

- · IsRelativeOf (sUserId) As Boolean

Esta API devuelve verdadero si la Id de usuario pasada corresponde con el usuario conectado o alguien del dictProxyList.

- · IsInternal As Boolean

Esta API se usa para conceder o restringir al usuario información basándose en si los datos son privados de la organización y si el usuario es interno o externo.

- · IsInRole (sRole) As Boolean

Esta API busca el valor sRole adecuado contenido en el atributo dictClientRoles para determinar si el usuario actual está autorizado para desempeñar ese papel.

Los siguientes descriptores de acceso se usan para obtener datos del objeto de perfil de cliente:

- · UserId: devuelve sTSId
- · EmployeeId: devuelve sEmployeeId
- · EmployeeName: devuelve sEmployeeName
- · EmployeeFirstName: devuelve sEmployeeFirst
- · EmployeeLastName: devuelve sEmployeeLast
- · EmployeeMiddleInitial: devuelve sEmployeeMI
- · ExpandTree: devuelve boolExpandTreePreference
- · TemplatePathPreference: devuelve sTemplatePathPreference

Objeto de seguridad

Métodos públicos

La siguiente API se expone por el objeto de seguridad y se usa por la aplicación para comprobar la seguridad:

• IsOperAuthorized (vMsg As Variant, nOperations As cmOperations, vContext As Variant) as Boolean

Esta API devolverá verdadero o falso dependiendo de lo que se devuelva desde las funciones de normas de negocio para determinar los niveles de acceso de usuario. Esta API se llama en dos situaciones:

1. Al ajustar el estado inicial antes de cargar el formulario. Si hay un requisito de seguridad, IsOperAuthorized se llama para la operación apropiada.

2. Después de cualquier cambio relevante en el estado de la IU. Por ejemplo, cuando una reclamación sensible se marca en la ventana del asistente de tareas. Un cambio relevante se uno que conlleva la necesidad de una comprobación de seguridad.

Los valores válidos para la enumeración y los datos de contexto correspondientes son:

· cmMaintainFormsCorr (ninguno)

· cmRunEventProcessor (ninguno)

· cmWorkOnSensitiveClaim (un objeto de reclamación)

· cmMaintainPersonalProfile (ninguno)

· cmMaintainWorkplan (ninguno)

· cmDeleteFileNote (un objeto de archivar nota)

· cmMaintainTaskLIbrary (ninguno)

· cmMaintainOrg (ninguno)

API de seguridad de servidor

• IsSVCOperAuthorized (vMsg As Variant. sOperations As String, vContext As Variant) as Boolean

Esta API se llama por cualquier método en el servidor que persiste datos o puede acceder potencialmente a datos sensibles (enfoque reactivo).

• IsOperAuthorized (vMsg As Variant, nOperations As cmOperations, vContext As Variant) as Boolean

Esta API está disponible para aquellos casos en los que es necesaria una comprobación de seguridad preactiva en el servidor.

Ejemplos de implementación

Los siguientes ejemplos muestran algunas formas de implementar las opciones descritas anteriormente:

Cliente

Lógica de negocio

IsOperAuthorized

Se considerará el caso de la ventana del asistente de tareas, en la que el usuario no debería estar autorizado a ver ninguna información sobre una reclamación sensible si no es el realizador de la reclamación o el supervisor del realizador. El siguiente código estaría en el controlador:

```
Private Sub TaskTree_NodeChanged(....)
    myController.SetCurrentTask
    myController.SetState
End Sub

Private Sub SetState()
    Dim objSecurity as Object
    Dim vContext(1) as Object

    Set objSecurity = taaApp.taoArch.objSecurity

    vContext(0) = CurrentClaim
    vContext(1) = CurrentTask

    tlbEditIcon.Enabled =
                objSecurity.IsOperAuthorized(vMsg, cmWorkOnSensitiveClaim, vContext)

End Sub
```

Se considerará el caso de la ventana de búsqueda de Mantener correspondencia, en la que sólo se debería permitir a un usuario que sea un bibliotecario de formularios y correspondencia que borre una plantilla. El siguiente código estaría en el controlador:

```
Private Sub SetWindowMode()
    Dim objSecurity as Object
    Set objSecurity = taaApp.taoArch.objSecurity

    . . .

    tlbEditIcon.Enabled = objSecurity.IsOperAuthorized(vMsg, cmMaintainFormsCorr)

End Sub
```

Servidor

Filtrado SQL

Se considerará el ejemplo de la ventana de Borrador de notas de archivo, en la que un usuario sólo puede ver el borrador de notas de archivo del que es autor. En el controlador se podría tener:

```
Public Sub GetDraftFNotes()
    Dim objCP as Object
    Set objCP = taoArch.objClientProfile

    Dim fntCCA as Object
    Set fntCCA = taaApp.taoArch.GetCCA(cmCCAFileNote)


    Call fntCCA.GetADraftFNote(vMsg, objCP.sOrgUserId, colFNotes)
End Sub
```

Y en el componente, la instrucción SQL sería:

```
Select  nFNoteId,
    sFNoteAuthor,
    dFNoteFinal,
    :
    :
    From  File Note
    Where sFileNoteSts = 'D'
    And   sFNoteAuthor = sAuthor
```

Aplicación de motor de tareas

Esta aplicación se ejecuta en el servidor como un proceso o servicio de fondo sin interacción directa con las aplicaciones de cliente, de modo que no necesita ninguna seguridad relacionada con la GUI. Básicamente, sus principales acciones se limitan a la generación de nuevas tareas en respuesta a eventos generados externamente o, más específicamente:

- lee información estática de las tablas de plantillas de tareas;
- lee eventos de las tablas de eventos;
- inserta tareas en la tabla de tareas.

En este sentido, su seguridad depende totalmente de las entidades externas tal como se describe a continuación:

- La aplicación biblioteca de tareas es el punto de entrada para cualquier cambio en las tablas de la base de datos de plantillas de tareas. Hará uso de las opciones descritas anteriormente con el fin de cumplir sus requisitos de seguridad.
- Los eventos se generan a partir de aplicaciones antiguas, por lo que el motor de tareas se fía completamente de la seguridad implementada por estas aplicaciones para controlar la generación de eventos.
- Otro nivel de seguridad para la generación de eventos se basa en las funciones de autenticación y autorización de la base de datos. Sólo los componentes autorizados tienen acceso a las tablas de las bases de datos (esto es válido también para todas las demás aplicaciones).

Carpeta de reclamación

Definición

La carpeta de reclamación gestiona la información de la reclamación desde su la primera noticia hasta el cierre y archivado. Lo hace proporcionando una interfaz estructurada y fácil de usar que soporta múltiples procesos de negocio para tratar reclamaciones. La información que captura se suministra a muchos otros componentes para permitir que los

profesionales de las reclamaciones hagan uso de aplicaciones de habilitación que reducen su carga de trabajo. Puesto que los archivos de reclamación físicos todavía son necesarios, la carpeta de reclamación proporciona capacidades que soportan un seguimiento de los archivos físicos. Funciona con el sistema LEGACY para soportar todas las capacidades que existen en el sistema actual.

Los procesos principales soportados por la carpeta de reclamación son:

• Primera noticia de pérdida

La carpeta de reclamación es el primer punto de entrada para nueva información de pérdida. Los archivos de reclamación existen en la carpeta de reclamación antes de que se "empujen" al sistema LEGACY para realizar el procesamiento final.

• Consultas de la reclamación

La carpeta de reclamación soporta consultas internas y externas sobre información de la reclamación. El diseño de la carpeta permite un acceso rápido a varios niveles de información en la reclamación por muchos motivos diferentes.

• Inicio del tratamiento de la reclamación

La carpeta de reclamación proporciona información de pérdida inicial al profesional de las reclamaciones para que pueda empezar el proceso de hacer los primeros contactos con participantes apropiados en la reclamación. Les permite ver e introducir datos recibidos a través de sus contactos y su investigación iniciales.

• Investigación y evaluación

La carpeta de reclamación proporciona acceso a información detallada necesaria para el proceso de investigación y evaluación. Permite a la persona que trata la reclamación navegar entre todas las aplicaciones y la información que necesitan para soportar estos procesos.

• Identificación de eventos de la reclamación

La carpeta de reclamación identifica eventos críticos que ocurren en la vida de una reclamación, tales como un cambio de estado, que puede desencadenar respuestas en otros componentes para realizar funciones automáticas, como desencadenar tareas en el asistente de tareas.

• Gestionar el archivo físico

La carpeta de reclamación soporta mejores capacidades de seguimiento de archivos físicos que van junto con el registro electrónico de una reclamación.

Valor

Al capturar información detallada sobre las reclamaciones, la carpeta de reclamación intenta mejorar la eficiencia de los profesionales de la reclamación de muchas formas. En primer lugar, dado que la información se organiza en un formato lógico fácil de usar, hay que rebuscar menos para encontrar información básica para soportar cualquier número de consultas. En segundo lugar, la carpeta de reclamación utiliza su información para soportar otras aplicaciones como formularios y correspondencia, de modo que la información de la reclamación no tiene que volverse a introducir cada vez que se necesita. En tercer lugar, proporciona mejores formas de encontrar archivos físicos para reducir el tiempo necesario para encontrarlos y trabajar con ellos. Además de esto, hay muchos otros usos potenciales de la información de la carpeta de reclamación.

La carpeta de reclamación también intenta superar algunos de los requisitos de procesamiento actuales que impone el sistema LEGACY, tal como registrar pérdidas sin reclamaciones, requerir números de póliza para abrir una reclamación, requerir reservas para líneas, y otras restricciones. Esto reducirá algo del trabajo de poco valor añadido requerido para abastecer al sistema LEGACY.

Finalmente, la carpeta de reclamación organiza y coordina información sobre los participantes y los realizadores de modo que todas las personas implicadas en una reclamación pueden identificarse rápida y fácilmente.

Usuarios clave

Aunque los profesionales de la reclamación son los usuarios principales de la carpeta de reclamación, cualquier profesional de las reclamaciones puede utilizar la carpeta de reclamación para conocer los detalles de una reclamación o responder a una consulta sobre una reclamación.

Funcionalidad de componente

Dado que la carpeta de reclamación es el punto de entrada primario de nuevas reclamaciones, debe capturar la información necesaria para abrir nuevas reclamaciones y poder pasar información al sistema LEGACY. Una vez pasada la información, el sistema LEGACY posee toda la información contenida en ambos sistemas, y no puede editarse en la carpeta de reclamación. Sin embargo, la carpeta de reclamación tiene más información que lo que está contenido en el sistema LEGACY, y por lo tanto permite introducir y modificar cierta información una vez que la reclamación se ha empujado al sistema LEGACY.

La carpeta de reclamación descompone una reclamación en diferentes niveles que reflejan la póliza, el asegurado, los reclamantes, y las líneas del reclamante. Cada nivel tiene un conjunto estructurado de información que se aplica al mismo. Por ejemplo, el nivel de reclamación de la reclamación tiene información sobre el estado de la reclamación, línea de negocio y realizadores. Una línea individual tiene información que incluye el tipo de línea, la jurisdicción y daños a la propiedad o al vehículo. El nivel de reclamante contiene información de contacto así como descripciones de las lesiones.

La información en cada nivel se agrupa en secciones con fines de organización. Cada nivel tiene una sección de detalles que incluye la información básica sobre el nivel.

Los niveles clave en la carpeta de reclamación y sus secciones de información son:

- El nivel de la póliza: detalles y coche cubierto para reclamaciones de vehículo, propiedad cubierta para reclamaciones de propiedad y yates cubiertos para reclamaciones marítimas.
- El nivel de la reclamación: detalles, hechos de pérdida, eventos, responsabilidad. La responsabilidad se considera parte del componente de negociación y que se describe ahí.
- El nivel de participante: detalles e información de contacto. Para el reclamante, se muestran secciones adicionales que muestran eventos, lesiones y gestión de discapacidad. El nivel de participante se comenta en el componente de participante.
- El nivel de la línea: detalles, vehículo dañado para líneas de vehículos, propiedad dañada para líneas de propiedad, yate dañado para líneas marítimas, eventos, daños y negociación. Los daños y la negociación se consideran parte del componente de negociación que se describe ahí.

Los eventos se desencadenan en la carpeta de reclamación realizando ciertas acciones como cambiar una jurisdicción, identificar una lesión, o cerrar una línea. Otros eventos generales se desencadenan en la sección de eventos en la mayoría de los niveles haciendo clic sobre el que ha ocurrido. Estos eventos se procesan por el procesador de eventos y podrían generar cualquier número de respuestas. En una realización de la presente invención, la respuesta principal es desencadenar nuevas tareas en el asistente de tareas para una reclamación.

Interfaces de usuario

- · Carpeta de reclamación IU
- · Nivel de póliza - Pestaña de detalles de póliza
- · Nivel de póliza - Pestaña de vehículo cubierto
- · Nivel de póliza - Pestaña de propiedad cubierta
- · Nivel de póliza - Pestaña de yate cubierto
- · Nivel de reclamación - Pestaña de detalles de la reclamación
- · Nivel de reclamación - Pestaña de hechos de pérdida
- · Nivel de reclamación - Pestaña de eventos
- · Nivel de reclamación - Pestaña de responsabilidad
- · Nivel de línea - Pestaña de detalles de línea
- · Nivel de línea - Pestaña de propiedad dañada
- · Nivel de línea - Pestaña de vehículo dañado

- · Nivel de línea - Pestaña de yate dañado
- · Nivel de línea - Pestaña de eventos
- · Nivel de línea - Pestaña de daños
- · Nivel de línea - Pestaña de negociación
- · Asistente de tareas
- · Archivar notas
- · Historial de reclamación
- · Buscar plantilla de reclamación
- · Buscar correspondencia
- · Encontrar reclamaciones
- · Versión 7
- · Ver carpeta de reclamación
- · Imprimir etiqueta

Diseño de árbol y menú de la carpeta de reclamación

Árbol de reclamación

El árbol de reclamación en la ventana de la carpeta de reclamación descompone la reclamación en los niveles de póliza, asegurado, reclamación, reclamante, y línea dependiendo de la composición específica de la reclamación.

El nivel de la póliza siempre es el primer nodo del árbol de reclamación y se identifica por el número de póliza. Antes de introducir el número de la póliza, el campo se enumera como "desconocido". Si se descodifica una reclamación, el campo se enumera como "descodificado". Al seleccionar el nivel de póliza, se lleva la pestaña del nivel de póliza al cuerpo de la carpeta de reclamación.

El nivel del asegurado es siempre el segundo nodo en el árbol de reclamación y se identifica por el nombre del asegurado. Antes de identificar al asegurado, el campo se enumera como "desconocido". Al seleccionar el nivel del asegurado se lleva la pestaña del participante asegurado al cuerpo de la carpeta de reclamación. Sólo se enumera un asegurado en este nivel, según se identifica en las pestañas del nivel de la póliza, sin embargo, pueden añadirse múltiples asegurados. Se muestran asegurados adicionales en la lista de participantes bajo el árbol de reclamación.

El nivel de la reclamación es siempre el tercer nodo en el árbol de reclamación y se identifica por el número de reclamación. Cuando se selecciona el nivel de la reclamación, la pestaña del nivel de la reclamación aparece en el cuerpo de la carpeta de reclamación.

Después del nivel de la reclamación, se enumeran todos los reclamantes con sus líneas asociadas en un formato jerárquico. Cuando se añade un reclamante, se añade un nodo al árbol, y el campo que identifica al reclamante se enumera como "desconocido". Una vez que se ha identificando al participante, parcial o cliente, el nombre del reclamante se enumera en el nivel. Cuando se selecciona el nivel, la pestaña del nivel participante se muestra en el cuerpo de la carpeta de reclamación.

Los niveles de línea se identifican por el tipo de línea. Antes de seleccionar un tipo de línea, el nivel de línea se enumera cono "desconocido". Cuando se selecciona un nivel de línea, las pestañas del nivel de línea para la línea específica se muestran en el cuerpo de la carpeta de reclamación.

Ha varias cosas que pueden alterar el árbol de reclamación una vez establecido. En primer lugar, si un reclamante o línea se borra, se elimina del árbol de reclamación. Una reclamación que se marca por error no cambia la apariencia de los niveles. En segundo lugar, los niveles de reclamación, reclamante y línea se identifican por diferentes iconos dependiendo de si se han empujado al V7 o no. En tercer lugar, cuando una línea o reclamante está compensado, se identifica como tal.

Lista de participantes

El cuadro de lista de participantes contiene todos los participantes no reclamantes y no asegurados en la reclamación. (Los reclamantes y asegurados se muestran en el árbol de reclamación y no se repiten aquí.) Los participantes se muestran con su nombre y papel. Cuando se selecciona un participante, la pestaña del nivel participante se muestra en la carpeta de la reclamación.

Elementos de menú de la carpeta de reclamación

Los menús de la carpeta de reclamación contienen las acciones que un usuario necesitaría realizar en la carpeta de reclamación. Puede accederse a todos ellos a través de selección con el teclado. Las opciones de menú se habilitan o deshabilitan basándose en el estado de la carpeta de reclamación. La carpeta de reclamación puede estar en modo ver o modo editar para un nivel específico en el árbol de reclamación. Cuando la carpeta de reclamación se encuentra en el modo editar, la mayoría de las opciones están deshabilitadas hasta que el usuario guarda sus cambios y vuelve al modo ver. La habilitación/deshabilitación de las opciones de menú depende también de si la reclamación o partes de la reclamación se han empujado al V7.

Barra de herramientas de la carpeta de reclamación

La barra de herramientas representa las acciones comunes que un usuario realiza y a las que puede accederse fácilmente haciendo clic sobre el icono apropiado. Hay cinco grupos de botones en la barra de herramientas de la carpeta de reclamación que representan, en orden, actividades comunes, añadir nuevos elementos a una reclamación, lanzar utilidades, realizar actividades V7, y acceder a funciones de ayuda. La habilitación/deshabilitación de los botones de la barra de herramientas sigue la misma lógica que para los elementos de menú.

