



Industria y Comercio

SUPERINTENDENCIA

DE

Delegatura de propiedad Industrial

División de Nuevas Creaciones

SOLICITUD PATENTE DE INVENCION

21. EXPEDIENTE No.

03-31595

54. TÍTULO MECANISMOS Y METODOS PARA SUCECOS DE UN
SISTEMA DE ACCESIBILIDAD

51. CLASIFICACIÓN INTERNACIONAL 7 G06F 13/14

71. SOLICITANTE MICROSOFT CORPORATION

DOMICILIO REDMOND, WASHINGTON, ESTADOS UNIDOS
DE AMERICA.

74. APODERADO DARIO CARDENAS ALVARO

22. BOGOTÁ, D. C., 2003 10 14

20 2038

04

PETITORIO

Industria y Comercio
SUPERINTENDENCIA

03 031595 00



Radicación : 03031595
Fecha (ORD) : 2003-04-14 13:15:22
Trámite : 002 PATENTES 3 1 REGISTRO y 1 PRESENTACIÓN
Dependencia : 2021 DIVISIÓN DE MARCAS Y DISEÑOS

FORMULARIO UNICO DE SOLICITUD DE PATENTE 2000-3

1

SOLICITUD DE:

☒

Patente de Invención

☐

Patente de Modelo de Utilidad

2

S LICITANTE
(71)

Nombre: MICROSOFT CORPORATION

Dirección: One Microsoft Way Redmond, Washington 98052

Teléfono: 3123600 Fax: 3122410

E-Mail: general@cardenasycardenas.com

Domicilio: REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA

IDENTIFICACION

C.C. ☐

NIT ☐

C.E. ☐

Otro ☒

Número

3

REPRESENTANTE
O APODERADO
(74)

Nombre: DARIO CARDENAS NAVAS

Dirección: Carrera 7 No 71-52 Torre B Piso 9

Teléfono: 3123600 Fax: 3122410

E-Mail: general@cardenasycardenas.com

IDENTIFICACION

C.C. ☒

NIT ☐

C.E. ☐

Otro ☐

Número 17.088.629 BTA
TP 1.250 C.S.J.

4

INVENTOR(ES)
(72)

Nombre:
1. ROBERT SINCLAIR
2. PATRICIA M. WAGONER
3. BRENDAN MCKEON
Dirección:
1. 24138 NE 5th Place, Sammamish, Washington 98074
2. 23723 NE 81st Street, Redmond, Washington 98053
3. 1705 Summit Avenue, #307, Seattle, Washington 98122
Teléfono: 3123600 Fax: 3123600
E-Mail: general@cardenasycardenas.com

Domicilio:
1. WASHINGTON, ESTADOS UNIDOS DE AMERICA
2. WASHINGTON, ESTADOS UNIDOS DE AMERICA
3. WASHINGTON, ESTADOS UNIDOS DE AMERICA

IDENTIFICACION

C.C. ☐

NIT ☐

C.E. ☐

Otro ☒

Número

5

Título(54):

MECANISMOS Y METODOS PARA SUCESOS DE UN SISTEMA DE ACCESIBILIDAD

6

Clasificación Internacional (51):

G06F 17/60

7

Prioridad

SI ☒

NO ☐

(33) País de Origen

U.S.A.

U.S.A.

(32) Fecha

Septiembre 30, 2002

Febrero 14, 2003

(31) N° de Solicitud

60/415,338

10/368,941

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

6 meses

☐ 12 meses

☐ 18 meses

☐ Otro

☐

9

Comprobante de Pago No.:

03 - 25,410

Fecha Abril 10, 2003

10

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.

X Recibos de Reivindicación de prioridades Nos 03 - 25,411 / 03 - 25,412

11

NOMBRE: DARIO CARDENAS NAVAS

FIRMA:

C.C. 17.086.629 BTA

TP. 1.250
C.SJ.

CARDENAS & CARDENAS

ABOGADOS

001508

Superintendencia de Industria y Comercio
 Radicación: 53.31595 (0000000)
 Fecha Recibida: 2003-04-10 11:15:22
 Trámite: 400 PATENTES D 1 REGISTRO AL PRESENTAR
 Dependencia: DE LA DIVISION DE NUEVAS CREACIONES
 Folios: 121

NIT: 800176089-2

RECIBO OFICIAL DE CAJA : 03 - 25,410
 FECHA : ABRIL 10 DE 2003

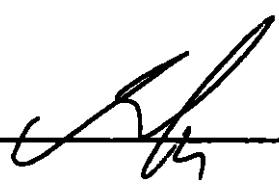
***** CONSIGNACION *****

DEPOSITANTE	TIPO PAGO	BANCO	CUENTA	No. PAGO	FECHA PGO	Vr. PAGO
CARDENAS Y CARDENAS P.I.	CONSIGNACION	BANCO POPULAR	050-00110-6	12173432	10/04/2003	3,367,000.00

***** CONCEPTO *****

CANT.	RENTISTICO	CONCEPTO	TOTAL CONCEPTO
1	50005-01-01	SOLICITUDES	
1		TRANITES DE SOL. DE PATENTE DE IN	400,000.00
		TOTAL :	\$ 400,000.00

SON: CUATROCIENTOS MIL PESOS

RESPONSABLE : 

RECIBO DE CAJA APLICADO AL EXPEDIENTE No. _____

CARDENAS

ABOGADOS

Superintendencia de Registros y Contratos
Radicacion : 03331595 (0000000)
Fecha (N.R.): 2003-04-14 15:10:22
Tramite : 002 PATENTES & 1 REGISTRO
Dependencia: 2da DIVISION DE MARCAS Y DISEÑOS

NIT : 800176089-2

RECIBO OFICIAL DE CAJA : 03 - 25,411
FECHA : ABRIL 10 DE 2003

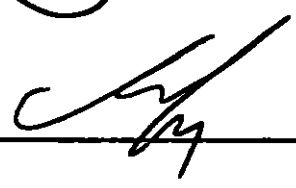
CONSIGNACION

DEPOSITANTE	TIPO PAGO	BANCO	CUENTA	No. PAGO	FECHA PAGO	Vr. PAGO
CARDENAS Y CARDENAS P.I. CONSIGNACION		BANCO POPULAR	050-00110-6	12173632	10/04/2003	3,367,000.00

CONCEPTO

CANT. RENTISTICO	CONCEPTO	TOTAL CONCEPTO
1 30003-01-01 SOLICITUDES	82 INVOCACION DE UNA PRIORIDAD EN NU	112,000.00
	TOTAL :	112,000.00

SON: CIENTO DOCE MIL PESOS

RESPONSABLE : 

RECIBO DE CAJA APLICADO AL EXPEDIENTE No. _____

CARDENAS & CARDENAS

ADOPADAC

Asociación : 0001595 00000000
Fecha (Año): 2003-04-10
Tramite : 000 PATENTES 0 1 REGISTRAR/ 411 PRESEVIAL
Dependencia : 2002 DIVISION DE N.E.H. EJECUCIONES

NIT : 800176089-2

RECIBO OFICIAL DE CAJA : 03 - 25,412
FECHA : ABRIL 10 DE 2003

***** CONSIGNACION *****

DEPOSITANTE	TIPO PAGO	BANCO	CUENTA	No. PAGO	FECHA PGD	Vr. PAGO
CARDENAS Y CARDENAS P.I.	CONSIGNACION	BANCO POPULAR	030-00110-6	12173632	10/04/2003	3,367,000.00

***** CONCEPTO *****

CANT. RENTISTICO	CONCEPTO	TOTAL CONCEPTO
1 50005-01-01 SOLICITUDES	02 INVOCACION DE UNA PRIORIDAD EN NU	112,000.
	TOTAL :	112,000.

SON: CIENTO DOCE MIL PESOS

RESPONSABLE : _____

RECIBO DE CAJA APLICADO AL EXPEDIENTE No. _____

ACCESSIBILITY SYSTEM EVENTS MECHANISM AND METHOD

TECHNICAL FIELD

The invention relates to the field of assistive technology, automated testing, and other products which collect user interface information and the interaction of these products with user interface information.

BACKGROUND OF THE INVENTION

Assistive technology (AT) products exist to help computer users who have a need for assistance in areas of learning, communication and access to information contained in and presented by computer software. These products have a need for information relevant to the computer interface. Similarly, existing automated testing products and user interface commanding utilities also have a need for information about the user interface. Currently, these products have no sufficient source of user interface (UI) information. These three types of products (clients) are required to have necessary support from elsewhere to enable them to: (1) gather information about an application's user interface; (2) programmatically discover and interrogate UI elements regardless of the technology used to build the UI; (3) generate keyboard and pointer input; and (4) understand what type of behavior or functionality is currently available. No single technology is available currently that gives an AT product all of these capabilities. Furthermore, current AT products are not always compatible with all graphical operating system (OS) technologies and lack the ability to filter and coordinate redundant or misleading notifications in a centralized way. An additional disadvantage is that current automation and accessibility infrastructures are not extensible and therefore require OS level changes to add new functionality.

Furthermore, currently to gather information about an application's user interface, the AT product must write application-specific code to obtain information for the user. The process of writing this application-specific code is time consuming and requires continuous maintenance. Current automation infrastructure also lacks the ability to filter and coordinate redundant

or misleading event notifications in a consistent manner. Thus, event consumers are required to independently filter information.

Current systems allow AT products to request event notifications in three levels of granularity: (1) everything on a ^{escritorio} desktop; (2) in a particular process (such as opening of a word processor); or (3) in a ^{ensarte} thread in the particular process (multiple objects doing work in the process). Currently, when the client receives an event, it receives a window handle for a specific window in which the event occurred and other ^{trozos} bits of information to indicate where the event occurred. A client can make a cross process call to retrieve the UI object that is related to the event. With this object, the client can make additional cross process calls to ask for information about that object. If the client needs five pieces of information, then the client must make five cross process calls. Cross process calls are exceedingly slow, so the performance cost of collecting UI information using current accessibility infrastructure is high. This type of known scenario is shown in FIG. 8. A server application 12 fires an event 6. A kernel 14 determines which clients must be notified and sends an event notification 18 to an interested client 10. The client 10 makes a request 16 from the server application 12 across a process boundary 2 for the object related to the event notification 18. The server application 12 returns the object 20 and then the client 10 can begin sending requests 16 for information about the UI control which fired the event. The server application 12 returns the requested information 20 across the process boundary 2 to the client 10.

Another current option allows client code to be loaded as a dynamic link library (.DLL) within a process. This option has several drawbacks. First, it requires cooperation from the system to load client code into a process. Second, it gives rise to security issues because once in the client code is loaded into an application's process, it is difficult to restrict the information it gathers. Third, to be an effective technique for the client, it must be loaded into every process on the system. Optimally, only trusted clients should be loaded into another application's process.

expand
primer peldaño --
y de aquí se ingresa
al sistema
7

Furthermore, a system is needed that gives the client the ability to specify what event notifications it wants to receive. In known systems, a client may need to make a number of cross process calls and then analyze the information to determine if it is interested in the event. A mechanism is needed
5 that can perform this event filtering in a more performant manner and that can be easily updated to support new system or application events. Furthermore, a system is needed that uses only trusted components in order to alleviate security concerns.

Currently, when seeking information about a user interface, the
10 AT product is required to access trees that are native to a particular UI framework. Accordingly, multiple trees are required to convey user interface information for multiple UI frameworks. These differing trees may contain information which is not of interest or is not visible to the user, such as hidden container objects which manage the visible UI controls manipulated by the end
15 user. Therefore, a need exists for a single unified tree having only those nodes that are of interest to the user.

A solution is needed that addresses the needs of AT products, automated testing tools, and commanding utilities. The solution should be usable by all graphical OS technologies and should allow all forms of UI and UI
20 components to become accessible.

BRIEF SUMMARY OF THE INVENTION

The present invention is directed to a method and computer application for providing a client with user interface information. In one aspect of the invention, an event mechanism is provided within a client-server
25 environment for keeping a client notified of changes in the user interface (UI). The event mechanism includes a registration mechanism that allows the client to register to receive interesting UI information. The event mechanism also includes a UI automation client for registering clients to receive event notifications and for conveying those notifications to the client when they occur.
30 The UI automation server additionally receives the notifications from the UI

framework and includes a filtering mechanism for removing events that are uninteresting to the client and a notification mechanism for notifying the UI automation client of the remaining interesting notifications.

5 A further aspect of the invention provides a computer-implemented method for collecting information about interesting UI elements. The method includes receiving a request from the client for specified user interface information and querying the UI engine used by a UI automation server to determine whether the requested UI information is available. The specified UI information is passed from the UI automation server to a UI automation client.

10 Additional advantages and novel features of the invention will be set forth in the description which follows, and in part will become apparent to those skilled in the art upon examination of the following, or may be learned from practice of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

15 The present invention is described in detail below with reference to the attached drawing figures, wherein:

FIG. 1 is a block diagram of a computing system environment suitable for use in implementing the present invention;

20 FIG. 2 is a block diagram of interaction between an accessibility system, a client environment, and a server environment;

FIG. 3 is a block diagram illustrating components of the accessibility system core;

FIGs. 4(A)-4(D) illustrate the creation of a logical tree from native elements;

25 FIG. 5 is a flowchart showing a sequence of procedures for building a logical tree;

FIG. 6 shows a dialog box and its components forming logical elements;

30 FIG. 7 is a flowchart illustrating procedures involved in activating an event mechanism of the invention; and

FIG. 8 shows a known system for event notification.

DETAILED DESCRIPTION OF THE INVENTION

Exemplary Operating Environment

FIG. 1 illustrates an example of a suitable computing system environment 100 on which the invention may be implemented. The computing system environment 100 is only one example of a suitable computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Neither should the computing environment 100 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the exemplary operating environment 100.

The invention may be described in the general context of computer-executable instructions, such as program modules, being executed by a computer. Generally, program modules include routines, programs, objects, components, data structures, etc. 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-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 remote computer storage media including memory storage devices.

With reference to FIG. 1, an exemplary system 100 for implementing the invention includes a general purpose computing device in the form of a computer 110 including a processing unit 120, a system memory 130, and a system bus 121 that couples various system components including the system memory to the processing unit 120.

Computer 110 typically includes a variety of computer readable media. By way of example, and not limitation, computer readable media may comprise computer storage media and communication media. The system memory 130 includes computer storage media in the form of volatile and/or nonvolatile memory such as read only memory (ROM) 131 and random access memory (RAM) 132. A basic input/output system 133 (BIOS), containing the basic routines that help to transfer information between elements within computer 110, such as during start-up, is typically stored in ROM 131. RAM 132 typically contains data and/or program modules that are immediately accessible to and/or presently being operated on by processing unit 120. By way of example, and not limitation, FIG. 1 illustrates operating system 134, application programs 135, other program modules 136, and program data 137.

The computer 110 may also include other removable/nonremovable, volatile/nonvolatile computer storage media. By way of example only, FIG. 1 illustrates a hard disk drive 141 that reads from or writes to nonremovable, nonvolatile magnetic media, a magnetic disk drive 151 that reads from or writes to a removable, nonvolatile magnetic disk 152, and an optical disk drive 155 that reads from or writes to a removable, nonvolatile optical disk 156 such as a CD ROM or other optical media. Other removable/nonremovable, volatile/nonvolatile computer storage media that can be used in the exemplary operating environment include, but are not limited to, magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid state ROM, and the like. The hard disk drive 141 is typically connected to the system bus 121 through a non-removable memory interface such as interface 140, and magnetic disk drive 151 and optical disk drive 155 are typically connected to the system bus 121 by a removable memory interface, such as interface 150.

The drives and their associated computer storage media discussed above and illustrated in FIG. 1, provide storage of computer readable instructions, data structures, program modules and other data for the computer 110. In FIG. 1, for example, hard disk drive 141 is illustrated as storing operating

system 144, application programs 145, other program modules 146, and program data 147. Note that these components can either be the same as or different from operating system 134, application programs 135, other program modules 136, and program data 137. Operating system 144, application programs 145, other program modules 146, and program data 147 are given different numbers here to illustrate that, at a minimum, they are different copies. A user may enter commands and information into the computer 110 through input devices such as a keyboard 162 and pointing device 161, commonly referred to as a mouse, ^{de guisa} ~~trackball~~ ^{bolita} ~~or touch pad.~~ ^{o tableta de graficación tacto-sensible} Other input devices (not shown) may include a ^{palanca de mando, paleta de mando para videojuegos} microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 120 through a user input interface 160 that is coupled to the system bus, but may be connected by other interface and bus structures, such as a parallel port, game port or a universal serial bus (USB). A monitor 191 or other type of display device is also connected to the system bus 121 via an interface, such as a video interface 190. In addition to the monitor, computers may also include other peripheral output devices such as speakers 197 and printer 196, which may be connected through an output peripheral interface 195.

