

IE



Industria y Comercio

SUPERINTENDENCIA

**SOLICITUD
PATENTE DE INVENCION**

EXPEDIENTE N°

04-112067

TITULO: EL MODELO DE OBJETO
PROGRAMABLE PARA EL NOMBRE UNICO
(NAMESPACE) O ESQUEMA DE APOYO DE
BIBLIOTECA EN UNA APLICACIÓN DE SOFTWARE

CLASIFICACION INTERNACIONAL: G06F 17 31

SOLICITANTE: MICROSOFT CORPORATION

DOMICILIO: REDMOND, WASHINGTON, ESTADOS
UNIDOS DE AMERICA

APODERADO: LUZ HELENA ADARVE GOMEZ

BOGOTÁ, D.C. _____

Pr
CS
Pd. sust

PETITORIO

Industria y Comercio
SUPERINTENDENCIA

SUPERINDUSTRIA Y COMERCIO
Radicación : 04112067 00000000 Folios: 63
Fecha (AMB): 2004-11-08 15:52:08
Trámite : 002 PATENTES B 1 REGISTRO/ 411 PRESENTA
Dependencia: 2020 DIVISION DE NUEVAS CREACIONES

FORMULARIO UNICO DE SOLICITUD DE PATENTE 2000-3

SOLICITUD DE:

☒ X

Patente de Invención

☐

Patente de Modelo de Utilidad

SOLICITANTE (71)	Nombre: MICROSOFT CORPORATION Dirección: One Microsoft Way Redmond, Washington 98052, Estados Unidos de America Teléfono: 3123800 Fax: 3122410 E-Mail: general@cardenasycardenas.com Lugar de Constitución: Estado de Washington, Estados Unidos de América Domicilio: REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA	IDENTIFICACION C.C. ☐ NIT ☐ C.E. ☐ Otro ☒ X Número _____
REPRESENTANTE O APODERADO (74)	Nombre: LUZ HELENA ADARVE GOMEZ Dirección: Carrera 7 No 71-52 Torre B Piso 8 Teléfono: 3123800 Fax: 3122410 E-Mail: general@cardenasycardenas.com	IDENTIFICACION C.C. ☒ X NIT ☐ C.E. ☐ Otro ☐ Número 41.575.435 BTA TP 1.4.884 C.S.J.
INVENTOR(ES) (72)	Nombre: 1. Brian M. Jones 2. Mark Sunderland 3. Marcin Sawicki 4. Robert A. Little Dirección: 1. One Microsoft Way Redmond, Washington 98052, Estados Unidos de America 2. One Microsoft Way Redmond, Washington 98052, Estados Unidos de America 3. One Microsoft Way Redmond, Washington 98052, Estados Unidos de America 4. One Microsoft Way Redmond, Washington 98052, Estados Unidos de America Teléfono: 3123800 Fax: 3123800 E-Mail: general@cardenasycardenas.com Domicilio: 1. WASHINGTON, ESTADOS UNIDOS DE AMERICA 2. WASHINGTON, ESTADOS UNIDOS DE AMERICA 3. WASHINGTON, ESTADOS UNIDOS DE AMERICA 4. WASHINGTON, ESTADOS UNIDOS DE AMERICA	IDENTIFICACION C.C. ☐ NIT ☐ C.E. ☐ Otro ☒ X Número _____

i

Título(54):

EL MODELO DE OBJETO PROGRAMABLE PARA EL NOMBRE UNICO (NAMESPACE) O ESQUEMA DE APOYO DE BIBLIOTECA EN UNA APLICACIÓN DE SOFTWARE

ii

Prioridad

SI ☒ X

NO ☐

(33) País de Origen

Estados Unidos de América

(32) Fecha

Diciembre 9, 2003

(31) Nº de Solicitud

10/731597

Para publicar a partir de la fecha de la presente solicitud a los:

☐

8 meses

☐

12 meses

☐

18 meses

☐

Otro



Comprobante de Pago No.:

Fecha

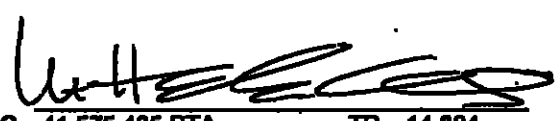
ANEXOS

- ☒ Comprobante de pago de la tasa de presentación de la solicitud.
- ☐ Comprobante de pago de la tasa por concepto de excedente de palabras en la publicación.
- ☒ Comprobante de pago de la tasa por concepto de reivindicación de prioridad.
- ☒ Poderes, si fuere el caso. (Protocolo No 16.069)
- ☒ Certificado de existencia y representación legal cuando el solicitante sea persona jurídica. Protocolo No 16.069)
- ☐ Copia certificada de la primera solicitud, si se reivindica prioridad.
- ☐ Traducción simple de la primera solicitud, si se reivindica prioridad.
- ☐ Documento de cesión del inventor al solicitante o a su causante.
- ☒ Descripción de la invención.
- ☒ Una o más reivindicaciones.
- ☒ Dibujos y/o planos necesarios.
- ☐ De ser el caso, copia del contrato de acceso.
- ☐ De ser el caso, documento que acredite la licencia o autorización de uso de conocimientos tradicionales de la comunidad de las comunidades indígenas.
- ☐ De ser el caso, certificado de depósito del material biológico.
- ☒ Arte final 12x12 y 6x6 cm.
- ☒ Resumen y extracto para publicación.

10

NOMBRE: LUZ HELENA ADARVE GOMEZ

FIRMA:


C.C. 41.575.435 BTA

TP. 14.884
C.SJ.

SUPERINDUSTRIA Y COMERCIO
Radicación : 84112067 00000000 Folios: 63
Fecha (AMD): 2004-11-08 15:52:08
Trámite : 002 PATENTES D 1 REGISTRO/ 411 PRESENTA
Dependencia: 2020 DIVISION DE NUEVAS CREACIONES

NIT : 900176089-2

RECIBO OFICIAL DE CAJA : 04 - 96,526
FECHA : NOVIEMBRE 5 DE 2004

***** CONSIGNACION *****

DEPOSITANTE	TIPO PAGO	BANCO	CUENTA	No. PAGO	FECHA PGO	Vr. PAGO
CARDENAS Y CARDENAS P.I.	CONSIGNACION	BANCO POPULAR	050-00110-6	29223822	05/11/2004	6,250,000.00

***** CONCEPTO *****

CANT. CONTABILITICO	CONCEPTO	TOTAL CONCEPTO
1 50005-01-01 SOLICITUDES	1 TRAMITES DE SOL. DE PATENTE DE IN	422,000.00
	TOTAL :	\$ 422,000.00

SON: CUATROCIENTOS VEINTIDOS MIL PESOS

RESPONSABLE : _____

RECIBO DE CAJA APLICADO AL EXPEDIENTE No. EL MODELO DE OBJETO PROGRAMABLE
PARA EL NOMBRE UNICO (NAMESPACE)
O ESQUEMA DE ARQUITECTURA DE BIBLIOTECA
EN UNA APLICACION DE SOFTWARE.

CARDENAS & CARDENAS
ABOGADOS

NIT : 890174089-2

SUPERINDUSTRIA Y COMERCIO
Radicacion : 04112067 00000000 Folios: 63
Fecha (AMD): 2004-11-08 15:52:08
Tramite : 002 PATENTES 1 REGISTRADO 411 PRESENTA
Dependencia: 2020 DIVISION DE NUEVAS CREACIONES

RECIBO OFICIAL DE CAJA : 04 - 66,627
FECHA : NOVIEMBRE 5 DE 2004

***** CONSIGNACION *****

DEPOSITANTE	TIPO PAGO	BANCO	CUENTA	No. PAGO	FECHA PAGO	Vr. PAGO
CARDENAS Y CARDENAS P.T. CONSIGNACION		BANCO POPULAR	050-00110-6	39723922	05/11/2004	6,250,000.00

***** CONCEPTO *****

CANT. CONTABILISTICO	CONCEPTO	TOTAL CONCEPTO
1 50005-01-01 SOLICITUDES	82 INVOCACION DE UNA PRIORIDAD EN NU	118,000.00
	TOTAL :	\$ 118,000.00

SON: CIENTO DIEY OCHO MIL PESOS

RESPONSABLE : _____

RECIBO DE CAJA APLICADO AL EXPEDIENTE No. EL MODELO DE OBJETO PROGRAMABLE PARA EL NOMBRE UNICO (NAMESPACE) O ESQUEMA DE APOYO DE BZB EN UNA APLICACION D

CARDENAS & CARDENAS
ABOGADOS

5

NOTE FINAL
12x12

15

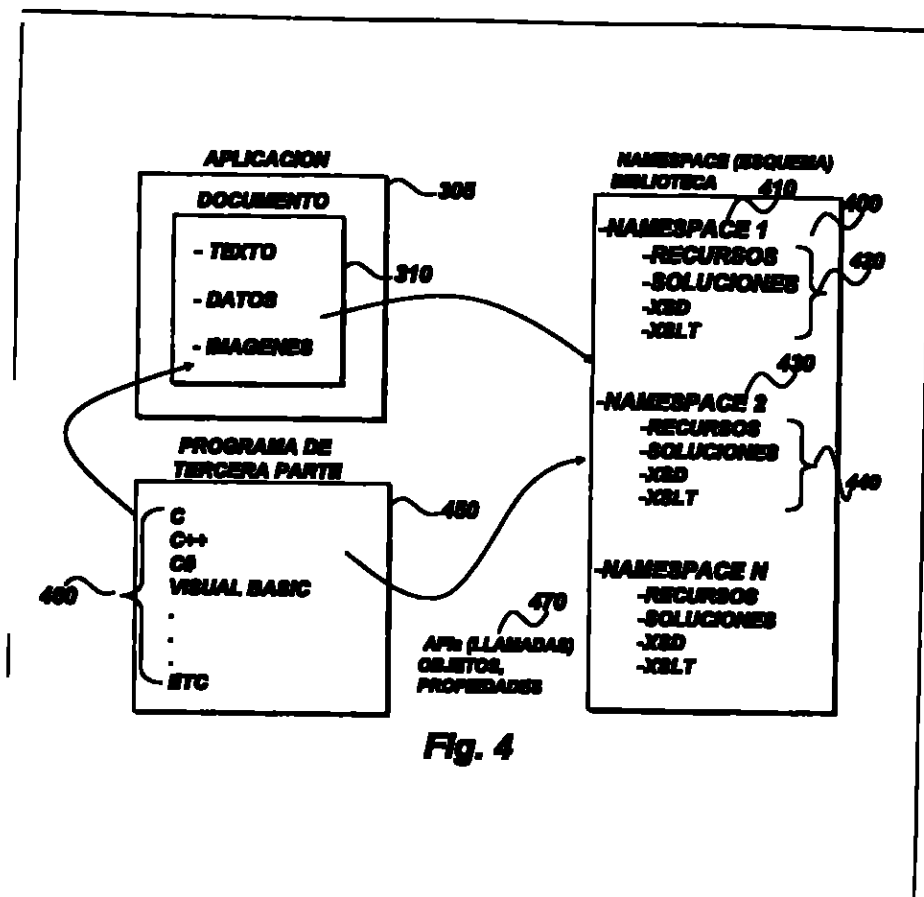


Fig. 4

**EL MODELO DE OBJETO PROGRAMABLE PARA EL NOMBRE UNICO
(NAMESPACE) O ESQUEMA DE APOYO DE BIBLIOTECA EN UNA
APLICACIÓN DE SOFTWARE**

Aplicaciones Relacionadas

La Aplicación de Utilidad de Patente de los Estados Unidos por la materia del solicitante número 60001.0263US01/MS303917.1, titulado "El Modelo de Objeto Programable para la Validación del Esquema del Lenguaje Enriquecido Extensible", y la Aplicación de Patente Utilidad de los Estados Unidos por la materia del solicitante número 60001.0264US01/MS303918.1, titulado "Modelo de Objeto Programable para el Encarecimiento de Idioma de Encarecimiento Extensible en una Aplicación", está por el Lenguaje Enriquecido del Margen de Ganancia Extendida en una Aplicación" están presentes incorporados por referencia.

Derechos de Propiedad de la Noticia

Una parte del descubrimiento de este documento patente contiene material que está sujeto a protección de derechos de propiedad literaria. El dueño de los derechos de propiedad literaria no tiene ninguna objeción a la reproducción del facsímil por cualquiera del documento patente o el descubrimiento patente, como aparece en la Patente de Estados Unidos y en el archivo o registros de la Oficina Marcas de fábrica, pero por otra parte reserva todos los correctos derechos de propiedad literaria en absoluto.

Campo de la Invención

La invención presente se relaciona con los modelos de objeto programables. Más particularmente, la invención presente se relaciona a un modelo de objeto programable para el nombre único (Namespace) o apoyo de biblioteca de esquema en una aplicación de software.

El fondo de la Invención

Las aplicaciones de software de computadora les permiten a los usuarios crear una variedad de documentos para ayudarlos en el trabajo, educación, y ocio. Por

ejemplo, las aplicaciones populares de procesador de palabras les permiten a los usuarios crear cartas, artículos, libros, memorandos, y similares. Las aplicaciones de hoja de cálculo les permiten a los usuarios guardar, manipular, imprimir, y desplegar una variedad de datos alfanuméricos. Tales aplicaciones tienen varias fortalezas bien conocidas que incluyen corrección especial, formateo, impresión, cálculo, asistencia en línea y corrección fuera de línea.

La mayoría de las aplicaciones de software de computadora no contienen toda la programación necesaria para proporcionar la funcionalidad requerida o deseada por cada usuario potencial. Muchos programadores desean a menudo aprovecharse de las capacidades de una aplicación existente en sus propios programas o personalizar la funcionalidad de una aplicación y hacerlo más conveniente para un juego específico de usuarios o acciones. Por ejemplo, un programador que trabaja en la industria financiera puede desear personalizar un procesador de texto para un público de usuarios que consiste en analistas financieros que revisan los informes financieros. En los recientes años, el Lenguaje Enriquecido Extensible se ha usado ampliamente como una estructura de datos intercambiable para muchos usuarios. A menudo los usuarios de funcionalidad de XML atan o asocian uno o más esquemas de archivos XML o soluciones basadas en XML a un documento siendo editado o creado por el usuario. Sin embargo, los usuarios/programadores están limitados en su habilidad de aplicar el esquema de archivos de XML y otras de funcionalidad basadas en XML a un documento dado porque el usuario/programador no tiene acceso directo y fácil al Namespace o al esquema de biblioteca que contienen el esquema de archivos XML u otras soluciones basadas en XML.

En concordancia, hay una necesidad en el arte para un modelo de objeto programable para permitir a un usuario/programador acceder a un Namespace o biblioteca del esquema de recursos de XML para personalizar o de otra manera manipular los recursos para reforzar el uso de la funcionalidad de XML del usuario/programador con un documento de aplicación de software. Con respecto a esto y a otras consideraciones es que la invención presente ha sido hecha.

Resumen de la Invención

La invención presente proporciona métodos y sistemas permitiéndole a un usuario acceder programáticamente y utilizar un Namespace o biblioteca del esquema que contienen el esquema de archivos XML y recursos basados en XML relacionados para asociar esos recursos basados en XML con uno o más documentos y personalizar la funcionalidad asociada con esos recursos Basados en XML. Una vez un usuario o programador obtiene el acceso al Namespace o biblioteca del esquema, el usuario puede asociar programáticamente el esquema de archivos XML con los datos de XML en un documento asociado, y *recíprocamente el usuario puede descubrir y puede quitar asociaciones del*

esquema de archivos XML con datos de contenidos XML en el documento. El usuario también puede asociar programáticamente los archivos de transformación con datos de XML contenidos en un documento y puede descubrir y puede quitar archivos de transformación existentes asociados con datos XML contenidos en el documento. El usuario también puede asociar otros archivos y el software ejecutable asociado con las soluciones del documento basadas en XML y con otros datos XML contenidos en el documento. Adicionalmente, el usuario puede detectar y eliminar la asociación de soluciones basadas en XML y otros tipos de software ejecutable de asociación con datos XML contenidos en un documento. Estas y otras características, ventajas, y aspectos de la invención presente pueden más claramente ser entendidos y apreciados de una revisión de la siguiente descripción detallada de las personalizaciones descubiertas y por referencia a los dibujos y demandas adjuntos.