Descripción de ventanas

Nombre de control	Tipo	Descripción	Valor / estado por defecto
Árbol de reclamación	Vista de árbol	El árbol de reclamación enumera la póliza, asegurado, todos los reclamantes y sus líneas relacionadas en un formato de árbol de reclamación	La actual estructura de árbol de reclamación para la reclamación seleccionada. El nivel de reclamación está seleccionado y se visualizan las pestañas del nivel de reclamación.
Lista de participantes	Vista de lista	Una lista de todos los participantes no asegurados y no reclamantes asociados con una reclamación	Todos los participantes que no son reclamantes o asegurados para la reclamación y su papel
Botón de la barra de herramientas,	Botón de comando	Cambia las pestañas al nivel seleccionado en la vista de	Habilitado cuando la reclamación está en modo

Editar		lista del árbol de reclamación o de participante al modo editar	editar.
Botón de la barra de herramientas, Regenerar	Botón de comando	Regenera la reclamación actual, incluyendo toda la información de participante y de línea.	Habilitado cuando la reclamación está en modo ver.
Botón de la barra de herramientas, Encontrar	Botón de comando	Abre la ventana de búsqueda de reclamación para permitir al usuario buscar otra reclamación	Habilitado
Botón de la barra de herramientas, Asignación de reclamación	Botón de comando	Abre la ventana de asignar reclamación.	Habilitado cuando la reclamación está en modo ver
Botón de la barra de herramientas, Gestionar archivo físico	Botón de comando	Abre la ventana de gestionar archivo físico	Habilitado cuando la reclamación está en modo ver
Botón de la barra de herramientas, Declarar evento	Botón de comando	Abre la ventana de declarar evento	Habilitado cuando la reclamación está en modo ver
Botón de la barra de herramientas, Reclamante	Botón de comando	Añade el reclamante y abre pestañas de participante en modo editar para la introducción de un nuevo nodo de nivel reclamante	Habilitado cuando la reclamación está en modo ver. El límite de V7 para reclamantes es de 999, no se editará esto aquí.
Botón de la barra de herramientas, Participante	Botón de comando	Añade un nuevo participante y abre pestañas participante en modo editar	Habilitado cuando la reclamación está en modo ver
Botón de la barra de herramientas, Línea	Botón de comando	Añade línea y abre pestañas de línea en modo editar para la introducción de un nuevo nodo de nivel línea	Habilitado cuando la reclamación está en modo ver y el contexto de cliente está seleccionado en el árbol de reclamación. El límite de V7 para líneas es de 15 por reclamante, este botón se deshabilitará después de añadir 15.

Botón de la barra de herramientas, Designar realizador	Botón de comando	Abre la ventana de designar realizador	Habilitado cuando la reclamación está en modo ver
Botón de la barra de herramientas, Imprimir pantalla	Botón de comando	Imprime la actual ventana de carpeta de reclamación	Habilitado
Botón de la barra de herramientas, Asistente de tareas	Botón de comando	Lanza Asistente de tareas para la reclamación actual	Habilitado cuando la reclamación está en modo ver
Botón de la barra de herramientas, Notas de archivo	Botón de comando	Lanza Notas de archivo para la reclamación actual	Habilitado cuando la reclamación está en modo ver
Botón de la barra de herramientas, Historial de reclamaciones	Botón de comando	Lanza Historial de reclamaciones para la reclamación actual	Habilitado cuando la reclamación está en modo ver
Botón de la barra de herramientas, Correspondencia	Botón de comando	Abre la ventana de formularios y correspondencia	Habilitado cuando la reclamación está en modo ver
Botón de la barra de herramientas, empujar a V7	Botón de comando	Abre la ventana de emulador terminal en la primera pantalla de ajuste V7	Habilitado cuando la reclamación está en modo ver y el estado de la reclamación se ha empujado previamente o abierto y hay nuevos reclamantes o líneas que empujar.
Botón de la barra de herramientas, Realizar pago	Botón de comando	Abre la pantalla V7 PUEM en la ventana de emulador terminal si se selecciona un reclamante o participante asociado a ese reclamante. De lo contrario, visualiza la ventana que requiere que el usuario seleccione un reclamante.	Habilitado cuando la reclamación se ha empujado a V7 y se ha seleccionado un participante.
Botón de la barra de herramientas, Ayuda	Botón de comando	Abre Ayuda	Habilitado

Editar reclamación	Opción de menú	Cambia pestañas de reclamación al modo editar de manera que el usuario puede realizar cambios	Habilitado cuando la reclamación está en modo ver
Regenerar reclamación	Opción de menú	Regenera la reclamación actual, incluyendo toda la información de participante y de línea	Habilitado cuando la reclamación está en modo ver
Encontrar reclamación	Opción de menú	Abre la ventana de búsqueda de reclamación	Habilitado
Guardar reclamación	Opción de menú	Guarda el nivel de reclamación cuando está en modo editar	Habilitado cuando el nivel de reclamación está en modo editar
Reclamación, estado de la reclamación, Primer informe completado	Opción de menú	Cambia el estado de la reclamación a "No designado" y crea el evento primer informe completado	Habilitado cuando la reclamación está en modo ver y el estado de la reclamación es "Nuevo".
Reclamación, estado de reclamación, Asignación completa	Opción de menú	Cambia el estado de la reclamación a "Abierto" y crea el evento Designación completada	Habilitado cuando la reclamación está en modo ver y el estado de la reclamación es "No designado"
Reclamación, estado de la reclamación, cerrar	Opción de menú	Inicia el proceso de cerrar reclamación	Habilitado cuando la reclamación está en modo ver, el estado de reclamación V7 está cerrado y el estado de la reclamación Millennium no es "cerrado" ni "Archivado"
Reclamación, estado de la reclamación, reabrir	Opción de menú	Cambia el estado de la reclamación a "Abierto"	Habilitado cuando la reclamación está en modo ver y "cerrada" o "archivada"
Reclamación, estado de la reclamación, marcar error	Opción de menú	Marca la reclamación actual y todas sus líneas en error. Finaliza todos los participantes.	Habilitado cuando la reclamación está en modo ver y no se ha empujado a V7.
Reclamación, asignar	Opción de menú	Abre la ventana de asignar reclamación	Habilitado cuando la reclamación está en modo ver

Reclamación, gestionar archivo físico	Opción de menú	Abre ventana de archivo físico	Habilitado cuando la reclamación está en modo ver
Reclamación, declarar evento	Opción de menú	Abre ventana de declarar evento	Habilitado cuando la reclamación está en modo ver
Reclamación, cerrar carpeta de reclamación	Opción de menú	Cierra la ventana de carpeta de reclamación actual	Habilitado
Editar, cortar	Opción de menú	Mueve el texto seleccionado al portapapeles	Deshabilitado
Editar, copiar	Opción de menú	Copia el texto seleccionado al portapapeles	Deshabilitado
Editar, pegar	Opción de menú	Pega texto desde el portapapeles	Deshabilitado
Vista, plegar todo	Opción de menú	Pliega el árbol de reclamación	Habilitado
Vista, expandir todo	Opción de menú	Expande el árbol de reclamación	Habilitado
Editar póliza	Opción de menú	Abre pestañas de póliza en el modo editar	Habilitado cuando la reclamación está en modo ver
Guardar póliza	Opción de menú	Guardar información en pestaña de póliza actual	Habilitado cuando el nivel de póliza está en modo editar
Participante, nuevo, reclamante	Opción de menú	Abre pestañas de participante en modo editar para introducir un nuevo nodo de nivel reclamante en el árbol de reclamación	Habilitado cuando la reclamación está en modo ver
Participante, nuevo, asegurado	Opción de menú	Abre pestañas de participante en modo editar para introducir nuevo nodo de nivel asegurado en el árbol de reclamación	Habilitado cuando la reclamación está en modo ver
Participante, nuevo, otros	Opción de menú	Abre pestañas de participante en modo editar para introducir nueva entrada en la lista de participantes	Habilitado cuando la reclamación está en modo ver
Editar participante	Opción de menú	Pone las pestañas de participante actualmente	Habilitado cuando la reclamación está en modo ver

		seleccionadas en modo editar	y el participante seleccionado en el árbol o cuadro de lista
Guardar participante	Opción de menú	Guarda la información cambiada en las pestañas de participante y devuelve la reclamación al modo ver	Habilitado sólo cuando un nivel de participante está en modo editar.
Borrar participante	Opción de menú	Borra participante seleccionado	Habilitado sólo cuando la reclamación está en modo ver y el participante está seleccionado
Nueva línea	Opción de menú	Añade nueva línea al árbol de reclamación y abre pestañas de línea en modo editar	Habilitado cuando la reclamación está en modo ver, se ha seleccionado reclamante y no se ha superado el límite de 15 líneas por reclamante
Editar línea	Opción de menú	Pone pestañas de línea en modo editar de manera que el usuario pueda cambiar los detalles de línea	Habilitado cuando la reclamación está en modo ver y se ha seleccionado una línea
Guardar línea	Opción de menú	Guarda la información que se ha introducido en las pestañas de línea y devuelve la reclamación al modo ver	Habilitado cuando una línea está en modo editar
Cambiar estado de línea, cerrar	Opción de menú	Cambia el estado de una línea en la carpeta de reclamación a "cerrado"	Habilitado cuando la reclamación está en modo ver, se ha seleccionado una línea, la línea no está cerrada y su estado V7 está cerrado.
Estado de cambio de línea, reabierto	Opción de menú	Cambia el estado de la línea seleccionada a "abierto"	Habilitado cuando la reclamación está en modo ver, se ha seleccionado una línea y la línea está "cerrada".
Estado de cambio de línea, marcar error	Opción de menú	Marca la línea seleccionada en error	Habilitado cuando la reclamación está en modo ver, se ha seleccionado una línea y la línea no se ha empujado.
Asignar línea	Opción de menú	Abre la ventana de asignación de reclamación	Habilitado
Designar	Opción de	Abre la ventana de asignar	Habilitado cuando la

realizadores	menú	realizadores	reclamación está en modo ver
Realizadores, ver todo	Opción de menú	Visualiza todos los realizadores de reclamación asignados a la reclamación en la IU ver realizador	Habilitado cuando la reclamación está en modo ver
Utilidades, imprimir pantalla	Opción de menú	Imprime pantalla actual	Habilitado
Utilidades, ver asistente de tareas	Opción de menú	Abre la ventana de asistente de tareas para la reclamación actual	Habilitado cuando la reclamación está en modo ver
Utilidades, crear nueva nota de archivo	Opción de menú	Abre la ventana notas de archivo para reclamación actual	Habilitado cuando la reclamación está en modo ver
Utilidades, ver historial de reclamaciones	Opción de menú	Abre la ventana historial de reclamaciones para reclamación actual	Habilitado cuando la reclamación está en modo ver
Utilidades, crear correspondencia	Opción de menú	Abre ventana de formularios y correspondencia	Habilitado cuando la reclamación está en modo ver
Versión 7, empujar reclamación	Opción de menú	Lanza V7 para iniciar el proceso de empujar	Habilitado cuando la reclamación está en modo ver y en estado de "empujado previamente" o abierto cuando hay reclamantes y líneas sin empujar
Versión 7, deshacer empujar	Opción de menú	Retorna la reclamación al estado empujado previamente	Habilitado cuando la reclamación está en modo ver y en estado "pendiente de empujarse"
Versión 7, realizar pago	Opción de menú	Abre la pantalla V7 PUEM en la ventana del emulador terminal si un reclamante o participante asociado a un reclamante está seleccionado. De lo contrario, visualiza ventana que requiere que el usuario seleccione un reclamante.	Habilitado cuando la reclamación se ha empujado a V7 y se ha seleccionado un participante.
Ayuda, contenidos	Opción de menú	Abre el archivo de ayuda para el menú de contenidos	Habilitado

Ayuda, buscar ayuda sobre	Opción de menú	Abre el archivo de ayuda para la ventana de búsqueda	Habilitado
Ayuda, acerca de	Opción de menú	Abre la ventana que muestra información acerca de la aplicación	Habilitado

Detalles de ventana

Nombre de control	Objetivo inicial	Botón por defecto	Orden de pestañas
Árbol de reclamación	Sí		1
Lista de participantes			2
Menú de reclamación			3
Menú edición			4
Menú vista			5
Menú póliza			6
Menú participante			7
Menú línea			8
Menú realizador			9
Menú utilidades			10
Menú versión 7			11
Menú ayuda			12

Diagrama CAR

Nombre de control	Acción	Respuesta	Clave cortocircuito	Clave mnemotécnica
Árbol de reclamación	Clic	- Destaca nodo en el árbol - Deshabilita participante en vista lista si se seleccionó previamente uno - Muestra pestañas relacionadas en modo vista - Habilita elementos de menú apropiados y botones de la barra de herramientas		
	Doble clic	- Nivel seleccionado en el árbol entra en modo editar		
Todos los campos de texto	Destacar	- Habilita cortar y copiar		
Lista de participantes	Clic	- Destaca participante en el cuadro de lista - Deselecciona nivel en el árbol de reclamación si se		

		seleccionó previamente uno - Muestra pestañas relacionadas en modo vista - Habilita elementos de menú apropiados y botones de la barra de herramientas
	Doble clic	- Participante seleccionado en vista lista entra en el modo edición
Botón barra de herramientas editar	Clic	- Cambia las pestañas para el nivel seleccionado en el árbol de reclamación o vista lista de participantes al modo editar
Botón barra de herramientas regenerar	Clic	- Regenera la reclamación actual, incluyendo toda la información de participante y de línea
Botón barra de herramientas encontrar	Clic	- Abre la ventana de búsqueda de reclamación para permitir al usuario buscar otra reclamación
Botón barra de herramientas asignar reclamación	Clic	- Abre la ventana de asignación de reclamación
Botón barra de herramientas gestionar archivo físico	Clic	- Abre la ventana de gestionar archivo físico
Botón barra de herramientas declarar evento	Clic	- Abre la ventana declarar eventos
Botón barra de herramientas reclamante	Clic	- Añade reclamante y abre pestañas de participante en modo editar para introducir un nuevo nodo de nivel reclamante
Botón barra de	Clic	- Añade nuevo participante

herramientas participante		y abre pestañas de participante en modo editar
Botón barra de herramientas línea	Clic	- Añade línea y abre pestañas de línea en modo editar para introducir un nuevo nodo de nivel línea
Botón barra de herramientas, designar realizador	Clic	- Abre ventana de designar realizador
Botón barra de herramientas, imprimir pantalla	Clic	- Imprime la ventana de carpeta de reclamación actual
Botón barra de herramientas, asistente de tareas	Clic	- Lanza el asistente de tareas para la reclamación actual
Botón barra de herramientas, notas de archivo	Clic	- Lanza Notas de archivo para la reclamación actual
Botón barra de herramientas, historial de reclamaciones	Clic	- Lanza el historial de reclamaciones para la reclamación actual
Botón barra de herramientas, correspondencia	Clic	- Abre la ventana formularios y correspondencia
Botón barra de herramientas, empujar a V7	Clic	- Abre la ventana del emulador terminal en la primera pantalla de ajuste V7
Botón barra de herramientas, realizar pago	Clic	- Abre la pantalla V7 PUEM en la ventana del emulador terminal si se ha seleccionado un reclamante o participante. De lo contrario, visualiza una ventana que requiere que el usuario seleccione un reclamante.

Botón barra de herramientas, Ayuda	Clic	- Abre Ayuda	
Editar reclamación	Clic	- Cambia pestañas de reclamación al modo editar de manera que el usuario pueda realizar cambios	
Regenerar reclamación	Clic	- Regenera la reclamación actual, incluyendo toda la información de participante y línea	Ctrl+R
Encontrar reclamación	Clic	- Abre la ventana de búsqueda de reclamación	Ctrl+F
Guardar reclamación	Clic	- Guarda el nivel de reclamación cuando está en modo edición	
Reclamación, estado de reclamación, primer informe completo	Clic	- Cambia el estado de la reclamación a "No designada" y crea el evento de primer informe completo	
Reclamación, estado de reclamación, designación completada	Clic	- Cambia el estado de la reclamación a "abierto" y crea el evento de designación completa	
Reclamación, estado de reclamación, cerrar	Clic	- Inicia el proceso cierre de reclamación	
Reclamación, estado de reclamación, reabrir	Clic	- Cambia el estado de la reclamación a "abierto"	
Reclamación, estado de reclamación, marcar error	Clic	- Marca la reclamación actual y todas sus líneas en error. Finaliza todos los participantes.	
Reclamación,	Clic	- Abre la ventana de	

asignar		asignación de reclamación
Reclamación, gestionar archivo físico	Clic	- Abre la ventana de archivo físico
Reclamación, declarar evento	Clic	- Abre la ventana de declarar evento
Reclamación, cerrar carpeta de reclamación	Clic	- Cierra la ventana de carpeta de reclamación actual
Editar, cortar	Clic	- Mueve el texto seleccionado al portapapeles Ctrl + X
Editar, copiar	Clic	- Copia el texto seleccionado al portapapeles Ctrl + C
Editar, pegar	Clic	- Pega el texto desde el portapapeles Ctrl + V
Vista, plegar todo	Clic	- Pliega el árbol de reclamación
Vista, expandir todo	Clic	- Expande el árbol de reclamación
Editar póliza	Clic	- Abre las pestañas de póliza en modo editar
Guardar póliza	Clic	- Guarda información de póliza y devuelve las pestañas al modo ver
Nuevo participante, reclamante	Clic	- Abre pestañas de participante en modo editar para introducir un nuevo nodo de nivel reclamante en el árbol de reclamación
Nuevo participante, asegurado	Clic	- Abre pestañas de participante en modo editar para introducir nuevo nodo de nivel asegurado en el árbol de reclamación
Nuevo participante, otros	Clic	- Abre pestañas de participante en modo editar para introducir una nueva entrada en la lista de participantes

Editar participante	Clic	- Pone las pestañas de participante actualmente seleccionadas en modo editar
Guardar participante	Clic	- Guarda la información cambiada en las pestañas de participante y devuelve la reclamación al modo ver
Borrar participante	Clic	- Borra participante seleccionado
Nueva línea	Clic	- Añade nueva línea al árbol de reclamación y abre pestañas de línea en modo editar
Editar línea	Clic	- Pone pestañas de línea en modo editar de manera que el usuario pueda cambiar los detalles de línea
Guardar línea	Clic	- Guarda información introducida en las pestañas de línea y devuelve la reclamación al modo ver
Cambiar estado de línea, cerrar	Clic	- Cambia el estado de una línea en la carpeta de reclamación a "cerrado"
Cambiar estado de línea, reabrir	Clic	- Cambia el estado de la línea seleccionada a "abierto"
Cambiar estado de línea, marcar error	Clic	- Marca línea seleccionada en error
Asignar línea	Clic	- Abre la ventana de asignación de reclamación
Asignar realizadores	Clic	- Abre la ventana de asignar realizadores
Realizadores, ver todo	Clic	- Visualiza todos los realizadores de la reclamación asignados a la reclamación en la IU ver

		realizador	
Utilidades, imprimir pantalla	Clic	- Imprime pantalla actual	Ctrl + P
Utilidades, ver asistente de tareas	Clic	- Abre la ventana del asistente de tareas para la reclamación actual	
Utilidades, cera nueva nota de archivo	Clic	- Abre la ventana de notas de archivo para la reclamación actual	
Utilidades, ver historial de reclamaciones	Clic	- Abre la ventana del historial de reclamaciones para la reclamación actual	
Utilidades, crear correspondencia	Clic	- Abre la ventana de formularios y correspondencia	
Versión 7, empujar reclamación	Clic	- Lanza V7 para iniciar el proceso de empujar	
Versión 7, deshacer empujar	Clic	- Devuelve la reclamación al estado previo a ser empujada	
Versión 7, realizar pago	Clic	- Abre la pantalla V7 PUEM en la ventana de emulador terminar si un reclamante o participante asociado a un reclamante se ha seleccionado. De lo contrario, visualiza una ventana que requiere que el usuario seleccione un reclamante	
Ayuda, contenidos	Clic	- Abre el archivo de ayuda en el menú de contenido	
Ayuda, buscar ayuda sobre	Clic	- Abre el archivo de ayuda en la ventana de búsqueda	
Ayuda, acerca de	Clic	- Abre la ventana que visualiza la información acerca de la aplicación	

Elementos de datos

Literal	Longitud	Tipo de control	Vínculo entidad datos	Normas edición	Tratamiento de errores
Árbol de reclamación		Vista de árbol			
- Póliza		Nodo de la vista de árbol	Número de póliza (póliza)		
- Asegurado		Nodo de la vista de árbol	Nombre preferido del participante (implicación en el seguro)		
- Reclamación		Nodo de la vista de árbol	Número de reclamación (reclamación)		
- Reclamante		Nodo de la vista de árbol	Nombre preferido del reclamante (implicación en el seguro)		
- Línea		Vista de árbol	Tipo de línea (línea)		
Cuadro desplegable de participantes		Vista desplegable	Nombre preferido y papel del participante (implicación en el seguro y papel de implicación)		

Puntos de confirmación

Reclamación opción de menú guardar– guarda todos los datos del nivel de reclamación

Póliza opción de menú guardar– guarda todos los datos del nivel de póliza

Participante opción de menú guardar– guarda todos los datos del nivel de participante

Línea opción de menú guardar– guarda todos los datos del nivel de línea

Reclamación opción de menú cerrar carpeta de reclamación – obliga al usuario a guardar cambios si está en modo editar

Historial de reclamación

Definición

El historial de reclamación muestra información en una interfaz de usuario que pretende incluir todos los elementos constitutivos de un archivo de reclamación. Puede buscarse en los cuatro tipos de historial incluidos en el componente mediante criterios de indexación comunes como participante, realizador, y fase de la reclamación. Puede producirse un informe de leyenda que muestre el historial seleccionado en un formato de documento.