The computer 110 in the present invention may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 180. The remote computer 180 may be a personal computer, and typically includes many or all of the elements described above relative to the computer 110, although only a memory storage device 181 has been illustrated in FIG. 1. The logical connections depicted in FIG. 1 include a local area network (LAN) 171 and a wide area network (WAN) 173, but may also include other networks.

When used in a LAN networking environment, the computer 110 is connected to the LAN 171 through a network interface or adapter 170. When used in a WAN networking environment, the computer 110 typically includes a modem 172 or other means for establishing communications over the WAN 173, such as the Internet. The modem 172, which may be internal or external, may be

connected to the system bus 121 via the user-input interface 160, or other appropriate mechanism. In a networked environment, program modules depicted relative to the computer 110, or portions thereof, may be stored in the remote memory storage device. By way of example, and not limitation, FIG. 1 illustrates
5 remote application programs 185 as residing on memory device 181. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Although many other internal components of the computer 110 are not shown, those of ordinary skill in the art will appreciate that such components
10 and the interconnection are well known. Accordingly, additional details concerning the internal construction of the computer 110 need not be disclosed in connection with the present invention.

Accessibility System Structure

As shown in FIG. 2, an accessibility system 200 interacts with a
15 client environment 300 and a server environment 400. The accessibility system 200 may be implemented in the computer environment 100 described above with respect to FIG. 1. The accessibility system 200 includes a client side accessibility interface 220 for facilitating interaction with the client 300, a server side accessibility interface 230 for facilitating interaction with the server side 400, and
20 an accessibility system core 201. The accessibility system 200 of the invention provides new application program interfaces (APIs), interfaces, and metaphors for programmatically accessing a user interface (UI). The accessibility system 200 allows applications to make themselves and any components they use accessible.

25 The client environment 300 preferably includes an assistive technology (AT) product or automated UI testing tool. The server side 400 may implement a variety of different technologies as shown in FIG. 2. A server system 410 includes an adapter 412 and a core 414, which may be found in a first type of UI. A server system 420 includes a proxies component 422 and controls
30 424 as may be found in a second type of UI, such as a Win32 UI available in

Microsoft Operating System products, from the Microsoft Corporation of Redmond, Washington. The server system 430 includes an adapter 432 and internal OM 434, which may be found in an alternative third type of UI.

As shown in FIG. 3, an event mechanism 210, which is included
5 in the accessibility system 200, relies on a UI automation client 202 and a UI automation server 204 for facilitating interaction with the client environment 300 and the server environment 400. The UI automation client 202 and the UI automation server 204 are described in greater detail below with reference to the events mechanism 210 of the invention. The accessibility system 200 of the
10 invention gives the client (AT Product) 300 the capability to: (1) gather information about an application's user interface; (2) programmatically discover and interrogate UI elements regardless of the technology used to build the UI.; (3) generate keyboard and ^{pointer} pointer input; and (4) understand what type of behavior or functionality is currently available. The accessibility system 200
15 allows applications to make themselves and their components accessible. The structure shown in FIGs. 2 and 3 enables five major aspects of the accessibility system 200 including: (1) logical UI tree; (2) Control Patterns; (3) Event Mechanism; (4) Input API (not covered in this document); and (5) properties.

UI access logical tree 222

20 An integral component of the accessibility system 200 is the logical tree 222, an example of which is shown in FIG. 4(D). The tree 222 is included in the client side accessibility interface 220.

The logical tree 222 is a filtered view of the underlying structural hierarchy of UI elements, not a separate tree that must be implemented by the
25 control or application developer. Instead, it ^{armaniza} keys off a few well-defined properties, interesting and uninteresting, which indicate whether a structural element should be exposed in the logical tree 222. The accessibility system core 201 consumes this information to produce the filtered UI logical tree 222 that is, in turn, presented to the AT products or test script.

The logical tree 222 is a tree of elements, each of which represents a control, an item in a control, or a grouping structure, which may be a dialog, pane, or frame. The structure of the logical tree 222 should represent the UI of the application as perceived by the user (even if the controls are actually implemented using a different underlying structure). The tree should be stable over time. As long as an application looks the same to a user, the logical tree 222 representing that application should remain the same, even if implementation details of the application behind the scenes have changed. Native elements that exist for structural and implementation reasons, such as a ^{esquema} shell's "ShDocView" window in the Microsoft OS products should not appear in this tree, since the user does not perceive them.

The logical tree 222 is a single tree built from a plurality of fragments that is capable of unifying a plurality of different processes so that they are the same to the client. The logical tree 222 enables bulk retrieval and is capable of obtaining a value for a list of properties. By the time a user normally would have invoked a cross process call to ask for values, the accessibility system 200 will have fetched them through the use of the logical tree 222.

Instead of being constructed in one step as in the known systems, the logical tree 222 is constructed from fragments that are used to build a raw tree. As shown in FIG. 5, three main procedures build the logical tree 222. In procedure 72, the accessibility system 200 locates native elements of underlying technologies and arrives at the native trees shown in FIG. 4(A). In procedure 74, the accessibility system 200 combines native elements to form the raw tree 20 as shown in FIG. 4(B). Finally, in procedure 76, the logical tree 222 is obtained by hiding uninteresting components in the raw tree 20 as shown in FIG. 4(D).

FIG. 4(A) illustrates two native trees 10 and 14, which are constructed from native elements of underlying technologies such as the Win32 UI, or any other available UI. The native tree 10 includes a parent node 11 and a plurality of descendants 12 having various relationships with one another. Similarly, the native tree 14 includes a parent node 15 having a plurality of child nodes 16. The child nodes 16 may be described as siblings of one another.

ممكن ان يكون...

As shown in FIG. 4(B), the native trees 10 and 14 may be combined to form a raw tree 20. The raw tree 20 includes a parent node 21, having two child nodes 22 and 30. The child node 22 has descendants 23- 29 and the child node 30 has descendants 31-33. This raw tree 20 is a combination of the native trees 10 and 14, with the nodes of the native tree 10 forming nodes 22-29 and the nodes of the native tree 14 forming nodes 30-33.

Through a method broadly shown in FIGs. 4(C) and 4(D), the raw tree 20 is converted into the logical tree 222. To move from the raw tree 20 to the logical tree 222, a developer can insert hints in the raw tree. The developer can mark nodes within the raw tree 20 as "hide self" or "hide self and children" or as "hide children of node", etc. The developer can also move nodes sideways or place nodes before children. These "hints" and modifications in the raw tree 20 are used to form the logical tree 222. For example, in FIG. 4(C), a developer has marked nodes 24-26 and 33 of the raw tree 20 as uninteresting as indicated by blocks 40 and 41. Typically, nodes that contain elements that will not be visible to the user are marked as uninteresting. Nodes related to the visible UI are typically considered to be interesting and will be included in the logical tree 222 for use by the AT client 300. As shown in FIG. 4(D), the nodes marked as uninteresting are not included in the logical tree 222.

The accessibility system 200 uses the logical tree 222 to find information about events, the state of the system, the location of objects and information about controls. Known systems have not had the capability to scope within their trees. The logical tree 222 can be navigated based on the preferences of the client 300 and is capable of providing information regardless of the server side application in use.

*avoir une
sofene de
action*

In operation, if a client 300 needs to obtain information for a user about an application, the client 300 looks for a button to press and observes text on the button. The client 300 would call an API "find an element". The API will return a value that is referenced to a position in the logical tree 222 of the client side interface 220. Through the logical tree 222, the accessibility system 200 gives the client 300 an abstract view of the UI regardless of the application in

use. The abstract model includes structures, properties, events, and functionality that a list box, button or other UI component can expect to have in common with one another.

5 The logical tree 222 is a single unifying tree that is a logical representation of the UI and is formed into a shape that includes only elements of interest to clients 300. Accordingly, instead of forcing AT products to filter the structural hierarchy of UI elements and guess at the model being presented to the end user, the logical tree 222 presents a hierarchy that closely maps to the structure being presented to the end user. This greatly simplifies the AT
10 product's task of describing the UI to the user and helps the user interact with the application.

Because this logical UI tree 222 is a fundamental part of the accessibility system 200, all of the other components of the accessibility system 200 are designed to work from the logical tree 222. For example, FIG. 6 shows a
15 simple dialog box 60 that appears to have a very simple structure. However, when viewed through the currently available accessibility technology, the structure of this dialog box 60 is surprisingly complex. It contains 264 objects that an AT product must filter through to discover those that are meaningful to the end user. With the accessibility system 200 and its support for the logical UI
20 Tree 222, the developer who owns this dialog box 60 can set a few properties to present the following structure shown in FIG. 6 to the AT products 300.

As shown in FIG. 6, for a "Run" dialog, the developer can indicate as interesting, the flying window graphic 62 and "Type the name of program, folder, document, or internet resource and windows will open it for you" at 63.
25 The developer can also indicate as interesting the combo box 64 including notepad, word, calculator, etc., and the OK 65, cancel 66 and browse 67 buttons. This offers developers a low-cost mechanism to tag their element hierarchy and thereby produce a logical representation of their application's UI through the UI accessibility system 200. Each of the features shown may be represented by a
30 node that has a specified relationship to each other node in the logical tree 222.

This logical representation offers an immediate benefit to a testing team and to AT products or clients 300.

A set of APIs allows the client 300 to get to the tree. Functionality includes: (1) logical element from point to point; (2) logical element from event; and (3) currently focused logical element. As set forth above, a logical element represents a UI component, possibly a control, a part of a control, or a container or logical grouping (i.e. dialog, pane, or frame). Controls can vary greatly in terms of their functionality. Accordingly, different interfaces are used to represent functionality associated with particular types of controls. These control-specific interfaces derive from a common base interface that represents functionality common to all controls. The common base interface contains: (1) methods for navigating the logical tree 222; (2) a general method for getting property values; and (3) methods for accessing supported control-specific interfaces. In navigating the logical tree 222, each underlying application UI technology will provide its own technique for navigation.

Although the logical tree 222 is of ultimate interest to the user, the raw element tree 20 also serves some important functions. While the logical tree 222 contains only elements that the user can relate to, the raw element tree 20 contains nodes, such as 22, that represent the implementation structure of the underlying framework. For a Win32 UI fragment, for example, this tree would contain nodes that represent HWNDs. In some respects, the raw element tree 20 is a 'half-way house' between the logical element tree 222 and the underlying frameworks' own native element trees. The raw element tree 20 is used as a basis for building the logical element tree 222, and it is where native elements first plug into the system.

The raw element tree 20 can also be used for debugging and testing. It is useful for pinpointing or describing where a particular problematic node is. Functionality on a base raw element node includes: methods for navigating the raw element tree; a method for jumping to a corresponding logical element (if one exists); property containing 'debug string' for this element – e.g. "HWND 0x483FE" for HWND nodes; and other 'behind the scenes

infrastructure' methods. These other methods enable hit testing and location; events; and exposing properties that frameworks can easily provide (e.g. focused, enabled).

5 The raw element tree 20 contains nodes 22-33 that represent elements from various rendering engines. The raw element tree is used as a starting point for rendering engines to plug themselves into the accessibility system 200 and is built from lightweight adapter objects which adapt native elements, such as HWNDs from Win32, into a unified logical tree 222. It is additionally used for handling hosting transitions, where one technology hosts another. Since the raw element tree 20 is the base on which the logical tree 222 is built, it can be used to check that the logical tree 222 is complete and connected and can be used to check for unaccounted-for elements. The raw element tree 20 may further be used for other infrastructure-like tasks: such as providing some basic element ID and providing some basic framework-provided 15 element properties, such as focused, enabled, and location.

The raw element tree 20 is not the primary source of information for AT products or clients 300, is not used for logical navigation and is not exposed to end-users. The raw element tree 20 also cannot be used to capture an element's position in tree so that it can be returned to at some future point in 20 time. The logical element tree 222 performs all these functions.

The raw element tree 20 can typically be built mechanically from the raw elements of the underlying rendering technology (HWNDs, Elements) without knowledge of the logical elements represented. It can therefore be used to look for raw elements, which have not been accounted for in the logical tree 222. 25 The raw element tree 20 is a useful debugging and diagnostic tool, as it allows a 'stack dump' / *(liste des enfants de la fenêtre)* like description of a node's location to be captured. Furthermore, known systems base their trees on code specific criteria and are difficult to implement with diverse technologies. The present approach uses a general abstract 'raw element' type, which can be implemented by or on behalf of any 30 underlying rendering technology.

In order to obtain the raw element tree, calling a raw element root will get a desktop element, verified by making sure that its parent is NULL and all other nodes have it as their ultimate ancestor. To obtain other elements, calling a method to obtain a raw element from a specified point will return the element using valid screen coordinates. After obtaining the raw element tree, it can be checked and verified by checking the elements (parents, siblings and children).

In operation, the client 300 can navigate the raw element tree using relationships such as: parent; next sibling, previous sibling, first child, last child, etc. The client 300 can then jump from the raw element to the corresponding logical element in the logical tree 222.

Events Mechanism

When a client 300 wants to keep informed of events, the client 300 is able to register through the UI automation client 202 as shown in FIG. 3 to obtain the information. The client 300 specifies object information it wants to receive, where it wants the information to go, and the list of properties it wants to get back. The client request goes to UI automation client 202. UI automation client 202 can monitor any process on the desktop. The UI automation server 204 keeps track of which clients 300 are listening and knows how to get back to UI automation client 202. The UI automation client 202 advises the UI engine 206 of client interest, so the UI engine 206 knows when to tell the UI automation server 204 of the event. The UI engine does not have to utilize the client advice but may choose instead to always notify the UI automation server 204 of events or notify the UI automation server only if a client is listening for any events. The advice is useful if the UI engine wants to turn on UI automation server notification only if there is a client listening for events. The UI engine would do this to avoid possible degradation of speed of the UI and to avoid loading code modules it doesn't otherwise need.

The UI Engine 206 then informs the UI automation server 204 of a UI event. UI automation server 204 returns the requested logical element to the

client 300 and sends information to the client 300 including properties of the event that the client 300 requested. The UI automation server 204 decides what information is within the scope of client request and only forms a logical element if the information is of interest. Forming a logical element includes pre-fetching,
5 on the UI automation server side, the set of properties that the client has indicated it will use when handling the event. For example, the UI automation server 204 can discover a logical element for a combo box. The scope would be the combo box or its children. The client 300 can request children/parent/dependents to define scope during the registration phase.

10 After the UI automation server 204 determines whether information is within the requested scope, it builds a logical element. The UI automation client 202 serves the client 300 by talking to target applications receiving the requested information from the UI automation server 204 and routing objects to a proper space on the client 300.

15 The UI automation server 204 is created when the client 300 registers to receive event notification. As an example, a UI engine 206 is running the Microsoft Word word processing application. The client 300 has registered for name property change. The client's registration causes the UI automation server 204 to be created. The client's registration also advises the UI engine 206
20 to start notifying the UI automation server 204 for the name property. The UI engine 206 doesn't get scope information. The UI engine 206 calls one of the APIs for the server side. The UI engine 206 specifies (1) what property changed; (2) the new value of the property; and (3) perhaps the old value. The UI automation server 204 is created based on events of interest to the client 300 and
25 therefore knows events, properties, clients, and scope of interest so it knows if any client 300 is interested in the created logical element. If more than one client 300 has registered for events with a particular UI automation server 204 and the clients 300 have registered for the same event and have asked for properties to be bulk fetched with the returned logical element, when the UI automation server
30 204 sends an event back to the clients 300, each client 300 will get the union of the requested bulk fetch properties returned with the logical element.

For each client 300 listening, the UI automation server 204 notifies the client 300 passing the client the logical element associated with the event. The UI automation server 204 creates only one logical element. This is an improvement over the current technology in which each client 300 would be
5 required to ask for their own copy of the object that is the source of the event.