Descripción breve de los Dibujos

Fig. 1 es un diagrama del bloque simplificado de un sistema de cómputo y periféricos asociados y dispositivos de red que proporciona un ambiente operativo ejemplar para la presente invención.

Fig. 2 es que un diagrama del bloque simplificado que ilustra la interacción entre los objetos de acuerdo con un modelo de programación orientado a objetos.

Fig. 3 es un diagrama de bloque que ilustra la interacción entre un documento, un archivo de esquema atado, y un modelo de la funcionalidad del esquema de validación.

Fig. 4 es un diagrama del bloque que ilustra la interacción entre un documento, un Namespace o biblioteca de esquema y una tercera parte de la aplicación de software.

Descripción detallada de la Personalización Preferida

Se dirigen personalizaciones de la invención presente a los métodos y sistemas para permitirle a un usuario llamar programáticamente una biblioteca del esquema nombre único (Namespace/Schema) del esquema de archivos XML y las soluciones basadas en XML y los recursos para controlar la asociación de esos archivos, soluciones y recursos a uno o más documentos. Estas personalizaciones pueden combinarse, pueden utilizarse otras personalizaciones, y pueden hacerse los cambios estructurales sin partir del espíritu o alcance de la invención presente. La descripción de tallada siguiente no puede ser tomada por consiguiente en un sentidos limitante y el alcance de la invención presente se define por las demandas adjuntas y sus equivalentes.

Refiriéndose ahora a los dibujos, en los cuales los numerales se representan como elementos a través de varias figuras. se describirán aspectos

de la invención presente y del ambiente operativo ejemplar. La Fig. 1 y la siguiente discusión intenta proporcionar una corta, descripción general de un ambiente de cómputo conveniente en que la invención puede llevarse a cabo. Mientras la invención se describirá en el contexto general de módulos del programa que se ejecutan junto con un programa de aplicación que corre en un sistema operativo en una computadora personal, eso experimentado en el arte reconocerá que la invención también puede llevarse a cabo en combinación con otros módulos del programa.

Generalmente, los módulos del programa incluyen rutinas, programas, componentes, estructuras de datos, y otros tipos de estructuras que realizan tareas particulares o implementan tipos de datos abstractos. Es más, estos más experimentados en el arte apreciarán que la invención puede practicarse con otras configuraciones de sistema de computadora, incluyendo los dispositivos portátiles, sistemas de multiprocesador, elementos basados en microprocesador o electrónica del consumidor programable, los mini ordenadores, los sistemas de cómputo grandes, y similares. La invención también puede practicarse en ambientes de cómputo distribuidos dónde las tareas son realizadas por dispositivos de proceso remotos que se unen a través de una red de comunicaciones. En un ambiente de cómputo distribuido, pueden localizarse los módulos de programa en los dispositivos de almacenamiento de memoria locales y remotos.

Volviendo ahora a la Fig. 1, se describirá una arquitectura de computadora ilustrativa para una computadora 2 personal para practicar varias personalizaciones de la invención. La arquitectura de la computadora mostrada en la Fig. 1 ilustra una computadora personal convencional, incluso un proceso central unidad 4 ("CPU"), un sistema memoria 6, incluso un acceso aleatorio memoria 8 (RAM) y una memoria de solo lectura ("ROM") 10, y un sistema de bus 12 eso acopla la memoria a la CPU 4. Un sistema de entrada/salida (input/output) básico que contiene las rutinas básicas que ayudan a transferir la información entre los elementos dentro de la computadora, como durante el inicio (startup), se guarda en la ROM 10. La computadora personal 2 incluye más allá un almacenamiento en masa dispositivo 14 para guardar un sistema operativo 16, la programas de aplicación, como el programa de aplicación 305, y los datos.

El almacenamiento en masa dispositivo 14 se conecta al CPU 4 a través de un director de almacenamiento en masa (no mostrado) conectado al bus 12. El almacenamiento en masa dispositivo 14 y sus medios de comunicación legibles por computadora asociados, mantienen el almacenamiento no volátil para la computadora personal 2. Aunque la descripción de medios de comunicación legibles por computadora contenidos aquí dentro se refiere a un dispositivo de almacenamiento en masa, como se maneja un disco duro o CD-ROM, debe ser apreciado por estos experimentados en el arte de los medios de comunicación

legibles por computadora que pueden ser cualquier medio de comunicación disponible que puede accederse por la computadora personal 2.

Por vía del ejemplo, y sin limitación, los medios de comunicación legibles por computadora pueden comprender medios de comunicación de almacenamiento de computadora y medios de comunicación. Los medios de comunicación de almacenamiento de computadora incluyen medios de comunicación volátiles y no-volátiles, trasladables y no-trasladables llevados a cabo en cualquier método o tecnología para el almacenamiento de información como las instrucciones legibles por computadora, las estructuras de datos, módulos de programa u otros datos. Los medios de comunicación de almacenamiento de computadora incluyen, pero no se limitan a, RAM, ROM, EPROM, EEPROM, memoria flash u otro estado sólido de la tecnología de memoria, CD-ROM, DVD, u otro almacenamiento óptico, casetes magnéticos, cintas magnéticas, almacenamiento de disco magnético u otros dispositivos de almacenamiento magnéticos, o cualquier otro medio que pueden usarse para guardar la información deseada y qué puede accederse por la computadora.

Según varias personalizaciones de la invención, la computadora personal 2 puede operar en un ambiente conectado a una red de computadoras que usa las conexiones lógicas a las computadoras remotas a través de un TCP/IP red 18, como Internet. La computadora personal 2 puede conectar al TCP/IP red 18 a través de una interfase de red unidad 20 conectada al bus 12. Debe apreciarse que la interfase de red unidad 20 también puede utilizarse para conectar a otros tipos de redes y los sistemas de computadora remotos. La computadora personal 2 también puede incluir un controlador de entrada salida 22 para recibir y procesar la entrada de varios dispositivos, incluso un teclado o ratón (no mostrado). Similarmente, un consolador de entrada/salida 22 puede proporcionar el rendimiento a una pantalla de despliegue, una copiadora, u otro tipo de dispositivo de salida.

Como se mencionó brevemente antes, pueden guardarse varios módulos de programa y archivos de datos en el almacenamiento en masa dispositivo 14 y RAM 8 de la computadora personal 2, incluso un sistema operativo 16 conveniente para controlar el funcionamiento de una computadora personal conectada una red de computadoras, como el sistema operativo WINDOWS XP de la CORPORACIÓN MICROSOFT de Redmond, Washington. El almacenamiento en masa dispositivo 14 y RAM 8 también pueden guardar uno o más programas de la aplicación. En particular, el almacenamiento en masa dispositivo 14 y RAM 8 pueden guardar una aplicación programa 305 para crear y revisar un documento electrónico 310. Por ejemplo, la aplicación programa 305 puede comprender un programa de aplicación de procesador de palabras, una aplicación de hoja de cálculo, una aplicación de contactos, y similares. El programa de aplicación para crear y revisar otros tipos de documentos electrónicos también puede usarse con

varias personalizaciones de la invención presente. Un esquema de archivo 330 y un esquema de nombre único (namespace/schema) de biblioteca 400, descrita debajo, también es mostrado.

Las personalizaciones ejemplares de la invención presente son llevadas a cabo por las comunicaciones entre los diferentes objetos de software en un ambiente de programación orientado a objetos. Para los propósitos de la descripción siguiente de personalizaciones de la invención presente, es útil describir componentes de un ambiente de programación orientado a objetos brevemente. La Fig. 2 es un diagrama de bloque simplificado que ilustra la interacción entre el software objeto según un modelo de programación orientado a objetos. De acuerdo con un ambiente de programación orientada a objetos, un primer objeto 210 puede incluir código de software, métodos ejecutables, propiedades, y parámetros. Similarmente, un segundo objeto 220 también puede incluir código de software, métodos ejecutables, propiedades, y parámetros.

Un primer objeto 210 puede comunicarse con un segundo objeto 220 para obtener información o funcionalidad del segundo objeto 220 llamando el segundo objeto 220 vía un mensaje de llamada 230. Como es bien sabido por estos experimentados en el arte de programación orientada a objetos, el primer objeto 210 puede comunicarse con el segundo objeto 220 vía las interfases del programa de aplicación (API) que le permite a dos objetos de software dispares 210, 220 comunicarse entre sí para obtener información y funcionalidad de cada uno. Por ejemplo, si el primer objeto 210 requiere la funcionalidad proporcionada por un método contenido en el segundo objeto 220, el primer objeto 210 puede pasar la llamada 230 a un mensaje del segundo objeto 220 en el que el primer objeto identifica el método requerido y en que el primer objeto pasa cualquier parámetro requerido al segundo objeto requerido por el segundo objeto para operar el método identificado. Una vez el segundo objeto 220 recibe la llamada del primer objeto, el segundo objeto ejecuta el método base llamado sobre los parámetros proporcionados y envía un retorno de mensaje 250 que contiene un valor obtenido atrás del primer objeto 210.

Por ejemplo, en términos de las personalizaciones de la invención presente, y como se describirá debajo, un primer objeto 210 puede ser una tercera parte personalizada de la aplicación que pasa un mensaje al segundo objeto tal como un objeto del esquema de validación XML (Extensible Markup Language) en donde el primer objeto identifica un método que requiere la aprobación de un elemento XML especificado en un documento dónde el elemento XML especificado es un parámetro pasado por el primer objeto con el método identificado. Recibida la llamada del primer objeto, según este ejemplo, el objeto del esquema ejecuta el método identificado sobre el elemento XML específico y retorna un mensaje al primer objeto en la forma de un resultado o valor asociado con el electo XML validado. El funcionamiento de los ambientes de programación

orientados a objeto, como brevemente se describió anteriormente, es bien conocido por estos experimentados en el arte.

Como se describe debajo, se llevan a cabo personalizaciones de la invención presente a través de la interacción de objetos de software en la utilización, la personalización, y aplicación de componentes del Lenguaje Enriquecido Extendido (XML). La Fig. 3 es un diagrama de bloque que ilustra la interacción entre un documento, un archivo del esquema atado, y un módulo de funcionalidad del esquema de validación. Como es bien conocido por los experimentados en el arte, el Lenguaje Extendido Enriquecido (XML) proporciona un método de descripción de texto y datos en un documento permitiéndole a un usuario crear nombres de etiqueta que se aplican al texto o datos en un documento que a su vez define el texto o los datos a que los cuales las etiquetas asociadas son aplicadas. Refiriéndose por ejemplo a la Fig. 3, el documento 310 creado con la aplicación 305 contiene texto que ha sido marcado con etiqueta XML 315, 320, 325. Por ejemplo, el texto "Saludos" es anotado con la etiqueta XML <title>. El texto "Mi nombre es Sarah" se anota con la etiqueta <body>. Según XML, el creador de las etiquetas <title> y <body> es libre de crear sus propias etiquetas para describir las etiquetas a que esas etiquetas se aplicarán. Entonces, tan largo como fluye cualquier aplicación de consumo o máquina de cómputo se proporciona las instrucciones acerca de la definición de las etiquetas aplicadas al texto que la aplicación o la máquina de cómputo pueden utilizar los datos de acuerdo con las etiquetas. Por ejemplo, si una aplicación que fluye se ha programado para extraer texto definido como los títulos de artículos o publicaciones procesadas por esta aplicación, la aplicación puede analizar el documento 310 y puede extraer el texto "Saludos", como es ilustrado en la Fig. 3 porque ese texto se anota con la etiqueta <title>. El creador de la etiqueta particular XML denominada para el documento 310, ilustrado en la Fig. 3, proporciona la descripción útil para el texto o los datos contenidos en el documento 310 que puede ser utilizado por terceras partes tan ampliamente como estas terceras partes se proporcionan con las definiciones asociadas con las etiquetas aplicadas al texto o los datos.

Según las personalizaciones de la invención presente, pueden ahorrarse el texto y la personalización de XML entrados en el documento 310 según una variedad de formatos de archivo diferentes y según el idioma de la programación nativo de la aplicación 305 con que el documento 310 se crea. Por ejemplo, pueden ahorrarse el texto y el enriquecimiento de XML según una aplicación de procesador de palabras, una aplicación de hoja de cálculo, y similares. Alternativamente, pueden ahorrarse el texto y el enriquecimiento de XML entrados en el documento 310 puede ser guardado como un formato XML ya sean texto o datos, cualquier enriquecimiento de XML aplicado, y cualquier formato tal como el de fuente, estilo, estructura de párrafo, etc. puede ser grabado como una representación de XML. En concordancia, el flujo o las aplicaciones de terceras

partes capaz de entender los datos grabados como XML puede abrir y puede consumir el texto o los datos grabados así como una representación de XML. Para una discusión detallada de grabar el texto y el enriquecimiento de XML, los formatos asociados y otros atributos de un documento 310 como XML, vea U.S. patente de Aplicación titulada "Documento Almacenado de Procesador de Palabra en un Archivo Sencillo XML que puede ser Manipulado por las Aplicaciones del entendimiento XML," ("Word Processing Document Stored in a Single XML File that may be manipulated by Applications that Understanding XML," U.S. Serial No. 10/187,060, archivado el 28 de junio de 2002 el cual está aquí incorporado por referencia como si estuviera totalmente puesto aquí dentro.

Para proporcionar un marco de trabajo definible para los elementos enriquecidos de XML (etiquetas) aplicados a texto o datos, como se ilustra en la Fig. 3, se crean los archivos de esquema XML que contienen la información necesaria para permitir a usuarios y consumidores del enriquecido y de los datos almacenados para entender las definiciones diseñadas del etiquetado de XML por el creador del documento. Cada esquema de archivo también referido preferentemente en el arte como un archivo XSD que incluye una inscripción de todos los elementos de XML (etiquetas) que pueden aplicarse a un documento según un archivo del esquema dado. Por ejemplo, un esquema archivo 330, ilustrado en la Fig. 3, puede ser un archivo del esquema que contiene definiciones de ciertos elementos XML que pueden aplicarse a un documento 310 incluso los atributos de elementos de XML o limitaciones y/o reglas asociado con texto o datos que pueden anotarse con los elementos XML según el esquema de archivo. Por ejemplo, refiriéndose al esquema de archivo 330 ilustrado en la Fig. 3, el archivo del esquema se identifica por el Namespace "intro" del archivo del esquema que incluye un elemento raíz de <introCard>.

Según el esquema archivo 330, el elemento <introCard> sirve como un elemento de la raíz para el archivo del esquema y también como un elemento del padre a dos elementos del hijo <title> y <body>. Como es bien conocido por estos experimentados en el arte, pueden definirse varios elementos del padre bajo un solo elemento raíz, y pueden definirse varios elementos del hijo bajo cada elemento del padre. Típicamente, sin embargo, un esquema dado archivo 330 contiene sólo un elemento raíz. Todavía refiriéndose a la Fig. 3, el esquema de archivo 330 también contiene atributos 340 y 345 para el <title> y <body> (<título> y <cuerpo>) de los elementos, respetuosamente. Los atributos 340 y 345 pueden proporcionar la definición extensa o las reglas asociadas con aplicar los elementos respectivos a texto o datos en el documento 310. Por ejemplo, el atributo 345 define ese texto anotado con el elemento <title> que debe ser menor o iguala a veinticinco caracteres de longitud. En concordancia, si el texto excede veinticinco caracteres de longitud se anota con el elemento <title> o etiqueta, la anotación intentada de ese texto no será válido según las definiciones contenidas en el esquema de archivo 330.

Aplicando tales definiciones o reglas como los atributos a los elementos de XML, el creador del esquema puede dictar la estructura de datos contenida en un documento asociado con un archivo del esquema dado. Por ejemplo, si el creador de un esquema archivo 330 para definir enriquecimiento de XML aplicado a un documento del curriculum vitae desea que la sección de experiencia del documento del curriculum vitae contenga no más de cuatro presentes o entradas de trabajo anteriores, el creador del esquema archivo 330 puede definir un atributo del elemento <experiencia>, por ejemplo, para permitir que no más de cuatro presentes o entradas de trabajo pasadas puedan entrarse entre las etiquetas <experiencia> para que el texto de experiencia sea válido según el esquema archivo 330. Como es bien conocido por estos experimentados en el arte, el esquema de archivo 330 puede atarse o por otra parte puede asociarse con un documento 310 dado para la aplicación de enriquecimiento de XML aceptable definida en el archivo del esquema atado al documento 310. Según una personalización, el documento 310 enriquecido con los elementos de XML de los adjuntos o esquemas asociados archivo 330 puede apuntar a los adjuntos o al archivo del esquema asociado apuntando a un identificador del recurso uniforme (URI) asociado con un Namespace que identifica los adjuntos o el esquema asociado de archivo 330.