Valor

El historial de reclamación proporciona a los usuarios una interfaz común a través de la cual ver una gran variedad de información sobre la reclamación. Incluye todo el historial disponible acerca de una reclamación y se expande a medida que se construyen las capacidades de la reclamación, como la captura de correo entrante. Los usuarios desarrollan vistas personalizadas del historial en base a cualquier criterio por el que pueda indexarse el historial, y estos informes se guardan como documentos de Word personalizables. La forma en que se indexa la información del historial proporciona acceso rápido a los datos pertinentes que se necesitan para responder a una variedad de consultas.

Usuarios clave

Todos los miembros de la organización de reclamación pueden usar el historial de reclamación como una forma para ver rápidamente la actividad realizada en una reclamación. Esta utilidad aumenta la habilidad para localizar información clave relativa a cualquier reclamación.

Funcionalidad de componente

El historial de reclamación es un componente que contiene un proceso simple para recuperar un historial a partir de otros componentes en el sistema. No contiene datos originales en sí mismo. Incluso ver un elemento de historial se hace en la ventana del componente en que se capturó en primer lugar el elemento.

El segundo proceso clave del historial de reclamación es producir un informe de leyenda de todos los elementos de historial según los elementos que el usuario quiere incluir.

Hay dos interfaces de usuario necesarias para este componente que corresponden a las dos funciones clave anteriores:

- Búsqueda en el historial de reclamación: esta ventana utiliza los campos fase de reclamación, participante, realizador y tipo de historial en cada registro de historial para ayudar al usuario a estrechar la búsqueda de historial específico.

- Informe de leyenda: este informe utiliza la funcionalidad de Word para producir un informe sobre cada elemento de historial que el usuario quiere ver y su detalle asociado. Puesto que el informe se produce en Word, puede personalizarse totalmente según múltiples necesidades diferentes.

Interfaces de usuario

- Búsqueda de historial de reclamación

- Informe de leyenda (documento de Word, no diseño IU).

Formularios y correspondencia

Definición

El componente formularios y correspondencia soporta comunicación y documentación de reclamación interna y externa a lo largo de todas las partes del proceso de tratamiento de reclamaciones.

La función formularios y correspondencia - crear correspondencia proporciona la habilidad de buscar una plantilla usando varios criterios de búsqueda, seleccionar una plantilla para su uso y entonces aprovechar los datos de reclamación en la plantilla seleccionada.

La función formularios y correspondencia - mantener plantilla es una herramienta para el bibliotecario para crear, borrar y actualizar plantillas de correspondencia y sus criterios asociados.

Algunos procesos específicos soportados por formularios y correspondencia son:

- Informar sobre la reclamación
    - a agencias estatales/federales, etc. en una primera noticia de pérdida
    - peticiones internas de información
- Aconsejar a participantes
- Contactar con participantes
- Realizar cálculos
- Crear correspondencia para reclamaciones o no reclamaciones

Valor

El componente formularios y correspondencia soporta al usuario a la hora de crear documentación.

Aprovechar la información de la reclamación directamente en la correspondencia reduce la cantidad de tecleo y dictado realizados para crear formularios y cartas. Los datos típicos disponibles en las plantillas deberían incluir: autor, remitente, número de reclamación, fecha de pérdida, nombre del asegurado, número de póliza, etc. Un bibliotecario añade y mantiene formularios y cartas estandarizados en grupos lógicos disponibles para toda la empresa.

Usuarios clave

Los empleados de reclamación son los principales usuarios del componente formularios y correspondencia, pero puede usarse por cualquiera que tenga acceso al sistema para crear documentos usando plantillas existentes.

Los bibliotecarios de formularios y correspondencia usan el sistema para crear, actualizar o eliminar plantillas.

Funcionalidad del componente

Formularios y correspondencia - crear correspondencia.

1. Buscar una plantilla basándose en criterios de búsqueda.
2. Crear una correspondencia a partir de una plantilla usando datos de la reclamación.
3. Crear una correspondencia a partir de una plantilla sin usar datos de la reclamación.
4. Ver los criterios para una plantilla seleccionada.
5. Ver la plantilla de Microsoft Word antes de aprovechar cualquier dato.

Formularios y correspondencia - mantener plantilla

1. Buscar una plantilla basándose en criterios de búsqueda.
2. Crear, duplicar, editar y borrar plantillas de correspondencia y sus criterios.
3. Probar y aprobar internamente plantillas recién creadas / editadas.
4. Copiar de manera adecuada plantillas de Word para distribución NAN.

Interfaces de usuario

- Búsqueda de correspondencia
- Detalles de correspondencia
- Campos asociados
- Búsqueda de mantener correspondencia
- Información de plantilla de correspondencia - pestaña de detalles

- Información de plantilla de correspondencia - pestaña de criterios

- Microsoft Word

Notas de archivo

Definición

Las notas de archivo capturan información textual que no puede reunirse en elementos de datos discretos como parte de la captación de datos de la reclamación. Son principalmente una herramienta de documentación, pero también se usan para comunicación interna entre profesionales de la reclamación. Los usuarios pueden clasificar las notas por participante o fase de la reclamación (médica, investigación, cobertura, etc.) para permitir una recuperación rápida y organización de esta información textual.

Valor

Las notas de archivo aceleran la recuperación y el informe de información de reclamación. Una utilidad de búsqueda de notas de archivo con múltiples criterios de indexación proporciona a los profesionales de la reclamación y supervisores la habilidad de encontrar rápidamente una nota de archivo escrita sobre una persona o asunto particular. La herramienta de notas de archivo utiliza capacidades de procesamiento de texto modernas que aceleran la introducción, reducen el error y permiten que se destaque información importante. Además, la categorización y la búsqueda de campos clave facilitan el proceso de encontrar y agrupar las notas de archivo. Finalmente, las notas de archivo mejoran la comunicación ya que pueden enviarse una y otra vez entre los implicados en gestionar la reclamación.

Usuarios clave

Todos los miembros de la organización de reclamaciones pueden utilizar notas de archivo. Partes externas pueden ver a través de RMS notas de archivo marcadas como General. Esta utilidad aumenta la habilidad de localizar información clave relativa a una reclamación. Cualquiera que quiera saber más sobre una reclamación o quiera grabar información sobre una reclamación utiliza la herramienta de notas de archivo.

Funcionalidad del componente

La búsqueda de notas de archivo se incluye como parte del componente del historial de reclamación que permite al usuario buscar los elementos históricos de un archivo de reclamación incluyendo tareas, cartas y eventos de cambio de reclamación significativos.

Las interfaces de usuario que se necesitan para este componente son:

- Búsqueda de notas de archivo (parte del componente historial de reclamación): esta ventana utiliza los campos de fase de la reclamación en el registro de las notas de archivo para ayudar al usuario a estrechar la búsqueda para notas de archivo específicas. Además, permite a los usuarios ver todas las notas de archivo que cumplen criterios especificados en un formato de estilo informe.

- Introducción de notas de archivo: la ventana usada para grabar la nota de archivo. Integra un sistema de procesamiento de texto y proporciona la habilidad de categorizar, indicar una nota como de la empresa (privada) o general (pública), guardar la nota como un borrador o una copia final y enviar la nota a otra persona.

Interfaces de usuario

- Notas de archivo

- Revisar borrador de nota de archivo

- Buscar participante

- Buscar realizador

Libro de direcciones

Definición

El libro de direcciones es la interfaz entre el sistema de reclamaciones y la base de datos de clientes. La aplicación de cliente es un nuevo componente diseñado para mantener un seguimiento de personas u organizaciones que interactúan con RELIANCE por cualquier motivo, aunque las reclamaciones son con la mayor probabilidad la primera aplicación para usar Cliente. Al libro de direcciones se accede directamente desde el escritorio y desde la carpeta de reclamación.

El libro de direcciones cumple varias necesidades en la organización de reclamaciones. Aunque su función principal es soportar la adición de participantes a una reclamación, actúa como pasillo hacia la base de datos de cliente para buscar participantes existentes, y añadir nuevas personas u organizaciones a la base de datos de la empresa.

La base de datos de cliente mantiene información de nombres, direcciones, números de teléfono y otra información que siempre se refiere a una persona u organización sin importar qué papel juegan en la reclamación.

Valor

El libro de direcciones proporciona una definición común de personas u organizaciones que interactúan con RELIANCE y por tanto proporciona un medio mucho más eficiente de capturar esta información. Cada entrada de la base de datos de cliente proporciona la habilidad de vincular una persona u organización a todos los papeles diferentes que desempeñan en la organización, y por tanto hace que la recuperación de información cliente por cliente sea rápida y fácil.

Hay muchos beneficios para RELIANCE por tener un libro de direcciones común. La información sobre personas y empresas se aprovecha en otras actividades como tareas habilitadas que buscan los números de teléfono de un cliente cuando debe hacerse una llamada. La información que se ha almacenado de manera redundante en el pasado puede introducirse una vez y reutilizarse. Una vez que todas las áreas de RELIANCE usen la aplicación de cliente, las diferentes áreas de la empresa pueden compartir definiciones de individuos y organizaciones.

Funcionalidad del componente

El libro de direcciones permite a los usuarios añadir, editar y borrar registros de la base de datos cliente. También proporciona una facilidad de búsqueda robusta, incluyendo búsquedas fonéticas por nombre para encontrar personas contenidas en la base de datos cliente.

Hay dos interfaces de usuario principales para el libro de direcciones:

- Encontrar entrada del libro de direcciones - ésta es una ventana de búsqueda que permite a un usuario encontrar registros en la base de datos de cliente usando nombres, direcciones, números de teléfono y otros identificadores. Desde esta ventana, pueden seleccionarse registros específicos y adjuntarse como participantes en reclamaciones.

- Mantener entrada del libro de direcciones - esta ventana permite a los usuarios añadir o editar información sobre un cliente especificando sus nombres, direcciones, números de teléfono, información de correo electrónico y números de identificación como SSN o TIN.

El libro de direcciones se crea al mismo tiempo que la aplicación cliente para asegurar que se sigue un enfoque de diseño consistente.

Usuarios clave

Todos los miembros de la organización de reclamaciones utilizan el libro de direcciones para buscar información sobre personas u organizaciones en la base de datos de cliente. Los que establecen y manejan reclamaciones usan el libro de direcciones para identificar participantes.

Interfaces de usuario

- Encontrar cliente

- Mantener cliente

Índice

Definición

El componente índice o búsqueda de reclamaciones proporciona la habilidad de localizar reclamaciones en el sistema usando varios criterios de búsqueda. Los criterios cubren una amplia variedad de capacidades de búsqueda que existen actualmente incluyendo, pero sin limitarse a, realizadores de reclamaciones, participantes, búsquedas fonéticas por nombre, direcciones, papeles, oficinas y líneas de negocio. Los resultados de búsqueda exhiben datos de la reclamación, el participante y el realizador seleccionados para ayudar a identificar cada reclamación.

El componente índice también permite una navegación fácil a varios componentes de reclamación como la carpeta de reclamaciones, una vez identificada una reclamación. Puede accederse al mismo desde el escritorio y desde cualquier carpeta de reclamación abierta.

El componente de índice está diseñado para soportar varios procesos de negocio en la organización de reclamación. Sus funciones son críticas para mejorar la productividad del personal de reclamación y el servicio al cliente en las siguientes áreas:

- Asociar Correo

Las capacidades de la búsqueda índice hacen más fácil identificar la reclamación a la que pertenece un correo basándose en criterios usados para identificar reclamaciones en formularios, correspondencia y facturas. Los realizadores de una reclamación también pueden identificarse con fines de encaminamiento de correos.

- Consultas telefónicas

Esta ventana es el punto principal para tratar consultas telefónicas entrantes para cualquier reclamación. Los usuarios pueden encontrar las reclamaciones rápidamente sin tener que saturar a la persona que llama con preguntas sobre información adicional.

- Duplicar reclamaciones

Antes de abrir nuevas reclamaciones, pueden realizarse comprobaciones para asegurarse de que la reclamación no se ha introducido ya en el sistema. Las capacidades de búsqueda adicional proporcionan una mayor seguridad de que no se introducirán reclamaciones duplicadas. Esto reduce la necesidad de borrar o fusionar registros de reclamación.

- Identificación de fraude

Al poder buscarse las reclamaciones fácilmente por participante y otros criterios, pueden buscarse fácilmente preguntas fraudulentas. Sin embargo, esta no es la finalidad principal de este componente.

Valor

El índice reduce el tiempo necesario para encontrar reclamaciones existentes y también reduce el potencial trabajo repetido de no encontrar reclamaciones cuando se necesita para asociar correos o comprobaciones de duplicados.

Usuarios clave

Los empleados de la reclamación son los principales usuarios de la ventana de índice, aunque puede usarse por cualquiera que tenga acceso al sistema para acceder a reclamaciones sin tener que memorizar números de seguimiento.

Funcionalidad del componente

El índice es principalmente un motor de búsqueda robusto que busca rápida y eficazmente reclamaciones. No es un componente que almacene sus propios datos, ya que está centrado principalmente en dirigir a los usuarios más rápida y directamente a los datos de la reclamación.

El índice se compone de una ventana de búsqueda que sigue el formato de todas las demás ventanas de búsqueda en el sistema.

Interfaces de usuario

- Encontrar reclamaciones

Lesión

Definición

El componente lesión captura versiones de las lesiones de un demandante a medida que evolucionan. Esta ventana captura información sobre lesiones en forma de campos de datos discretos, reduciendo la necesidad de notas de archivo de texto de formato libre. Capturar los datos en lugar de texto, permite seguir de cerca la lesión e informar de ella rápidamente. Los datos también sirven como estadísticas de retroalimentación, es decir, para construir mejores prácticas de reclamación y en la selección de riesgos. El método preferido para identificar y documentar lesiones es el código ICD-9. El usuario puede introducir o buscar el código ICD-9 usando descriptores o números.

Valor

Los datos en cada lesión se capturan y resumen en un formato accesible y consistente, haciendo que se tarde considerablemente menos en grabar y revisar el caso y se haga de forma más organizada, permitiendo al tasador centrarse en los resultados deseados. Esta “foto instantánea” del estado actual y el historial de una lesión facilita enormemente rechazar o transferir archivos entre profesionales de la reclamación. Además, la captura en campos de

datos discretos permite el uso de eventos para identificar puntos de acción en el ciclo de vida de una reclamación que incluye lesiones.

Usuarios clave

Todos los miembros de la organización de reclamaciones pueden utilizar el componente lesión. Este componente aumenta la habilidad de localizar y resumir información clave relativa a una lesión.

Funcionalidad del componente

Las lesiones son un aspecto de la información del participante que está relacionada con los participantes que reclaman en la reclamación. El componente de participante relaciona clientes con todas las demás entidades relacionadas con la reclamación. La información sobre lesiones se relacionará con los registros de participante y se visualizarán en la información del nivel participante en la carpeta de reclamación. Son necesarias nuevas entidades para implementar la captura de datos de lesión: búsqueda de lesión e ICD-9. El componente de lesión interactúa con otros cinco componentes: la carpeta de reclamación, que contiene datos de gestión de discapacidad sobre un reclamante; participante, que enumera los individuos asociados con la reclamación; así como notas de archivo, asistente de tareas y procesador de eventos. El componente de lesión también usa Microsoft Word para crear informes de lesión históricos, formateados, para una persona particular.

Las interfaces de usuario necesarias para este componente son:

- Lesión: se trata de la ventana de lesión principal que captura datos de informe de lesión básicos, incluyendo: la fuente del informe de lesiones, la fecha del informe de lesiones, un indicador del historial médico anterior, y después una lista detallada de las lesiones asociadas a ese informe. La lista detallada incluye campos discretos para los siguientes datos. código ICD-9, parte del cuerpo, tipo, clase, gravedad, tratamiento, diagnóstico, un campo de descripción de texto libre y un indicador de la relación causal.

- ICD-9: ésta es la ventana de búsqueda para localizar códigos ICD-9 y descripciones asociadas

- Gestión de discapacidad: esta ventana contiene un subconjunto de campos de datos de participante que permite una gestión más eficaz de la lesión.