If the UI engine 206 does not utilize the UI automation client's advice when clients register for events, the UI engine 206 can ask the UI automation server 204 if there are any accessibility clients 300 listening and if no one is listening, then can avoid the work of creating the information and sending
10 it to the UI automation server 206. For example, a screen reader is the client 300 and specifies where it wants information to go, the focus change object to receive events, and the specific list of properties of interest. The UI engine 206 is advised and knows it should send events to the UI automation server 204. Upon detecting focus changes, the UI engine 206 notifies the UI automation server 204.
15 The UI automation server 204 converts to a well-known interface and sends the event and object to the UI automation client 202. The UI automation client 202 routes the object to an appropriate space on the client 300.

The above-described components improve upon the known systems by eliminating the central repository in the kernel for events. Instead, the
20 UI automation server 204 knows all clients 300 interested in getting information about the context in which it is running. The elimination of the kernel repository also creates a more peer-to-peer interaction, since the UI automation server 204 fulfills the function previously performed in the kernel. The accessibility system 200 of the invention gives client 300 the ability to specify what it wants to see
25 such that filtering is accomplished on the server side using the UI automation server 204.

FIG. 7 is a flow chart showing the procedures involved in the event registration and notification method. In step 80, the client 300 requests event notification. In step 82, the UI automation client 202 communicates the
30 request to the UI automation server 204. In step 84, the UI automation client advises the UI engine 206 that it wants notification. In step 86, the UI

automation server 204 receives notification from the UI engine 206. In step 88, the UI automation server 204 filters the received information. If the received information is found uninteresting to the user in step 90, the UI automation server 204 discards the information and continues to wait for notification in step 92.

- 5 Alternatively, if the information is found to be interesting in step 90, the UI automation server 204 creates a logical element and sends it to the UI automation client 202 in step 94. In step 96, the UI automation client 202 puts the received information in its proper place on the client 300.

- 10 The event mechanism 210 of the accessibility system 200 allows the client 300 to register to receive event notifications for property changes in the UI, tree changes in a control's structure, multimedia events, and related information. Without these capabilities, clients 300 have to continually poll all the UI elements in the system to see if any information, structure, or state has changed. The accessibility system 200 events mechanism 210 also allows clients
15 300 to receive events out-of-process, request a collection of properties to be returned with the event notification, and to register for events on multiple elements.

- The event mechanism 210 exposes: the interfaces the AT product or test application uses to register for events; interfaces the AT product
20 implements on objects used to receive event notifications; and the interfaces control implementers use to notify the event engine of UI events. The event mechanism 210 is used to allow AT products and test applications to receive events independently of the UI engines used to render UI and allows AT products and test applications to track top-level application windows and focus without
25 regard to the underlying technology.

- FIG. 2 shows how clients 300 interact with the accessibility system 200. There is no global management of events across processes. Instead, each accessibility system client 300 or server 400 has its own context. Client applications will only receive events for applications that support the accessibility
30 system 200. To start using accessibility system events in an application, the client 300 can do one of four things: (1) Use the "AddTopLevelWindowListener"

API to discover new UI appearing on the desktop and in the handler for that event, register for other events and by this means receive events from any process; (2) Use one of the Find APIs to locate interesting UI and target a specific piece of UI; (3) Use some other means to discover interesting UI such as finding a window handle or a point on the screen and, using that handle or point, acquire a logical element to use as a reference for listening to events; or (4) Use the "AddFocusChangeListener" API to track the input focus and register for events on the UI that currently has focus.

A top-level window is a window whose ^{generator}parent is the desktop. The use of the term "opened" indicates that a new top-level window has appeared to the user. The use of the term "closed" indicates that a top-level window has been destroyed.

The accessibility system server side interface 230 includes two APIs for notifying the accessibility system 200 of events being added and removed. The UI automation client calls these APIs when clients register and unregister for events. This allows the UI engine 206 to get information about what accessibility events are being requested in its context.

Where applicable, events are associated with logical elements from the application's logical element tree 222. Where logical elements are not applicable, events are associated with a human readable string or other well-known object that represents the source of an event. The event mechanism 210 provides filtering based on user supplied preferences during event registration. By using the client's filtering preferences at the server, before creating the event-related data and sending it cross process to the client, the event mechanism 210 inherently improves out-of-process performance by reducing the number of cross process calls. The event mechanism 210 provides a way to specify the properties of a logical element that are interesting for the event during event registration. This further reduces the number of cross-process calls. The event mechanism 210 is extensible without requiring major operating system (OS) changes. Although the event mechanism 210 may be implemented using managed code, unmanaged applications can access it through COM interoperability.

There are two tasks the AT client 300 performs to handle events: (1) event registration; and (2) handling the event in a callback. An important requirement for either task is to be usable in both managed and unmanaged code. In one embodiment of the invention, the client 300 may pass interfaces when registering for events. The client 300 passes objects that implement well-known interfaces when registering and the UI engine 206 calls back on those interfaces to notify the listener. In the cases where there is a source object in the logical element tree 222, each of the callbacks receives the source element and an additional parameter.

Well-known interfaces implemented by the AT product are used as the way to return events. The following sections show some types of events, what registration looks like and how the event is received.

1. Top-Level Window Events

Top level window events include events related to menus and combo box dropdowns or any feature having the desktop as a parent. In an embodiment of the invention, an `AddTopLevelWindowListener` method is used to receive notification of top level windows being opened and closed. Calling `AddTopLevelWindowListener` will get notifications for new top-level windows opening or top-level windows being closed on the current desktop. A `RemoveTopLevelWindowListener` method provides a mechanism to stop receiving notifications for top-level windows being opened, or closed, on the desktop. This method uses a callback object to identify this listener. Therefore, the object passed in to the `RemoveTopLevelWindowListener` method is the same as that passed to `AddTopLevelWindowListener`. An `OnTopLevelWindowOpened` method is called by the accessibility system 200 once for each new, top-level window that is opened. Similarly, an `OnTopLevelWindowClosed` method may be called by the accessibility system 200 once when a top-level window has closed. To receive these notifications, the client 300 calls the `AddTopLevelWindowListener` method.

2. Focus Events

Clients 300 often require a method for tracking focus. Doing this in the Microsoft Windows OS is difficult. For instance, when a menu is dropped down (e.g. the File menu in Microsoft Word) the items in the menu get focus as the user moves the cursor down each one. When the menu closes (e.g. the user presses the ESC key) today there is no focus event sent. Instead the client interested in focus changes must listen to a number of events and figure out which of those logically represents a focus change. In an embodiment of the invention, an `AddFocusChangeListener` method may be used to notify the listener of focus change events. The client 300 may specify a set of properties to be returned. This feature of specifying properties to return with the logical element is part of all event registration APIs. The client 300 may call a `RemoveFocusChangeListener` method to stop receiving notifications for focus changes. This method may use the callback object to identify this listener and in this case the objects passed in to the `RemoveFocusChangeListener` method will be the same as that passed to the `AddFocusChangeListener` procedure. The accessibility system 200 calls an `OnFocusChanged` method when focus has changed. A `FocusEventArgs` parameter used in the `OnFocusChanged` method exposes information related to the focus change. If the information about the last focused logical element is available, then that element will be available in the event args parameter used in the `OnFocusChanged` element. There are cases when no UI element has focus until something explicitly puts focus on an element.

3. Property Change Events

Property change events are fired when properties of logical elements change. In an embodiment of the invention, a client 300 calls an `AddPropertyChangedListener` method to receive notifications for property changes. An `OnPropertyChanged` method is called by the accessibility system 200 when the value of a property has changed on a logical element within the logical element tree specified in `AddPropertyChangedListener`. A scope

parameter indicates for what elements an event should be fired. For instance, passing the root logical element of a window and a request for descendants limits property change callbacks to that window. If the scope parameter is set to all elements in the tree, then the scope is ignored and any changes to the specified properties that occur on the desktop are sent. A client 300 may call `AddPropertyChangedListener` multiple times with different sets of properties and/or a different callback object. The notification provided by the accessibility system 200 indicates: the property that changed; the new property value; and the old property value if available .

10 A client 300 may call a `RemovePropertyChangedListener` method to stop receiving notifications for property changes. This method may use the scope element and callback object to identify this listener and in this case the objects passed in must be the same as that passed to `AddPropertyChangedListener`.

15 **4. Events from Controls Patterns**

The events fired from controls patterns need to be extensible and therefore these events are identified by a GUID. Any GUID value is accepted when registering. For any new control pattern, the events it documents need to be unique GUIDs. AT products may need to be modified to listen for new control pattern events. The listener also needs to be able to scope these events. For instance, directed testing may want to limit these events to a particular application or control within an application. The controls patterns define what the source is and event consumers will need to refer to that part of the documentation in order to know how to use a source element and event argument object.

25 An `AddEventListener` method will enable the client 300 to receive events for controls. The scope parameter can be used to indicate for what elements to fire events. For instance, passing the root logical element of a window and requesting descendants will limit events to that window. If all elements in the tree are desired, then all events on the desktop will be sent. A
30 `RemoveEventListener` method may be used to stop receiving an event for

controls. This method may use the scope element, callback object and an event identifier to identify the listener. In this case, the objects passed in must be the same as that passed to AddEventListener.

When an application has invoked the AddEventListener method, an OnEvent method is called by the accessibility system 200 when a control-specific event is fired and the source of the event is a logical element within the logical element tree specified in AddEventListener. When events for controls are fired there is often event-specific information available.

A RemoveAllListeners method may be called to stop receiving any events. This is a quick way to clean up before a client application shuts down. The removal method may optimally be used when terminating an application.

5. Logical Structure Change Events

Logical structure change events are fired when the logical element tree structure changes. An AddLogicalStructureChangeListener method may be implemented to receive notifications for structural changes in the logical element tree. When logical elements are added, removed or invalidated, methods on the specified callback object are called. The scope parameter can limit the elements as set forth above. A RemoveLogicalStructureChangeListener method may be called to stop receiving events for logical element tree changes. This method may use a callback object and a scope element to identify this listener and in this case the objects passed in must be the same as those passed to AddEventListener.

An OnChildAdded method may be called by the accessibility system 200 when a child element is added and the parent is a logical element within the logical element tree 222 specified in AddLogicalStructureChangeListener. An OnChildRemoved method is called by the accessibility system 200 when a child element is removed and the old parent is a logical element within the logical element tree specified in AddLogicalStructureChangeListener.

An OnChildrenBulkAdded method is called by the accessibility system when a large number of children are added (e.g. greater than 20 children) within a short period of time. An OnChildrenBulkRemoved method is called by the accessibility system when a large number of children are removed within a short period of time. An OnChildrenInvalidated method is called by the accessibility system 200 when a large number of children (e.g. greater than 20 children) are both added and removed within a short period of time.

6. Multimedia Events

Another type of event is the multimedia event. Multimedia may include sound, video, and animation. The methods will support multimedia events and notify a client of actions including "stopped", "paused", "fastforwarded", "rewound", and "muted". Methods similar to those described above may be implemented for adding and removing multimedia listeners.

7. Simple Sound Events

Simple sound events may be handled separately from multimedia events. Simple sound events represent simple, short-duration, sounds that exist to convey to the user some event other than the sound itself. Simple sound events may include: the sound played when new mail has arrived; the sound generated when the battery on a laptop is low; or the sound played when a message box is displayed with the IconExclamation type. An AddSoundListener method can be called to receive notification of simple sounds being played and a RemoveSoundListener method may be implemented to stop receiving notifications for simple sound events.

An OnSound method is called by the accessibility system 200 when a simple sound has played. To receive this notification, the listening application calls AddSoundListener. The OnSound method retrieves the following information: the name of the sound; the source of the sound; and an alert level value indicating the importance of the sound to the user. Possible alert levels include: 'unknown', indicating that the importance is unknown; 'informational', indicating that information has been presented; 'warning',

indicating a warning condition; 'question', indicating that a user response is required; 'exclamation', indicating that the event is non-critical but may be important; and 'critical', indicating the occurrence of a critical event.

8. Soft Focus Events

5 Soft focus events appear on the desktop but stay in the background. Some examples of soft focus events are: a balloon help window indicating "New updates are available" in the notification area; a flashing icon in the task bar for a background application that wants to gain focus; and a printer icon appearing and disappearing from the notification tray when printing starts
10 and ends. These events may seem to overlap somewhat with other event categories (multimedia may involve animation events as does ^{ditto} soft focus) However, the event will be categorized based on what it conveys to the user rather than how it is conveyed.

 An AddSoftFocusListener method may be implemented to receive
15 notification of events that try to get the user's attention without taking the input focus. A RemoveSoftFocusListener method stops the notification. This method may use the callback object to identify this listener and therefore the object passed in should be the same as that passed to AddSoftFocusListener.

 An OnSoftFocus method may be called by the accessibility system
20 200 when a soft focus event has occurred. To receive this notification, the listening application or client 300 calls AddSoftFocusListener. A source element, if available, can be used to get more information about the event. An example source element would be the logical root element of a ^{circulo de identificacion} balloon help window used by one of the notification applications in the system tray. The
25 OnSoftFocus method returns: name of the event; source of the event; and the alert level indicating importance to the user.

 The following chart illustrates the actions of the client 300 and the accessibility system 200 when the client 300 uses an AddTopLevelWindowListener API to listen to events from a specific process.

Client	Accessibility system and the target UI window
Calls AddTopLevelWindowListener	- Accessibility system client starts a thread to watch for top-level application windows being created and destroyed. - New UI appears and the Accessibility system client calls the client's OnTopLevelWindowOpened method.
From the OnTopLevelWindowOpened method calls other APIs to register for additional events happening in the target UI window.	- Accessibility system client communicates the event Id's to the target UI window so it can be selective in notifying of events. - Accessibility system client communicates the event Id's and filtering information to the Accessibility system server so it can further filter events. - Target UI window uses the accessibility system server API to notify the accessibility system server side of the events of interest.
Handles events in the callback objects	- Accessibility system server communicates events back to the accessibility system client. - Accessibility system client calls back on the objects the client application supplied with registration.
Calls RemoveListener APIs to stop receiving events	- Accessibility system client communicates to the accessibility system server and to the target UI window the events that are no longer of interest. - The target app stops notifying the accessibility system server of events.

Event Notification

Corresponding event notification methods are used by the server 400 or underlying UI engine to support the accessibility system events enumerated above. The UI automation server APIs include methods that the server or underlying UI engine may call to accomplish this. For instance, there is a NotifyPropertyChanged method for the server to call to notify when a particular property changes on a logical element. It is up to the server 400 or underlying UI

engine to generate the appropriate parameters and call these notification methods when the UI changes.

Server Methods

5 An AdviseEventAdded method and an AdviseEventRemoved method are called by the UI automation client to notify the server 400 when clients 300 are requesting events. This allows the server 400 to not propagate events to the accessibility system 200 when there is no one interested. Servers can use these notifications to make performance dependent on whether there are clients using the events.

10 **Control Patterns**

 The accessibility model offers a unique approach to categorizing and exposing the functionality supported by a particular UI element or control. Instead of associating functionality with a specific control type, (for example, a button, an edit box, or a list box) as in the prior art, the accessibility model
15 defines a set of common control patterns that each define one aspect of UI behavior. Because these patterns are independent of each other, they can be combined to describe the full set of functionality supported by a particular UI element.

 For example, instead of describing an element in terms of its class
20 name, such as Button, the accessibility system 200 describes it as supporting the invokable control pattern. A control pattern defines the structure, properties, events, and methods supported by an element. Therefore, these patterns not only allow the client to query a control's behavior, they also allow it to programmatically manipulate the control by using interfaces designed for a
25 particular pattern. For example, a SelectionContainer pattern provides methods to query for the selected items, to select or deselect a specific item, or to determine if the control supports single or multiple selection modes.

 The control patterns currently defined for the accessibility system 300 include: (1) Selection Container; (2) Hierarchy; (3) Invokable; (4) Simple

Grid; (5)Text; (6) Value; (7) Represents Object; (8) Scrollable; (9) Sortable; (10) Drawing; and (11) Other Container.

This technique enables control developers to implement a new type of control while still having a well-defined approach for exposing its behavior to AT products and test scripts. If a new type of behavior is introduced, a new control pattern can be defined to express the required functionality.