Según las personalizaciones de la invención presente, un documento 310 puede tener una pluralidad de archivos del esquema atados. Es decir, un creador del documento 310 puede asociar o puede atar más de un esquema archivo 330 al documento 310 para proporcionar un marco de trabajo para la anotación de enriquecimiento XML de más de un archivo del esquema. Por ejemplo, un documento 310 puede contener texto o datos asociados con los datos financieros. Un creador del documento 310 puede desear asociar el esquema de Archivos de XML 330 que contiene el enriquecimiento y las definiciones de XML conteniendo asociadas con las instituciones financieras múltiples. En concordancia, el creador del documento 310 puede asociar un esquema de archivo XML 330 de uno o más instituciones financieras con el documento 310. Igualmente, un esquema de XML dado de archivo 330 puede asociarse con una estructura del documento particular como una plantilla para poner los datos financieros en un formato deseable.

Según las personalizaciones de la invención presente, una colección de esquema de Archivos XML pueden mantener las soluciones del documento asociadas en un Namespace o biblioteca de esquema localizadas separadamente del documento 310. El documento 310 puede contener los indicadores a su vez a URIs en el Namespace o biblioteca del esquema asociadas con uno o más archivos del esquema adjuntos ó por otra parte asociados con el documento 310. Como el documento 310 requiere información de uno o más esquemas de archivos asociados, Los puntos del documento 310 para el Namespace o biblioteca del esquema pueda obtener las definiciones del esquema requeridas. Para una descripción detallada del uso de una operación de Namespace o

bibliotecas del esquema, vea U.S. Aplicación de Patente titulada "Sistema y Método para Proporcionar Información Relacionada de Namespace", U.S. Serial No. 10/184,190, archivado el 27 de junio de 2002, y U.S. Aplicación de Patente titulada "Sistema y Método para Obtener y Usar Información Relacionada de Namespace para Documentos de XML Abiertos" U.S. Serial No. 10/185,940, archivado el 27 de junio de 2002, ambas aplicaciones de patentes americanas que están aquí dentro incorporadas por referencia como si totalmente fueran puestas aquí dentro. Para una descripción detallada de un mecanismo para transmitir los componentes del software como el esquema de Archivos XML y soluciones asociadas de un Namespace o biblioteca del esquema, vea U.S. Aplicación de Patente titulada "Mecanismo para Descargar los Componentes de Software de una Fuente Remota para el Uso por una Aplicación de Software Local, U.S. Serial No. 10/164,260, archivado el 5 de junio de 2002.

Todavía refiriéndose a la Fig. 3, una funcionalidad de aprobación de esquema módulo 350 se ilustra para validar el enriquecimiento de XML aplicado a un documento 310 contra un esquema de archivo XML 330 adjunto ó por otra parte asociado con el documento 310, como se describió anteriormente. Como se describió anteriormente, el archivo del esquema 330 conjuga afuera los elementos de XML aceptables y los atributos asociados y define las reglas para la anotación válida del documento 310 con el enriquecimiento de XML de un esquema asociado de archivo 330. Por ejemplo, como es mostrado en el esquema de archivo 330, dos elementos del hijo <title> y <body> se definen bajo la raíz o elemento del padre <introCard>. Los atributos 340, 345 que definen la longitud del cordón aceptable de texto asociado con los elementos del hijo <title> y <body> también se ilustra. Como se describió anteriormente, si un usuario intenta anotar el documento 310 con el enriquecimiento de XML de un esquema de archivo 330 adjunto ó asociado con el documento en la violación de las definiciones de enriquecimiento de XML contenidas en el esquema de archivo 330, una invalidez o estado del error será presentada. Por ejemplo, si el usuario intenta entrar una cadena de título que excede veinticinco caracteres, la entrada del texto violará el atributo de longitud máximo de carácter del elemento <title> del esquema archivo 330. Para validar enriquecimiento de XML aplicado a un documento 310, contra un esquema asociado de archivo 330, se utiliza un módulo de aprobación del esquema 350. Como debería ser entendido por estos experimentados en el arte, la aprobación del esquema de módulo 350 es un módulo de software que incluye las instrucciones ejecutables de computadora suficientes para comparar el enriquecimiento de XML y el texto asociados entrados en un documento 310 contra uno asociado o adjunto del esquema de archivo XML 330 como el enriquecimiento de XML y el texto asociado son ingresados en el documento 310.

Según las personalizaciones de la invención presente, la aprobación del esquema de módulo 350 compara cada elemento de enriquecimiento de XML y texto asociado o datos aplicados al documento 310 contra los adjuntos o el

esquema asociado de archivo 330 para determinar si cada elemento y texto asociados o los datos obedecen las reglas y definiciones impartidas por el esquema adjunto de archivo 330. Por ejemplo, si un usuario intenta entrar una cadena de caracteres que excede veinticinco caracteres anotados para el <title> de elementos 320, el módulo de aprobación de esquema comparará esa cadena de texto contra la cadena del atributo de texto 340 del esquema adjunto de archivo 330 y determinará que la cadena de texto entrada por el usuario excede la longitud de la cadena de texto aceptable máxima. En concordancia, un mensaje de error o diálogo se presentará al usuario para alertar al usuario que la cadena de texto entrada por el usuario excede la longitud de caracteres aceptable máxima según el esquema adjunto de archivo 330. Igualmente, si el usuario intenta agregar un elemento enriquecimiento de XML entre los elementos <title> y el <body>, el esquema del módulo de validación 350 determinará que el elemento de enriquecimiento XML aplicado por el usuario no es un elemento válido permitido entre los elementos <title> y <body> según el esquema adjunto de archivo 330. En concordancia, la aprobación del esquema de módulo 350 generará un mensaje de error o diálogo para usuario y alertar al usuario que el enriquecimiento de XML no es válido.

Modelo de Objeto programable para las Bibliotecas de Namespaces/Schema

Como se describió anteriormente con referencia a la Fig. 3 para proporcionar un marco de trabajo definible y orientado a reglas para aplicar el Lenguaje de Enriquecimiento Extensible (XML) para un documento 310, uno o más esquemas de archivos 330 pueden asociarse o pueden atarse al documento para poner las definiciones y reglas que gobiernan la aplicación de los elementos de enriquecimiento XML que corresponden a un esquema de archivo dado 330 a un documento 310. Como se describió, una pluralidad de esquema de Archivos XML y otras soluciones del documento, las plantillas por ejemplo pre-estructuradas, pueden adjuntarse o asociarse con un solo documento XML 310. Es más, como se describió anteriormente, varios esquemas de archivos diferentes XML identificados por un Namespace de identificación y un número de soluciones de documento pueden guardarse en un Namespace o biblioteca del esquema aparte del documento 310. Según las personalizaciones de la invención presente, se permiten a los usuarios llamar al Namespace programáticamente o la biblioteca del esquema asociado con uno o más documentos 310 para personalizar o manipular el archivo del esquema por otra parte Namespaces y definiciones asociadas, reglas, recursos, y soluciones asociadas con varios identificadores de Namespace contenidos en el Namespace o biblioteca del esquema.

La Fig. 4 es un diagrama de bloque que ilustra la interacción entre un documento 310, un Namespace o esquema de biblioteca 400 y una tercera parte la aplicación 450. Según las personalizaciones de la invención presente, los

usuarios pueden llamar programáticamente el Namespace de biblioteca 400 vía un conjunto de llamadas de mensajes orientados a objeto o interfases de programa de aplicación 470 para modificar los volúmenes o funcionamiento del esquema individual de archivo 410, 430 o los recursos 420, 440 asociados con archivos del esquema identificados en el Namespace de biblioteca 400. El usuario puede comunicarse con la biblioteca de Namespace de la aplicación 305 o desde la tercera parte programa 450 vía un conjunto de llamadas de mensajes orientadas a objeto, y la tercera parte el programa se desarrolla usando una variedad de lenguajes de programación, como C, C++, C #, Visual Basic, y similares.

Teniendo el acceso a la biblioteca de Namespace a través de un conjunto de interfases de programa de aplicación 470, el usuario puede asociar programáticamente uno o más archivos de esquema XML adicionales o Namespaces con datos XML, y recíprocamente, el usuario puede detectar y puede eliminar las asociaciones existentes entre uno o más esquemas de Archivos XML y datos de XML o enriquecimientos aplicados al documento 310. El usuario también puede programáticamente asociar La Transformación del Lenguaje de Hoja de Estilo Extensible (XSLT) con los datos XML aplicados a un documento, y recíprocamente, el usuario puede detectar y puede eliminar las transformaciones de XSLT existentes de la asociación con los datos de XML aplicados al documento 310. Es más, el usuario puede asociar otros archivos y las aplicaciones de software ejecutables programáticamente con los datos XML aplicados al documento 310 y puede detectar y eliminar las asociaciones existentes de otras aplicaciones de software y archivos con los datos XML.

[Por ejemplo, el Namespace 430 ilustrado en la biblioteca de Namespace 400 puede contener una solución comprendida de una estructura pre-formateada para una plantilla de documento de curriculum vitae. Cuando esa solución se aplica al documento 310, se aplicarán las definiciones del esquema asociadas y reglas diseñadas por el creador del documento de plantilla de curriculum vitae al enriquecimiento de XML y el texto asociado entrados en el documento 310. Si un archivo del esquema se asociara con una plantilla de documento de curriculum vitae que requiere que una sección de experiencia de un documento del curriculum vitae debe tener al menos tres pasados o presentes descripciones de trabajo que la definición del esquema aplicará al documento 310 de tal forma que al menos tres descripciones de trabajo deben ser entradas por un usuario subsiguiente en la sección de experiencia para que los documentos XML 310 puedan ser validados por una aprobación del módulo de esquema 350. Continuando con este ejemplo, si tal plantilla de documento de esquema de del curriculum vitae es asociado con el documento 310, y un usuario desea quitar la asociación de ese archivo del esquema con el documento 310, el usuario lo puede hacer programáticamente desde una tercera parte, el programa, enviando una llamada de mensaje orientada a objeto a la biblioteca de Namespace 450 o a la aplicación 305 con tal de que la interfase del programa de aplicación pueda dirigir

el levantamiento de la asociación del documento plantilla curriculum vitae de esquema de archivo del documento 310.

Lo siguiente es una descripción de objetos y las propiedades asociadas que comprenden las llamadas de mensaje orientado a objeto o las interfaces de Programa de aplicación que le permiten a un usuario programáticamente acceder a la biblioteca del Namespace 400 como se describió anteriormente. Siguiendo cada uno de los objetos y conjunto de propiedades asociadas debajo es una descripción de la operación y funcionalidad del objeto o propiedad asociada.

Objeto de Aplicación

Lo siguientes son los métodos y propiedades del objeto.

.XMLNamespaces property (Propiedad de Nombres únicos de XML)

Una lectura sólo apunta a una colección de XMLNamespaces que representan la biblioteca de Namespace disponibles para la aplicación.

Objeto de colección de XMLNamespaces - un objeto que proporciona el acceso a los objetos de XMLNamespace. Representa la biblioteca de Namespace. Cada objeto de XMLNamespace en la colección representa un solo y único Namespace en la biblioteca de Namespace. Los siguientes son los métodos y propiedades del objeto.

.Add() method - (Agregue () el método)

La creación de un método y la adición a la colección de un nuevo objeto de XMLNamespace. Se usa para registrar un nuevo Namespace en la biblioteca de Namespace. Devuelve un nuevo objeto de XMLNamespace. Puede aceptar los siguientes parámetros.

Path – (La Ruta) - El apuntador al archivo del esquema para el Namespace. El indicador puede ser un camino del archivo representado como una cadena.

NamespaceURI - El URI del Namespace que representa el esquema. El URI puede ser una cadena de texto.

Alias – Una cadena de texto que representa un alternante (más de usuario amigable) es el nombre para el Namespace que el programador puede especificar.

InstallForAllUsers – Instalar Para Todos Los Usuarios - Una bandera que indica si el nuevo Namespace en la biblioteca de Namespace debe estar disponible a todos los usuarios de la computadora o sólo al usuario actual.

.Application Property - .Propiedad de la aplicación

Un indicador de solo lectura para la representación del objeto de aplicación que representa la aplicación de este modelo del objeto.

.Count property - . Propiedad de conteo

Una propiedad de solo lectura que devuelve el número de Namespaces registrados en la biblioteca de Namespace. La propiedad es la misma que el número total de objetos de XMLNamespace en la colección de XMLNamespaces.

.Creator property - .Propiedad de creador

Un indicador de solo lectura para el creador del objeto.

. InstallManifest () method - Método de Instalación manifiesta

Un método para instalar manifiestos de la solución que registran Namespaces en la biblioteca de Namespace. Puede aceptar los parámetros siguientes.

Path – (La Ruta) - El apuntador al archivo manifiesto para el manifiesto. El apuntador puede ser un camino del archivo representado por una cadena del texto.

InstallForAllUsers – Instalar Para Todos Los Usuarios - Una bandera que indica si el nuevo Namespace en la biblioteca de Namespace debe estar disponible a todos los usuarios de la computadora o sólo al usuario actual.

.Item() method -.Artículo() método

Un método para acceder a los miembros individuales de esta colección que usan un índice numérico o una palabra clave de búsqueda. El método puede aceptar los parámetros siguientes.

Index - Índice - un número que representa la posición del objeto Namespace de XML pedido en la biblioteca de Namespace. El Índice también puede ser una cadena de texto que representa el alias (seudónimo) o el URI del Namespace pedido.

.Parent property - . La propiedad del padre

Un propiedad de solo lectura que devuelve el objeto del padre de la colección. Esta propiedad devuelve un apuntador a la aplicación desde la cual la colección de XMLNamespaces se accede.

XMLNamespace object – Objeto XMLNamespace - un objeto que representa una entrada de Namespace individual en la biblioteca de Namespace (y un artículo individual en la colección de XMLNamespaces). Los siguientes son los métodos y propiedades del objeto.

.Alias property - .Propiedad alias

Una propiedad para controlar el seudónimo que el programador asocia con el Namespace. Puede apoyar el parámetro siguiente.

AllUsers – Todos los usuarios - una bandera que indica si el seudónimo está disponible a todos los usuarios o simplemente el usuario actual.

.Application property - .Propiedad de la aplicación

Un apuntador de sólo lectura para la representación del objeto de aplicación de este modelo de objeto.

.AttachToDocument () method - Método Adjuntar-a-documento()

Un método para adjuntar el esquema del Namespace representado por el objeto al documento seleccionado. Apoya los parámetros siguientes:

Document - Documento - un apuntador al documento al cual el esquema se pide ser atado.

.Creator Property - .Propiedad creador

Un indicador de solo lectura para el creador del objeto.

.DefaultTransform property - .Propiedad Transformación Predefinida

Una propiedad que apunta a la transformación de XSLT predefinida asociada con este Namespace. Puede apoyar el parámetro siguiente.

AllUsers – Todos los Usuarios - una bandera que indica si la configuración de la transformación predefinida debe afectar a todos los usuarios de la máquina o sólo al usuario actual.

.Delete() method - .Eliminar() método

Un método para eliminar el XMLNamespace objeto de la colección y destruirlo, quitando efectivamente la asociación de Namespace representada por este objeto de la biblioteca de Namespace.

.Location property - .Propiedad de Ubicación

Una propiedad de solo lectura que controla la situación del esquema asociado con el Namespace representado por el objeto de XMLNamespace. Puede apoyar el parámetro siguiente.

AllUsers – *Todos los usuarios* - una bandera que indica si la configuración de ubicación del esquema debe afectar a todos los usuarios de la máquina o sólo al usuario actual.

.Parent property - .La propiedad padre

Una propiedad de solo lectura que devuelve el objeto del padre del objeto de XMLNamespace. Esta propiedad devuelve un apuntador a la colección de XMLNamespaces de la cual el objeto es un miembro.

.URI property - .La propiedad de URI

Una propiedad de solo lectura que devuelve el URI del Namespace representada por el objeto.

.XSLTransforms property - .Propiedad Transforma XSL

Un apuntador de solo lectura a la colección de XSLTransforms que representa las transformaciones de XSLT asociadas con el Namespace representado por el objeto.