Interfaces de usuario

- Carpeta de reclamación - nivel participante - pestaña de lesión

- Ventana de búsqueda ICD-9

- Carpeta de reclamación - nivel participante - pestaña de gestión de discapacidad

Negociación

Definición

La figura 10 es una ilustración del componente negociación de una realización de la presente invención. La negociación proporciona una única plantilla estructurada que se complementa con vistas de soporte, para capturar eventos relativos a una negociación. La interfaz 1000 de negociación captura elementos clave de una negociación, tales como un intervalo objetivo de acuerdo, demandas y ofertas actuales y puntos fuertes de apoyo y afirmaciones de oposición de la reclamación. La información de negociación se reúne en elementos 1002 de datos discretos, que permiten la capacidad de generar eventos 1006 basados en atributos clave o cambios en una negociación. Estos eventos 1006 se envían entonces a una cola 1008 de eventos común. El componente 1000 de negociación hace interfaz con el componente 1004 de notas de archivo para proporcionar capacidad de documentación adicional, en un formato no estructurado. La plantilla de negociación se soporta en todos los demás datos contenidos en la carpeta de reclamación.

Valor

Los datos en todos los casos se resumen en un formato accesible y consistente, haciendo que la grabación y la revisión del caso lleven considerablemente menos tiempo y sea más organizada, permitiendo al tasador centrarse en la estrategia de negociación y en los resultados deseados. Esta “foto instantánea” del estado actual facilita enormemente el rechazo o la transferencia de archivos entre profesionales de la reclamación. Además, la captura en campos de datos discretos permite el uso de eventos para identificar puntos de acción en una negociación.

Usuarios clave

Todos los miembros de la organización de reclamación pueden utilizar la negociación. Este componente aumenta la habilidad de localizar y resumir información clave relativa a una negociación.

Funcionalidad del componente

La negociación es un tipo de actividad de resolución que forma parte del componente de reclamación del modelo de entidad de reclamación. El componente de reclamación es el objetivo central del modelo de entidad de reclamación, porque contiene la información esencial sobre una reclamación. El componente de reclamación soporta la funcionalidad de captura de datos de reclamación central, los procesos de primera noticia, y actividad de resolución para reclamaciones. Los principales tipos/clases de datos en el componente de reclamación son: reclamación, reclamante, línea, historial de la reclamación, actividad de resolución, elemento de reserva y cambio del elemento de reserva. Se necesitan tres entidades para implementar una negociación: actividad de resolución, reclamación e historial de la reclamación. También hay interacción entre el componente de negociación y los componentes asistente de tareas, notas de archivo y procesador de eventos.

Las interfaces de usuario necesarias para la negociación son:

- Negociación: esta ventana captura datos de oferta y de demanda, incluyendo: cantidad, fecha, tipo y modo de comunicación. El intervalo de acuerdo objetivo, el más bajo y el más alto, se captura, junto con los puntos fuertes y débiles del caso.

Interfaces de usuario de soporte, que también forman parte de la carpeta de reclamación, incluyen:

- Responsabilidad (pestaña de nivel de reclamación): esta ventana se usa para documentar factores de responsabilidad en la evaluación y el establecimiento de precios de una reclamación. Los factores de responsabilidad incluyen el porcentaje de responsabilidad para todas las partes implicadas; forma de negligencia que prevalece para esa jurisdicción; teorías de responsabilidad que la persona que trata la reclamación cree que son aplicables a la reclamación. Usada antes de desarrollar la estrategia de negociación.

- Daños (pestaña en el nivel de línea): esta ventana proporciona la capacidad de establecer precios y evaluar una reclamación basándose en daños incurridos y esperados. Usada antes de desarrollar la estrategia de negociación.

Interfaces de usuario

- Carpeta de reclamación - nivel de línea - pestaña de negociación

- Carpeta de reclamación - nivel de reclamación - pestaña de responsabilidad

- Carpeta de reclamación - nivel de línea - pestaña de daños

Organización

Definición

La figura 11 es un diagrama de flujo de las operaciones utilizadas por el componente de organización según una realización de la presente invención. El componente 1100 de organización permite almacenar, buscar y reutilizar la información común a todas las personas que realizan un trabajo en las reclamaciones en todas las reclamaciones en las que están trabajando.

En una realización del componente 1100 de organización, todos los registros de empleado se mantienen en una base 1102 de datos común de modo que pueden adjuntarse a las reclamaciones específicas en las que están trabajando, situadas en una base 1104 de datos de reclamación. La información común que se mantiene en el registro de empleado incluye nombre, localización, teléfono y alguna información contextual organizativa mínima como oficina o departamento. Este es el requisito mínimo para soportar el seguimiento de realizadores en reclamaciones. La información 1102 de empleado se vincula 1106 entonces a la información 1104 de reclamación y se actualizan las bases 1108 de datos. Al vincular los empleados 1102 con las reclamaciones 1104 en las que están trabajando, la base de datos puede buscar por empleado o por reclamación 1110.

Sin embargo, esta versión de la organización puede expandirse para incluir relaciones de organización (específicamente hacer un seguimiento de dónde se sitúa un empleado en la estructura de la organización), grupos de personas como realizadores para la designación de reclamaciones y la asignación de reclamaciones dentro de la estructura de organización. Estas capacidades deben soportar cualquier noción de análisis del número de casos, informes de gestión o asignación automática que sería necesario incluir.

Valor

Al hacer un seguimiento de definiciones comunes de empleados en las reclamaciones, las capacidades de indexación se mejoran y puede hacerse un seguimiento preciso de los realizadores en las reclamaciones.

Usuarios clave

Los usuarios principales de las capacidades de organización son el personal administrativo que establece realizadores; así como los técnicos que hacen un seguimiento de quién está trabajando en una reclamación.

Funcionalidad del componente

El diseño de mínimo alcance del componente de organización incluye una ventana de búsqueda para encontrar empleados en la organización y una ventana de detalle para ver información específica sobre cada empleado.

Interfaces de usuario

- Búsqueda de entidad de organización
- Añadir/editar entidad de organización

Participante

Definición

La figura 12 es una ilustración del componente de participante según una realización de la presente invención. El participante 1200 proporciona el enlace entre reclamaciones y personas y organizaciones almacenadas en la base de datos de cliente y a las que se accede a través del libro 1202 de direcciones. El participante enlaza clientes con reclamaciones 1204 definiendo los papeles que desempeñan, por ejemplo, reclamante, conductor o doctor. Reutiliza la información contenida en el libro 1202 de direcciones de modo que no tiene que volver a introducirse para cada participante.

El componente de participante también permite realizar enlaces 1206 entre participante y varios elementos en las reclamaciones. Puede enlazarse un doctor al reclamante que trata y puede enlazarse un conductor al vehículo dañado que estaban conduciendo.

Una vez añadido un participante a una reclamación, puede adjuntarse información 1208 adicional que es específica a esa reclamación. Esta información incluye lesión, empleo, y muchos otros tipos de información que son específicos del papel que desempeña esa persona u organización en una reclamación.

Los procesos de negocio soportados principalmente por el participante 1200 son:

- Registrar la implicación en una reclamación

Hay un requisito de captura de datos básicos para hacer un seguimiento de personas y organizaciones implicadas en una reclamación, y esto se hace de manera más eficaz usando el enfoque de participante.

- Registrar información específica del papel

El libro 1202 de direcciones almacena información que puede reutilizarse en las reclamaciones, pero el componente 1200 de participante debe mantener la información que es específica de la implicación de una persona o una organización en una reclamación específica.

- Ponerse en contacto con clientes

Dado que el participante está relacionado con el libro 1202 de direcciones común, cualquier información de contacto contenida en el mismo puede obtenerse rápida y fácilmente.

- Formularios y correspondencia 1210

Aprovechar la información de correspondencia en las cartas proporciona una habilidad de eficiencia a todos los usuarios que no necesitan buscar información de nombre y dirección.

- Categorizar información de historial

Los participantes se usan para categorizar elementos del historial como tareas y notas de archivo de modo que la información relativa a un único participante en una reclamación puede recuperarse fácilmente.

- Indexación de reclamación

Adjuntar participantes a una reclamación permite que el componente de índice sea más eficaz en el procesamiento de consultas de la reclamación.

Usuarios clave

Los usuarios principales de los componentes 1200 de participante son los que trabajan directamente en el procesamiento de reclamaciones. Son los que mantienen las relaciones de participante.

Los profesionales de la reclamación que tratan lesiones usan la pestaña de participante en la carpeta de reclamación para hacer un seguimiento de las lesiones y gestionar discapacidades para un mejor resultado en la reclamación.

Valor

Dado que el componente 1200 de participante solo trata de definir los papeles que desempeñan personas y organizaciones en todas las reclamaciones, no hay introducción redundante de la información de nombre, dirección y teléfono. Esto está almacenado todo en el libro 1202 de direcciones.

El número de papeles de participante potenciales que pueden definirse es prácticamente ilimitado, y por tanto expansible, según debe capturarse la implicación de personas y organizaciones adicionales.

Funcionalidad del componente

La mayor parte de la funcionalidad de participante se ejecuta en el contexto de la carpeta de reclamación. La carpeta de reclamación contiene niveles de participante de dos formas. En primer lugar, los reclamantes se muestran en el árbol de reclamación a la izquierda de la ventana. Por debajo de esto, se muestran otros participantes en una lista. Al seleccionar cualquier participante se muestra un conjunto de pestañas de información del participante que muestran la siguiente información:

- Detalles del participante - información básica sobre el papel que desempeña un participante en una reclamación y todos los demás participantes asociados al mismo.

- Información de contacto - Información del libro de direcciones sobre nombres, direcciones y números de teléfono.

- Lesión - información específica de la naturaleza de las lesiones sufridas por los reclamantes lesionados.

- Gestión de discapacidad - información sobre reclamantes lesionados con discapacidades.

Sólo las dos primeras pestañas se mostrarán de manera consistente para todos los participantes. Otras pestañas pueden aparecer basándose en el papel y las características de la implicación de un participante en una reclamación.

Añadir o editar información sobre el papel del participante se hace actualmente a través de la ventana de búsqueda del libro 1202 de direcciones. El proceso es tan simple como encontrar el registro del libro 1202 de direcciones para el participante buscado y especificar el papel que el participante desempeña en la reclamación. Una vez hecho esto, el participante se mostrará en la carpeta de reclamación, y puede añadirse información adicional.

La noción de un participante es un concepto genérico que no es específico solamente de las reclamaciones. Se basa en un patrón de diseño que puede expandirse a medida que se construyen capacidades de reclamación adicionales. Cualquier implicación de una persona u organización puede modelarse de esta forma.

Interfaces de usuario

- Nivel de participante - pestaña de detalles de participante

- Nivel de participante - pestaña de información de contacto

- Nivel de participante - pestaña de eventos

- Nivel de participante - pestaña de lesión (componente de lesión)

- Nivel de participante - pestaña de gestión de discapacidad (componente de lesión)

- Vista de lista de participantes

Realizador

Definición

El componente de realizador permite que entidades organizativas (personas, grupos, oficinas, etc.) se asignen a varios papeles en el tratamiento de la reclamación desde el informe hasta la resolución. El componente de realizador se utiliza en un planteamiento a nivel de reclamación.

Un realizador se define como cualquier persona u organización que puede designarse para cumplir un papel en una reclamación.

El componente de realizador soporta los procesos de asignación en el proceso de tratamiento de la reclamación. Esto va más allá de la asignación de la reclamación en FNOL. Este componente también permite la asignación de trabajo (tareas).

Algunos procesos específicos soportados por el realizador son:

- Asignar reclamaciones

Identificación de papeles diferentes en las reclamaciones para asignar la reclamación (iniciar reclamación - trabajo de proceso DC)

- Mantiene papeles y relaciones de los realizadores en las reclamaciones

- Asignar tareas

- Reasignaciones

- Soporta el proceso de inicio de reclamación - asignación

- Mecanismo de búsqueda de empleados, oficinas

- Todos los realizadores deberían estar en el componente de organización

- Proporciona historial de asignaciones

Valor

El componente de realizador permite la asignación de papeles o tareas a personas o grupos. Los datos sobre los realizadores residen en un almacén común: el componente de organización.

El componente de realizador reduce el tiempo requerido para encontrar empleados, equipos o cualquier realizador potencial, y garantiza la consistencia de los datos.

Usuarios clave

Los usuarios principales del componente de realizador son aquellos que trabajan directamente en el procesamiento de reclamaciones. Son los que mantienen la asignación de papeles o tareas relacionados con una reclamación.

Funcionalidad del componente

El componente de realizador soporta una función informativa y una función de asignación

1. Vista de detalles para realizadores (empleado, oficina, unidad, etc.). Estos detalles pueden sugerir relaciones de entidad organizativa pero no de un modo que las defina o las mantenga.

2. Vista de todos los realizadores asignados a una reclamación, actualmente y en el pasado (incluye personas, grupos, oficinas, etc.)

3. Asignar realizadores a una reclamación - en el nivel de reclamación, niveles de reclamante, y de suplemento (incluyendo personas, oficinas, grupos, etc.)

Interfaces de usuario

- Asignar realizador

- Papeles de los realizadores

- Vista de lista de realizadores

Asistente de tareas

Definición

El asistente de tareas es la piedra angular de un entorno de trabajo para un profesional de las reclamaciones. Proporciona funciones de agenda en un nivel de etapas de trabajo que permiten la gestión de eventos de reclamación

complejos. Permite la ejecución consistente de las mejores prácticas de reclamación al reunir y volver a reunir todas las tareas que deben realizarse para una reclamación basándose en características de reclamación detalladas. Estas características se derivan de requisitos de conformidad a las normas, compromisos del servicio de cuentas, y las mejores prácticas para el tratamiento de todo tipo de reclamaciones. El asistente de tareas también proporciona mecanismos que automatizan una parte o todo el trabajo en la realización de una tarea para ayudar al profesional de las reclamaciones a la hora de completar su trabajo. Una vez completada una tarea, el asistente de tareas genera un registro histórico para documentar las acciones del elemento de tratamiento de la demanda.

El asistente de tareas es ...

- Un método para garantizar la ejecución consistente de requisitos de normas, compromisos del servicio de cuentas y mejores prácticas de tratamiento de reclamaciones
- Una fuente de ayuda automatizada para profesionales de la reclamación
- Una herramienta de comunicación para toda la organización en el contexto de una reclamación (no sustituye a Lotus Notes).
- Un mecanismo para que la estrategia de reclamaciones sea una práctica común y compartir experiencia corporativa
- Una aplicación de agenda para hacer un seguimiento de las reclamaciones
- Una herramienta de seguimiento histórico
- Una forma de obtener la atención de un profesional o el jefe de un equipo de una demanda
- Un mecanismo para hacer cambios del proceso en la organización rápidamente

En el asistente de tareas, los profesionales de la reclamación tienen el control definitivo para determinar si y cuándo deben completarse las tareas. También tienen la habilidad de añadir tareas a la lista para representar un trabajo que hacen y que no está reflejado en las definiciones estándar de tareas en el sistema. Esto soporta una visión del profesional de las reclamaciones como un trabajador entendido que dedica la mayor parte de su tiempo a centrarse en un resultado exitoso a lo largo de la investigación, evaluación y negociación del mejor resultado posible.

Valor

El asistente de tareas reduce el tiempo necesario para tratar una reclamación al proporcionar al profesional de las reclamaciones la planificación automática de la actividad de la reclamación. Ayuda al profesional de las reclamaciones a recordar, realizar y registrar tareas completadas para cada reclamación. Las tareas completadas se documentan automáticamente y siguen siendo parte del historial de la reclamación.

El asistente de tareas también garantiza el tratamiento consistente de reclamaciones en toda la organización, y al hacer eso, puede tener un impacto significativo en los costes de gastos y pérdidas. Además, ayuda a garantizar la conformidad a normas y el cumplimiento de promesas de cuentas. Ayuda al trabajo en equipo requerido a la hora de tratar reclamaciones difíciles como un mecanismo de comunicación de estructuras.

Las habilitaciones automáticas de tareas reducen la cantidad de tiempo que los profesionales de la reclamación tienen que perder en actividades de poco valor añadido tales como escribir correspondencia. Por lo tanto pueden gastar mucho más tiempo en investigar, evaluar y negociar cada reclamación.

Usuarios clave

Aunque los profesionales de la reclamación son los principales usuarios del asistente de tareas, otros también usan la aplicación. Todo el departamento de reclamaciones utiliza el asistente de tareas para estructurar el trabajo y comunicarse entre sí. Los jefes de equipo utilizan el asistente de tareas para llevar a cabo revisiones de archivos y para guiar el trabajo del profesional de las reclamaciones. El personal administrativo usa el asistente de tareas como un medio para recibir trabajo y comunicar que se ha completado ese trabajo. Los procedimientos de la reclamación usan el asistente de tareas para completar el trabajo y para pedir asistencia de los jefes de equipo y los profesionales de la reclamación especiales.

El asistente de tareas requiere un nuevo tipo de usuario para establecer y mantener una variedad de tareas que se crean. Un bibliotecario de tareas mantiene la biblioteca de tareas, que contiene la lista de todas las tareas estandarizadas en toda la organización. El bibliotecario define normas para hacer que las tareas se sitúen en listas de tareas basándose en características de la reclamación, fechas que definen cuándo deben realizarse las tareas y habilitación de tareas a través de otras aplicaciones.

Funcionalidad del componente

La figura 13 es un diagrama de flujo de las operaciones utilizadas por el componente de asistente de tareas de la presente invención. El procesamiento de tareas a través del asistente de tareas comprende el ciclo de vida de la tarea desde su creación hasta completarse o borrarse. En una primera operación 1300, el motor de tareas proporciona tareas al asistente de tareas. En la segunda operación 1302, el asistente de tareas visualiza entonces la lista de tareas proporcionadas por el motor de tareas. En la tercera operación 1304, se permite al usuario añadir tareas y editar tareas proporcionadas por el motor de tareas. La cuarta operación 1306 tiene lugar cuando se procesa la reclamación. Cuando se procesa la reclamación, el usuario y el motor de tareas determinan cuándo se completan las diversas tareas. Cuando se completa una tarea, tiene lugar la quinta operación 1308. En la quinta operación 1308, se genera un registro histórico para cualquier tarea que se haya determinado que esté completa.

Las interfaces de usuario clave para este componente son:

- Asistente de tareas: es la utilidad que soporta la ocupación, ejecución y seguimiento histórico de tareas. Permite a los usuarios realizar tareas, completar tareas y eliminar tareas que se han añadido automáticamente.

- Plan de trabajo de tareas: esta interfaz de usuario permite al usuario trazar una estrategia del plan para una reclamación específica. Muestra tareas adjuntadas a sus niveles respectivos de la reclamación incluyendo líneas, participantes y la propia reclamación.