Assistive technology products and test scripts can now be written to understand how to work with each pattern, instead of each UI control. Because there are far fewer control patterns than control classes, this techniques minimizes necessary code. This approach also encourages a more flexible architecture that can effectively interrogate and manipulate new controls (as long as they support known control patterns).

The following table provides some examples of common controls and the patterns they will support.

Control	Relevant Control Patterns
Button	Invokable
Checkbox, Radiobutton	Value
Listbox	SelectionContainer, Scrollable
Combobox	SelectionContainer, Scrollable, Value
Treeview	SelectionContainer, Scrollable, Hierarchy
Listview	SelectionContainer, Scrollable, Sortable
Textbox, Edit	Value, Text, Scrollable

More specific interfaces will be used to expose functionality associated with common control patterns. Examples of these patterns include: (1) selection managing containers; (2) grid layout containers; (3) UI elements that contain values; (4) Icons that represent objects (files, emails, etc); and (5) UI elements that may be invoked. In general, these patterns are not tightly bound to specific controls and different controls may implement the same patterns. For example, listboxes, combo boxes, and treeviews all implement the 'selection managing container' pattern. Some controls may implement multiple patterns if

appropriate: a selection grid would implement both the 'Grid layout container' pattern and the 'Selection managing container' pattern.

- There is no single 'role' property as in previous applications. Instead, two separate mechanisms are used. Control patterns determine the available functionality of a control and a human-readable localizable property provides a control-type name that the user can understand, such as 'button', 'list box', etc.

Properties

- The accessibility system 200 will feature a general GetProperty method. Properties are preferably represented by GUIDs, with utility methods used to translate to and from a non-localizable ^{mnemonic} mnemonic form (useful for scripting and config files) and also to obtain localized descriptions. The two key advantages of a general GetProperty method instead of individual methods are that (a) it allows for new properties to be added over time without changing the interface, and (b) it allows for implementation techniques – such as ^{series} array-driven bulk property fetching – that are not possible when using separate methods. Each property must have a clearly defined intent. Whether the property is intended for human or machine consumption, whether the property is to be localized, etc., must be clearly defined.

- The present invention has been described in relation to particular embodiments, which are intended in all respects to be illustrative rather than restrictive. Alternative embodiments will become apparent to those skilled in the art to which the present invention pertains without departing from its scope.

- From the foregoing, it will be seen that this invention is one well adapted to attain all the ends and objects set forth above, together with other advantages which are obvious and inherent to the system and method. It will be understood that certain features and sub-combinations are of utility and may be employed without reference to other features and sub-combinations. This is contemplated and with the scope of the claims.

CLAIMS

What is claimed is:

1. An event mechanism within a client-server environment for keeping a client notified of changes in the user interface, the event mechanism
5 comprising: a registration mechanism for allowing the client to register to receive interesting user interface information; a user interface automation client for receiving registration information from the client and for conveying the interesting user interface information to the client upon receipt, wherein the user
10 interface automation client uses the registration information to advise a user interface server; and a user interface automation server for receiving registration information from the user interface automation client and for receiving the user interface information from a user interface engine, the user interface automation server including a filtering device for filtering out information that is
15 uninteresting to the client and a notification device for notifying the user interface automation client of information that is interesting to the client.
2. The event mechanism of claim 1, wherein the registration mechanism accepts registration information from the client, the registration information including: object information; a client location for the object information; and a list of properties to be returned with a source object.
- 20 3. The event mechanism of claim 2, wherein the UI automation client comprises means for delivering the object information to the specified client location.
4. The event mechanism of claim 2, wherein the filtering means of the UI automation server determines whether the user interface
25 information from the user interface engine includes the object information and the list of required properties.
5. The event mechanism of claim 1, wherein the registration mechanism receives a request for a specified scope of information.

6. The event mechanism of claim 5, wherein the user interface automation server forms a logical element if the user interface information is interesting to the client and is within the specified scope and discards the user interface information if the user interface information is
5 uninteresting or is not within the specified scope.

7. The event mechanism of claim 1, wherein the user interface automation server notification from the user interface engine including: a name of a changed property; a new value; and any available old value.

8. The event mechanism of claim 1, wherein the user
10 interface information comprises information pertaining to at least one event type selected from the group of: top-level window events; focus events; property change event; control patterns events; logical structure change events; multimedia events; simple sound events; and soft focus events.

9. The event mechanism of claim 8, wherein each event falls
15 into only one of the event types.

10. The event mechanism of claim 8, wherein each event type includes a set of methods for adding a listener, removing the listener, ^{as part of} prompting the event mechanism to listen for the event type, and for the user interface engine to notify the UI automation server of its events.

11. A computer-implemented method for notifying a client of
20 interesting user interface events, the method comprising: receiving a registration request from the client for specified user interface information; monitoring user interface engine information from a user interface automation server to determine whether the specified user interface information is available; passing the specified
25 user interface information from the user interface automation server to a user interface automation client; and delivering the specified user interface information from the user interface automation client to an appropriate client location.

30
37

12. The method of claim 11, wherein receiving a registration request comprises receiving registration information including: object information; a client location for the object information; and a list of required properties.

5 13. The method of claim 12, wherein delivering the specified user information comprises delivering the object information to the specified client location.

10 14. The method of claim 12, further comprising filtering user interface information with the UI automation server to determine whether the user interface information from the user interface engine includes the object information and the list of required properties.

15 15. The method of claim 11, wherein receiving a registration request comprises receiving a request for a specified scope of information.

16. The method of claim 15, further comprising forming a logical element with the user interface automation server if the user interface information is interesting to the client and is within the specified scope and discarding the user interface information if the user interface information is uninteresting or is not within the specified scope.

17. The method of claim 11, further comprising receiving information from the user interface engine including: a name of a changed property; a new value; and any available old value.

18. The method of claim 11, further comprising receiving user interface information pertaining to at least one event type selected from the group of: top-level window events; focus events; property change event; control patterns events; logical structure change events; multimedia events; simple sound events; and soft focus events.

-33-

19. The event mechanism of claim 8, wherein each event falls into only one of the event types.

ABSTRACT OF THE DISCLOSURE

A method and system providing a client with user interface information are described. An event mechanism within a client-server environment for keeping a client notified of user interface events. The event
5 mechanism comprises a registration mechanism for allowing the client to register to receive interesting user interface information. A user interface automation client receives registration information from the client and conveys the interesting user interface information to the client upon receipt. A user interface automation server receives registration information from the user interface
10 automation client and advises a user interface engine of the registration. The user interface automation server additionally receives the user interface information from the user interface engine. The user interface automation server includes a filtering device for filtering out information that is uninteresting to the client and a notification device for notifying the user interface automation client of
15 information that is interesting to the client.

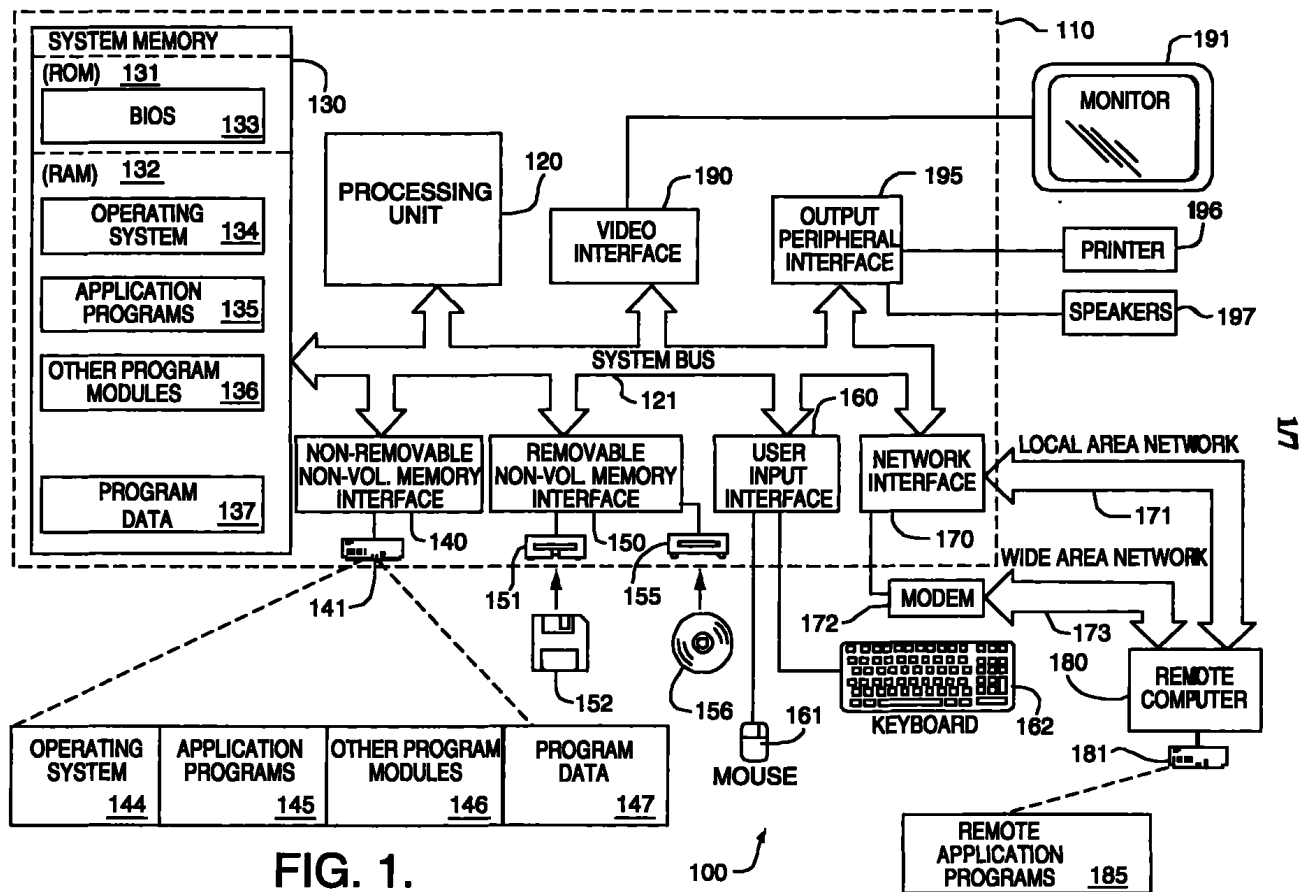


FIG. 1.

2/2

FIG. 2

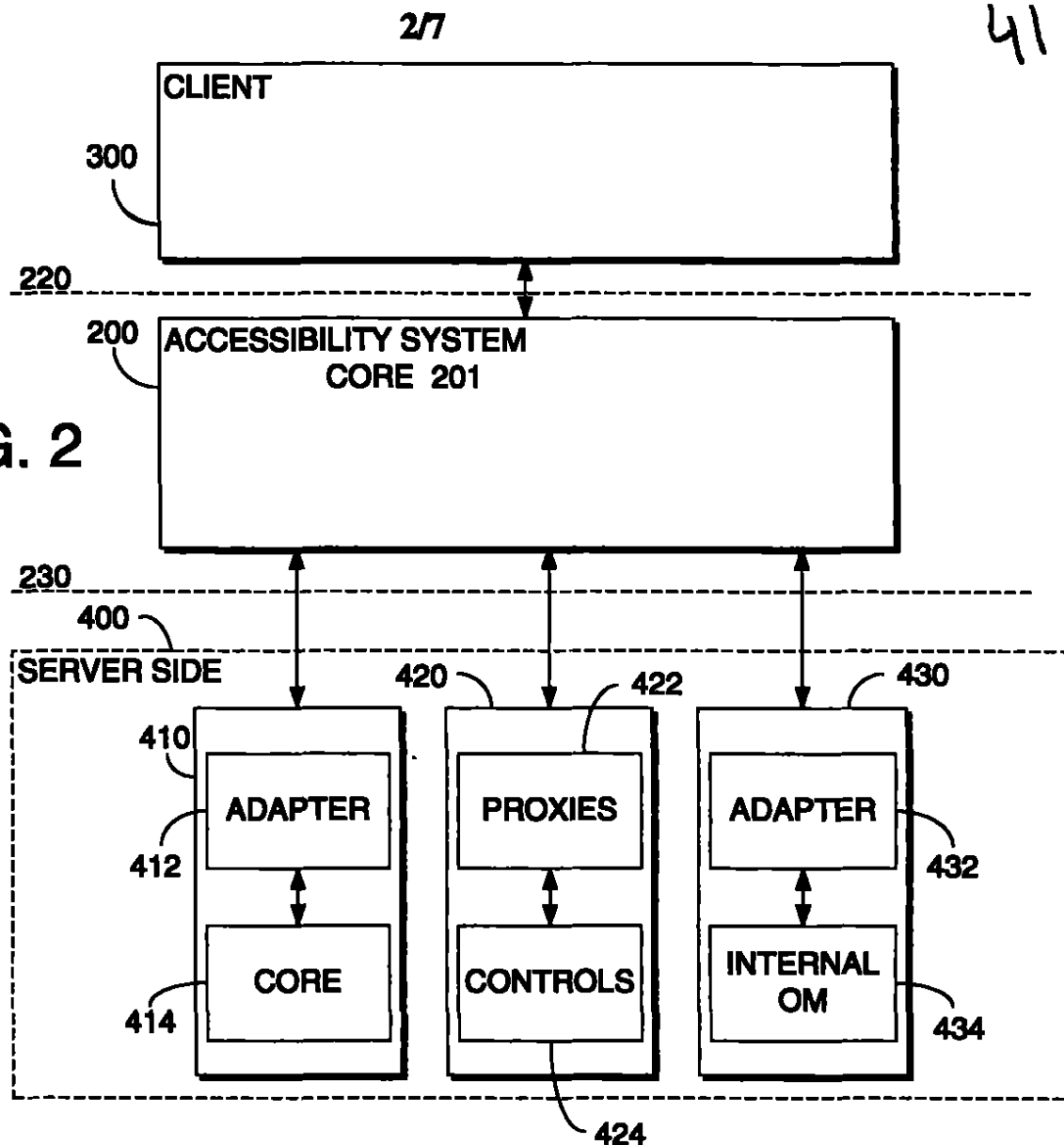
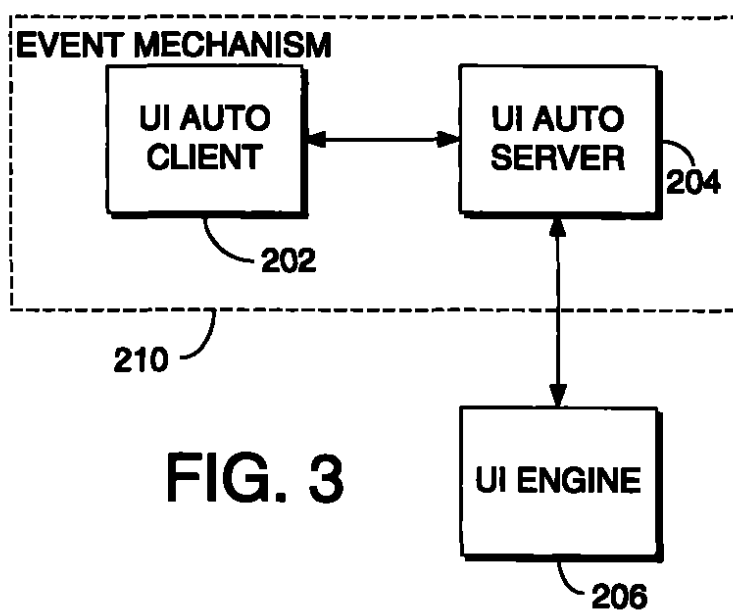