XSLTransforms object – Objeto Transforma XSL - un objeto que proporciona el acceso a los objetos XSLTransforms cada uno de los cuales representa un solo y único XSLT transformación asociada con un Namespace en la biblioteca de Namespace. Lo siguientes son los métodos y propiedades del objeto.

.Add() method - .Adicionar() método

Un método para crear y agregar a la colección un nuevo objeto de XSLTransform. Se usa para asociar una nueva transformación de XSLT con un Namespace en la biblioteca de Namespace. Devuelve un nuevo objeto de XSLTransform. Puede aceptar los parámetros siguientes.

Location – *Ubicación* - apuntador al archivo de XSLT; puede ser una ruta de archivo representada como una cadena de texto.

Alias – Pseudónimo – Una cadena de texto que representa un alternante (más de usuario amigable) el nombre para la transformación de XSLT que el programador puede especificar.

InstallForAllUsers - una bandera que indica si el nuevo Namespace en la biblioteca de Namespace debe estar disponible a todos los usuarios de la computadora o sólo al usuario actual.

.Application property - .Propiedad de la aplicación

Un apuntador de solo lectura al objeto de Aplicación que representa la aplicación de este modelo del objeto.

.Count property - .Propiedad de cuenta

Una propiedad de solo lectura que devuelve el número de registro de las transformaciones de XSLT para un Namespace dado en la biblioteca de Namespace. Es igual que el número total de objetos de XSLTransform en la colección de XSLTransforms.

.Creator property - .Propiedad creador

Un apuntador de solo lectura al creador del objeto.

.Item() method - .Artículo() método

Un método para acceder a los miembros individuales de esta colección que usan un índice numérico o una palabra clave de búsqueda. Puede aceptar los parámetros siguientes.

Index - Índice - un número que representa la posición del objeto XSLTransform requerido en la biblioteca de Namespace. El índice también puede ser una cadena de texto que representa el pseudónimo de la transformación de XSL requerida.

.Parent property - .Propiedad padre

Una propiedad de solo lectura retornando el objeto del padre de la colección. Esta propiedad devuelve un apuntador a la aplicación desde la cual la colección de XSLTransforms es accedida.

XSLTransform object – objeto XSLTransform - un objeto que representa una transformación de XSLT asociada con un Namespace en la biblioteca de Namespace. Los siguientes son los métodos y propiedades del objeto.

.Alias property - .Propiedad Alias

Una propiedad para controlar el seudónimo asociado al programador con el XSLT transform en la biblioteca de Namespace. Puede apoyar el parámetro siguiente.

AllUsers – Todos los Usuarios - una bandera que indica si el seudónimo está disponible a todos los usuarios o simplemente al usuario actual.

.Application property - .La propiedad de aplicación

Un apuntador de solo lectura al objeto de la Aplicación que representa la aplicación de este modelo del objeto.

.Creator property - .La propiedad del creador

Un apuntador de solo lectura al creador del objeto.

.Delete() method - .Eliminar() método

Un método para eliminar el objeto XSLTransform de la colección y destruirlo, quitando la asociación eficazmente entre las transformaciones XSLT y su Namespace en la biblioteca de Namespace.

.Location property - Propiedad de ubicación

Una propiedad de solo lectura que controla la ubicación de la transformación de XSLT asociada con el Namespace dado y representado por el objeto de XSLTransform. Puede apoyar el parámetro siguiente.

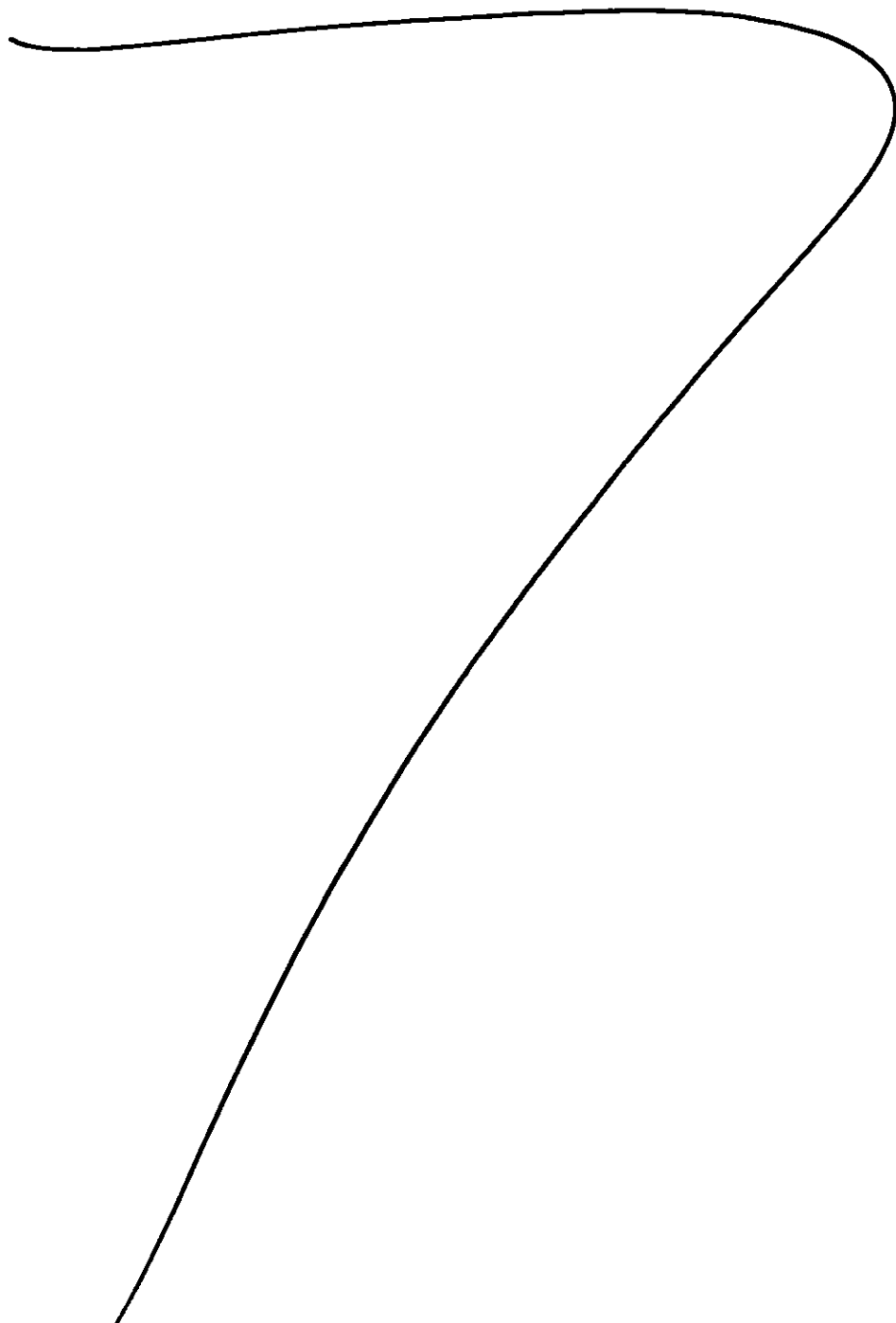
AllUsers – Todos los usuarios - una bandera que indica si la configuración de ubicación de la transformación XSLT debe afectar a todos los usuarios de la máquina o sólo al usuario actual.

.Parent property - .Propiedad padre

Una propiedad de solo lectura que devuelve el objeto del padre del objeto de XSLTransform. Esta propiedad devuelve un indicador a la colección de XSLTransforms de que el objeto es un miembro.

Como está aquí descrito, se mantienen métodos y sistemas permitiéndole a un usuario llamar programáticamente a los recursos identificados en un Namespace de Lenguaje Enriquecido Extensible o biblioteca de esquema para personalizar o modificar la asociación de recursos ó por otra parte modificando la asociación de los recursos identificados o contenidos en el Namespace o biblioteca del esquema

con uno o más documentos asociados. Estará claro para estos experimentados en el arte que pueden hacerse varias modificaciones o variaciones en la invención presente sin partir del alcance o espíritu de la invención. Otras personalizaciones de la invención estarán claras a estos experimentados en el arte desde la consideración de la especificación y práctica de la invención descubierta aquí dentro.



NOSOTROS EXIGIMOS:

1. Un modelo de objeto programable por acceder a los recursos de una biblioteca del esquema de Lenguaje de Enriquecimiento Extensible (XML), comprendiendo:

una interfase del programa de aplicación para permitirle a un usuario acceder los recursos programáticamente identificados en una biblioteca de esquema XML; |

la interfase del programa de aplicación que comprende una llamada del mensaje para pedir asociación de un esquema archivo XML a un enriquecimiento de XML aplicado a un documento; y

la interfase del programa de aplicación operativa para recibir un valor de retorno del esquema de biblioteca XML sensible a la asociación del esquema de archivo XML al enriquecimiento de XML aplicado al documento.

2. El método de la Demanda 1, con la cual la interfase del programa de aplicación más allá comprende una llamada del mensaje para pedir la eliminación de una asociación del esquema archivo XML al enriquecimiento de XML aplicado al documento; y

En donde la interfase del programa de aplicación es más operativa para recibir un valor de retorno del esquema biblioteca XML sensible para quitar de la asociación del esquema archivo XML al enriquecimiento de XML aplicado al documento.

3. Un modelo de objeto programable para acceder los recursos de una biblioteca del esquema de Lenguaje Enriquecido Extensible (XML), comprendiendo:

una interfase del programa de aplicación para permitirle a un usuario acceder programáticamente a los recursos identificados en un esquema de biblioteca XML;

la interfase del programa de aplicación comprende una llamada de mensaje para solicitar asociación de una transformación de XSLT a un enriquecimiento de XML aplicado a un documento; y

la interfase del programa de aplicación operativa para recibir un valor de retorno del esquema de biblioteca XML sensible a la asociación de la transformación de XSLT al enriquecimiento de XML aplicado al documento.

4. El método de la Demanda 3, en donde la interfase del programa de aplicación más allá comprende una llamada del mensaje para pedir la eliminación de una asociación de la transformación de XSLT al enriquecimiento de XML aplicado al documento; y

En donde la interfase del programa de aplicación sea más operativa para recibir un valor del retorno del esquema de biblioteca XML sensible para eliminar

de la asociación de la transformación de XSLT al enriquecimiento de XML aplicado al documento.

5. Un modelo de objeto programable para acceder los recursos de una biblioteca de esquema del Lenguaje Enriquecido Extensible (XML), comprendiendo:

una interfase del programa de aplicación para permitirle a un usuario acceder programáticamente a los recursos identificados en un esquema de biblioteca XML;

la interfase del programa de aplicación que comprende una llamada del mensaje para pedir asociación de un uno o más recursos basados en XML a un enriquecimiento de XML aplicado a un documento; y

la interfase del programa de aplicación operativa para recibir un valor de retorno del esquema de biblioteca XML sensible a la asociación de uno o más recursos basados en XML para el enriquecido XML aplicado al documento.

6. El método de la Demanda 5, en donde la interfase del programa de aplicación más allá comprende una llamada del mensaje para pedir la eliminación de una asociación de uno o más recursos basados en XML para el enriquecido XML aplicado al documento; y

En donde la interfase del programa de aplicación es más operativa para recibir un valor de retorno del esquema de biblioteca XML sensible para remover la asociación de uno o más recursos basados en XML para el enriquecido XML aplicado al documento.

7. Un modelo de objeto programable para acceder los recursos de una biblioteca del esquema de Lenguaje Enriquecido Extensible (XML), comprendiendo:

llamada del esquema de biblioteca XML vía una llamada del mensaje orientada a objeto;

pasar una propiedad de objeto al esquema de biblioteca XML, la propiedad del objeto que es asociado con un objeto del software asociado con la funcionalidad identificada en el esquema de biblioteca XML; y

en respuesta a la llamada del mensaje y a la propiedad del objeto pasada al esquema de biblioteca XML, recibiendo acceso para la funcionalidad identificada en el esquema de biblioteca XML asociado con la propiedad del objeto pasada al esquema de biblioteca XML.

8. El método de la Demanda 7, en donde pasar una propiedad de objeto al esquema de biblioteca XML incluye el paso de una propiedad del método para crear nuevos Namespace XML y para agregar el nuevo Namespace XML a una colección de Namespaces XML en dónde una ruta a un archivo del esquema asociado con el nuevo Namespace XML y un identificador del recurso uniforme

para el nuevo Namespace XML se pasa al esquema de biblioteca XML como los parámetros del objeto del método.

9. El método de la Demanda 7, en donde pasar una propiedad de objeto al esquema de biblioteca XML incluye el paso de una propiedad del método para instalar la solución manifiesta para registrar Namespaces XML en el esquema de biblioteca XML.

10. El método de la Demanda 7, en donde pasar una propiedad de objeto para el esquema de biblioteca XML incluye el paso de una propiedad del método para acceder los recursos de XML individuales de una colección de recursos de XML que usan un índice numérico o búsqueda de palabra clave en donde un índice numérico asociado con un recurso de XML individual se pasa como un parámetro con la propiedad del método.

11. El método de la Demanda 7, pasando la propiedad del objeto con que a la XML esquema biblioteca incluye el paso una propiedad del objeto por controlar un alias nombre asociado con un Namespace especificado identificó en la XML esquema biblioteca.

12. El método de la Demanda 7, en donde pasar una propiedad de objeto para el esquema de biblioteca XML incluye el paso de una propiedad del método para adjuntar un esquema de archivo XML especificado de un Namespace XML especificado a un documento especificado, en donde un apuntador al documento especificado se pasa al esquema de biblioteca XML como un parámetro de la propiedad del método.

13. El método de la Demanda 7, en donde pasar una propiedad de objeto para el esquema de biblioteca XML incluye pasar una propiedad del objeto que apunta a una transformación de XSLT predefinida asociada con un Namespace especificado.

14. El método de la Demanda 7, pasando una propiedad del objeto con que a la XML esquema biblioteca incluye el paso una propiedad del método por quitar un XML Namespace objeto de una colección de objetos de Namespace.

15. El método de la Demanda 7, en donde pasar una propiedad de objeto para el esquema de biblioteca XML incluye el paso de una propiedad del método para crear una nueva transformación de XSLT y para agregar la nueva transformación de XSLT a una colección de transformaciones de XSLT donde un indicador de la nueva transformación de XSLT se pasa al esquema de biblioteca XML como un parámetro de la propiedad del método.

16. El método de la Demanda 7, más allá comprende una propiedad del método para acceder las transformaciones de XSLT individuales contenidas en una colección de transformaciones de XSLT que usan un índice numérico o búsqueda de palabra clave en donde un índice numérico que representa la posición de una transformación de XSLT pedida en el esquema de biblioteca XML se pasa como un parámetro al esquema de biblioteca XML con la propiedad del método.

17. El método de la Demanda 7, en donde pasar una propiedad de objeto para el esquema de biblioteca XML incluye el paso de una propiedad del objeto para controlar un alias del nombre asociado con una transformación de XSLT identificada en el esquema de biblioteca XML.

18. El método de la Demanda 7, en donde pasar una propiedad de objeto para el esquema de biblioteca XML incluye el paso una propiedad del método para eliminar una transformación de XSLT de una colección de transformaciones de XSLT.

**EL MODELO DE OBJETO PROGRAMABLE PARA EL NOMBRE ÚNICO
(NAMESPACE) O ESQUEMA DE APOYO DE BIBLIOTECA EN UNA
APLICACIÓN DE SOFTWARE**

Resumen

Un modelo de objeto programable le permite a un usuario acceder programáticamente y utilizar un Namespace o biblioteca del esquema que contienen el esquema de Archivos XML y los recursos Basados en XML relacionados para asociar el esquema de archivos y los recursos Basados en XML con uno o más documentos y para personalizar la funcionalidad asociada con los archivos del esquema y los recursos Basados en XML. El modelo del objeto programable también permite al usuario/programador para remover el esquema de archivos y otros recursos Basados en XML desde la asociación con los documentos previamente asociados.