- Ventana de habilitación de tareas: hay muchas ventanas que pueden añadirse para habilitar tareas con otras aplicaciones tales como soporte telefónico, formularios y correspondencia, y notas de archivo. El número de habilitaciones de tarea posibles es prácticamente ilimitado.

- Introducción de tareas: permite a un usuario añadir nuevas tareas que no se añadieron automáticamente a la lista de tareas para cubrir situaciones en las que el elemento de tratamiento de reclamaciones quiere indicar un trabajo a realizar que no está reflejado por las definiciones estándares de tarea en la biblioteca de tareas.

Por detrás del funcionamiento del asistente de tareas, el motor de tareas evalúa de forma continua mensajes enviados desde otros componentes y determina basándose en las normas establecidas por el bibliotecario de tareas, qué tareas deberían ocuparse en el asistente de tareas. Los mensajes se envían al asistente de tareas cuando algo significativo ocurre en otro componente. Los mensajes contienen las características que el motor de tareas necesita evaluar para disponer las tareas adecuadas en la lista de tareas.

Interfaces de usuario

- Asistente de tareas
- Tarea de reasignación
- Tarea editar/añadir
- Tarea borrar
- Marcar tarea errónea
- Construir plan de trabajo
- Búsqueda de participantes
- Número de teléfono de participante
- Tarea de telefonear
- Perfil personal
- Búsqueda de cuentas
- Búsqueda de organización
- Búsqueda de realizador

Procesador de eventos/motor de tareas

Definición

La figura 14 es una ilustración del procesador 1400 de eventos en combinación con otros componentes del sistema según una realización de la presente invención. El procesador 1400 de eventos trabaja por detrás de las escenas de todas las aplicaciones de reclamación para escuchar eventos significativos que hayan ocurrido en la vida de varias entidades en el sistema como reclamaciones (pero potencialmente muchos más, como cuentas o pólizas en el futuro). Determina cuál debería ser la respuesta para cada evento y la pasa al componente de sistema que la procesará. El procesador de eventos es totalmente genérico para cualquier entidad o evento específico en el sistema y por tanto permite la automatización basándose en un número casi ilimitado de eventos y respuestas que podrían definirse.

La figura 15 es una ilustración del motor 1404 de tareas según una realización de la presente invención. El motor 1404 de tareas procesa el conjunto de respuestas de evento más común, los que necesiten generar tareas 1406 basándose en eventos 1006 que han ocurrido. Compara las tareas que se han definido para el sistema con un conjunto de criterios de reclamación para decir qué tareas deberían añadirse y qué tareas deberían marcarse ahora como completadas.

La única interfaz que el usuario ve en estos componentes es la biblioteca 1500 de tareas, que permite a los bibliotecarios 1502 de tareas definir las tareas y las normas que las crean, que se usan por el motor 1404 de tareas. El trabajo con estos componentes es casi totalmente una función realizada por especialistas que entienden la complejidad de las normas implicadas en garantizar que los eventos 1006 y las tareas 1406 se traten de manera adecuada.

El procesador 1400 de eventos también gestiona la sincronización de comunicación y datos entre nuevos componentes de reclamación y sistemas de reclamación LEGACY. Este único punto de contacto encapsula de manera eficaz los complejos procesos de traslación y notificación de eventos entre los dos sistemas.

Valor

La determinación automática de respuestas a eventos proporciona enormes beneficios a los usuarios del sistema al reducir el mantenimiento que deben realizar para asegurar la correcta disposición de reclamaciones.

Los usuarios desencadenan eventos mediante los datos que introducen y las actividades de sistema que realizan, y el sistema responde automáticamente con actividades automáticas apropiadas como la generación de tareas.

Las normas de generación de tareas definidas en la biblioteca de tareas proporcionan una definición extremadamente flexible de procesos de tratamiento de reclamaciones limitada sólo por los datos disponibles en el sistema en los que pueden basarse las normas de creación de tareas. Los bibliotecarios pueden implementar cambios de proceso rápidamente y pueden ejecutarse a través del asistente de tareas.

Usuarios clave

Aunque todo el personal de reclamaciones se beneficia directamente del funcionamiento del procesador de eventos y el asistente de tareas, sólo usuarios especialmente formados controlan el procesamiento de estos componentes. Los bibliotecarios de tareas que usan la interfaz de usuario de la biblioteca de tareas tratan el proceso de definir nuevas tareas y las normas que las desencadenan en el motor de tareas.

El personal de operaciones que garantiza que todos los eventos se procesan correctamente y que los recursos de sistema apropiados están disponibles para gestionar el rendimiento, tratan el procesamiento de eventos.

Funcionalidad del componente

Tal como se muestra en la figura 14, el procesador 1400 de eventos utiliza una cola 208 común de eventos 1006 que se ocupan por un componente 1402 del sistema para identificar qué eventos han tenido lugar. Al trabajar sobre esta cola, el procesador de eventos determina la respuesta adecuada para un evento y proporciona información a otros componentes que deben procesarlos. El procesador de eventos no procesa eventos por sí mismo y mantiene una encapsulación clara de las responsabilidades del sistema. Por ejemplo, un evento que afecta a datos de una reclamación se procesa por el componente de reclamación.

El motor 1404 de tareas sigue un proceso de evaluación de eventos 1006, determinando características de reclamación y asociando las características de la reclamación a tareas definidas en la biblioteca 1500 de tareas.

La interfaz de usuario clave para el motor 1404 de tareas es la biblioteca 1500 de tareas. La biblioteca 1500 de tareas mantiene las plantillas que contienen los campos y valores con los que se han establecido las tareas. Una plantilla de tarea puede contener instrucciones como “cuando el evento = litigio Y línea de negocio = vehículo comercial, entonces...”. Las plantillas también identifican cuál debería ser la fecha prevista para las tareas y cómo se habilita la tarea con otras aplicaciones.

Interfaces de usuario

- Buscar plantilla de tarea
- Buscar plantillas desencadenantes
- Detalles de la plantilla de tarea

Aunque se han descrito anteriormente varias realizaciones, debería entenderse que se han presentado a modo de ejemplo únicamente y no como limitación. Por tanto, el alcance de la invención no debería estar limitado por ninguna de las realizaciones anteriormente descritas a modo de ejemplo, sino que sólo se definen según las siguientes reivindicaciones.

REIVINDICACIONES

1. Aparato para hacer funcionar un software basado en componentes que puede seleccionar tareas, incluyendo dicho aparato un ordenador y comprendiendo:

un componente (906) de datos que hace que el ordenador almacene, recupere y manipule datos utilizando una pluralidad de funciones; y

un componente de cliente que incluye:

un adaptador (208) de componente de cliente que hace que el ordenador transmita y reciba datos hacia/desde el componente de datos;

un objeto (207) de negocio que hace que el ordenador proporcione datos memorizados en caché e incluye una lógica para hacer que el ordenador manipule lo datos;

un controlador (206) de interfaz de usuario que hace que el ordenador trate eventos generados por un usuario utilizando el objeto (207) de negocio para memorizar datos en caché y el adaptador (208) de componente de cliente para persistir datos en última instancia a un almacén (908) de datos,

en el que el componente de datos hace que el ordenador permite a un usuario definir tareas que deberá realizar un empleado que logran un objetivo al completarse, que permite al usuario introducir normas que dictan qué tareas deberían seleccionarse basándose en eventos generados por el usuario predeterminados, que recibe al menos un evento, y extraer y visualizar la tarea que se ha seleccionado basándose en el evento recibido según las normas definidas por el usuario.

2. Aparato según la reivindicación 1, en el que el componente de cliente hace además que el ordenador indique qué tareas se han completado.

3. Aparato según la reivindicación 1, en el que el evento recibido se proporciona desde una cola de eventos.

4. Aparato según la reivindicación 3, en el que la cola de eventos está ocupada con eventos de otros componentes de un sistema.

5. Aparato según la reivindicación 3, en el que la cola de eventos está ocupada con eventos de otras aplicaciones.

6. Aparato según la reivindicación 1, en el que el objetivo está relacionado con los seguros.

7. Aparato según la reivindicación 1, en el que las tareas extraídas se proporcionan a un asistente de tareas.

8. Aparato según cualquiera de las reivindicaciones 1 a 7, que además comprende:

un segmento (204) de código de un formulario de interfaz de usuario que hace que el ordenador recoja datos de una entrada de usuario; y

un segmento (208) de código de un adaptador de componente de cliente adaptado para transmitir datos a un componente (222) de servidor;

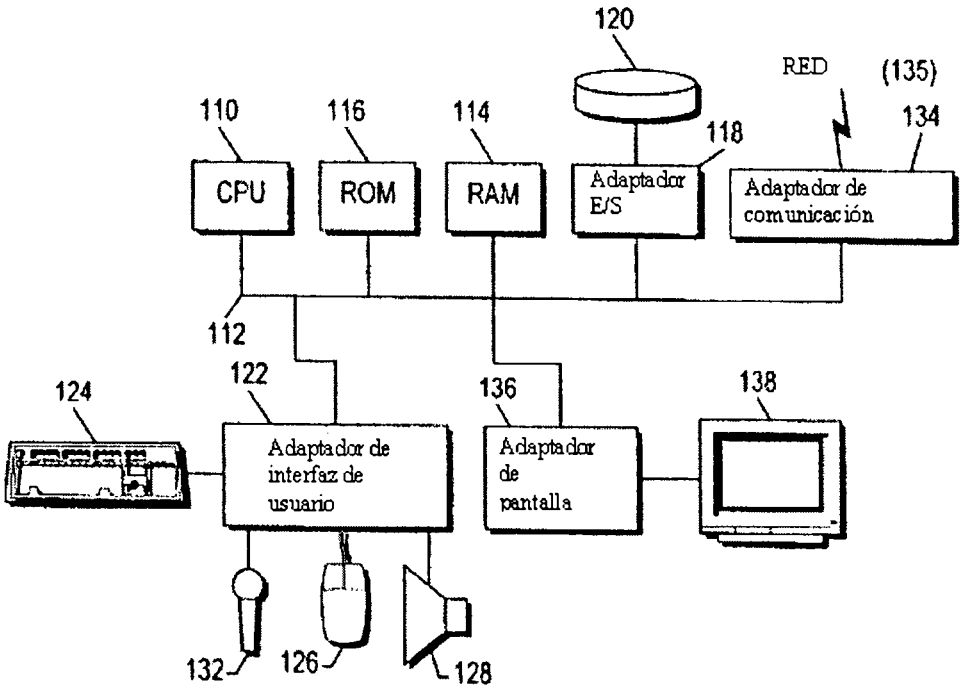
en el que dicho controlador (206) de interfaz de usuario comprende un segmento de código de controlador de interfaz de usuario que hace que dicho ordenador trate eventos generados por el usuario interactuando con el segmento (204) de código del formulario de interfaz de usuario para proporcionar la validación en una unidad lógica de trabajo, conteniendo la lógica para interactuar con uno o más objetos (207) de negocio, para interactuar con el adaptador (208) de componente de cliente, para añadir, recuperar, modificar o borrar objetos (207) de negocio, y para proporcionar un procesamiento de indicación sucia para notificar a un usuario el procesamiento de cambio.

9. Aparato según cualquiera de las reivindicación 1 a 8, para permitir la comunicación entre una pluralidad de clientes y un servidor con el fin de tratar tareas, que comprende:

uno o más componentes de cliente incluidos en cada cliente, haciendo cada componente de cliente de cada cliente que el ordenador se comunique y manipule datos con un primer tipo de datos;

uno o más componentes (222) de servidor que hacen que el ordenador se comunique y manipule datos con un segundo tipo de datos; y

uno o más adaptadores (208) de componente de cliente incluidos en cada cliente que hacen que el ordenador traduzca datos desde el uno o más componentes de cliente al segundo tipo de datos al comunicar datos desde el cliente al servidor y traduciendo además datos desde el uno o más componentes de servidor al primer tipo de datos al comunicar datos desde el servidor al cliente.



Técnica anterior Figura 1

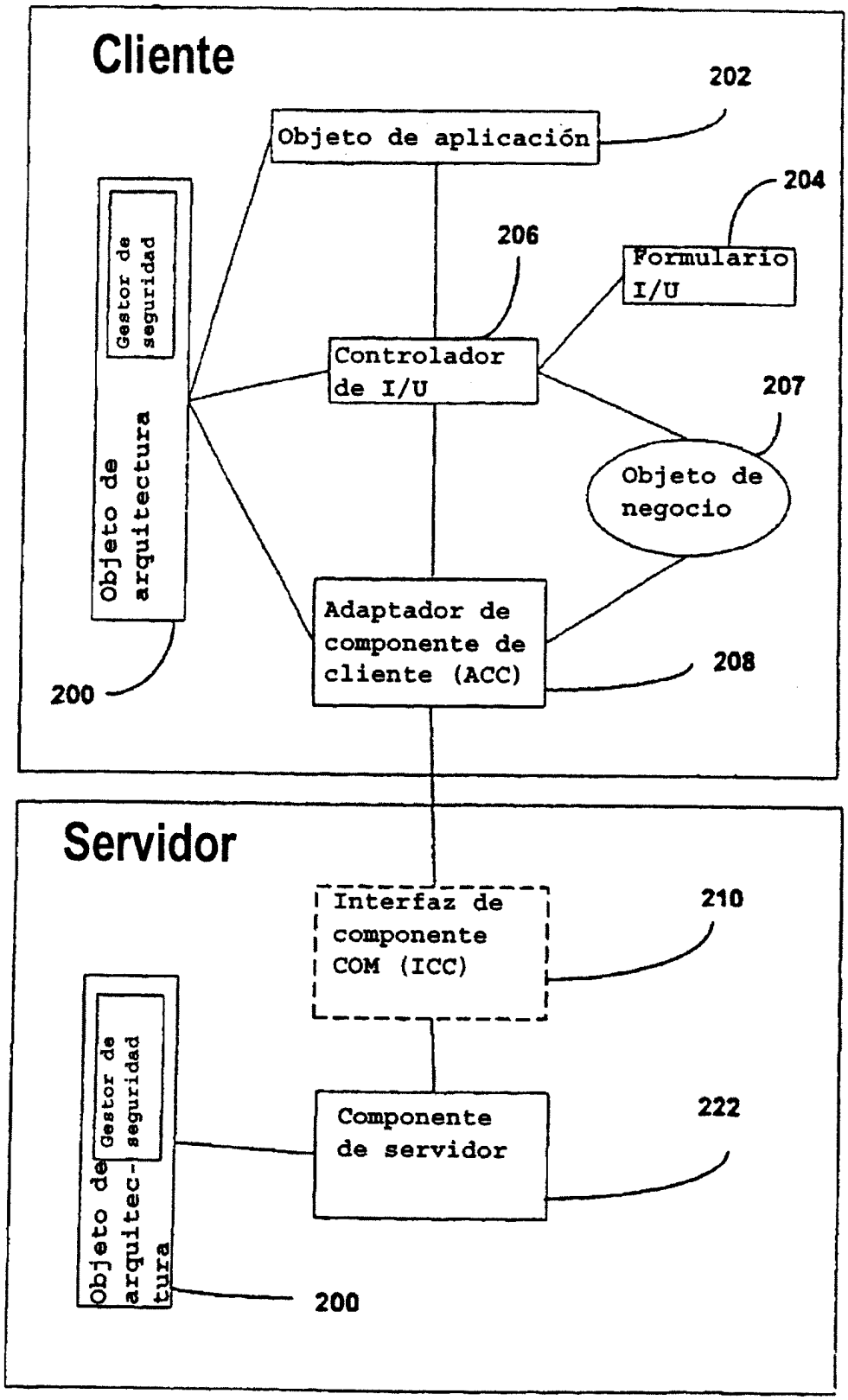

Cliente
202
Objeto de aplicación
204
206
Formulario I/U
Gestor de seguridad
Objeto de arquitectura
Controlador de I/U
207
Objeto de negocio
Adaptador de componente de cliente (ACC)
208
200
Servidor
Interfaz de componente COM (ICC)
210
Gestor de seguridad
Componente de servidor
222
Objeto de arquitec-tura
200

Figura 2A

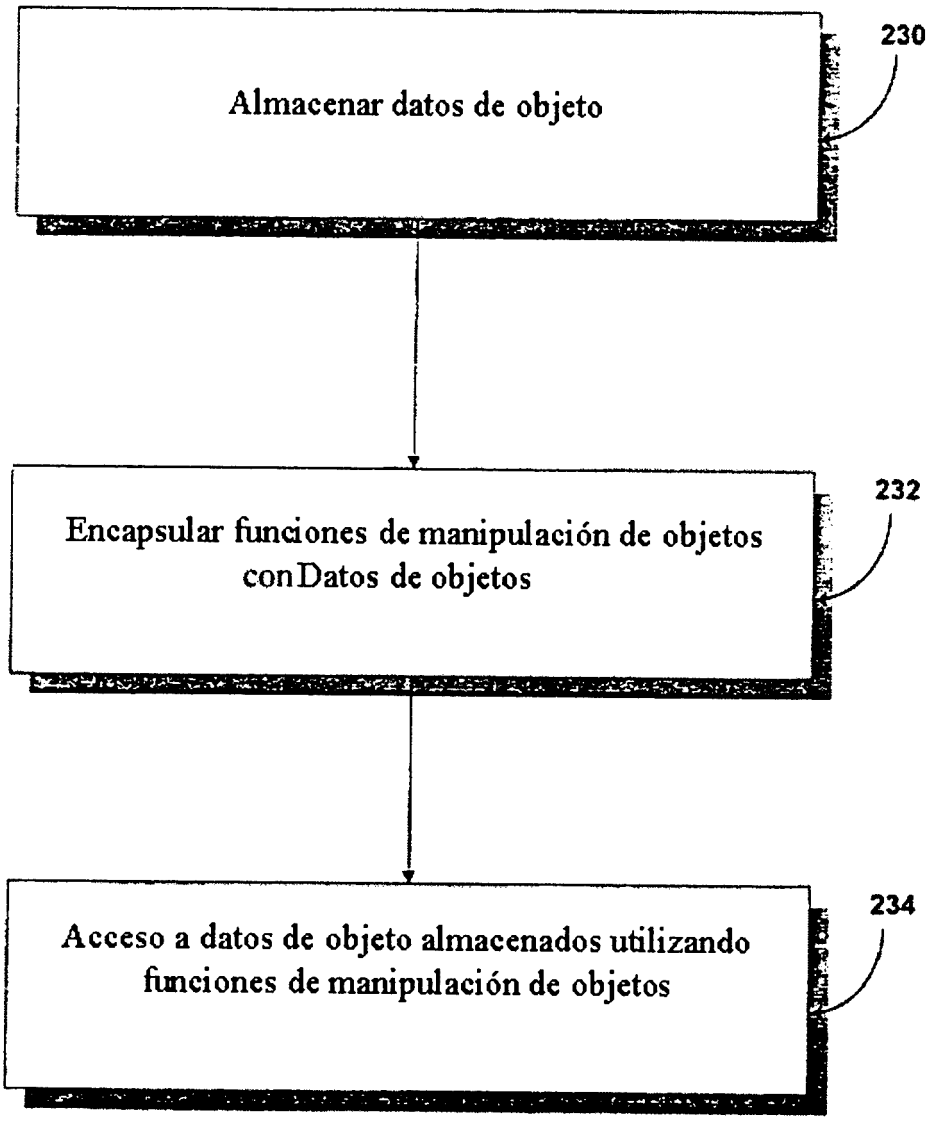

Almacenar datos de objeto
230
Encapsular funciones de manipulación de objetos con Datos de objetos
232
Acceso a datos de objeto almacenados utilizando funciones de manipulación de objetos
234