FIG. 3



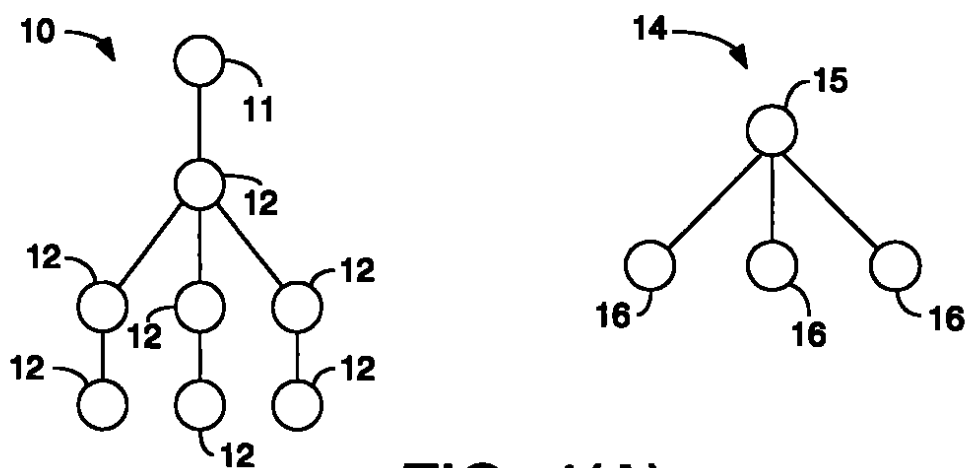


FIG. 4(A)

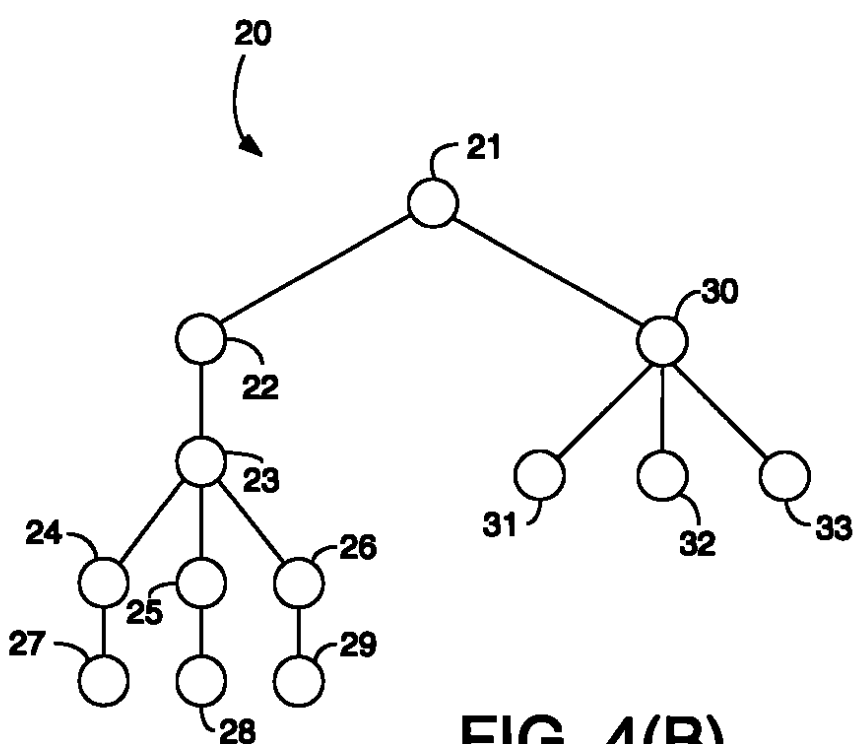
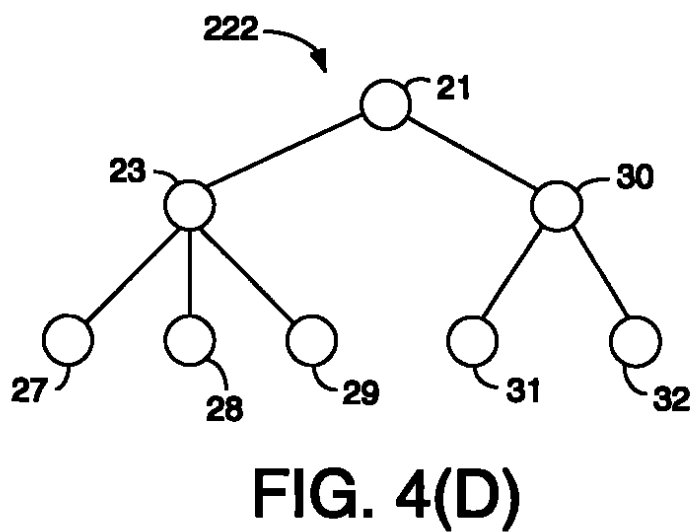
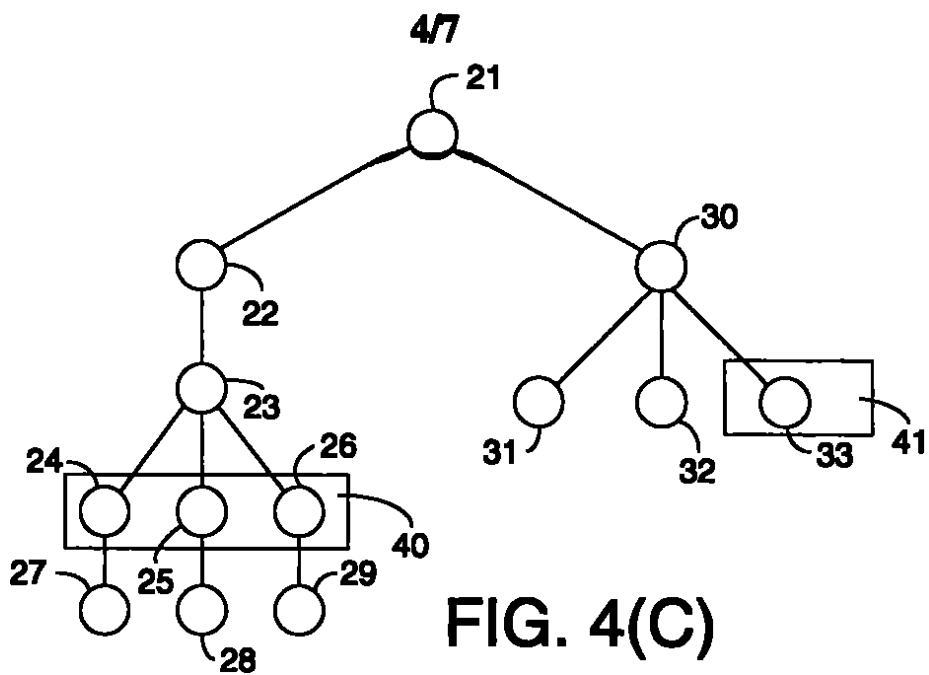


FIG. 4(B)

42
43



43
44

57

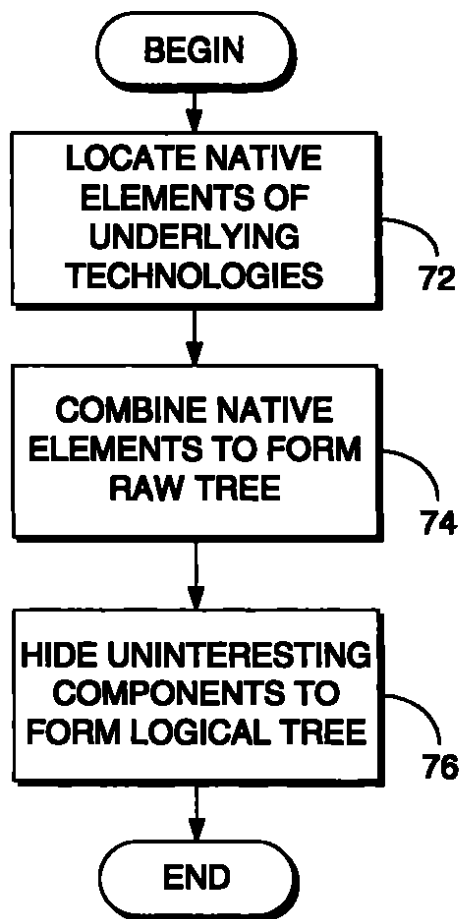


FIG. 5

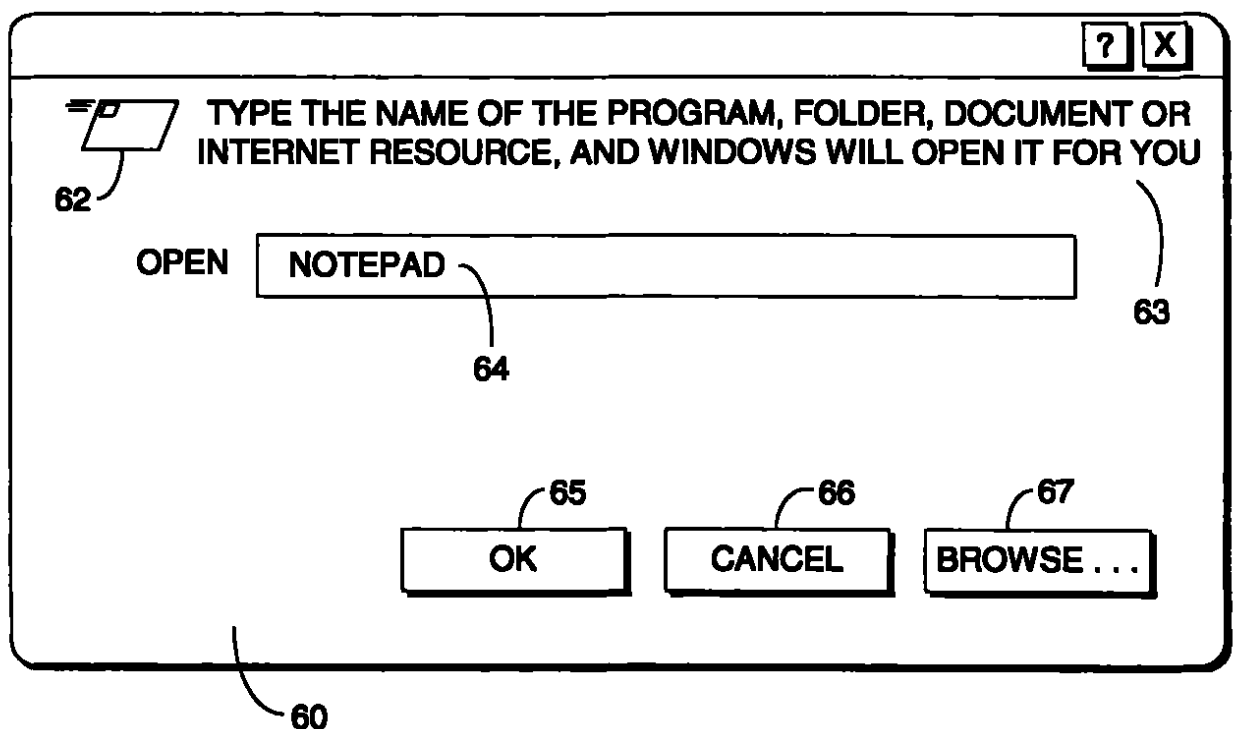


FIG. 6

44
45

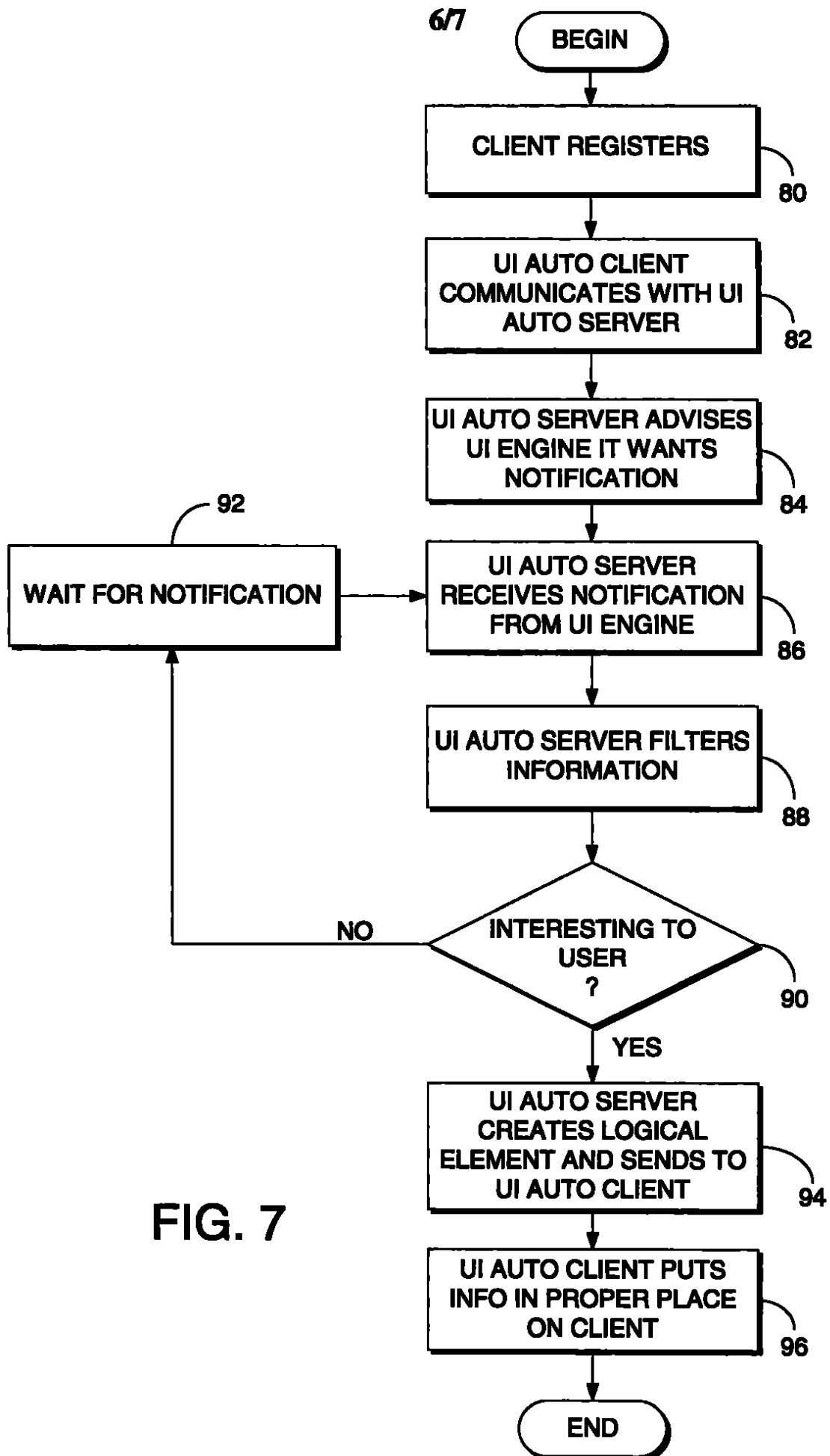
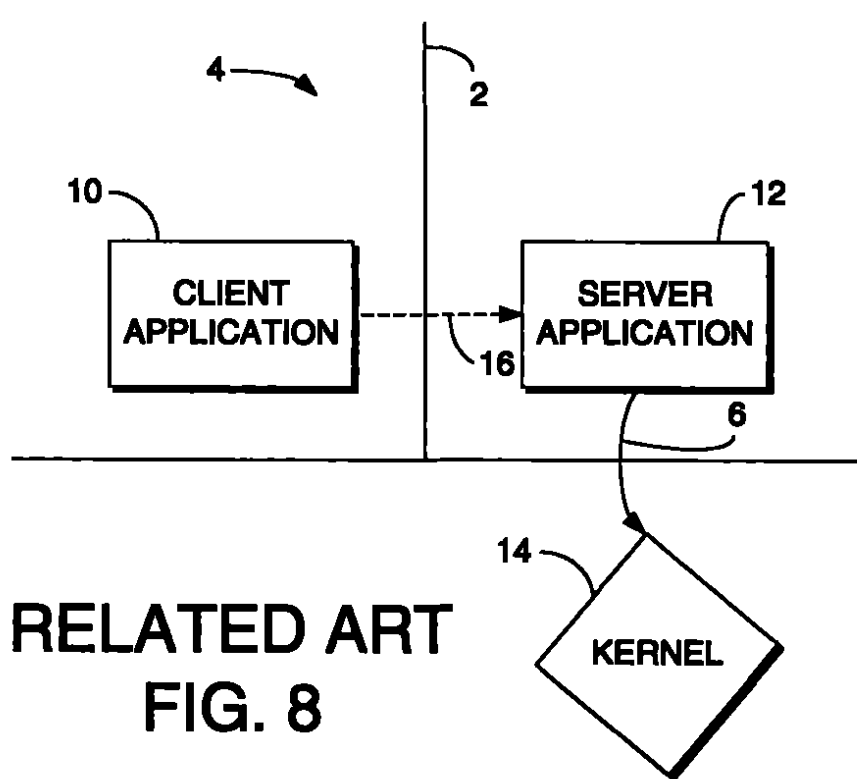


FIG. 7

45

46

77



RELATED ART
FIG. 8

46
47

**MECANISMOS Y MÉTODOS PARA SUCESOS DE UN
SISTEMA DE ACCESIBILIDAD**

CAMPO TÉCNICO

El invento se relaciona con el campo de tecnología de asistiva o de ayuda, pruebas automatizadas, y otros productos que recopilan información de la interfaz con el usuario, y también se relaciona con la interacción de estos productos con la información de la interfaz del usuario.