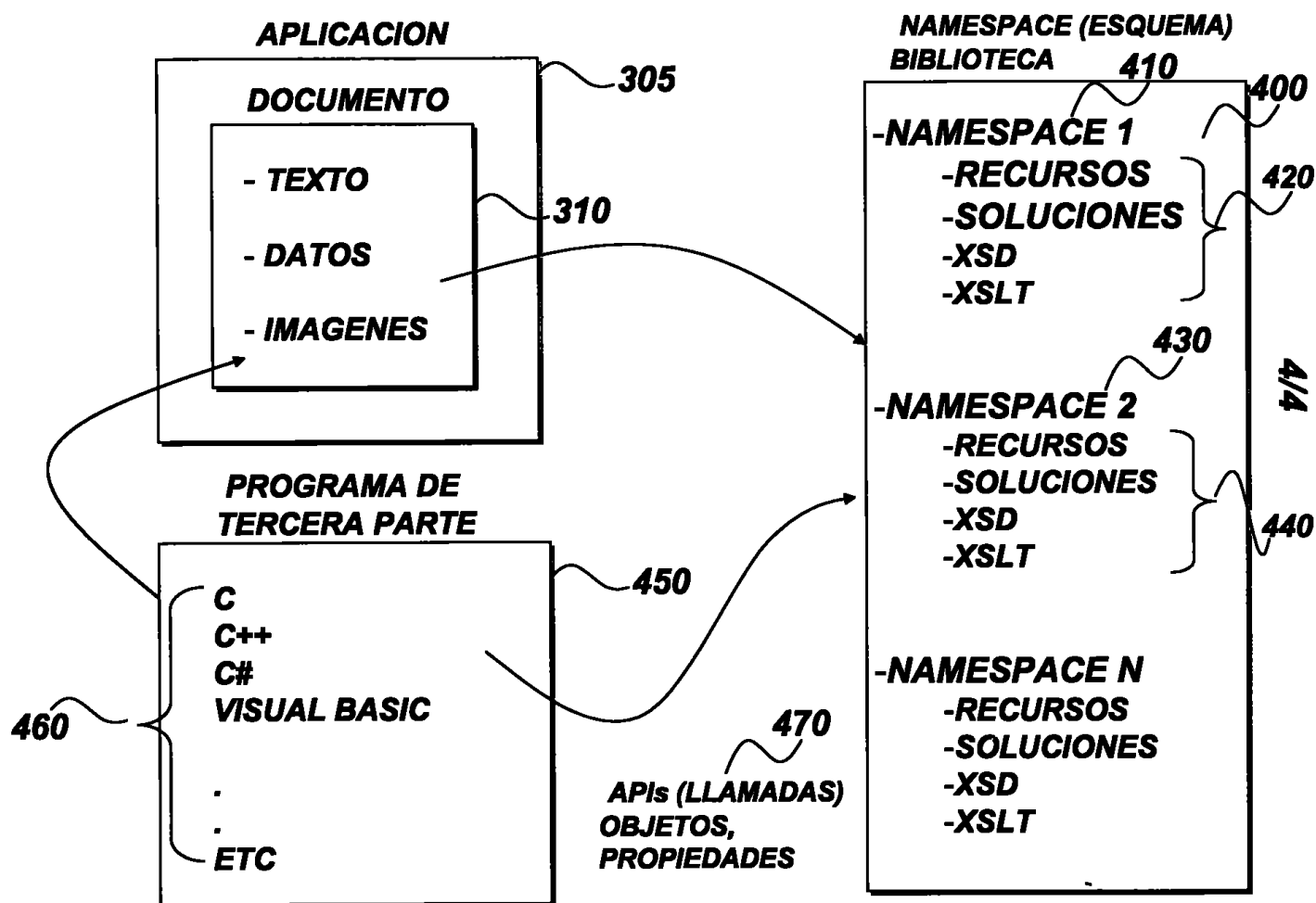


Fig. 4

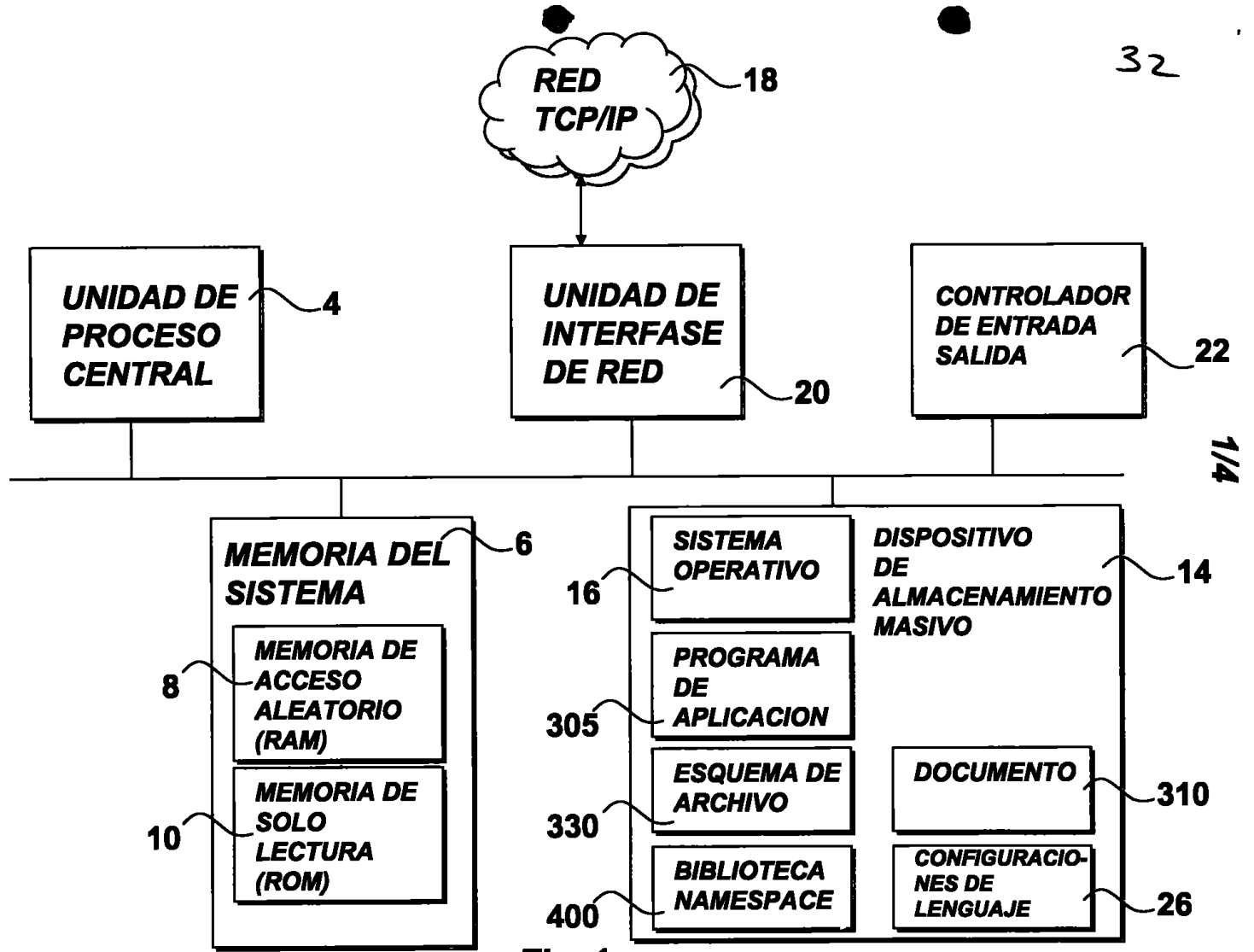


Fig. 1

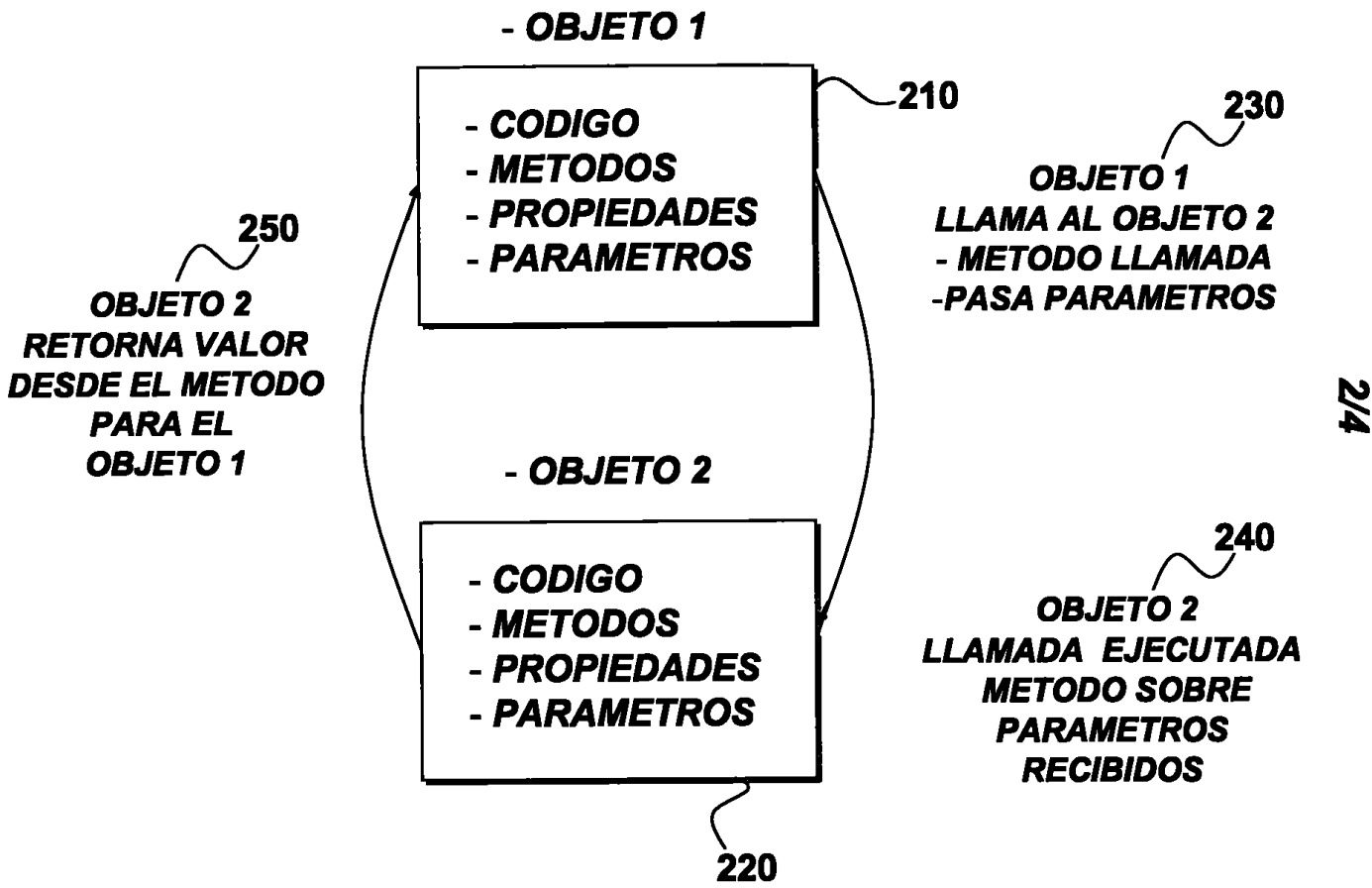


Fig. 2

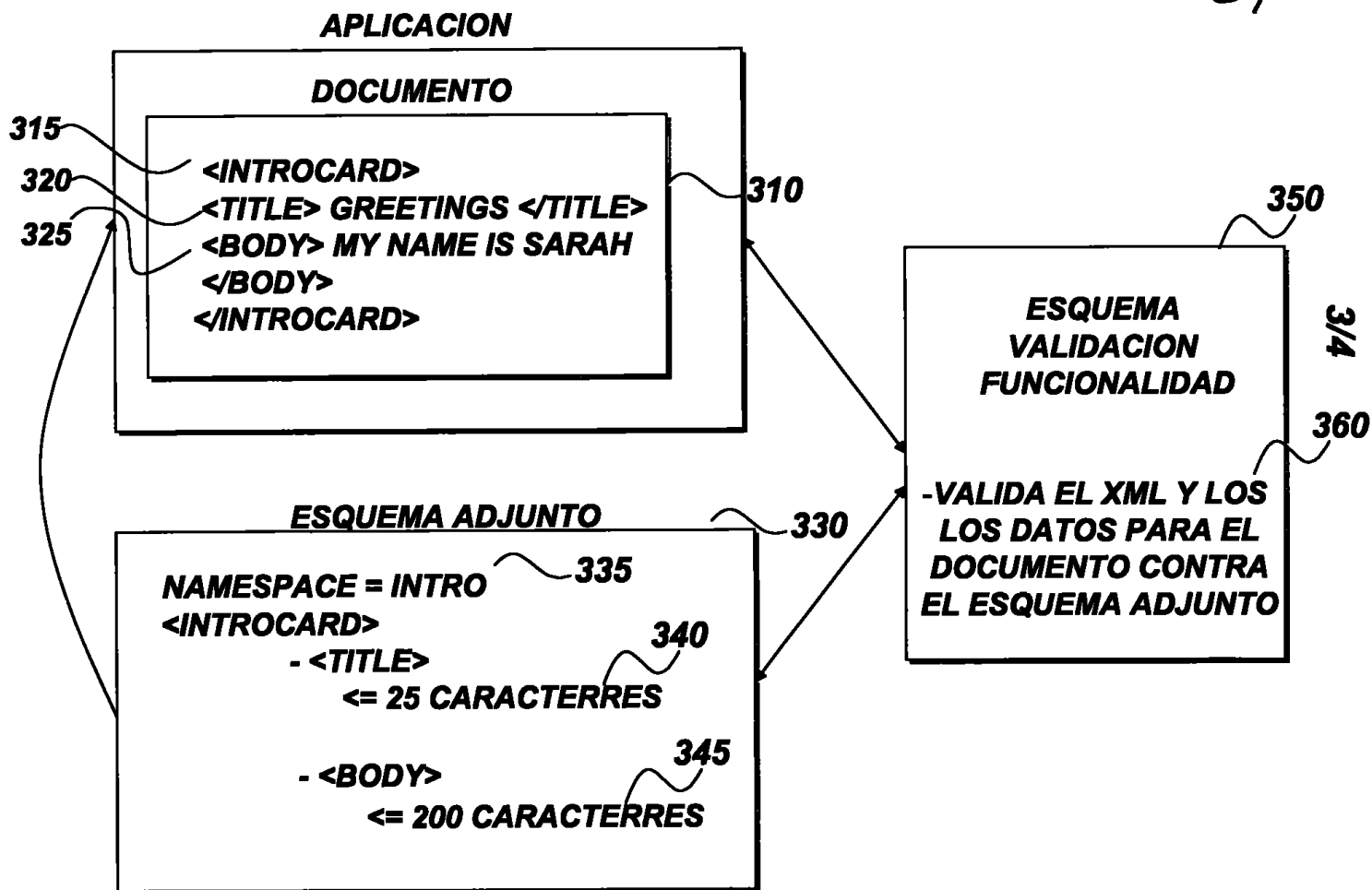


Fig. 3

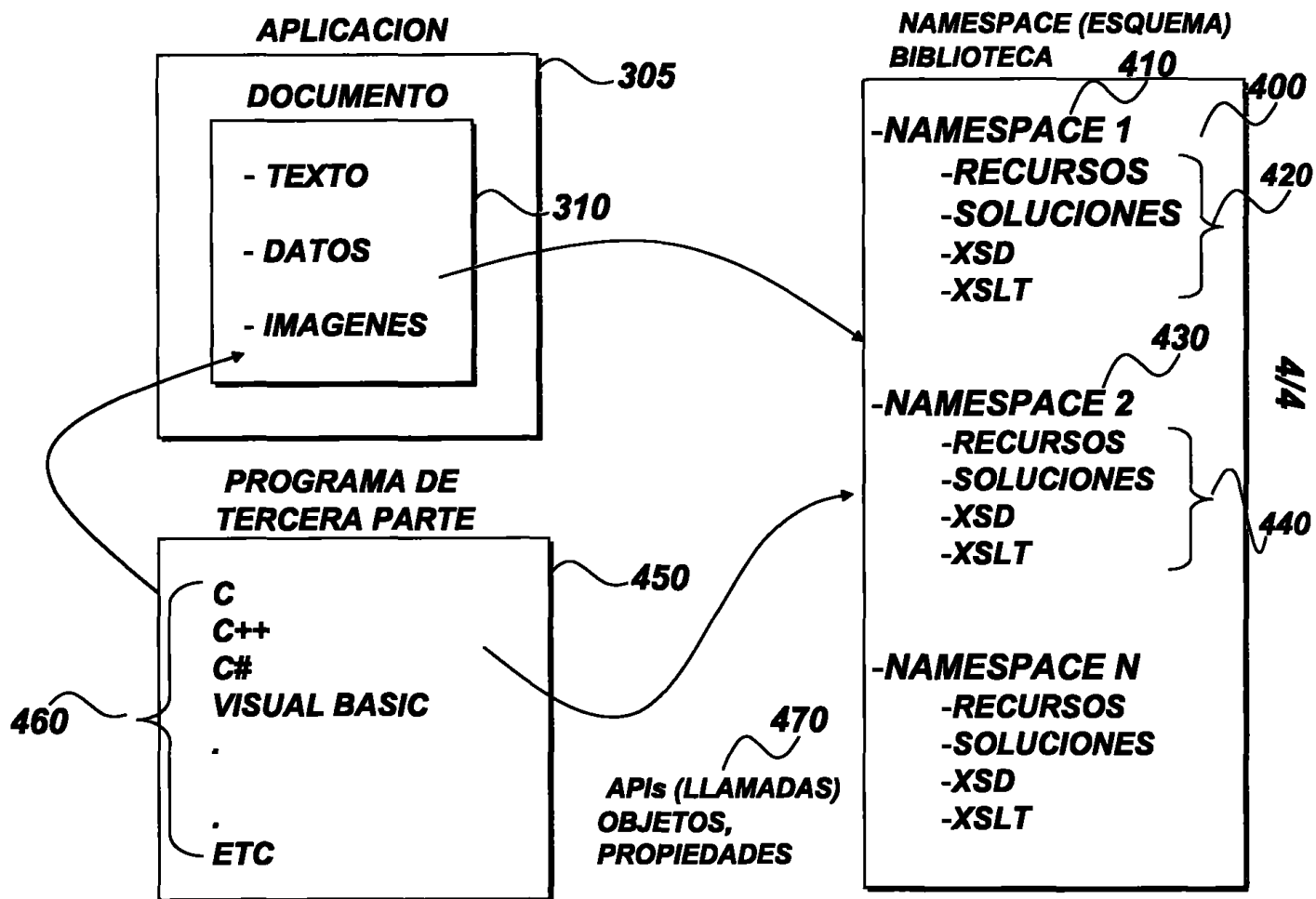


Fig. 4

**PROGRAMMABLE OBJECT MODEL FOR NAMESPACE OR SCHEMA
LIBRARY SUPPORT IN A SOFTWARE APPLICATION**

5

Related Applications

United States Utility Patent Application by applicant matter number 60001.0263US01/MS303917.1, entitled "Programmable Object Model for Extensible Markup Language Schema Validation," and United States Utility Patent Application by applicant matter number 60001.0264US01/MS303918.1, entitled "Programmable
10 Object Model for Extensible Markup Language Markup in an Application," are hereby incorporated by reference.

Copyright Notice

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the
15 facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the United States Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

Field of the Invention

The present invention relates to programmable object models. More
20 particularly, the present invention relates to a programmable object model for Namespace or schema library support in a software application.

Background of the Invention

Computer software applications allow users to create a variety of
25 documents to assist them in work, education, and leisure. For example, popular word processing applications allow users to create letters, articles, books, memoranda, and the like. Spreadsheet applications allow users to store, manipulate, print, and display a variety of alphanumeric data. Such applications have a number of well known strengths including rich editing, formatting, printing, calculation and on-line and off-line editing.

30 Most computer software applications do not contain all necessary programming for providing functionality required or desired by every potential user. Many programmers often wish to take advantage of an existing application's capabilities in their own programs or to customize the functionality of an application and make it more suitable for a specific set of users or actions. For example, a
35 programmer working in the financial industry may wish to customize a word processor for a user audience consisting of financial analysts editing financial reports. In recent years, the Extensible Markup Language has been used widely as an interchangeable

data format for many users. Often users of XML functionality attach or associate one or more XML schema files or XML-based solutions to a document being edited or created by the user. However, users/programmers are limited in their ability to apply XML schema files and other XML-based solutions functionality to a given document because
5 the user/programmer does not have direct and easy access to the Namespace or schema library containing the XML schema files or other XML-based solutions.

Accordingly, there is a need in the art for a programmable object model for allowing a user/programmer to access a Namespace or schema library of XML resources for customizing or otherwise manipulating the resources to enhance the
10 user/programmer's use of XML functionality with a software application document. It is with respect to these and other considerations that the present invention has been made.

Summary of the Invention

15 The present invention provides methods and systems for allowing a user to programmatically access and utilize a Namespace or schema library containing XML schema files and related XML-based resources for associating those XML-based resources with one or more documents and for customizing the functionality associated with those XML-based resources. Once a user or programmer obtains access to the
20 Namespace or schema library, the user may programmatically associate XML schema files with XML data in an associated document, and conversely, the user may detect and remove associations of XML schema files with XML data contained in the document. The user may also programmatically associate transformation files with XML data contained in a document and detect and remove existing transformation files associated
25 with XML data contained in the document. The user may also associate other files and executable software associated with XML-based and other document solutions with XML data contained in the document. Additionally, the user may detect and delete the association of XML-based solutions and other types of executable software from association with XML data contained in a document.

30 These and other features, advantages, and aspects of the present invention may be more clearly understood and appreciated from a review of the following detailed description of the disclosed embodiments and by reference to the appended drawings and claims.

35 **Brief Description of the Drawings**

Fig. 1 is a simplified block diagram of a computing system and associated peripherals and network devices that provide an exemplary operating environment for the present invention.

Fig. 2 is a simplified block diagram illustrating interaction between software objects according to an object-oriented programming model.

Fig. 3 is a block diagram illustrating interaction between a document, an attached schema file, and a schema validation functionality model.

5 Fig. 4 is a block diagram illustrating interaction between a document, a Namespace or schema library and a third party software application.

Detailed Description of the Preferred Embodiment

Embodiments of the present invention are directed to methods and systems for allowing a user to programmatically call a Namespace/Schema library of
10 XML schema files and XML-based solutions and resources for controlling the association of those files, solutions and resources to one or more documents. These embodiments may be combined, other embodiments may be utilized, and structural changes may be made without departing from the spirit or scope of the present invention. The following detailed description is therefore not to be taken in a limiting
15 senses and the scope of the present invention is defined by the appended claims and their equivalents.

Referring now to the drawings, in which like numerals represent like elements through the several figures, aspects of the present invention and the exemplary operating environment will be described. Fig. 1 and the following discussion are
20 intended to provide a brief, general description of a suitable computing environment in which the invention may be implemented. While the invention will be described in the general context of program modules that execute in conjunction with an application program that runs on an operating system on a personal computer, those skilled in the art will recognize that the invention may also be implemented in combination with other
25 program modules.

Generally, program modules include routines, programs, components, data structures, and other types of structures that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand-
30 held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, minicomputers, mainframe computers, and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and
35 remote memory storage devices.

Turning now to Fig. 1, an illustrative computer architecture for a personal computer 2 for practicing the various embodiments of the invention will be described. The computer architecture shown in Fig. 1 illustrates a conventional personal computer, including a central processing unit 4 ("CPU"), a system memory 6,
40 including a random access memory 8 ("RAM") and a read-only memory ("ROM") 10,

and a system bus 12 that couples the memory to the CPU 4. A basic input/output system containing the basic routines that help to transfer information between elements within the computer, such as during startup, is stored in the ROM 10. The personal computer 2 further includes a mass storage device 14 for storing an operating system 16, application programs, such as the application program 305, and data.

The mass storage device 14 is connected to the CPU 4 through a mass storage controller (not shown) connected to the bus 12. The mass storage device 14 and its associated computer-readable media, provide non-volatile storage for the personal computer 2. Although the description of computer-readable media contained herein refers to a mass storage device, such as a hard disk or CD-ROM drive, it should be appreciated by those skilled in the art that computer-readable media can be any available media that can be accessed by the personal computer 2.

By way of example, and not limitation, computer-readable media may comprise computer storage media and communication media. Computer storage media includes volatile and non-volatile, removable and non-removable media implemented in any method or technology for storage of information such as computer-readable instructions, data structures, program modules or other data. Computer storage media includes, but is not limited to, RAM, ROM, EPROM, EEPROM, flash memory or other solid state memory technology, CD-ROM, DVD, or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by the computer.

According to various embodiments of the invention, the personal computer 2 may operate in a networked environment using logical connections to remote computers through a TCP/IP network 18, such as the Internet. The personal computer 2 may connect to the TCP/IP network 18 through a network interface unit 20 connected to the bus 12. It should be appreciated that the network interface unit 20 may also be utilized to connect to other types of networks and remote computer systems. The personal computer 2 may also include an input/output controller 22 for receiving and processing input from a number of devices, including a keyboard or mouse (not shown). Similarly, an input/output controller 22 may provide output to a display screen, a printer, or other type of output device.

As mentioned briefly above, a number of program modules and data files may be stored in the mass storage device 14 and RAM 8 of the personal computer 2, including an operating system 16 suitable for controlling the operation of a networked personal computer, such as the WINDOWS XP operating system from MICROSOFT CORPORATION of Redmond, Washington. The mass storage device 14 and RAM 8 may also store one or more application programs. In particular, the mass storage device 14 and RAM 8 may store an application program 305 for creating and editing an electronic document 310. For instance, the application program 305 may comprise a word processing application program, a spreadsheet application, a contact application,