Figura 2B

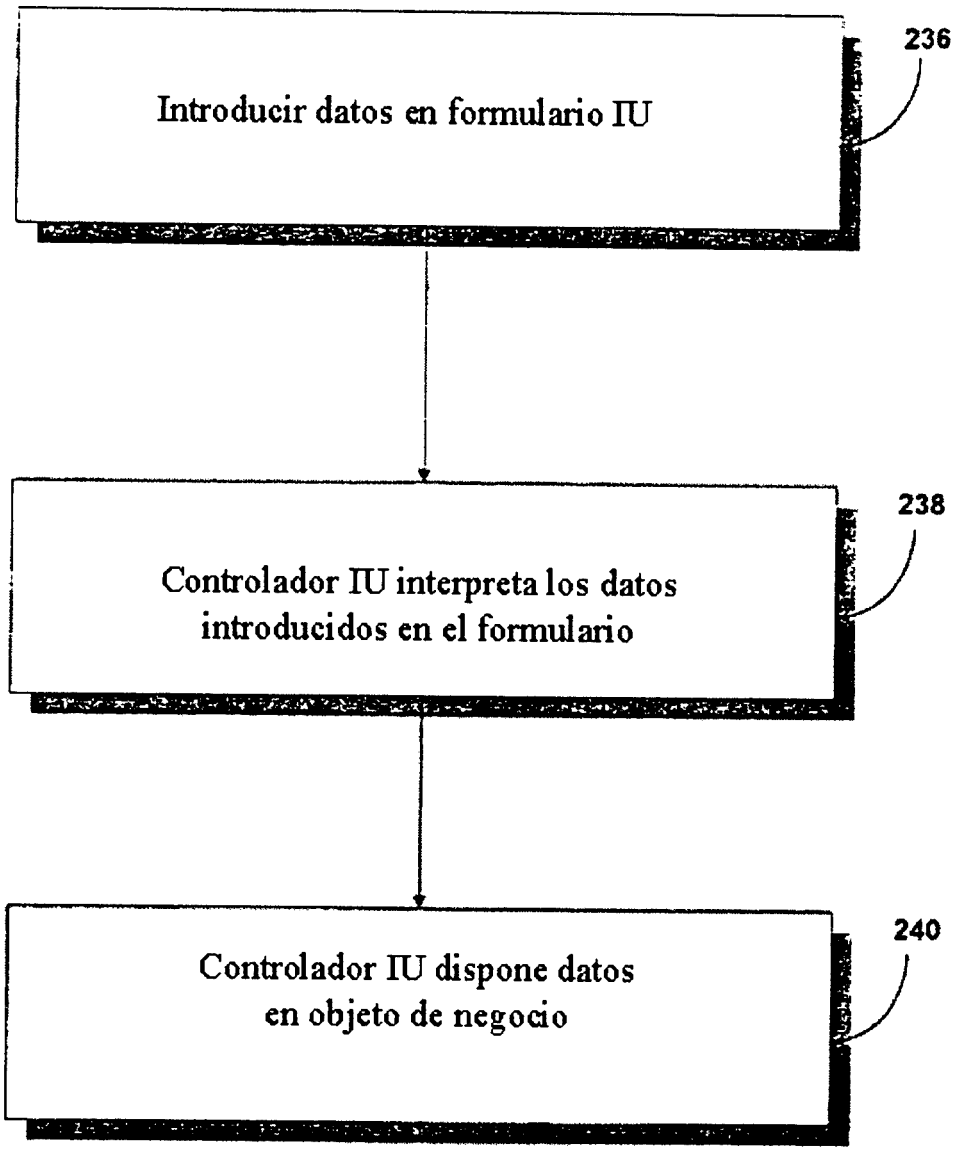

Introducir datos en formulario IU
236
Controlador IU interpreta los datos introducidos en el formulario
238
Controlador IU dispone datos en objeto de negocio
240

Figura 2C

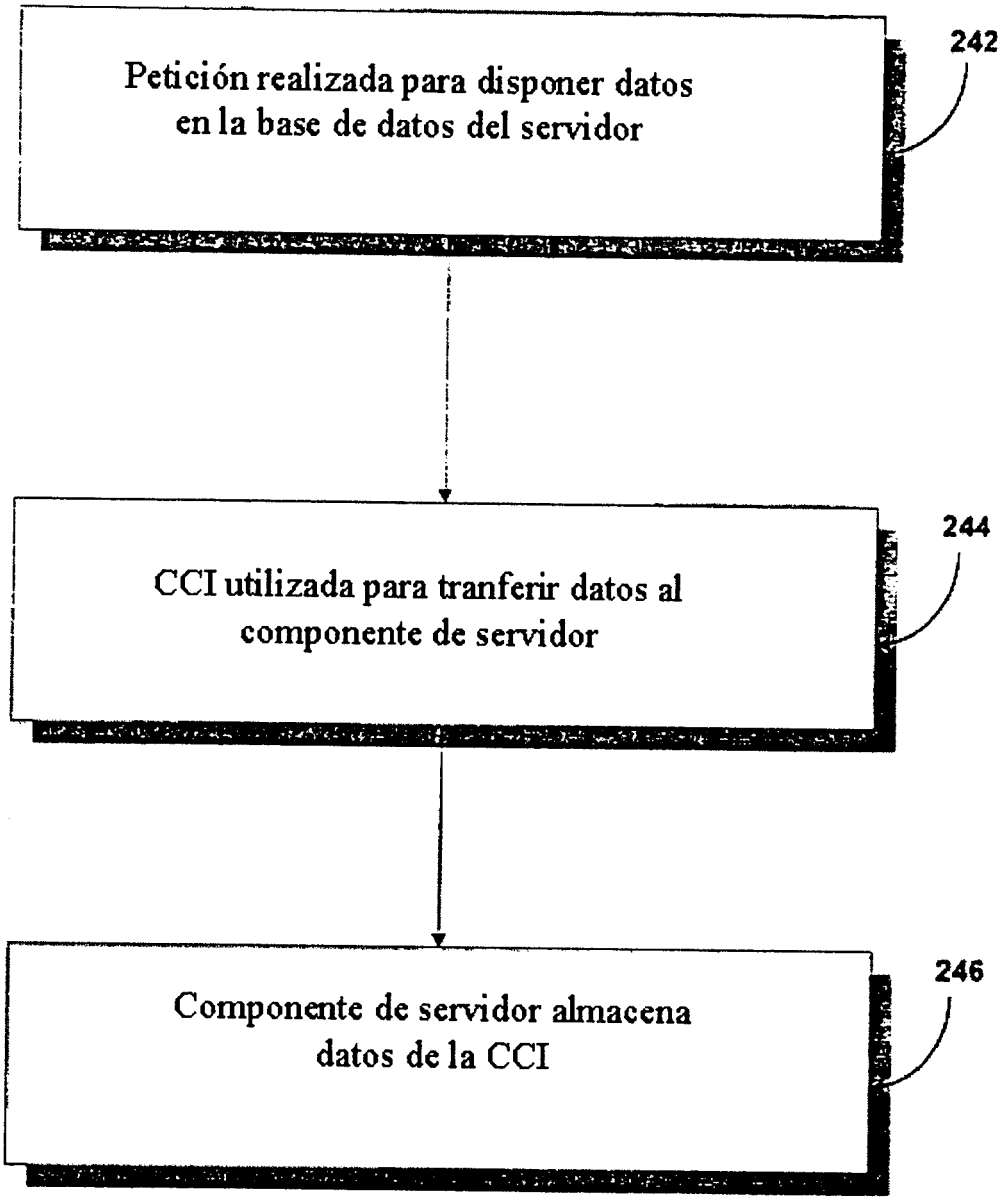

Petición realizada para disponer datos en la base de datos del servidor
242
CCI utilizada para tranferir datos al componente de servidor
244
Componente de servidor almacena datos de la CCI
246

Figura 2D

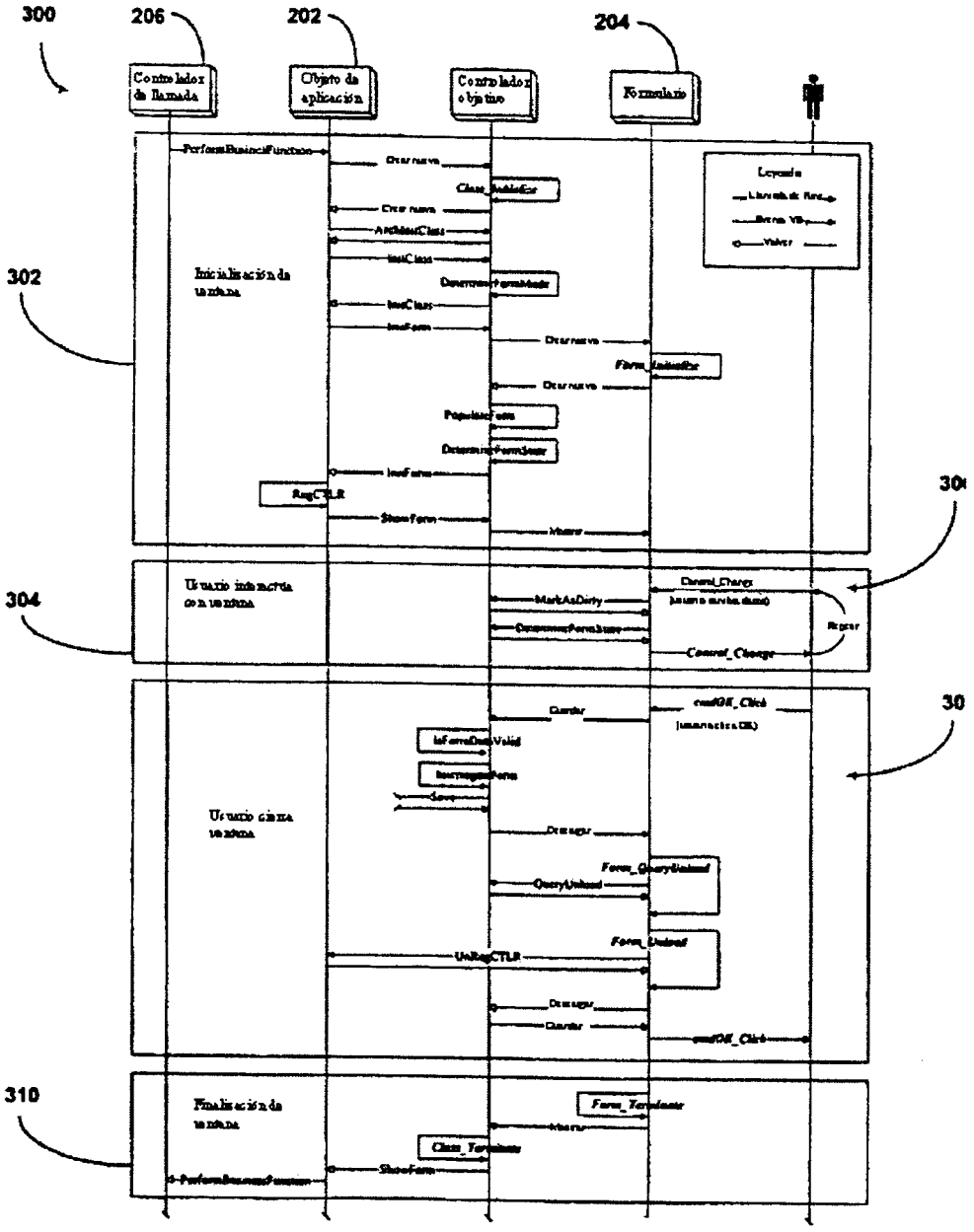

300
206
202
204
Controlador de llamada
Objeto de aplicación
Controlador objetivo
Formulario
Leyenda
302
Inicialización de ventana
304
Usuario interactúa con ventana
Usuario cierra ventana
310
Finalización de ventana

Figura 3

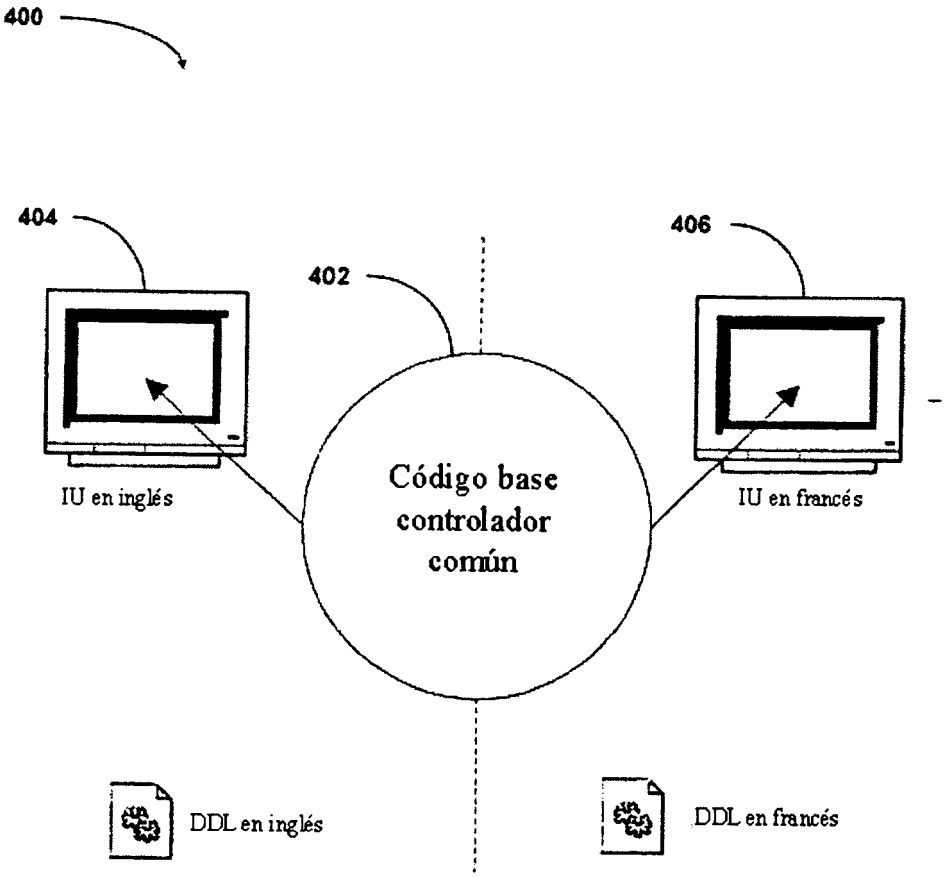
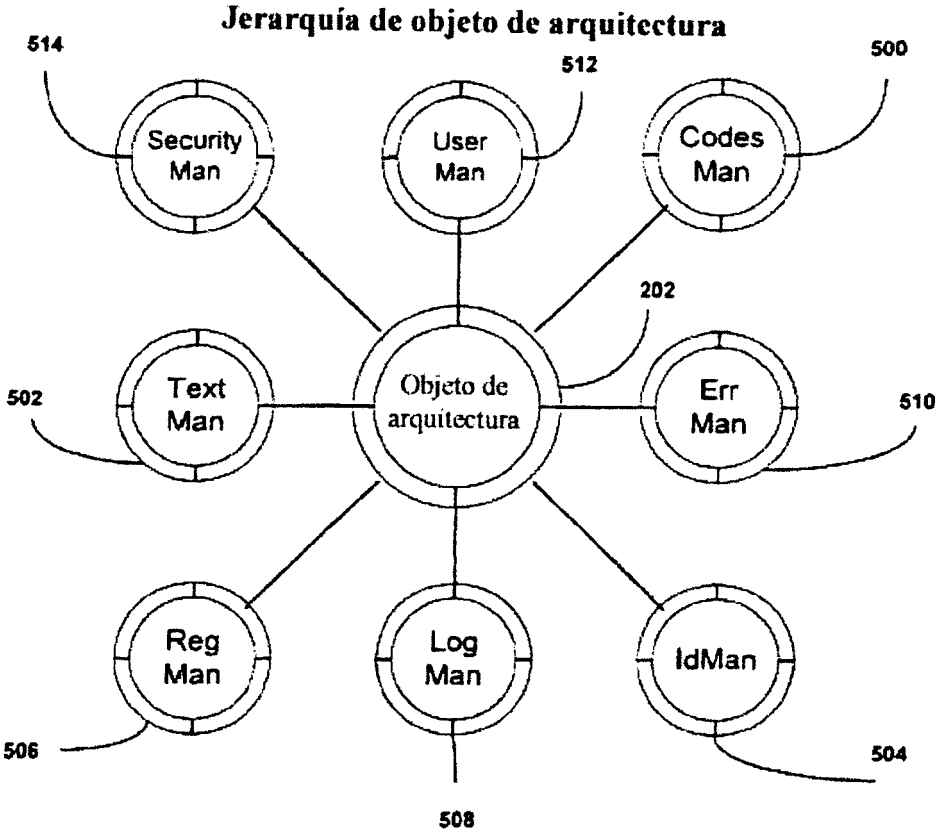

400
404
402
406
IU en inglés
Código base controlador común
IU en francés
DDL en inglés
DDL en francés

Figura 4

Figura 5


Jerarquía de objeto de arquitectura
514
512
500
Security Man
User Man
Codes Man
202
502
Text Man
Objeto de arquitectura
Err Man
510
Reg Man
Log Man
IdMan
506
504
508