X
Dong

ANTECEDENTES DEL INVENTO

Los productos de tecnología asistiva o de ayuda (AT) existen para ayudar a los usuarios de computador que tienen la necesidad de recibir asistencia en áreas de aprendizaje, comunicación y acceso a información contenida en software de computador y presentada en el mismo. Estos productos tienen la necesidad de información relevante a la interfaz de computador. Similarmente, los productos existentes de pruebas automatizadas y las utilidades de comando de la interfaz del usuario también tienen la necesidad de información acerca de la interfaz del usuario. Actualmente estos productos no tienen suficiente información sobre la fuente de la interfaz del usuario (UI, por sus iniciales en inglés). Estos tres

Luis
?

47
48

tipos de productos (clientes) requieren obtener el apoyo necesario en otra parte para permitirles: (1) recopilar información acerca de una interfaz del usuario para la aplicación; (2) descubrir programáticamente e interrogar los elementos de la interfaz del usuario independientemente de la tecnología utilizada para construir la misma; (3) generar entradas para el teclado y el ratón; y (4) entender qué tipo de comportamiento o funcionalidad está actualmente disponible. En la actualidad no existe una sola tecnología que le de a un producto AT todas estas capacidades. Además, los productos actuales de tecnología asistiva o de ayuda (AT, por sus iniciales en inglés) no siempre son compatibles con todas las tecnologías de sistemas operativos gráficos (OS, por sus iniciales en inglés) y carecen de la capacidad para filtrar y coordinar notificaciones redundantes o que se presten a malas interpretaciones en una forma centralizada. Una desventaja adicional es que la automatización actual y las infraestructuras de accesibilidad no se pueden extender y por consiguiente requieren cambios al nivel del Sistema Operativo para añadir una nueva funcionalidad.

Además, actualmente para recopilar información acerca de una interfaz del usuario para una aplicación, el producto AT debe escribir el código específico de la aplicación para

48
49

obtener información para el usuario. El proceso de describir este código específico de la aplicación es dispendioso y requiere mantenimiento continuo. La infraestructura de automatización actual también carece de capacidad para filtrar y coordinar notificaciones de sucesos redundantes o que conduzcan a malas interpretaciones en una forma consistente. Por consiguiente, los consumidores de sucesos requieren filtrar independientemente la información.

Los sistemas actuales permiten que los productos AT soliciten notificaciones de sucesos en tres niveles de granularidad: (1) todo en un escritorio ("desktop"); (2) en un proceso particular (tal como la apertura de un procesador de palabra); o (3) en un ensarte ("thread") en el proceso particular (múltiples objetos realizando trabajo en el proceso). Actualmente, cuando el cliente recibe un suceso, recibe el manejo de una ventana para una ventana específica en la que ocurrió el suceso y otros trozos de información para indicar en dónde ocurrió el suceso. El cliente puede realizar una llamada de proceso cruzado para recuperar el objeto de la interfaz del usuario (UI, por sus iniciales en inglés) que se relaciona con el suceso. Con este objeto, el cliente puede efectuar llamadas adicionales de proceso cruzado para solicitar información acerca de ese objeto. Si

49
50

el cliente necesita cinco pedazos de información, debe entonces efectuar cinco llamadas de proceso cruzado. Las llamadas de proceso cruzado son excesivamente lentas, de manera que el costo del desempeño de recopilar información UI utilizando la infraestructura de accesibilidad actual es alto. Este tipo de escenario conocido se muestra en la FIGURA 8. Una aplicación de servidor (12) activa un suceso (6). Un núcleo (14) determina qué clientes deben ser notificados y envía una notificación del suceso (18) a un cliente interesado (10). El cliente (10) hace una solicitud (16) desde la aplicación del servidor (12) a través de un límite del proceso (2) para el objeto relacionado con la notificación del suceso (18). La aplicación de servidor (12) devuelve el objeto (20) y luego el cliente (10) puede comenzar a enviar solicitudes (16) de información acerca del control UI que disparó el suceso. La aplicación del servidor (12) devuelve la información solicitada (20) a través del límite del proceso (2) al cliente (10).

Otra opción corriente permite que el código del cliente se cargue como una biblioteca de enlace dinámico (.DLL, por sus iniciales en inglés) dentro de un proceso. Esta opción tiene varias desventajas. Primero, requiere cooperación del sistema para cargar el código del cliente en un proceso. Como

50
51

segunda medida da lugar a problemas de seguridad porque una vez que el código del cliente está cargado en un proceso de la aplicación es difícil restringir la información que recopila. Tercero, para ser una técnica eficiente para el cliente, debe cargarse en cada proceso del sistema. Óptimamente, sólo los clientes en quienes se puede confiar deberían cargarse en otro proceso de la aplicación.

Además, se necesita un sistema que le de al cliente la capacidad de especificar qué notificaciones de sucesos desea recibir. En los sistemas conocidos, un cliente puede necesitar efectuar una serie de llamadas de proceso cruzado y luego analizar la información para determinar si está interesado en el suceso. Se necesita un mecanismo que pueda realizar esta filtración de sucesos en una forma más funcional que pueda actualizarse fácilmente para soportar sucesos de la aplicación o de un nuevo sistema. Además, se necesita un sistema que utilice solamente componentes en los que se confie con el fin de aliviar los problemas de seguridad.

Actualmente, cuando se busca información acerca de una interfaz del usuario, se requiere que el producto AT acceda a árboles que son nativos a un marco UI particular. Por

51
52

consiguiente, se requieren varios árboles para transportar información de la interfaz del usuario para múltiples marcos UI. Estos diferentes árboles pueden contener información que no sea de interés o que no sea visible para el usuario, tal como objetos ocultos en un contenedor que maneje los controles visibles UI manipulados por el usuario final. Por consiguiente, existe la necesidad de un sólo árbol unificado que tenga solamente aquellos nodos de interés para el usuario.

Se necesita una solución que aborde las necesidades de los productos AT, las herramientas de pruebas automatizadas, y las utilidades de comando. La solución debe poder utilizarse por parte de todas las tecnologías de Sistemas Operativos gráficos y debería permitir todas las formas de Interfaces del Usuario y componentes de las Interfaces del Usuario para ser accesibles.

BREVE RESUMEN DEL INVENTO

El presente invento está dirigido a un método y aplicación de computador para proporcionarle a un cliente información de interfaz del usuario. En un aspecto del invento, un mecanismo de sucesos se provee dentro de un ambiente de servidor del cliente para mantener a un cliente

15
e

52
53

notificado de los cambios en la interfaz del usuario (UI). El mecanismo de sucesos incluye un mecanismo de registro que le permite al cliente registrarse para recibir información interesante de la Interfaz del Usuario. El mecanismo de sucesos también incluye un cliente de automatización de UI para registrar clientes que quieran recibir notificaciones de sucesos y para transportar esas notificaciones a los clientes cuando éstas ocurran. El servidor de automatización de la Interfaz del Usuario (UI) recibe las notificaciones del marco de la UI e incluye un mecanismo de filtración para remover los sucesos que no sean interesantes para el cliente, y un mecanismo de notificación para notificar al cliente de automatización de UI todas las notificaciones interesantes restantes.

Otro aspecto del invento proporciona un método implementado en computador para recopilar información acerca de los elementos interesantes de la interfaz del usuario (UI). El método incluye recibir una solicitud de un cliente para información especificada de la Interfaz del Usuario y preguntarle al motor de la Interfaz del Usuario utilizado por un servidor de automatización de la Interfaz del Usuario para determinar si la solicitud de información de la Interfaz del Usuario está disponible. La información especificada de la

15
E

53
54

Interfaz del Usuario se pasa del servidor de automatización de la Interfaz del Usuario a un cliente automatizado de la Interfaz del Usuario.

En la descripción que sigue se estipularán las ventajas adicionales y las características novedosas del invento, y en parte serán evidentes para quienes conozcan la técnica cuando examinen lo siguiente, o podrán aprenderlo cuando practiquen el invento.

BREVE DESCRIPCIÓN DE LOS DIBUJOS

El presente invento se describe en detalle a continuación con referencia a las figuras de los dibujos adjuntos, en donde:

La FIGURA 1 es un diagrama de bloque de un ambiente de un sistema de computación adecuado para ser utilizado en la implementación del presente invento;

La FIGURA 2 es un diagrama de bloque de la interacción entre un sistema de accesibilidad, un ambiente del cliente, y un ambiente del servidor;

La FIGURA 3 es un diagrama de bloque que ilustra los componentes del núcleo del sistema de accesibilidad;

54
55

Las FIGURAS 4(A)-4(D) ilustran la creación de un árbol lógico a partir de elementos nativos;

La FIGURA 5 es un cuadro de flujo que muestra una secuencia de procedimientos para construir un árbol lógico;

La FIGURA 6 muestra una caja de diálogo y sus componentes formando elementos lógicos;

La FIGURA 7 es un cuadro de flujo que ilustra los procedimientos involucrados en activar un mecanismo de sucesos del invento; y

La FIGURA 8 muestra un sistema conocido para la notificación de sucesos.

DESCRIPCIÓN DETALLADA DEL INVENTO

Ambiente Operativo del Ejemplo

La FIGURA 1 ilustra un ejemplo de un ambiente de un sistema de computación adecuado (100) en el cual se puede implementar el invento. El ambiente del sistema de computación (100) constituye solamente un ejemplo de un ambiente de computación idóneo y no pretende sugerir ninguna limitación en cuanto al alcance de la utilización o de la funcionalidad del invento. Tampoco el ambiente de computación

85
56

del invento (100) se deberá interpretar como que tiene cualquier dependencia o exigencia relacionada con cualesquiera de las combinaciones de componentes ilustrados en el ambiente operativo del ejemplo (100).

El invento se puede describir en el contexto general de instrucciones que se pueden ejecutar por computador, tales como módulos de programas, que son ejecutados por un computador. Por lo general, los módulos del programa incluyen rutinas, programas, objetos, componentes, estructuras de datos, etc. que realizan tareas particulares o implementan tipos de datos abstractos particulares. Además, quienes conocen la técnica apreciarán que el invento se puede practicar con otras configuraciones de sistemas de computador, incluyendo artefactos manuales, sistemas de multiprocesadores, elementos electrónicos para el consumidor programables o basados en microprocesadores, minicomputadores, computadores principales y similares. El invento se podrá también practicar en ambientes de computación distribuidos en donde las tareas son realizadas por dispositivos de procesamiento remotos que están enlazados a través de una red de comunicaciones. En un ambiente de computación distribuido, los módulos de los programas pueden estar localizados en medios de almacenamiento de computación

182

56
ST

tanto locales como remotos, incluyendo dispositivos de almacenamiento de memoria.

Con referencia a la FIGURA 1, un sistema del ejemplo (100) para implementar el invento incluye un dispositivo del computación de propósito general en la forma de un computador (110) que incluye una unidad de procesamiento (120), una memoria del sistema (130), y un bus del sistema (121) que acopla diversos componentes del sistema a la unidad de procesamiento (120) incluyendo la memoria del mismo.

El computador (110) típicamente incluye una variedad de medios que se pueden leer por computador. A manera de ejemplo y no de limitación, los medios que se pueden leer por computador pueden incluir medios de almacenamiento de computador y medios de comunicaciones. La memoria del sistema (130) incluye medios de almacenamiento de computador en la forma de memoria volátil y/o no volátil, tal como memoria de solo lectura (ROM) (131) y memoria de acceso aleatorio (RAM) (132). Un sistema de entrada/salida básico (133) (BIOS), que contiene las rutinas básicas que ayudan a transferir la información entre los elementos dentro del computador (110), tal como durante el arranque, se almacena típicamente en (ROM) (131). RAM (132) típicamente contiene módulos de datos

57
58

y/o programas que son inmediatamente accesibles y/o que actualmente están siendo operados por la unidad de procesamiento (120). A manera de ejemplo y no de limitación, la FIGURA 1 ilustra el sistema de operación (134), los programas de la aplicación (135), otros módulos del programa (136), y los datos del programa (137).

El computador (110) también puede incluir otros medios de almacenamiento de computador volátiles/no volátiles, removibles/no removibles. A manera de ejemplo únicamente, la FIGURA 1 ilustra una unidad de disco duro (141) que lee de medios magnéticos o que escribe a medios magnéticos no volátiles, no removibles, una unidad de disco magnético (151) que lee de un disco magnético o escribe a un disco magnético no volátil removible (152), y una unidad de disco óptico (155) que lee de un disco óptico o escribe a un disco óptico no volátil removible (156), tal como un CD ROM u otro medio óptico. Otros medios de almacenamiento de computador volátiles/no volátiles, removibles/no removibles que se pueden utilizar en el ambiente de operación del ejemplo incluyen casetes de cintas magnéticas, tarjetas de memoria flash, discos versátiles digitales, cinta de video digital, RAM de estado sólido, ROM de estado sólido, y similares, sin que estos impliquen limitación. La unidad de disco duro (141)

con v. p

58
59

está típicamente conectada al bus del sistema (121) a través de una interfaz de memoria no removible tal como la interfaz (140) y la unidad de disco magnético (151) y la unidad de disco óptico (155) están típicamente conectadas al bus del sistema (121) mediante una interfaz de memoria removible, tal como la interfaz (150).

Las unidades (drives) y sus medios de almacenamiento de computación asociados discutidos arriba e ilustrados en la FIGURA 1 proporcionan almacenamiento de instrucciones que se pueden leer por computador, estructuras de datos, módulos de programas y otros datos para computador (110). En la FIGURA 1, por ejemplo, la unidad de disco duro (141) se ilustra como Sistema de Operación de almacenamiento (144), programas de aplicación (145), otros módulos del programa (146), y datos del programa (147). Nótese que estos componentes pueden bien sea ser los mismos del sistema operativo o diferentes (134), programas de aplicación (135), otros módulos del programa (136) y datos del programa (137). Al sistema operativo (144), los programas de aplicación (135), otros módulos del programa (146), y a los datos del programa (147) se les otorgan diferentes números aquí para ilustrar que, como mínimo, son diferentes copias. Un usuario puede ingresar comandos e información al computador (110) a través de dispositivos de

59
60

entrada, tales como un teclado (162) y un dispositivo de indicación (161), comúnmente denominado ratón, bala de guía ("trackball") o tablilla de graficación tacto-sensible ("touch pad"). Otros dispositivos de entrada (que no se muestran) pueden incluir un micrófono, palanca de mando ("joystick"), paleta de mando para videojuegos ("game pad"), antena satelital, escáner y similares. Estos y otros dispositivos de entrada con frecuencia están conectados a la unidad de procesamiento (120) a través de una interfaz de entrada del usuario (160) que está acoplada al bus del sistema, pero puede estar conectada mediante otras interfaces y estructuras de bus, tales como un puerto paralelo, un puerto de juegos o un bus de serie universal (USB, por sus iniciales en inglés). Un monitor (191) u otro tipo de dispositivo de exhibición también está conectado al bus del sistema (121) a través de una interfaz, tal como una interfaz de video (190). Además del monitor, los computadores pueden también incluir otros implementos periféricos de salida, tales como parlantes (197) e impresora (196), que pueden estar conectados a través de una interfaz periférica de salida (195).

El computador (110) del presente invento puede operar en un ambiente de red utilizando conexiones lógicas a uno o más

60
61

computadores a remoto, tal como un computador remoto (180). El computador remoto (180) puede ser un computador personal y típicamente incluye muchos o todos los elementos descritos arriba en relación con el computador (110), aún cuando en la FIGURA 1 sólo se vea ilustrado un dispositivo de almacenamiento de memoria (181). Las conexiones lógicas ilustradas en la FIGURA 1 incluyen una red de área local (LAN) (171) y una red de área ancha (WAN) (173), pero también pueden incluir otras redes.