and the like. Application programs for creating and editing other types of electronic documents may also be used with the various embodiments of the present invention. A schema file 330 and a namespace/schema library 400, described below, are also shown.

Exemplary embodiments of the present invention are implemented by communications between different software objects in an object-oriented programming environment. For purposes of the following description of embodiments of the present invention, it is useful to briefly to describe components of an object-oriented programming environment. Fig. 2 is a simplified block diagram illustrating interaction between software objects according to an object-oriented programming model. According to an object-oriented programming environment, a first object 210 may include software code, executable methods, properties, and parameters. Similarly, a second object 220 may also include software code, executable methods, properties, and parameters.

A first object 210 may communicate with a second object 220 to obtain information or functionality from the second object 220 by calling the second object 220 via a message call 230. As is well known to those skilled in the art of object-oriented programming environment, the first object 210 may communicate with the second object 220 via application programming interfaces (API) that allow two disparate software objects 210, 220 to communicate with each other in order to obtain information and functionality from each other. For example, if the first object 210 requires the functionality provided by a method contained in the second object 220, the first object 210 may pass a message call 230 to the second object 220 in which the first object identifies the required method and in which the first object passes any required parameters to the second object required by the second object for operating the identified method. Once the second object 220 receives the call from the first object, the second object executes the called method based on the provided parameters and sends a return message 250 containing a value obtained from the executed method back to the first object 210.

For example, in terms of embodiments of the present invention, and as will be described below, a first object 210 may be a third party customized application that passes a message to a second object such as an Extensible Markup Language schema validation object whereby the first object identifies a method requiring the validation of a specified XML element in a document where the specified XML element is a parameter passed by the first object with the identified method. Upon receipt of the call from the first object, according to this example, the schema validation object executes the identified method on the specified XML element and returns a message to the first object in the form of a result or value associated with the validated XML element. Operation of object-oriented programming environments, as briefly described above, are well known to those skilled in the art.

As described below, embodiments of the present invention are implemented through the interaction of software objects in the use, customization, and

application of components of the Extensible Markup Language (XML). Fig. 3 is a block diagram illustrating interaction between a document, an attached schema file, and a schema validation functionality module. As is well known to those skilled in the art, the Extensible Markup Language (XML) provides a method of describing text and data in a document by allowing a user to create tag names that are applied to text or data in a document that in turn define the text or data to which associated tags are applied. For example referring to Fig. 3, the document 310 created with the application 305 contains text that has been marked up with XML tags 315, 320, 325. For example, the text "Greetings" is annotated with the XML tag <title>. The text "My name is Sarah" is annotated with the <body> tag. According to XML, the creator of the <title> and <body> tags is free to create her own tags for describing the tags to which those tags will be applied. Then, so long as any downstream consuming application or computing machine is provided instructions as to the definition of the tags applied to the text, that application or computing machine may utilize the data in accordance with the tags. For example, if a downstream application has been programmed to extract text defined as titles of articles or publications processed by that application, the application may parse the document 310 and extract the text "Greetings," as illustrated in Fig. 3 because that text is annotated with the tag <title>. The creator of the particular XML tag naming for the document 310, illustrated in Fig. 3, provides useful description for text or data contained in the document 310 that may be utilized by third parties so long as those third parties are provided with the definitions associated with tags applied to the text or data.

According to embodiments of the present invention, the text and XML markup entered into the document 310 may be saved according to a variety of different file formats and according to the native programming language of the application 305 with which the document 310 is created. For example, the text and XML markup may be saved according to a word processing application, a spreadsheet application, and the like. Alternatively, the text and XML markup entered into the document 310 may be saved as an XML format whereby the text or data, any applied XML markup, and any formatting such as font, style, paragraph structure, etc. may be saved as an XML representation. Accordingly, downstream or third party applications capable of understanding data saved as XML may open and consume the text or data thus saved as an XML representation. For a detailed discussion of saving text and XML markup and associated formatting and other attributes of a document 310 as XML, see U.S. Patent Application entitled "Word Processing Document Stored in a Single XML File that may be Manipulated by Applications that Understanding XML," U.S. Serial No. 10/187,060, filed June 28, 2002, which is incorporated herein by reference as if fully set out herein.

In order to provide a definitional framework for XML markup elements (tags) applied to text or data, as illustrated in Fig. 3, XML schema files are created which contain information necessary for allowing users and consumers of marked up and stored data to understand the XML tagging definitions designed by the creator of

the document. Each schema file also referred to in the art as a XSD file preferably includes a listing of all XML elements (tags) that may be applied to a document according to a given schema file. For example, a schema file 330, illustrated in Fig. 3, may be a schema file containing definitions of certain XML elements that may be applied to a document 310 including attributes of XML elements or limitations and/or rules associated with text or data that may be annotated with XML elements according to the schema file. For example, referring to the schema file 330 illustrated in Fig. 3, the schema file is identified by the Namespace "intro" the schema file includes a root element of <introCard>.

10 According to the schema file 330, the <introCard> element serves as a root element for the schema file and also as a parent element to two child elements <title> and <body>. As is well known to those skilled in the art, a number of parent elements may be defined under a single root element, and a number of child elements may be defined under each parent element. Typically, however, a given schema file 15 330 contains only one root element. Referring still to Fig. 3, the schema file 330 also contains attributes 340 and 345 to the <title> and <body> elements, respectfully. The attributes 340 and 345 may provide further definition or rules associated with applying the respective elements to text or data in the document 310. For example, the attribute 20 345 defines that text annotated with the <title> element must be less than or equal to twenty-five characters in length. Accordingly, if text exceeding twenty-five characters in length is annotated with the <title> element or tag, the attempted annotation of that text will be invalid according to the definitions contained in the schema file 330.

By applying such definitions or rules as attributes to XML elements, the creator of the schema may dictate the structure of data contained in a document associated with a given schema file. For example, if the creator of a schema file 330 for defining XML markup applied to a resume document desires that the experience section of the resume document contain no more than four present or previous job entries, the creator of the schema file 330 may define an attribute of an <experience> element, for example, to allow that no more than four present or past job entries may be entered 30 between the <experience> tags in order for the experience text to be valid according to the schema file 330. As is well known to those skilled in the art, the schema file 330 may be attached to or otherwise associated with a given document 310 for application of allowable XML markup defined in the attached schema file to the document 310. According to one embodiment, the document 310 marked up with XML elements of the 35 attached or associated schema file 330 may point to the attached or associated schema file by pointing to a uniform resource identifier (URI) associated with a Namespace identifying the attached or associated schema file 330.

According to embodiments of the present invention, a document 310 may have a plurality of attached schema files. That is, a creator of the document 310 40 may associate or attach more than one schema file 330 to the document 310 in order to provide a framework for the annotation of XML markup from more than one schema

file. For example, a document 310 may contain text or data associated with financial data. A creator of the document 310 may wish to associate XML schema files 330 containing XML markup and definitions associated with multiple financial institutions. Accordingly, the creator of the document 310 may associate an XML schema file 330
5 from one or more financial institutions with the document 310. Likewise, a given XML schema file 330 may be associated with a particular document structure such as a template for placing financial data into a desirable format.