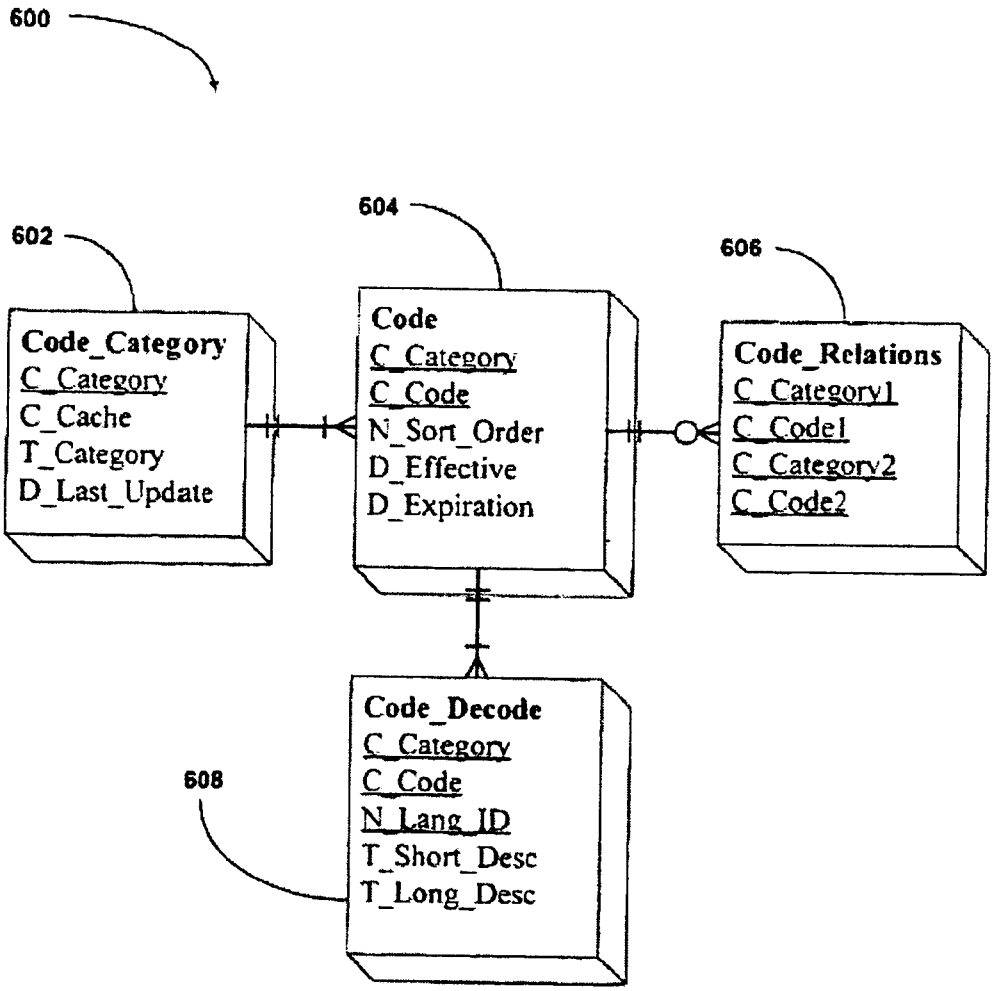

600
602
Code_Category
C_Category
C_Cache
T_Category
D_Last_Update
604
Code
C_Category
C_Code
N_Sort_Order
D_Effective
D_Expiration
606
Code_Relations
C_Category1
C_Code1
C_Category2
C_Code2
608
Code_Decode
C_Category
C_Code
N_Lang_ID
T_Short_Desc
T_Long_Desc

Figura 6

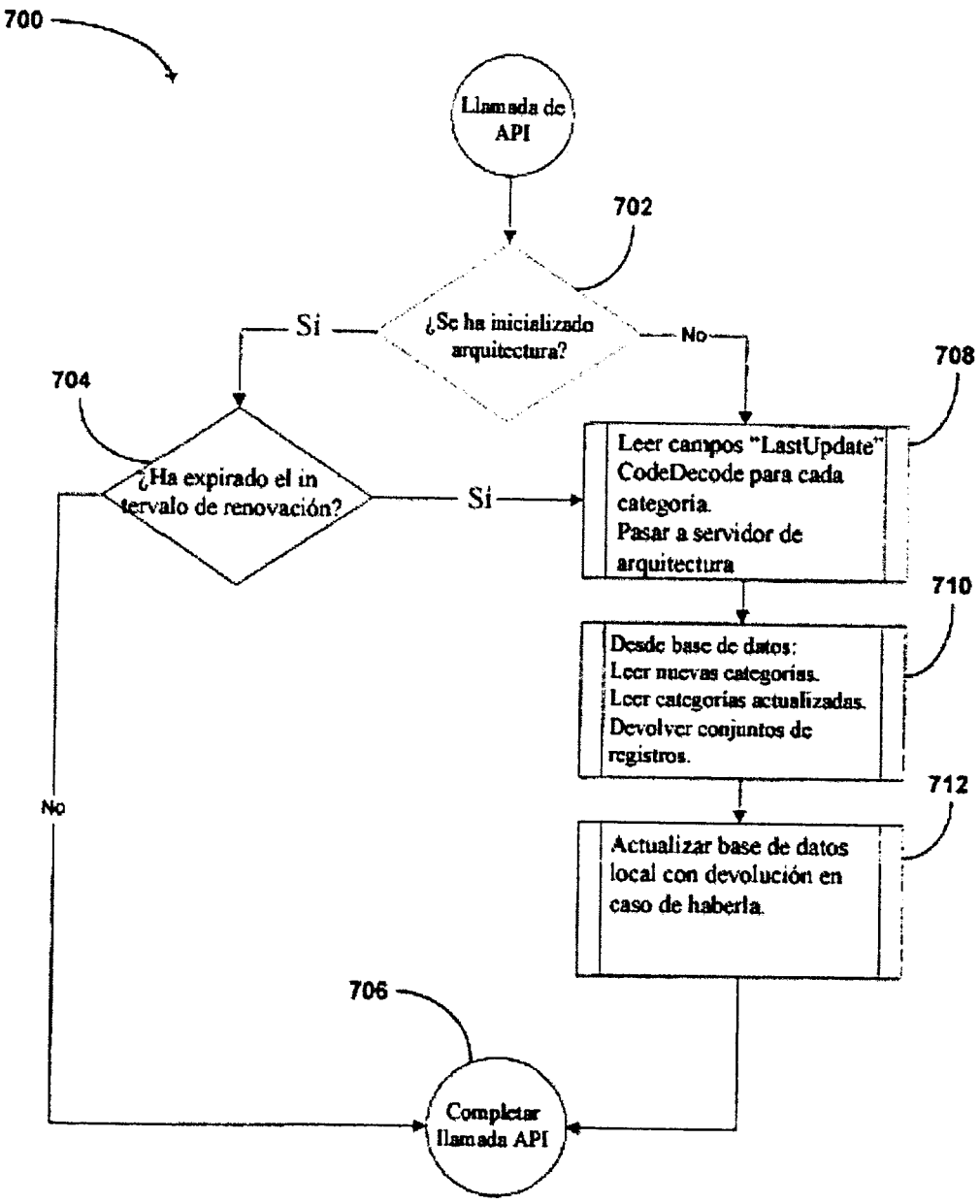

700
Llamada de API
702
¿Se ha inicializado arquitectura?
Sí
No
704
¿Ha expirado el intervalo de renovación?
Sí
No
708
Leer campos "LastUpdate" CodeDecode para cada categoría.
Pasar a servidor de arquitectura
710
Desde base de datos:
Leer nuevas categorías.
Leer categorías actualizadas.
Devolver conjuntos de registros.
712
Actualizar base de datos local con devolución en caso de haberla.
706
Completar llamada API

Figura 7

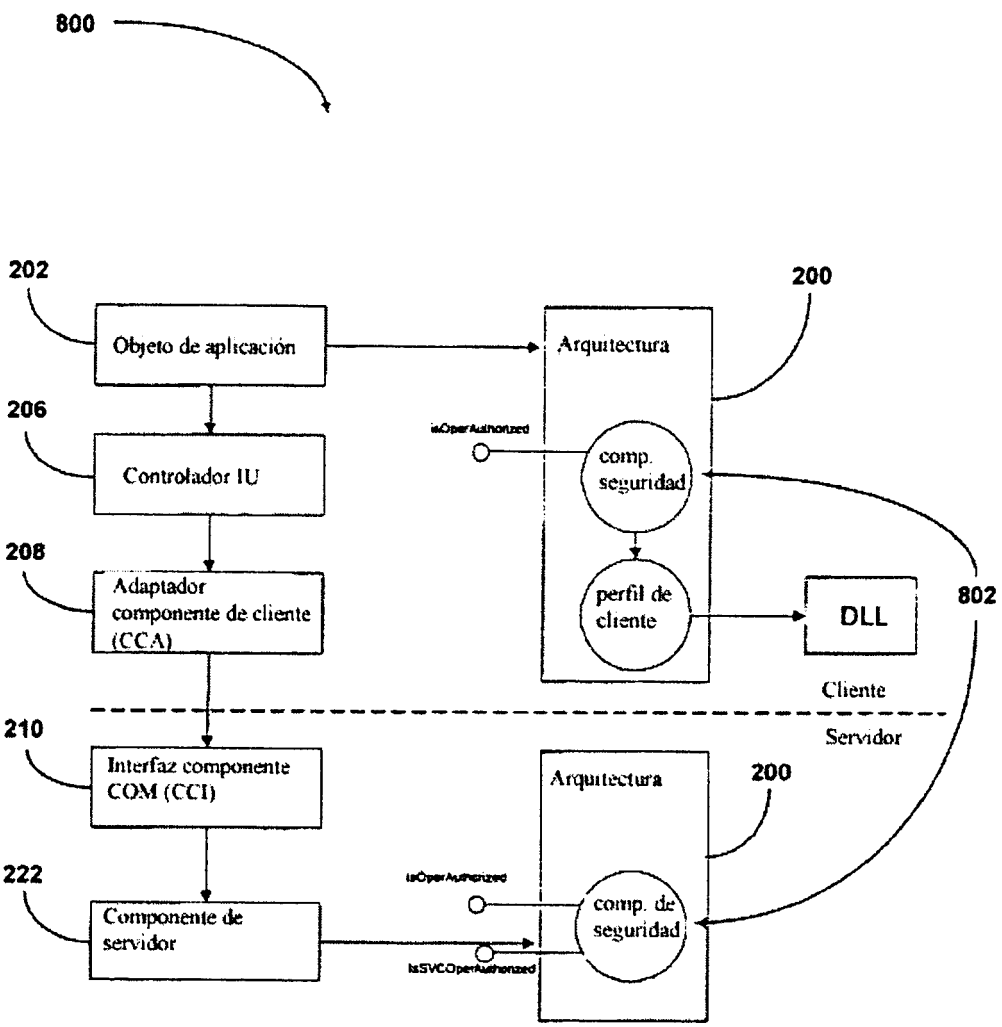

800
202
Objeto de aplicación
206
Controlador IU
208
Adaptador componente de cliente (CCA)
210
Interfaz componente COM (CCI)
222
Componente de servidor
200
Arquitectura
comp. seguridad
perfil de cliente
DLL
802
Cliente
Servidor
200
Arquitectura
comp. de seguridad

Figura 8

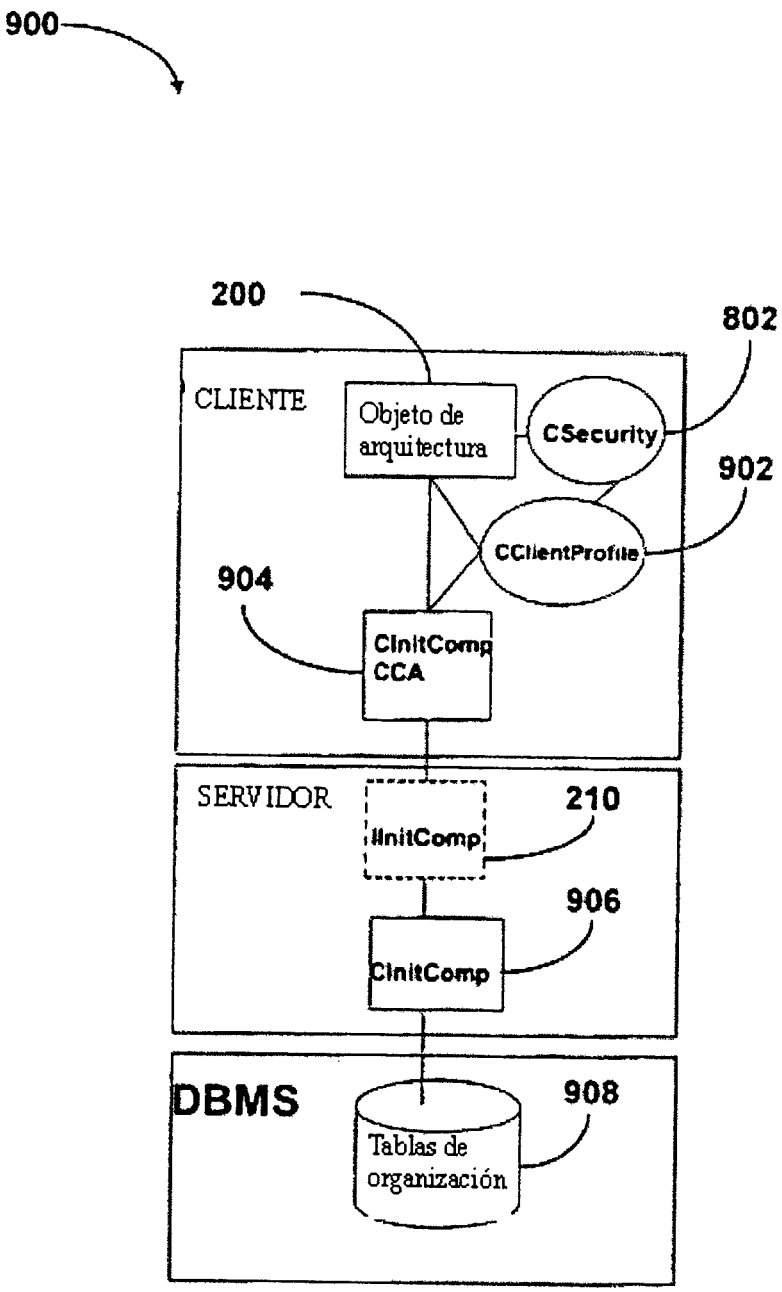

900
200
802
CLIENTE
Objeto de arquitectura
CSecurity
902
CClientProfile
904
CInitComp CCA
SERVIDOR
210
IInitComp
906
CInitComp
DBMS
908
Tablas de organización