Cuando se utiliza en un ambiente de red LAN, el computador (110) está conectado al LAN (171) a través de una interfaz de red o adaptador (170). Cuando se utiliza en un ambiente de red WAN, el computador (110) típicamente incluye un modem (172) u otros medios para establecer comunicaciones sobre la WAN (173), tal como la Internet. El modem (172), que puede ser interno o externo puede estar conectado al bus del sistema (121) a través de la interfaz de entrada del usuario (160) u otro mecanismo apropiado. En un ambiente de red, los módulos del programa ilustrados en relación con el computador (110), o las porciones de los mismos, pueden estar almacenados en un dispositivo de almacenamiento de memoria remoto. A manera de ejemplo y no de limitación, la FIGURA 1 ilustra programas de aplicación remota (185) que residen en

el dispositivo de memoria (181). Se apreciará que las conexiones de la red que se muestran son del ejemplo y que se pueden utilizar otros medios de establecer un enlace de comunicaciones entre los computadores.

Aún cuando muchos otros componentes internos del computador (110) no se muestran, quienes conocen la técnica apreciarán que esos componentes y la interconexión son bien conocidos. Por consiguiente, no debemos hacer revelar detalles adicionales respecto de la construcción interna del computador (110) en relación con el presente invento.

Estructura el Sistema de Accesibilidad

Tal como se muestra en la FIGURA 2, un sistema de accesibilidad (200) interactúa con un ambiente del cliente (300) y un ambiente del servidor (400). El sistema de accesibilidad (200) se puede implementar en un ambiente de computador (100) arriba descrito con respecto a la FIGURA 1. El sistema de accesibilidad (200) incluye una interfaz de accesibilidad lateral del cliente (220) para facilitar la interacción con el cliente (300), una interfaz de accesibilidad lateral del servidor (230) para facilitar la interacción con el costado del servidor (400), y un núcleo del sistema de accesibilidad (201). El sistema de

62
63

accesibilidad (200) del invento proporciona nuevas interfaces de los programas de la aplicación (APIs), interfaces, y metáforas para acceder programáticamente a la Interfaz del Usuario (UI). El sistema de accesibilidad (200) permite hacer que éstas y cualesquiera de los componentes que éstas utilizan sean accesibles.

El ambiente del cliente (300) preferiblemente incluye un producto de tecnología asistida o de ayuda (AT, por sus iniciales en inglés) o una herramienta de pruebas automatizadas de la Interfaz del Usuario (UI). El costado del servidor (400) puede implementar una variedad de tecnologías diferentes, tal como se muestra en la FIGURA 2. Un sistema de servidor (410) incluye un adaptador (412) y un núcleo (414), que se pueden encontrar en un primer tipo de Interfaz del Usuario (UI). Un sistema de servidor (420) incluye un componente de representantes (proxies) (422) y controles (424), tal como se encuentran en un segundo tipo de Interfaz del Usuario (UI), tal como una Win32 UI disponible en los productos de Sistema Operativo de Microsoft, en Microsoft Corporation de Redmond, Washington. El sistema del servidor (430) incluye un adaptador (432) y un módulo interno (OM) (434), que se pueden encontrar en un tercer tipo alternativo de UI.

63
64

Tal como se muestra en la FIGURA 3, un mecanismo de sucesos (210), que se incluye en el sistema de accesibilidad (200), se basa en un cliente de automatización UI (202) y un servidor de automatización UI (204) para facilitar la interacción con el ambiente del cliente (300) y el ambiente del servidor (400). El cliente de automatización UI (202) y el servidor de automatización UI (204) se describen con más detalle a continuación haciendo referencia al mecanismo de sucesos (210) del invento. El sistema de accesibilidad (200) del invento le otorga el cliente (Producto AT) (300) la capacidad de: (1) recuperar información acerca de la Interfaz del Usuario para una aplicación; (2) descubrir programáticamente e interrogar a los elementos UI independientemente de la tecnología utilizada para construir la Interfaz del Usuario (UI); (3) generar el teclado y la entrada del puntero; y (4) entender qué tipo de comportamiento o funcionalidad está actualmente disponible. El sistema de accesibilidad (200) permite que las aplicaciones y sus componentes sean accesibles. La estructura que se muestra en las FIGURAS 2 y 3 permite cinco aspectos principales del sistema de accesibilidad (200), que incluyen: (1) árbol lógico de la Interfaz del Usuario (UI); (2) Patrones de Control; (3) Mecanismo de Sucesos; (4) API de

effect
used

entrada (que no se cubre en este documento); y (5) propiedades.

Árbol lógico de acceso a la Interfaz del Usuario (UI) (222)

Un componente integral del sistema de accesibilidad (200) es el árbol lógico (222), cuyo ejemplo se muestra en la FIGURA 4(D). El árbol (222) se incluye en la interfaz de accesibilidad del costado del cliente (220).

El árbol lógico (222) es una vista filtrada de la jerarquía estructural subyacente de los elementos de la Interfaz del Usuario (UI) y no un árbol separado que deba implementarse por el desarrollador de la aplicación o el control. En vez, este armoniza unas pocas propiedades definidas, interesantes y no interesantes, que indican si un elemento estructural debe exponerse en el árbol lógico (222). El núcleo del sistema de accesibilidad (201) consume esta información para producir el árbol lógico filtrado de la Interfaz del Usuario (UI) (222) que a su vez se presenta a los productos de Tecnología Asistida (AT) o instrucciones de prueba ("test script").

El árbol lógico (222) es un árbol de elementos, cada uno de los cuales representa un control, un ítem en un control, o

una estructura de agrupación, que puede ser un diálogo, cuadro o marco. La estructura del árbol lógico (222) debe representar la Interfaz del Usuario de la aplicación, tal como lo percibe el usuario (inclusive si los controles se implementan realmente utilizando una estructura subyacente diferente). El árbol debe ser estable a través del tiempo. En tanto que una aplicación se vea igual para un usuario, el árbol lógico (222) que representa esa aplicación debería seguir siendo el mismo, inclusive si los detalles de la implementación de la aplicación por detrás de las escenas han cambiado. Los elementos nativos que existen por razones estructurales y de implementación, tales como la ventana del esqueleto "ShDocView" en los productos Microsoft de Sistemas Operativos (OS) no deberían aparecer en este árbol, puesto que el usuario no los percibe.

El árbol lógico (222) es un simple árbol construido a partir de una pluralidad de fragmentos que puede unificar una pluralidad de procesos diferentes de manera que sean iguales para el cliente. El árbol lógico (222) permite la recuperación y volumen y puede obtener un valor para una lista de propiedades. Para el momento en que un usuario normalmente habría invocado una llamada de proceso cruzado para preguntar por valores, el sistema de accesibilidad (200)

66
67

les habrá buscado dichos valores a través de la utilización del árbol lógico (222).

En lugar de estar construido en una etapa, tal como en los sistemas conocidos, el árbol lógico (222) está construido a partir de fragmentos que se utilizan para edificar un árbol en bruto. Tal como se muestra en la FIGURA 5, tres procedimientos principales construyen el árbol lógico (222). En el procedimiento (72), el sistema de accesibilidad (200) localiza los elementos nativos de las tecnologías subyacentes y llega a los árboles nativos que se muestran en la FIGURA 4(A). En el procedimiento (74), el sistema de accesibilidad (200) combina los elementos nativos para formar el árbol en bruto (20), tal como se muestra en la FIGURA 4(B). Finalmente, en el procedimiento (76), el árbol lógico (222) se obtiene ocultando los componentes no interesantes en el árbol en bruto (20), tal como se muestra en la FIGURA 4(D).

La FIGURA 4(A) ilustra dos árboles nativos (10) y (14), que están contruidos con elementos nativos de tecnologías subyacentes, tales como la Win32 UI, o cualquier otra UI disponible. El árbol nativo (10) incluye un nodo padre (11) y una pluralidad de descendientes (12) que tienen diversas relaciones entre sí. Similarmente, el árbol nativo '(14)

67
68

incluye un nodo padre (15) que tiene una pluralidad de nodos hijos (16). Los nodos hijos (16) se pueden describir como codescendientes el uno del otro.

Tal como se muestra en la FIGURA 4(B), los árboles nativos (10) y (14) se pueden combinar para formar un árbol en bruto (20). El árbol en bruto (20) incluye un nodo padre (21), que tiene dos nodos hijos (22) y (30). El nodo hijo (22) tiene descendientes (23-29) y el nodo hijo (30) tiene descendientes (31-33). Este árbol en bruto (20) es una combinación de los árboles nativos (10) y (14), con los nodos del árbol nativo (10) que forman los nodos (22-29) y los nodos del árbol nativo (14) forman los nodos (30-33).

A través de un método que se muestra ampliamente en las FIGURAS 4(C) y 4(D), el árbol en bruto (20) se convierte en el árbol lógico (222). Para moverse del árbol en bruto (20) al árbol lógico (222), un desarrollador puede insertar insinuaciones en el árbol en bruto. El desarrollador puede marcar nodos dentro del árbol en bruto (20) como "ocúltese solo" u "ocúltese con los descendientes" o como "oculte los descendientes del nodo", etc. El desarrollador también puede mover los nodos hacia los lados o colocar nodos antes de los descendientes. Estas "insinuaciones" y modificaciones en el

árbol en bruto (20) se utilizan para formar el árbol lógico (222). Por ejemplo, en la FIGURA 4(C) un desarrollador ha marcado los nodos (24-26) y (33) del árbol en bruto (20) como no interesantes según lo indicado por los bloques 40 y 41. Típicamente, los nodos que contienen elementos que no estarán visibles para el usuario, se marcan como no interesantes. Los nodos relacionados con la Interfaz del Usuario visibles típicamente se consideran interesantes y se describirán en el árbol lógico (222) para uso por parte del cliente de Tecnología Asistiva o de ayuda (AT) (300). Tal como se muestra en la Figura 4(D), los nodos marcados como no interesantes no se incluyen en el árbol lógico (222).

El sistema de accesibilidad (200) utiliza el árbol lógico (222) para encontrar información acerca de sucesos, el estado del sistema, la localización de los objetos y la información acerca de los controles. Los sistemas conocidos no incluyen la capacidad de abrir un ámbito dentro de sus árboles. En el árbol lógico (222) se puede navegar basándose en las preferencias del cliente (300), y es capaz de proporcionar información independientemente de la aplicación del lado del servidor en uso.

En operaciones, si un cliente (300) necesita obtener información para un usuario acerca de una solicitud, el

68
70

cliente (300) busca un botón para oprimir y los observadores escriben el botón. El cliente (300) llamaría "encuentre un elemento" de la Interfaz del Programa de aplicación (API). La API devolverá el valor que se muestre con respecto a una posición que el árbol lógico (222) del cliente tenga como interfaz lateral (220). A través del árbol lógico (222), el sistema de accesibilidad (200) le otorga al cliente (300) una visión general abstracta de la UI, independientemente de la aplicación en uso. El modelo abstracto incluye estructuras, propiedades, sucesos y funcionalidad que una casilla de listas, botones u otros componentes de la UI podrían esperar tener en común entre sí.

El árbol lógico (222) es un sólo árbol unificador que constituye una representación lógica de la Interfaz del Usuario y está formado de una configuración que incluye solamente elementos de interés para los clientes (300). Por consiguiente, en lugar de forzar a los productos AT a filtrar la jerarquía estructural de los elementos de la UI y conjeturar en cuanto al modelo que se está presentando al usuario final, el árbol lógico (222) presenta una jerarquía que mapea de cerca la estructura que se está presentando al usuario final. Esto simplifica enormemente la tarea del

69
71

producto AT de describir la UI al usuario y ayuda al usuario a interactuar con la aplicación.

Como este árbol lógico de la UI (222) constituye parte fundamental del sistema de accesibilidad (200), todos los demás componentes del sistema de accesibilidad (200) están diseñados para funcionar a partir del árbol lógico (222). Por ejemplo, la FIGURA 6 muestra una casilla de diálogos simple (60) que parece tener una estructura muy simple. Sin embargo, cuando se le mira a través de la tecnología de accesibilidad actualmente disponible, la estructura de esta casilla de diálogo (60) es sorprendentemente compleja. Contiene 264 objetos que un producto AT debe filtrar para descubrir los que tienen significado para el usuario final. Con el sistema de accesibilidad (200) y su soporte para el Árbol lógico de la UI (222), el desarrollador que posea esta casilla de diálogo (60) puede establecer unas pocas propiedades para presentar la siguiente estructura que se muestra en la FIGURA 6 a los productos AT (300).

Tal como se muestra en la FIGURA 6, para un diálogo "Run" (Corrida) el desarrollador puede indicar como interesante la gráfica de la ventana volante (62) y "Digitar el nombre del programa, fólder, documento o recurso de

70
72

Internet y Windows lo abrirá para usted" en el 63. El Desarrollador puede también indicar como interesante la casilla combo (64) que incluye la libreta de notas (notepad), word, la calculadora, etc. y el OK (65), y los botones de cancelar (66) y ojear (67). Esto le ofrece a los desarrolladores un mecanismo de bajo costo para rotular su jerarquía de elementos y producir así una representación lógica de su Interfaz del Usuario para su aplicación a través del sistema de accesibilidad de la UI (200). Cada una de las características mostradas se puede representar mediante un nodo que tiene una relación específica con cada uno de los demás nodos en el árbol lógico (222). Esta representación lógica ofrece un beneficio inmediato para un equipo de prueba y para los productos o clientes AT (300).

Un conjunto de APIs le permite al cliente (300) llegar al árbol. La funcionalidad incluye: (1) un elemento lógico de punto a punto; (2) un elemento lógico del suceso; y (3) un elemento lógico enfocado actualmente. Tal como arriba se dijo, un elemento lógico representa un componente de la UI, posiblemente un control, una parte de un control, o un contenedor o agrupación lógica (v.g. diálogo, cuadro (pane) o marco). Los controles pueden variar ampliamente en términos de su funcionalidad. Por consiguiente, se utilizan diferentes

77
73

interfaces para representar la funcionalidad asociada con los tipos particulares de controles. Estas interfaces de control específico se derivan de una interfaz de base común que representa funcionalidad común para todos los controles. La interfaz de base común contiene: (1) métodos para navegar el árbol lógico (222); (2) un método general para obtener valores de la propiedad; y (3) métodos para acceder a las interfaces soportadas por un control específico. Al navegar el árbol lógico (222), cada tecnología de la UI de la aplicación subyacente proporcionará su propia técnica para navegación.

Aún cuando el árbol lógico (222) es de máximo interés para el usuario, el árbol del elemento en bruto (20) también presta unas funciones importantes. Mientras que el árbol lógico (222) contiene solamente elementos con los cuales se puede relacionar el usuario, el árbol de elementos en bruto (20) contiene nodos, tales como (22) que representa la estructura de implementación del marco subyacente. Para un fragmento Win 32 UI, por ejemplo, este árbol contendría nodos que representan HWNDs. En algunos aspectos el árbol de elementos en bruto está a "media casa" entre el árbol de elementos lógicos (222) y los árboles de elementos nativos de los propios marcos de trabajo subyacentes. El árbol de

72
74

elementos en bruto (20) se utiliza como una base para construir el árbol de elementos lógicos (222) y es en donde los elementos nativos se enchufan por primera vez al sistema.

El árbol de elementos en bruto (20) también se puede utilizar para retirar virus y hacer pruebas. Es útil para mostrar o describir en dónde se encuentra un nodo problemático en particular. La funcionalidad en un nodo base de elementos en bruto incluye: métodos para navegar el árbol de elementos en bruto; un método para resetiar a un elemento lógico correspondiente (si existe); propiedad que contiene la "sarta para retirar virus" (debug string) para este elemento -v.g. "HWND 0x483FE" para los nodos HWND; y otros métodos "detrás de la infraestructura de la escena". Estos otros métodos permiten la prueba de aciertos y la localización, los sucesos, y exponer las propiedades que los marcos pueden proporcionar fácilmente (v.g. enfocados, y habilitados).