According to embodiments of the present invention, a collection of XML schema files and associated document solutions may be maintained in a Namespace or
10 schema library located separately from the document 310. The document 310 may in turn contain pointers to URIs in the Namespace or schema library associated with the one or more schema files attached to otherwise associated with the document 310. As the document 310 requires information from one or more associated schema files, the document 310 points to the Namespace or schema library to obtain the required schema
15 definitions. For a detailed description of the use of an operation of Namespace or schema libraries, see U.S. Patent Application entitled "System and Method for Providing Namespace Related Information," U.S. Serial No. 10/184,190, filed June 27, 2002, and U.S. Patent Application entitled "System and Method for Obtaining and Using Namespace Related Information for Opening XML Documents," U.S. Serial No.
20 10/185,940, filed June 27, 2002, both U.S. patent applications of which are incorporated herein by reference as if fully set out herein. For a detailed description of a mechanism for downloading software components such as XML schema files and associated solutions from a Namespace or schema library, see US Patent Application entitled Mechanism for Downloading Software Components from a Remote Source for Use by a
25 Local Software Application, US Serial No. 10/164,260, filed June 5, 2002.

Referring still to Fig. 3, a schema validation functionality module 350 is illustrated for validating XML markup applied to a document 310 against an XML schema file 330 attached to or otherwise associated with the document 310, as described above. As described above, the schema file 330 sets out acceptable XML elements and
30 associated attributes and defines rules for the valid annotation of the document 310 with XML markup from an associated schema file 330. For example, as shown in the schema file 330, two child elements <title> and <body> are defined under the root or parent element <introCard>. Attributes 340, 345 defining the acceptable string length of text associated with the child elements <title> and <body> are also illustrated. As
35 described above, if a user attempts to annotate the document 310 with XML markup from a schema file 330 attached to or associated with the document in violation of the XML markup definitions contained in the schema file 330, an invalidity or error state will be presented. For example, if the user attempts to enter a title string exceeding twenty-five characters, that text entry will violate the maximum character length
40 attribute of the <title> element of the schema file 330. In order to validate XML markup applied to a document 310, against an associated schema file 330, a schema

validation module 350 is utilized. As should be understood by those skilled in the art, the schema validation module 350 is a software module including computer executable instructions sufficient for comparing XML markup and associated text entered in to a document 310 against an associated or attached XML schema file 330 as the XML markup and associated text is entered in to the document 310.

According to embodiments of the present invention, the schema validation module 350 compares each XML markup element and associated text or data applied to the document 310 against the attached or associated schema file 330 to determine whether each element and associated text or data complies with the rules and definitions set out by the attached schema file 330. For example, if a user attempts to enter a character string exceeding twenty-five characters annotated by the <title> elements 320, the schema validation module will compare that text string against the text string attribute 340 of the attached schema file 330 and determine that the text string entered by the user exceeds the maximum allowable text string length. Accordingly, an error message or dialogue will be presented to the user to alert the user that the text string being entered by the user exceeds the maximum allowable character length according to the attached schema file 330. Likewise, if the user attempts to add an XML markup element between the <title> and the <body> elements, the schema validation module 350 will determine that the XML markup element applied by the user is not a valid element allowed between the <title> and <body> elements according to the attached schema file 330. Accordingly, the schema validation module 350 will generate an error message or dialogue to the user to alert the user of the invalid XML markup.

25 Programmable Object Model for Namespaces/Schema Libraries

As described above with reference to Fig. 3, in order to provide a definitional and rules-oriented framework for applying Extensible Markup Language (XML) markup to a document 310, one or more schema files 330 may be associated or attached to the document for setting definitions and rules governing the application of XML markup elements corresponding to a given schema file 330 to a document 310. As described, a plurality of XML schema files and other document solutions, for example pre-structured templates, may be attached to or associated with a single XML document 310. Moreover, as described above, a number of different XML schema files identified by a Namespace identification and a number of document solutions may be stored in a Namespace or schema library apart from the document 310. According to embodiments of the present invention, users are allowed to programmatically call the Namespace or schema library associated with one or more documents 310 for customizing or otherwise manipulating schema file Namespaces and associated definitions, rules, resources, and solutions associated with various Namespace identifiers contained in the Namespace or schema library.

Fig. 4 is a block diagram illustrating interaction between a document 310, a Namespace or schema library 400 and a third party application 450. According to embodiments of the present invention, users may programmatically call the Namespace library 400 via a set of object-oriented message calls or application programming interfaces 470 for modifying the contents or operation of individual schema files 410, 430 or resources 420, 440 associated with schema files identified in the Namespace library 400. The user may communicate with the Namespace library from the application 305 or from a third party program 450 via a set of object-oriented message calls, and the third party program may be developed using a variety of programming languages, such as C, C++, C#, Visual Basic, and the like.

By having access to the Namespace library through a set of application programming interfaces 470, the user may programmatically associate one or more additional XML schema files or Namespaces with XML data, and conversely, the user may detect and remove existing associations between one or more XML schema files and XML data or markup applied to the document 310. The user may also programmatically associate Extensible Stylesheet Language Transformation (XSLT) with XML data applied to a document, and conversely, the user may detect and remove existing XSLT transforms from association with XML data applied to the document 310. Moreover, the user may programmatically associate other files and executable software applications with XML data applied to the document 310 and detect and remove existing associations of other software applications and files with XML data.

For example, the Namespace 430 illustrated in the Namespace library 400 may contain a solution comprised of a pre-formatted structure for a resume document template. When that solution is applied to the document 310, associated schema definitions and rules designed by the creator of the resume template document will be applied to XML markup and associated text entered into the document 310. If a schema file associated with a resume document template requires that an experience section of a resume document must have at least three past or present job descriptions, that schema definition will be applied to the document 310 such that at least three job descriptions must be entered by a subsequent user in the experience section in order for the XML document 310 to be validated by a schema validation module 350. Continuing with this example, if such a resume document template schema file is associated with the document 310, and a user desires to remove the association of that schema file with the document 310, the user may do so programmatically from a third party program by sending an object-oriented message call to the Namespace library 450 or to the application 305 with a provided application programming interface for directing the removal of the association of the resume document template schema file from the document 310.

The following is a description of objects and associated properties comprising object-oriented message calls or application programming interfaces that allow a user to programmatically access the Namespace library 400 as described above.

Following each of the objects and associated properties set out below is a description of the operation and functionality of the object or associated property.

Application object

5

The following are methods and properties of the object.

.XMLNamespaces property

10

A read only pointer to an XMLNamespaces collection which represents the Namespace library available to the application.

XMLNamespaces collection object - an object providing access to the XMLNamespace objects. It represents the Namespace library. Each XMLNamespace object in the collection represents a single and unique Namespace in the Namespace library. The following are methods and properties of the object.

15

.Add() method

20

A method creating and adding to the collection a new XMLNamespace object. It is used to register a new Namespace in the Namespace library. It returns a new XMLNamespace object. It can accept the following parameters.

25

Path – pointer to the schema file for the Namespace. The pointer can be a file path represented as a string.

NamespaceURI – the URI of the Namespace that represents the schema. The URI can be a text string.

Alias – a text string representing an alternate (more user-friendly) name for the Namespace that the programmer may specify.

30

InstallForAllUsers – a flag indicating whether the new Namespace in the Namespace library should be available to all users of the computer or only the current user.

.Application property

35

A read only pointer to the application object representing the application of this object model.

.Count property

40

A read only property returning the number of registered Namespaces in the Namespace library. The property is the same as the total number of XMLNamespace objects in the XMLNamespaces collection.

.Creator property

A read only pointer to the creator of the object.

5

.InstallManifest() method

A method for installing solution manifests that register Namespaces in the Namespace library. It can accept the following parameters.

10

Path – pointer to the manifest file for the manifest. The pointer can be a file path represented by a text string.

InstallForAllUsers - a flag indicating whether the new Namespaces installed in the Namespace library by the manifest should be available to all users of the computer or only the current user.

15

.Item() method

A method for accessing the individual members of this collection using an numerical index or a search keyword. The method can accept the following parameters.

20

Index – a number representing the position of the requested XML Namespace object in the Namespace library. The index can also be a text string representing the alias or the URI of the requested Namespace.

.Parent property

25

A read only property returning the parent object of the collection. This property returns a pointer to the application from which the XMLNamespaces collection is accessed.

30

XMLNamespace object – an object representing an individual Namespace entry in the Namespace library (and an individual item in the XMLNamespaces collection). The following are methods and properties of the object.

.Alias property

35

A property for controlling the alias the programmer associates with the Namespace. It can support the following parameter.

AllUsers – a flag indicating whether the alias is available to all users or just the current user.

40

.Application property

A read only pointer to the Application object representing the application of this object model.

48

.AttachToDocument() method

5 A method for attaching the schema of the Namespace represented by the object to the selected document. It supports the following parameters:

Document – a pointer to the document to which the schema is requested to be attached.

10

.Creator property

A read only pointer to the creator of the object.

.DefaultTransform property

15 A property that points to the default XSLT transformation associated with this Namespace. It can support the following parameter.

AllUsers – a flag indicating whether the default transformation setting should affect all users of the machine or only the current user.

20

.Delete() method

A method for removing the XMLNamespace object from the collection and destroying it, effectively removing the Namespace association represented by this object from the Namespace library.

25

.Location property

A read only property that controls the location of the schema associated with the Namespace represented by the XMLNamespace object. It can support the following parameter.

30

AllUsers – a flag indicating whether the schema location setting should affect all users of the machine or only the current user.

.Parent property

35 A read only property returning the parent object of the XMLNamespace object. This property returns a pointer to the XMLNamespaces collection of which the object is a member.

.URI property

40 A read only property returning the URI of the Namespace represented by the object.

.XSLTransforms property

5 A read only pointer to the XSLTransforms collection representing XSLT transformations associated with the Namespace represented by the object.

XSLTransforms object - an object providing access to the XSLTransforms objects each of which represents a single and unique XSLT transform associated with a
10 Namespace in the Namespace library. The following are methods and properties of the object.

.Add() method

15 A method for creating and adding to the collection a new XSLTransform object. It is used to associate a new XSLT transformation with a Namespace in the Namespace library. It returns a new XSLTransform object. It can accept the following parameters.

20 *Location* – pointer to the XSLT file; can be a file path represented as a text string.

Alias – a text string representing an alternate (more user-friendly) name for the XSLT transformation that the programmer may specify.

25 *InstallForAllUsers* – a flag indicating whether the new Namespace in the Namespace library should be available to all users of the computer or only the current user.

.Application property

30 A read only pointer to the Application object representing the application of this object model.

.Count property

35 A read only property returning the number of registered XSLT transforms for a given Namespace in the Namespace library. It is the same as the total number of XSLTransform objects in the XSLTransforms collection.

.Creator property

A read only pointer to the creator of the object.

40 **.Item() method**

A method for accessing the individual members of this collection using an numerical index or a search keyword. It can accept the following parameters.

5 *Index* – a number representing the position of the requested XSLTransform object in the Namespace library. The index can also be a text string representing the alias of the requested XSL transform.

.Parent property

10 A read only property returning the parent object of the collection. This property returns a pointer to the application from which the XSLTransforms collection is accessed.

15 **XSLTransform object** - an object representing an XSLT transformation associated with a Namespace in the Namespace library. The following are methods and properties of the object.

.Alias property

20 A property for controlling the alias the programmer associated with the XSLT transform in the Namespace library. It can support the following parameter.

25 *AllUsers* – a flag indicating whether the alias is available to all users or just the current user.

.Application property

 A read only pointer to the Application object representing the application of this object model.

30

.Creator property

 A read only pointer to the creator of the object.

.Delete() method

35 A method for removing the XSLTransform object from the collection and destroying it, effectively removing the association between the XSLT transform and its Namespace in the Namespace library.

.Location property

51
A read only property that controls the location of the XSLT transform associated with the given Namespace and represented by the XSLTransform object. It can support the following parameter.

5
AllUsers – a flag indicating whether the XSLT transform location setting should affect all users of the machine or only the current user.

.Parent property

10 A read only property returning the parent object of the XSLTransform object. This property returns a pointer to the XSLTransforms collection of which the object is a member.

As described herein, methods and system are provided for allowing a user to programmatically call the resources identified in an Extensible Markup
15 Language Namespace or schema library for customizing or otherwise modifying the association of resources identified or contained in the Namespace or schema library with one or more associated documents. It will be apparent to those skilled in the art that various modifications or variations may be made in the present invention without departing from the scope or spirit of the invention. Other embodiments of the invention
20 will be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein.

52

WE CLAIM:

1. A programmable object model for accessing the resources of an Extensible Markup Language (XML) schema library, comprising:
an application programming interface for allowing a user to
5 programmatically access resources identified in an XML schema library;
the application programming interface comprising a message call for requesting association of an XML schema file to an XML markup applied to a document; and
the application programming interface operative to receive a return value
10 from the XML schema library responsive to association of the XML schema file to the XML markup applied to the document.

2. The method of Claim 1, whereby the application programming interface further comprises a message call for requesting removal of an association of the XML
15 schema file to the XML markup applied to the document; and
whereby the application programming interface is further operative to receive a return value from the XML schema library responsive to removal of the association of the XML schema file to the XML markup applied to the document. .

3. A programmable object model for accessing the resources of an Extensible Markup Language (XML) schema library, comprising:
20 an application programming interface for allowing a user to programmatically access resources identified in an XML schema library;
the application programming interface comprising a message call for requesting association of an XSLT transformation to an XML markup applied to a
25 document; and
the application programming interface operative to receive a return value from the XML schema library responsive to association of the XSLT transformation to the XML markup applied to the document.

30 4. The method of Claim 3, whereby the application programming interface further comprises a message call for requesting removal of an association of the XSLT transformation to the XML markup applied to the document; and
whereby the application programming interface is further operative to receive a return value from the XML schema library responsive to removal of the association of
35 the XSLT transformation to the XML markup applied to the document.

5. A programmable object model for accessing the resources of an Extensible Markup Language (XML) schema library, comprising:

an application programming interface for allowing a user to programmatically access resources identified in an XML schema library;

the application programming interface comprising a message call for requesting association of an one or more XML-based resources to an XML markup applied to a document; and

the application programming interface operative to receive a return value from the XML schema library responsive to association of the one or more XML-based resources to the XML markup applied to the document.

6. The method of Claim 5, whereby the application programming interface further comprises a message call for requesting removal of an association of the one or more XML-based resources to the XML markup applied to the document; and

whereby the application programming interface is further operative to receive a return value from the XML schema library responsive to removal of the association of the one or more XML-based resources to the XML markup applied to the document.

7. A programmable object model for accessing the resources of an Extensible Markup Language (XML) schema library, comprising:

calling the XML schema library via an object-oriented message call; passing an object property to the XML schema library, the object property being associated with a software object associated with functionality identified in the XML schema library; and

in response to the message call and the object property passed to the XML schema library, receiving access to the functionality identified in the XML schema library associated with the object property passed to the XML schema library.

8. The method of Claim 7, whereby passing an object property to the XML schema library includes passing a method property for creating a new XML Namespace and for adding the new XML Namespace to a collection of XML Namespaces, where a path to a schema file associated with the new XML Namespace and a uniform resource identifier for the new XML Namespace are passed to the XML schema library as parameters of the method object.

9. The method of Claim 7, whereby passing an object property to the XML schema library includes passing a method property for installing solution manifests for registering XML Namespaces in the XML schema library.

10. The method of Claim 7, whereby passing an object property to the XML schema library includes passing a method property for accessing individual XML resources from a collection of XML resources using a numerical index or search key

word, where a numerical index associated with an individual XML resource is passed as a parameter with the method property.

54

11. The method of Claim 7, whereby passing the object property to the XML
5 schema library includes passing an object property for controlling an alias name associated with a specified Namespace identified in the XML schema library.

12. The method of Claim 7, whereby passing the object property to the XML
schema library includes passing a method property for attaching a specified XML
10 schema file of a specified XML Namespace to a specified document, whereby a pointer to the specified document is passed to the XML schema library as a parameter of the method property.

13. The method of Claim 7, whereby passing the object property to the XML
15 schema library includes passing an object property that points to a default XSLT transformation associated with a specified Namespace.

14. The method of Claim 7, whereby passing an object property to the XML
schema library includes passing a method property for removing an XML Namespace
20 object from a collection of Namespace objects.

15. The method of Claim 7, whereby passing an object property to the XML
schema library includes passing a method property for creating a new XSLT
transformation and for adding the new XSLT transformation to a collection of XSLT
25 transformations, where a pointer to the new XSLT transformation is passed to the XML schema library as a parameter to the method property.