Figura 9

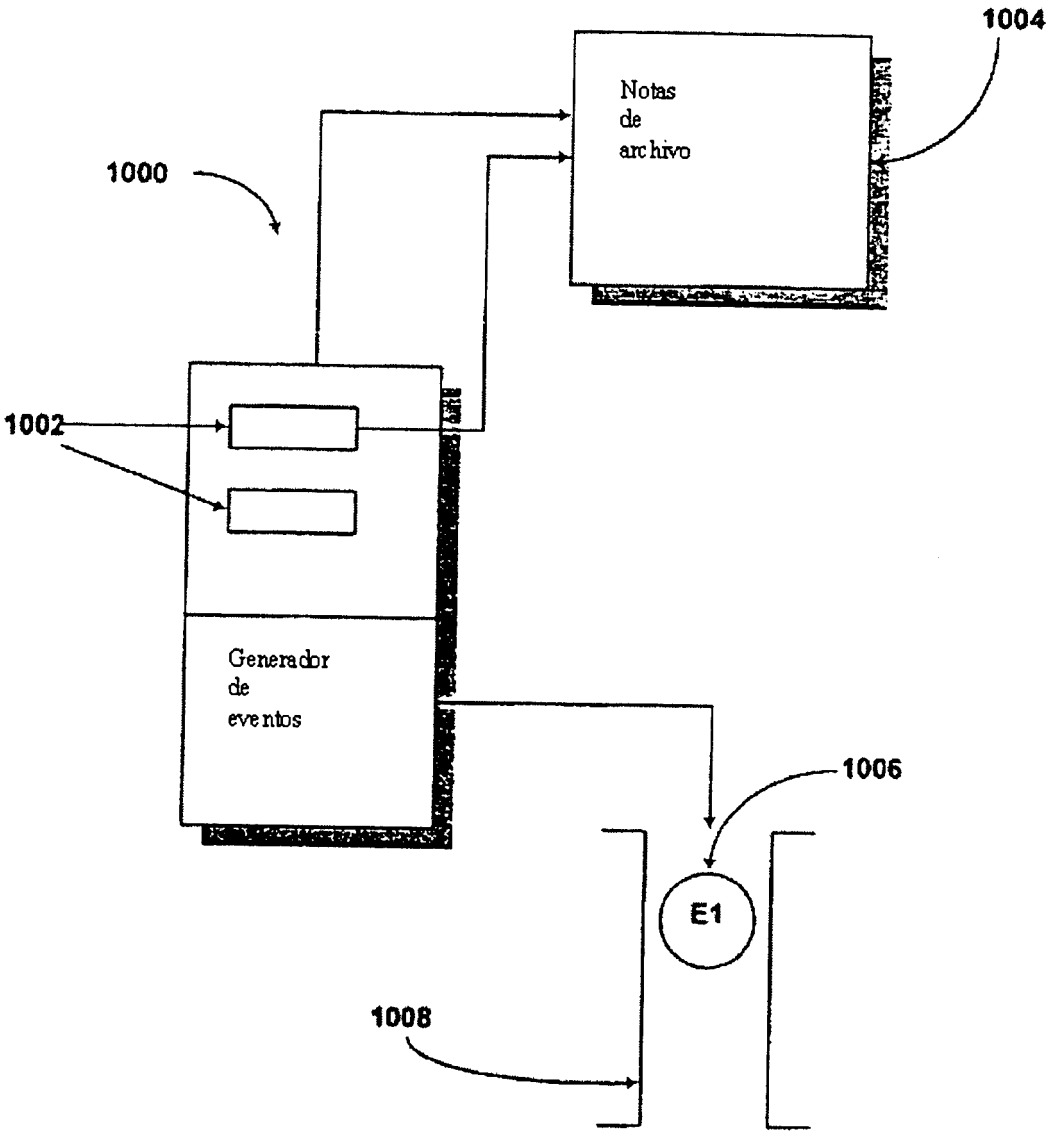

1004
Notas
de
archivo
1000
1002
Generador
de
eventos
1006
E1
1008

Figura 10

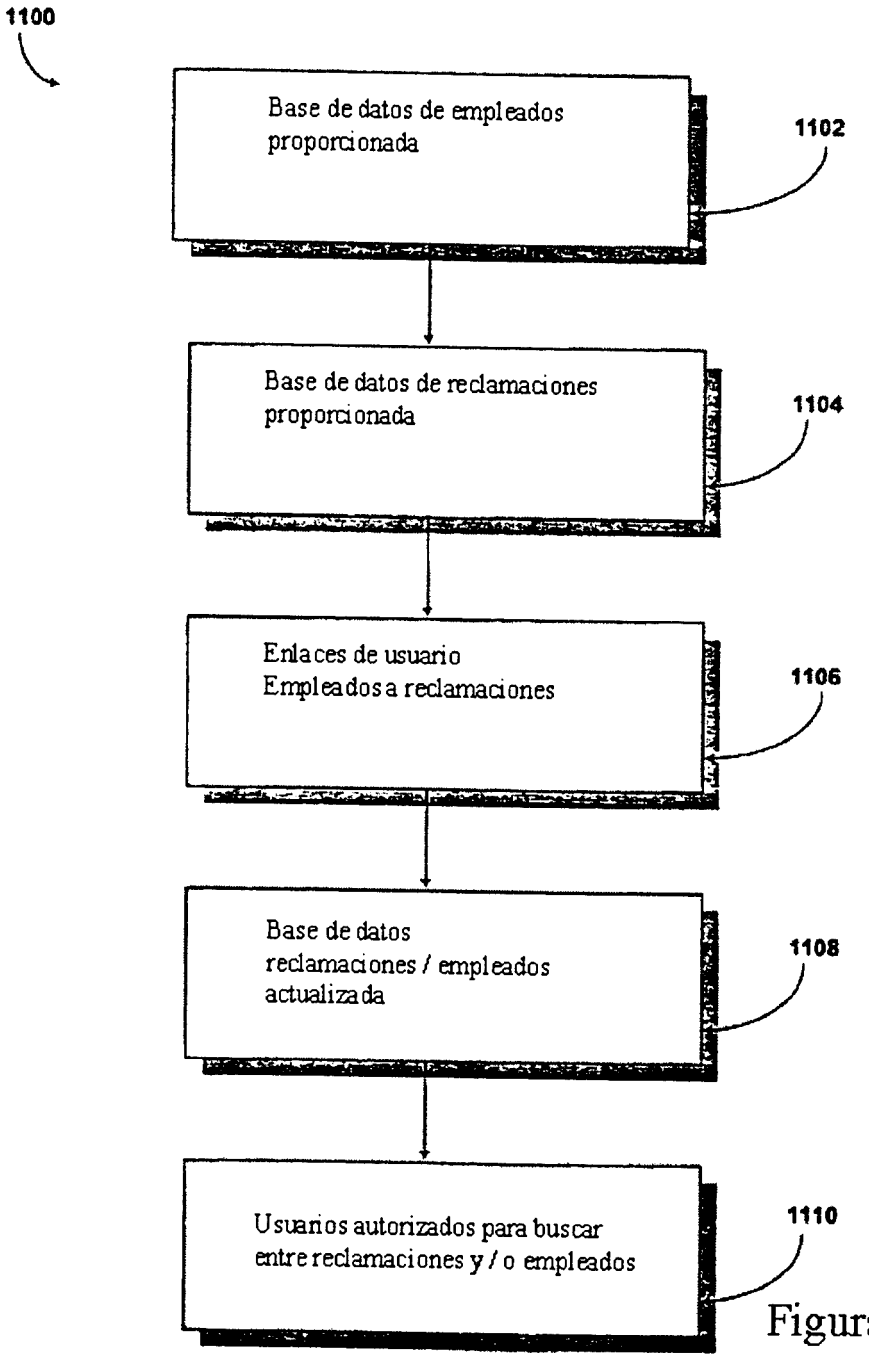

1100
Base de datos de empleados proporcionada
1102
Base de datos de reclamaciones proporcionada
1104
Enlaces de usuario Empleados a reclamaciones
1106
Base de datos reclamaciones / empleados actualizada
1108
Usuarios autorizados para buscar entre reclamaciones y / o empleados
1110

Figura 11

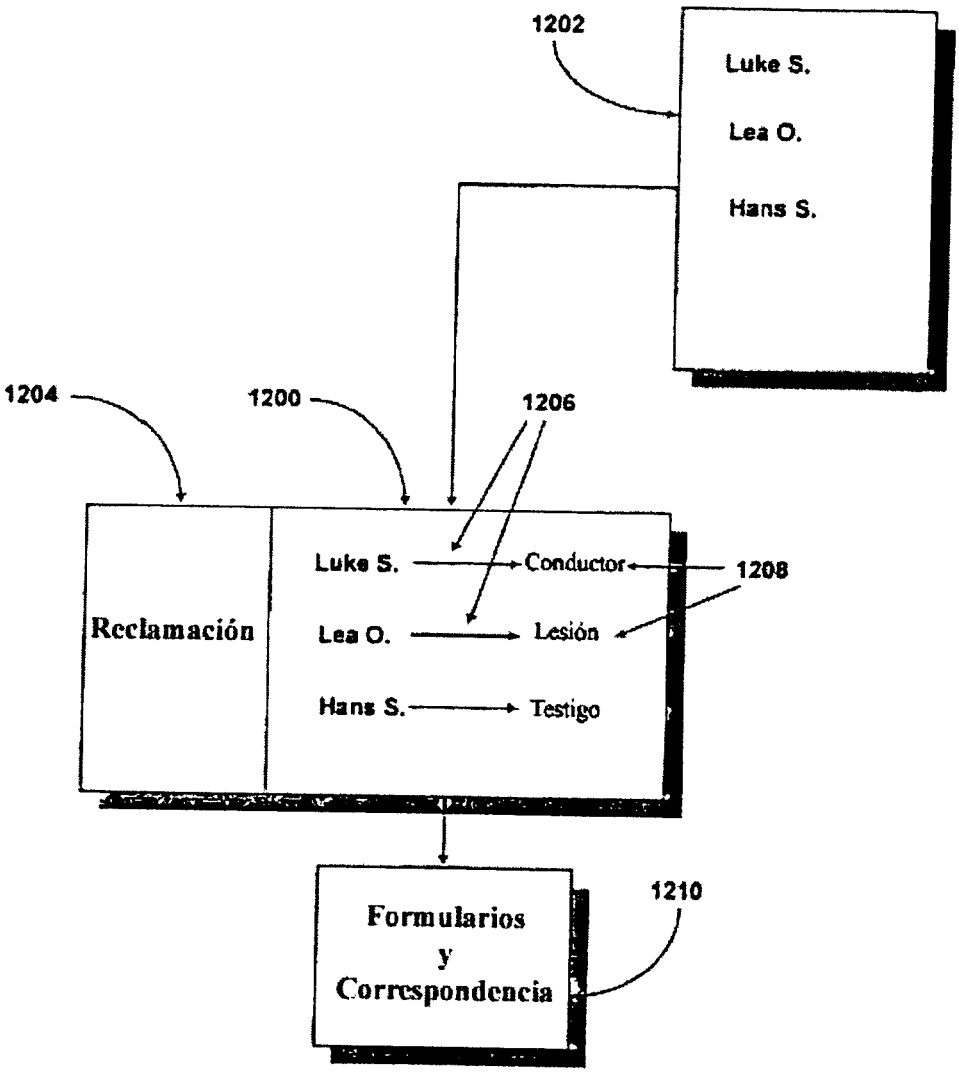

1202
Luke S.
Lea O.
Hans S.
1204
1200
1206
Reclamación
Luke S.
Conductor
1208
Lea O.
Lesión
Hans S.
Testigo
Formularios
y
Correspondencia
1210

Figura 12

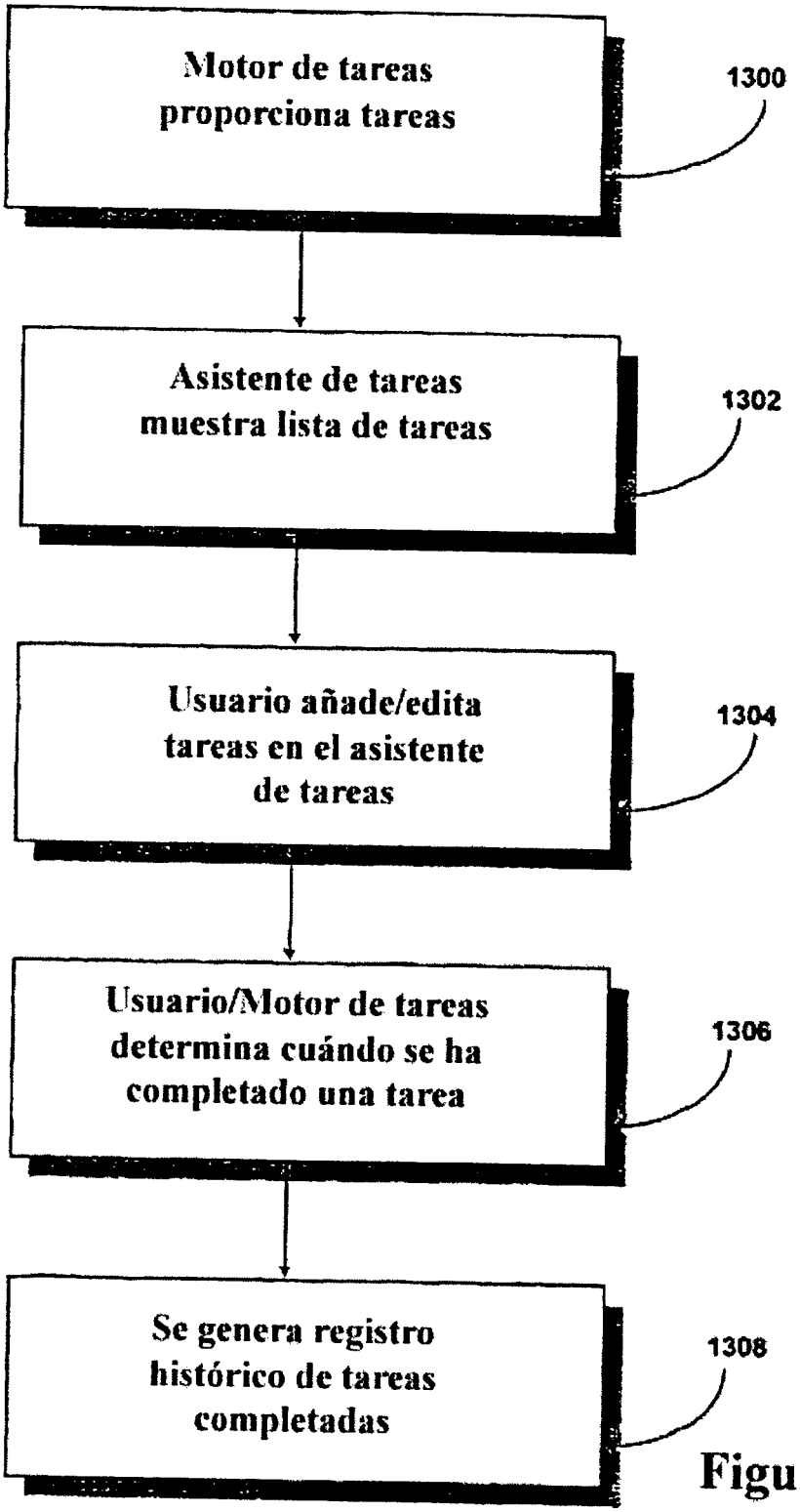

Motor de tareas proporciona tareas
1300
Asistente de tareas muestra lista de tareas
1302
Usuario añade/edita tareas en el asistente de tareas
1304
Usuario/Motor de tareas determina cuándo se ha completado una tarea
1306
Se genera registro histórico de tareas completadas
1308

Figura 13

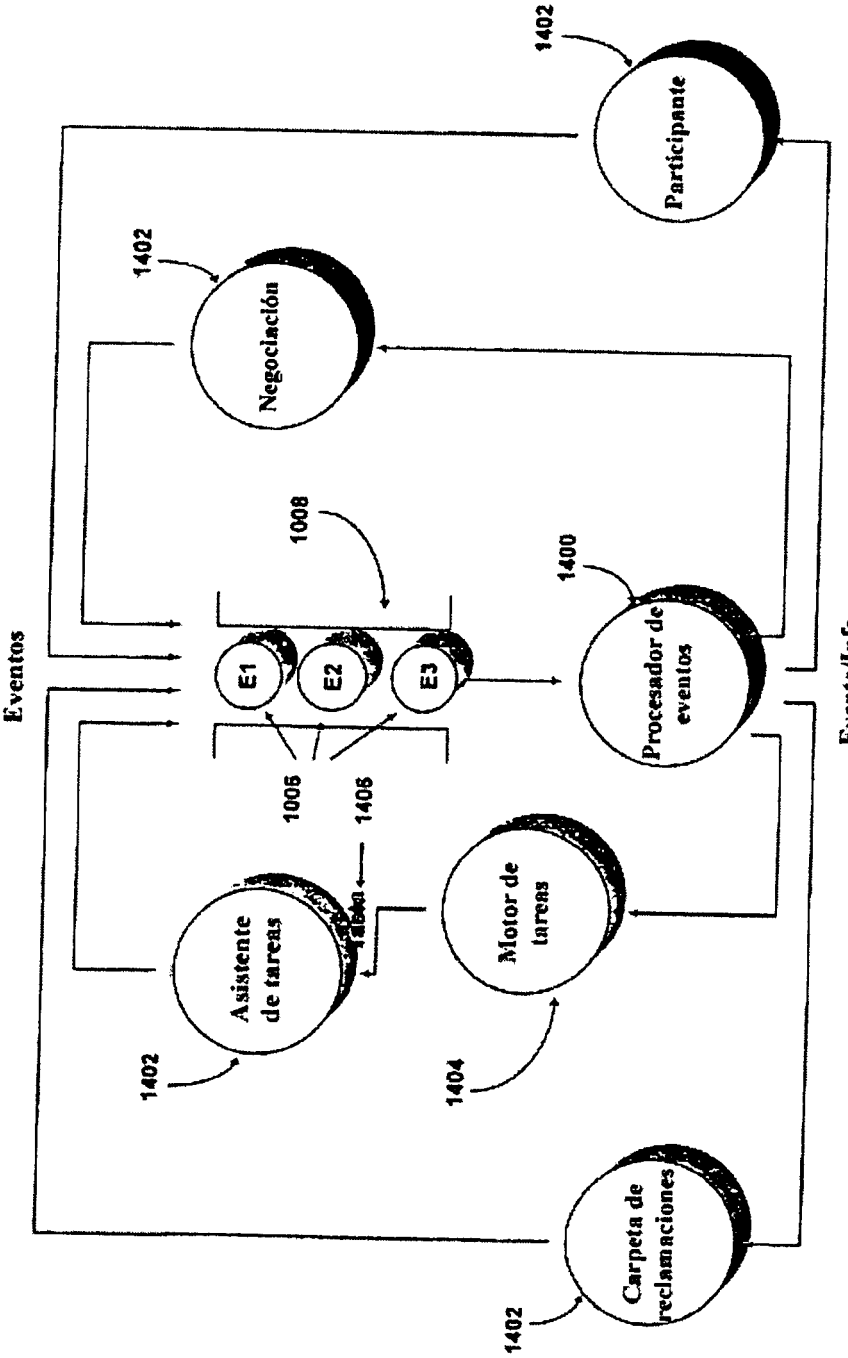

Eventos
1402
Participante
1402
Negociación
1008
1400
Procesador de eventos
E1
E2
E3
1006
1406
Evento/Info
Asistente de tareas
Motor de tareas
1402
1404
Carpeta de reclamaciones
1402

Figura 14

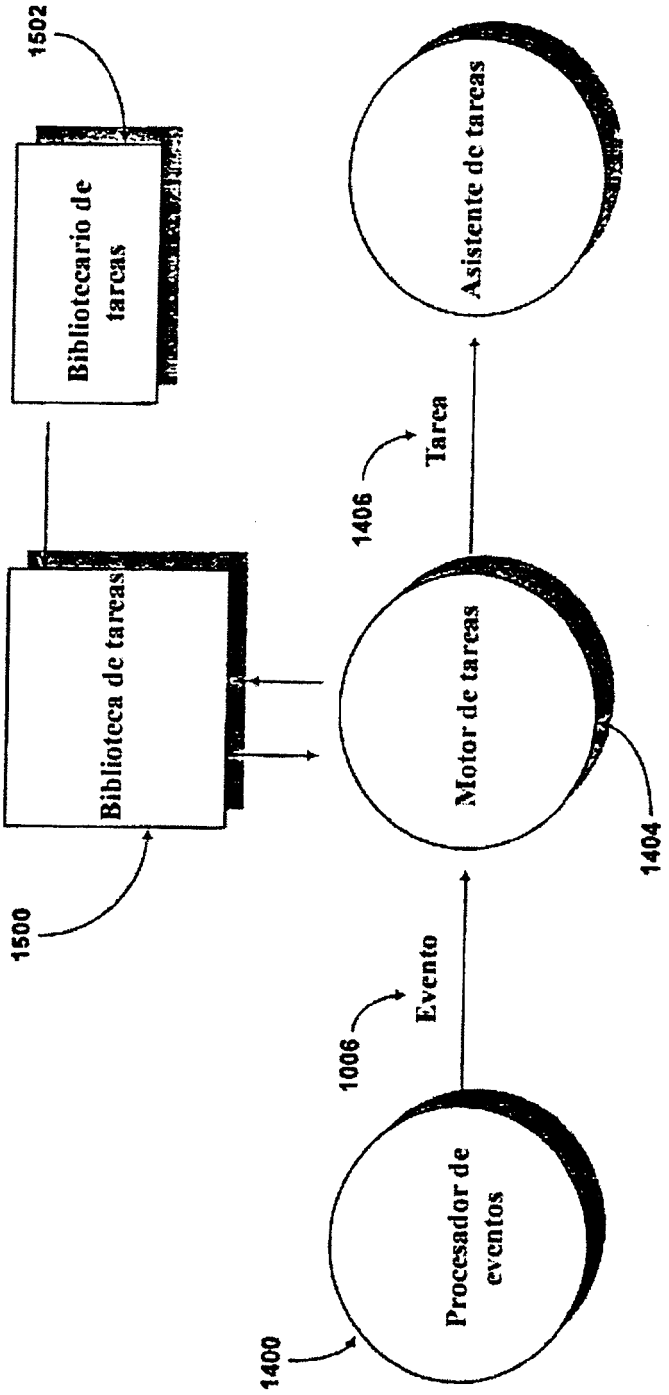

1502
Bibliotecario de tareas
Asistente de tareas
Tarea
1406
Biblioteca de tareas
Motor de tareas
1404
1500
Evento
1006
Procesador de eventos
1400

Figura 15