16. The method of Claim 7, further comprising a method property for
accessing individual XSLT transformations contained in a collection of XSLT
30 transformations using a numerical index or search key word, where a numerical index representing the position of a requested XSLT transformation in the XML schema library is passed as a parameter to the XML schema library with the method property.

17. The method of Claim 7, whereby passing an object property to the XML
35 schema library includes passing an object property for controlling an alias name associated with an XSLT transformation identified in the XML schema library.

18. The method of Claim 7, whereby passing the object property to the XML
schema library includes passing a method property for removing an XSLT
40 transformation from a collection of XSLT transformations.

hoja n° 20 en blanco

PROGRAMMABLE OBJECT MODEL FOR NAMESPACE OR SCHEMA LIBRARY SUPPORT IN A SOFTWARE APPLICATION

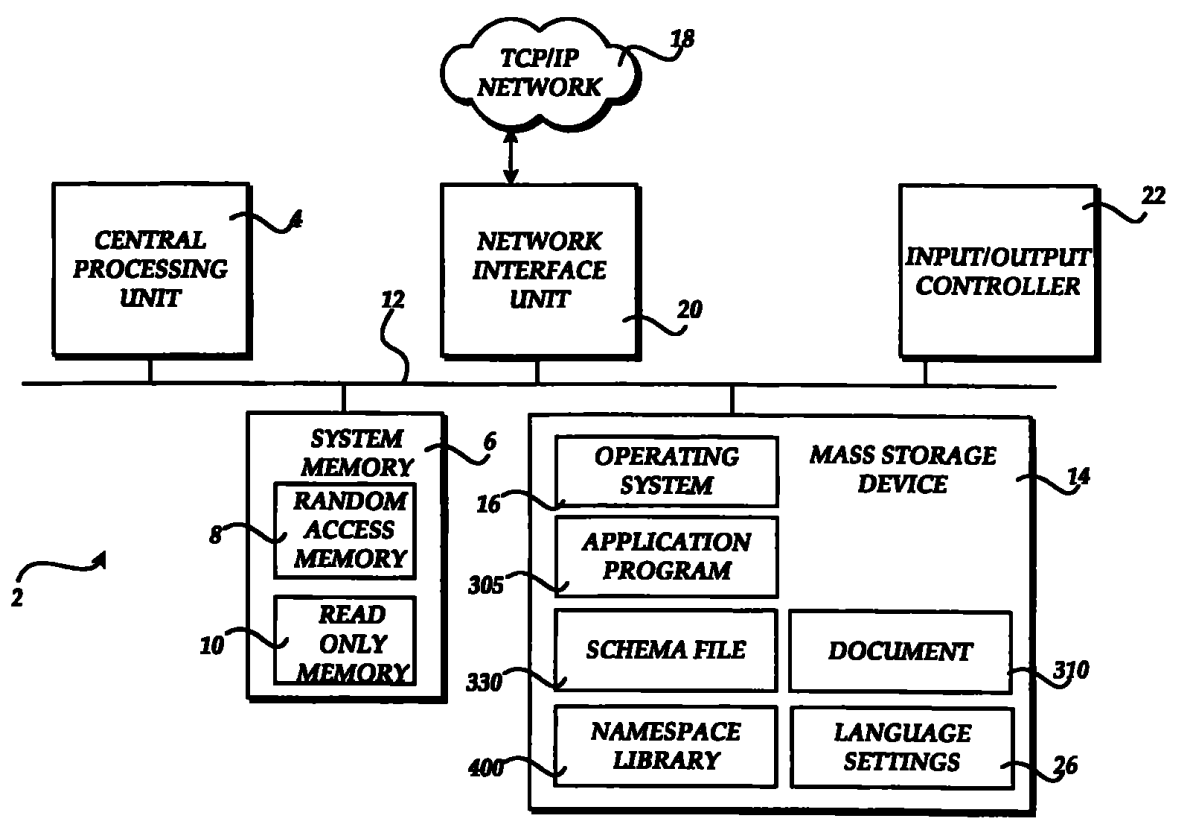
SS

Abstract

5

A programmable object model allows a user to programmatically access and utilize a Namespace or schema library containing XML schema files and related XML-based resources for associating the schema files and XML-based resources with one or more documents and for customizing the functionality associated with the schema files and XML-based resources. The programmable object model also allows the user/programmer to remove schema files and other XML-based resources from association with previously associated documents.

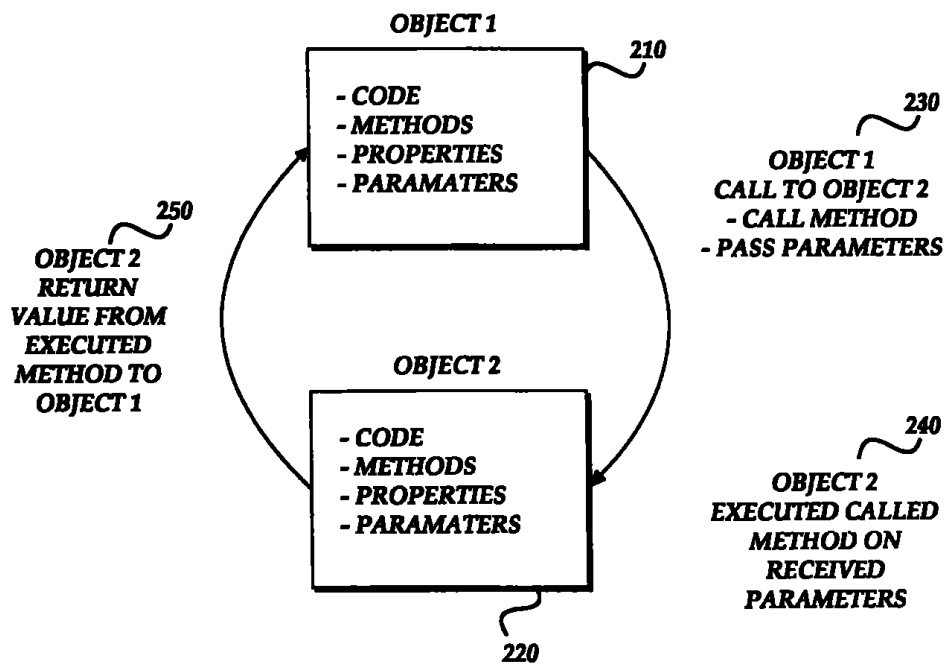
10



1/4

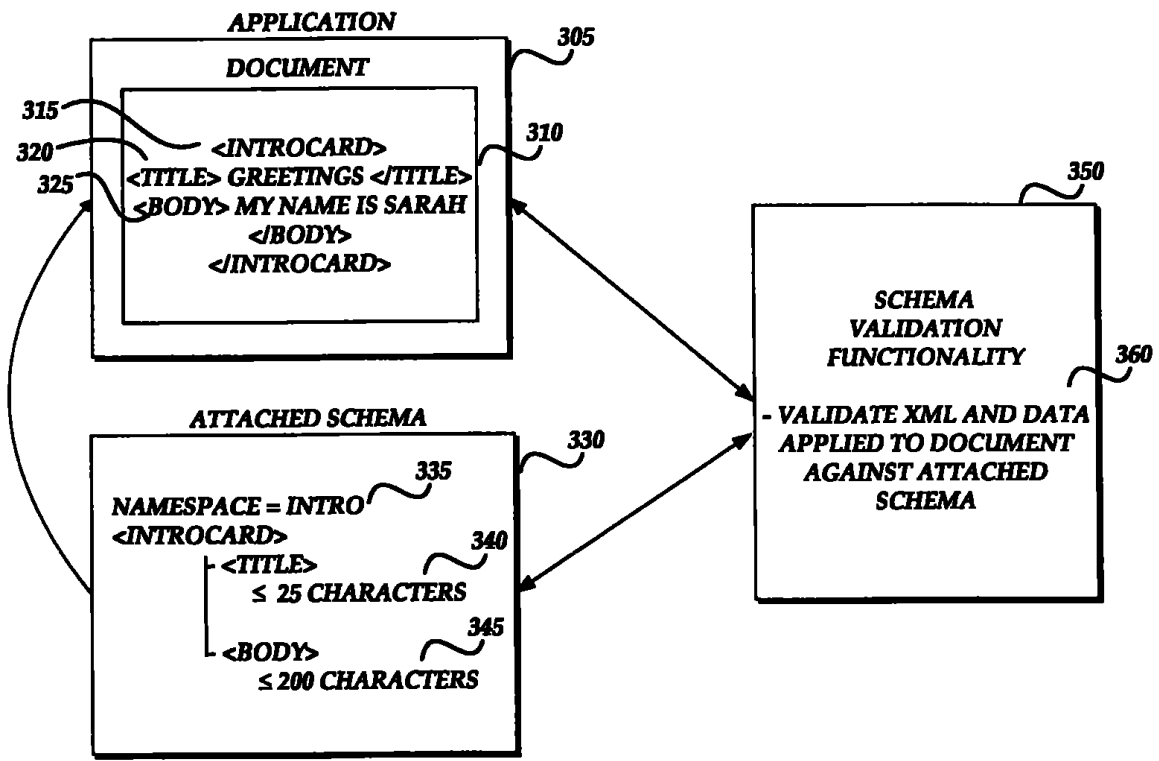
Fig. 1

20



2/4

Fig. 2



3/4

Fig. 3

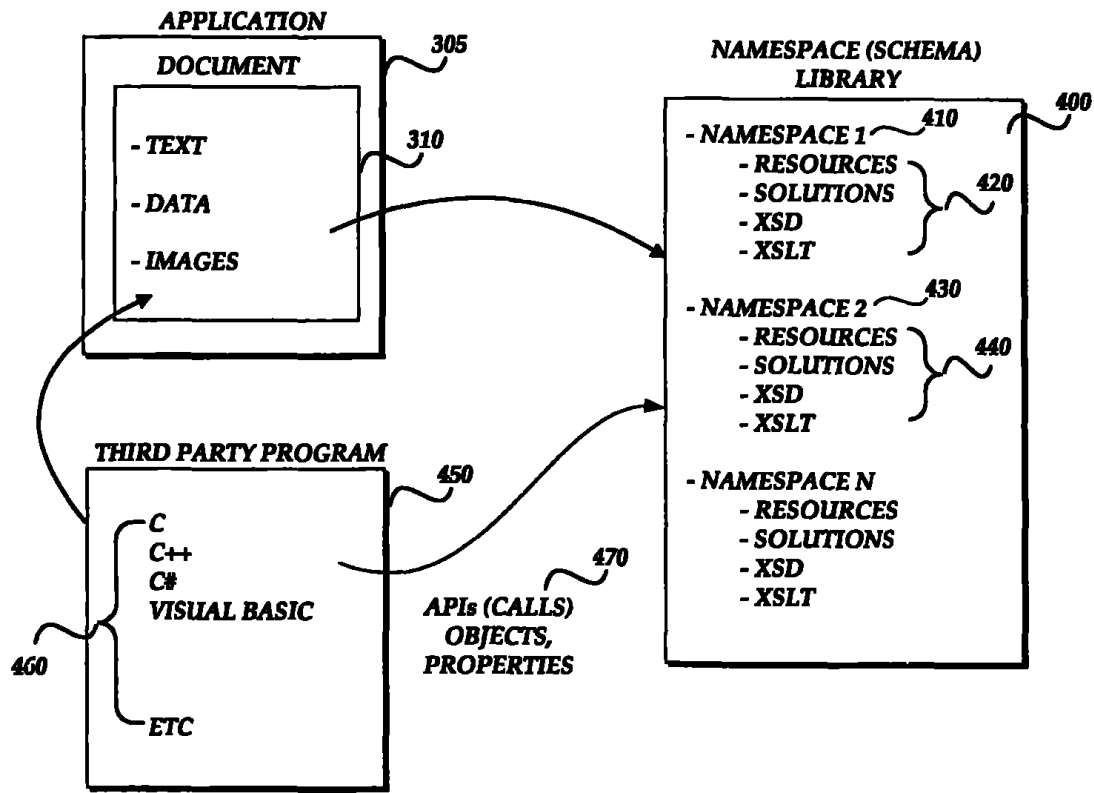


Fig. 4

4/4

93

60 64

SUPERINDUSTRIA Y COMERCIO
Radecacion : 04112067 00000000
Fecha (MM): 2004-11-08 15:52:08
Tramite 002 PATENTES D 1 REGISTRO/ 411 PRESENTA
Dependencia: 2020 DIVISION DE NUEVAS CREACIONES

REQUISITOS NECESARIOS PARA PRESENTACION Y AGILIZACION DEL TRAMITE

	USUARIO	RADIADOR
PATENTE DE INVENCION		
MODELO DE UTILIDAD	()	()
Indicacion de que se solicita la concesion de una patente	()	()
Datos de identificacion del solicitante o de la persona que se presenta la solicitud.	()	()
Descripcion de la invencion	()	()
Dibujos de ser estos pertinentes.	()	()
Comprobante de pago de las tasas establecidas.	()	()
COMPLETA ()		
INCOMPLETA ()		
DISEÑO INDUSTRIAL ()		
Indicacion de que se solicita el registro de Diseño Industrial	()	()
Datos de identificacion del solicitante o de la persona que presenta la solicitud	()	()
Representacion grafica o fotografica del Diseño Industrial o muestra del Material que incorpora el diseno	()	()
Comprobante de pago de las tasas establecidas	()	()
COMPLETA ()		
INCOMPLETA ()		
ESQUEMA DE TRAZADO ()		
Indicacion de que solicita el registro de un Esquema de trazado	()	()
Datos de identificacion del solicitante o de la persona que presenta la solicitud	()	()
Representacion grafica del esquema de trazado	()	()
Comprobante de pago de las tasas establecidas	()	()
COMPLETA ()		
INCOMPLETA ()		

Firma Responsable *Claudia M. Nieto*

Fecha

2020
Bogotá, D. C.

Doctor(a)
LUZ HELENA ADARVE GOMEZ

Oficio No.

REFERENCIA: Radicación 04-112087
 Trámite 002
 Evento 001
 Actuación 430
 Folios 001

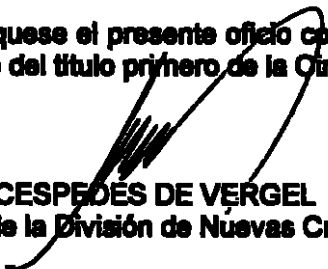
12051

Como resultado del examen de forma, realizado con fundamento en el artículo 38 de la Decisión 486 de la Comisión de la Comunidad Andina, se le comunica que:

- Allegar copia del documento en que conste la cesión del derecho a la patente otorgado por los inventores al solicitante o a su causante. (art. 28-k, Dec 486/2000).
- Debe allegar el poder otorgado al abogado que aparece en la solicitud como apoderado, con el lleno de los requisitos exigidos en el Código de Procedimiento Civil. Cuando se requiera debe allegarse la sustitución del poder, otorgada por quien aparece como apoderado principal. Adicionalmente, deberá presentar el documento que acredita la existencia y representación legal de la sociedad solicitante.

En cumplimiento del artículo 39 de la Decisión 486, se le requiere para que dentro del término de dos (2) meses siguientes a la fecha de notificación del presente oficio, complete los requisitos anteriormente enunciados. En caso de no dar cumplimiento al presente requerimiento, la solicitud se considerará abandonada y perderá su prelación.

Notifíquese el presente oficio conforme a lo dispuesto en el numeral 5.2, literal e), del capítulo quinto del título primero de la Circular Única No.10 de 2001.


ALIX CESPEDÉS DE VERGEL
Jefe de la División de Nuevas Creaciones

El anterior requerimiento fue notificado por fijación en lista de fecha 10 FEB. 2005

Se recuerda al interesado que debe presentar, dentro de los dieciséis meses siguientes contados a partir de la fecha de presentación de la solicitud cuya prioridad se invoca, copia de la misma certificada por la autoridad que la expidió y, en los casos en que haya lugar, la traducción simple del capítulo reivindicatorio. El incumplimiento de éste plazo, acarreará la pérdida de la prioridad invocada.

2020277

Carrera 13 No. 27-00 Pisos 5, 7 y 10
Conmutador: 382 0840
Fax: 350 5220
E-mail: info sic.gov.co
www.sic.gov.co
Bogotá, D.C., Colombia