El árbol de elementos en bruto (20) contiene nodos (23-33) que representan elementos de diversos motores de representación. El árbol de los elementos en bruto se utiliza como un punto de partida para los motores de representaciones para conectarse al sistema de accesibilidad (200) y está construido de objetos del adaptador de peso liviano que

73
75

adaptan los elementos nativos, tales como HWNDs del Win32 en un árbol lógico unificado (222). Se utiliza adicionalmente para manejar transiciones de distribución de la información donde una tecnología hospeda otra. Como el árbol de los elementos en bruto (20) es la base sobre la cual el árbol lógico (222) está construido, éste se puede utilizar para verificar que el árbol lógico (222) está completo y conectado y se puede utilizar para verificar los elementos que no se han contabilizado. El árbol de elementos en bruto (20) se puede utilizar además para otras tareas de tipo de infraestructura: tales como proporcionar algún elemento básico ID y proporcionar algunas propiedades de elementos proporcionados por el marco básico, tales como enfocado, habilitado, y localización.

El árbol de elementos en bruto (20) no es la fuente primaria de información para los productos o clientes AT (300), no se utiliza para navegación lógica si no está expuesto a los usuarios finales. El árbol de elementos en bruto (20) tampoco se puede utilizar para capturar una posición del elemento en el árbol de manera que se pueda regresar a él en algún punto futuro en el tiempo. El árbol de elementos lógicos (22) realiza todas estas funciones.

El árbol de elementos en bruto (20) se puede construir típicamente en forma mecánica a partir de los elementos en bruto de la tecnología de representación subyacente (HWNDs, Elementos) sin conocimiento de los elementos lógicos representados. Por consiguiente se puede utilizar para buscar elementos en bruto que no hayan sido contabilizados en el árbol lógico (222). El árbol de elementos en bruto (20) constituye una herramienta útil de diagnóstico y de retiro de virus, puesto que permite capturar una descripción de la localización del nodo como un listado del contenido de la memoria ("stack dump"). Además, los sistemas conocidos basan sus árboles en criterios de códigos específicos y son difíciles de implementar con diversas tecnologías. El presente enfoque utiliza un compendio general tipo "elemento en bruto" que se puede implementar por o en nombre de cualquier tecnología de representación subyacente.

Para obtener el árbol del elemento en bruto, llamar a la raíz del elemento en bruto obtendrá un elemento del escritorio, verificado asegurándose de que su generador es NULL y de que todos los demás nodos lo tengan como su ancestro final. Para obtener otros elementos, llamar a un método para obtener un elemento en bruto desde un punto especificado devolverá el elemento utilizando coordenadas de

24
77

pantalla válida. Después de obtener el árbol de los elementos en bruto, éste se puede chequear y verificar chequeando los elementos (generadores, codescendientes y descendientes).

En operación, el cliente (300) puede navegar el árbol de los elementos en bruto (20) utilizando relaciones tales como: generador; codescendiente siguiente, codescendiente anterior, primer descendiente, último descendiente, etc. el Cliente (300) puede entonces saltar desde el elemento en bruto hasta el elemento lógico correspondiente en el árbol lógico (222).

Mecanismo de Sucesos

Cuando un cliente (300) quiere mantenerse informado de los sucesos, el cliente (300) puede registrarse a través del cliente de automatización de la UI (202) tal como se muestra en la FIGURA 3 para obtener la información. El cliente (300) especifica la información objeto que desea recibir, a dónde desea que vaya la información, y la lista de propiedades que quiere obtener. La solicitud del cliente va al cliente de automatización de la UI (202). El cliente de automatización de la UI (202) puede monitorear cualquier proceso en el escritorio. El servidor de automatización de la UI (204) lleva la cuenta de cuáles clientes (300) están escuchando y sabe cómo regresar al cliente de automatización de la UI

25
28

(202). El cliente de la automatización de la UI (202) informa al motor de la UI (206) sobre el interés del cliente, de manera que el motor de la UI (206) sepa cuándo informarle al servidor de automatización de la UI (204) sobre el suceso. El motor de la UI no tiene que utilizar la notificación del cliente pero puede elegir en vez notificar siempre al servidor de automatización de la UI (204) sobre los sucesos o notificar al servidor de automatización de la UI solamente si un cliente está escuchando cualesquiera de los sucesos. El aviso es útil si el motor de la UI desea activar la notificación del servidor de automatización de la UI solamente si hay un cliente escuchando los sucesos. El motor de la UI haría esto para evitar la posible degradación de velocidad de la UI y para evitar cargar los códigos de los módulos que de otra manera no necesita.

El Motor de la UI (206) informa luego al servidor de automatización de la UI (204) sobre un suceso de la UI. El servidor de automatización de la UI (204) devuelve el elemento lógico solicitado al cliente (300) y envía la información al cliente (300) incluyendo las propiedades del suceso que el cliente (300) solicitó. El servidor de automatización de la UI (204) decide qué información está dentro del alcance de la solicitud del cliente y solamente

76
79

forma un elemento lógico si la información es de interés. Formar un elemento lógico incluye la prebúsqueda en el lado del servidor de automatización de la UI, el establecimiento de las propiedades que el cliente ha indicado que utilizará cuando maneje el suceso. Por ejemplo, el servidor de automatización de la UI (204) puede descubrir un elemento lógico para una casilla de combinación de correo. El alcance sería la casilla de combinación de correo o sus descendientes. El cliente (300) puede solicitar a los descendientes/generadores/dependientes definir el alcance durante la fase de registro.

Después de que el servidor de automatización de la UI (204) determina si la información está dentro del alcance solicitado, construye un elemento lógico. El cliente de automatización de la UI (202) atiende al cliente (300) hablándole a las aplicaciones meta recibiendo la información solicitada del servidor de automatización de la UI (204) y enrutando los objetos a un espacio adecuado sobre el cliente (300).

El servidor de automatización de la UI (204) se crea cuando el cliente (300) se registra para recibir notificaciones de sucesos. Como ejemplo, un motor de la UI

77
80

(206) está corriendo una aplicación de procesamiento de palabras Microsoft Word. El cliente (300) se ha registrado para un cambio del nombre de la propiedad. El registro del cliente hace que se cree el servidor de automatización de la UI (204). El registro del cliente también informa al motor de la UI (206) que comience a notificar al servidor de automatización de la UI (204) para el nombre de la propiedad. El motor de la UI (206) no recibe el alcance de la información. El motor de la UI (206) llama a una de las APIs para el costado del servidor. El motor de la UI (206) especifica (1) qué propiedad cambió; (2) el nuevo valor de la propiedad; y (3) quizá el valor antiguo. El servidor de automatización de la UI (204) se crea basándose en los sucesos de interés para el cliente (300) y por consiguiente conoce sucesos, propiedades, clientes y ámbitos de interés, así que sabe si cualquier cliente (300) está interesado en crear el elemento lógico. Si más de un cliente (300) se ha registrado para sucesos con un servidor particular de automatización de la UI (204) y los clientes (300) se han registrado para el mismo suceso y han solicitado que las propiedades sean buscadas en bloque con el elemento lógico devuelto, cuando el servidor de automatización de la UI (204) envía un suceso de regreso a los clientes (300), cada cliente

78
81

(300) obtendrá la unión de las propiedades de búsqueda en bloque solicitadas devueltas con el elemento lógico.

Por cada cliente (300) que esté a la escucha, el servidor de automatización de la UI (204) notifica al cliente (300) pasándole el elemento lógico asociado con el suceso. El servidor de automatización de la UI (204) crea solamente un elemento lógico. Esto constituye una mejora sobre la tecnología actual en la que cada cliente (300) tendría que solicitar su propia copia del objeto que constituye la fuente del suceso.

Si el motor de la UI (206) no utiliza el artefacto del cliente para la automatización de la UI cuando los clientes se registran para sucesos, el motor de la UI (206) lo podrá solicitar al servidor de automatización de la UI (204) si existen cualesquiera clientes de accesibilidad (300) escuchando y si ninguno lo está, puede entonces evitar el trabajo de crear la información y de enviarla al servidor de automatización de la UI (206). Por ejemplo, un lector de pantalla es el cliente (300) y especifica a dónde quiere que la información vaya, el objeto del cambio del foco para recibir el suceso, y la lista específica de las propiedades de interés. El motor de la UI (206) está avisado y sabe que

78
82

debe enviar los sucesos al servidor de automatización de la UI (204). Cuando detecte los cambios de foco, el motor de la UI (206) notifica al servidor de automatización de la UI (204). El servidor de automatización de la UI se convierte a una interfaz bien conocida y envía el suceso y el objeto al cliente de automatización de la UI (202). El cliente de la automatización de la UI (202) enruta el objeto a un espacio apropiado en el cliente (300).

Los componentes arriba descritos mejoran en los sistemas conocidos eliminando el repositorio central en el núcleo para sucesos. En vez, el servidor de automatización de la UI (204) conoce a todos los clientes (300) interesados en obtener información acerca del contexto en el cual está corriendo. La eliminación del repositorio del núcleo también crea una interacción más de semejante a semejante puesto que el servidor de automatización de la UI (204) cumple la función previamente desempeñada en el núcleo. El sistema de accesibilidad (200) del invento le da al cliente (300) la capacidad de especificar lo que éste quiere ver de manera que el filtrado se logre en el lado del servidor que utiliza el servidor de automatización de la UI (204).

La FIGURA 7 es un diagrama de flujo que muestra los procedimientos involucrados en el registro de sucesos y en el

método de notificación. En el paso (80), el cliente (300) solicita una notificación del suceso. En el paso (82), el cliente para la automatización de la UI (202) comunica la solicitud al servidor de automatización de la UI (204). En el paso (84), el cliente de automatización de la UI informa al motor de la UI (206) que desea una notificación. En el paso (86), el servidor de automatización de la UI (204) recibe notificación del motor de la UI (206). En el paso (88), el servidor de automatización de la UI (204) filtra la información recibida. Si se encuentra que la información recibida no es de interés para el usuario en el paso (90), el servidor de automatización de la UI (204) descarta la información y continúa esperando una notificación en el paso (92). Alternativamente, si en el paso (90) se encuentra que la información es de interés, el servidor de automatización de la UI (204) crea un elemento lógico y lo envía al cliente de automatización de la UI (202) en el paso (94). En el paso (96), el cliente de automatización UI (202) coloca la información recibida en su propio lugar en el cliente(300).

El mecanismos de los sucesos (210) del sistema de accesibilidad (200) permiten que el cliente (300) se registre para recibir notificaciones de sucesos por cambios a la propiedad en la UI, cambios en el árbol en una estructura de

84

control, sucesos de multimedia e información relacionada. Sin estas capacidades, los clientes (300) tienen que escrutar continuamente todos los elementos de la UI en el sistema para ver si cualquier información, estructura o estado ha cambiado. El mecanismo de sucesos (210) del sistema de accesibilidad (200) también le permite a los clientes (300) recibir sucesos fuera del proceso, solicitar una recopilación de propiedades para ser devueltas con la notificación del suceso y registrarse para sucesos en elementos múltiples.

El mecanismo de sucesos (210) expone: las interfaces que el producto AT o la aplicación de prueba utiliza para registrarse para los sucesos; las interfaces que el producto AT implementa en los objetos utilizados para recibir las notificaciones de sucesos; y las interfaces que quienes implementan el control utilizan para notificar al motor de sucesos sobre los sucesos de la UI. El mecanismo de sucesos (210) se utiliza para permitirle a los productos AT y a las aplicaciones de prueba recibir sucesos independientemente de los motores de la UI utilizados para representar la UI y permite que los productos AT y las aplicaciones de prueba rastreen las ventanas de aplicación de nivel superior y se enfoquen sin tener en cuenta la tecnología subyacente.

82
85

La FIGURA 2 muestra cómo los clientes (300) interactúan con el sistema de accesibilidad (200). No existe un manejo global de eventos a lo largo de los procesos. En vez, cada cliente del sistema de accesibilidad (300) o servidor (400) tiene su propio contexto. Las aplicaciones del cliente solamente recibirán sucesos para aplicaciones que soportan el sistema de accesibilidad (200). Para comenzar a utilizar los sucesos del sistema de accesibilidad en una aplicación, el cliente (300) puede hacer una de cuatro cosas: (1) utilizar el API "AddTopLevelWindowListener" para descubrir la nueva UI que aparece en el escritorio y en el manejador para ese suceso, registrarse para otros sucesos y recibir por este medio sucesos de cualquier proceso: (2) Utilizar una de las "Find APIs" (encontrar una de las Interfaces del Programa de Aplicación) para localizar la UI interesante y tener como meta una pieza específica de la UI; (3) Utilizar algún otro medio para descubrir las UI interesantes, tales como encontrar una manija de ventana o un punto en la pantalla y, utilizando esa manija o punto, adquirir un elemento lógico para utilizarlo como referencia para escuchar los sucesos; o (4) Utilizar el API "AddFocusChangedListener" para rastrear el focalizador de entrada y registrarse para sucesos en la UI que actualmente tengan el enfoque.

83
86

Una ventana de alto nivel es una ventana cuyo generador se encuentra en el escritorio. La utilización del término "abierta" indica que una nueva ventana de alto nivel ha aparecido para el usuario. La utilización del término "cerrada" indica que una ventana de alto nivel ha sido destruida.

La interfaz del costado del servidor del sistema de accesibilidad (230) incluye dos APIs para notificar al sistema de accesibilidad (200) los eventos que se están añadiendo y removiendo. El cliente de automatización de la UI llama a estas APIs cuando el cliente se registra y se retira del registro para sucesos. Esto le permite al motor de la UI (206) obtener información acerca de cuáles sucesos de accesibilidad están siendo solicitados en este contexto.

Cuando se aplique, los eventos se asocian con elementos lógicos del árbol de elementos lógicos de la aplicación (222). En donde los elementos lógicos no son aplicables, los sucesos se asocian con una sarta que pueda ser leída por los humanos o por otro objeto bien conocido que represente la fuente de un suceso. El mecanismo del suceso (210) proporciona filtración basándose en las preferencias suministradas por el usuario durante el evento de registro.

84
85

Al utilizar las preferencias de filtrado del cliente en el servidor antes de crear los datos relacionados con el suceso y de enviarlos a través del proceso al cliente, el mecanismo de sucesos (210) inherentemente mejora el desempeño fuera del proceso reduciendo el número de llamadas cruzadas del proceso. El mecanismo de sucesos (210) proporciona una forma para especificar las propiedades de un elemento lógico que sean interesantes para el suceso durante el registro del mismo. Esto reduce aún más el número de llamadas de procesamiento cruzado. El mecanismo de sucesos (210) se puede extender sin requerir mayores cambios del sistema operativo (OS). Aún cuando el mecanismo del suceso (210) se puede implementar utilizando un código manejado, las aplicaciones sin manejar pueden accederlo a través de la interoperabilidad COM.

Existen dos tareas que el cliente AT (300) realiza para manejar sucesos: (1) registro del suceso; y (2) manejo del suceso en una visita suplementaria. Un requisito importante para cualquier tarea es el de que se pueda utilizar tanto el código manejado como sin manejar. En una incorporación del invento, el cliente (300) puede pasar interfaces cuando se está registrando para sucesos. El cliente (300) pasa objetos que implementan interfaces bien conocidas cuando se registran

85
88

y el motor UI (206) hace visitas suplementarias a aquellas interfaces para notificar al oyente. En los casos en donde existe un objeto fuente en el árbol del elemento lógico (222), cada una de las visitas suplementarias recibe el elemento fuente y un parámetro adicional.

Las interfaces bien conocidas e implementadas por el producto AT se utilizan como forma para devolver sucesos. Las siguientes secciones muestran algunos tipos de sucesos; cómo se ve el registro y cómo se recibe el suceso.

1. Eventos de Ventana de Alto Nivel

Los eventos de ventana de alto nivel incluyen eventos relacionados con menús y bajadas de casillas combo o en cualquier característica que tenga al escritorio como generador. En una incorporación del invento, un método `AddTopLevelWindowListener` se utiliza para recibir notificación de las ventanas de alto nivel que están siendo abiertas y cerradas. Llamar a `AddTopLevelWindowListener` obtendrá las notificaciones para nuevas ventanas de alto nivel que se abran o para las ventanas de alto nivel que se están cerrando en el escritorio corriente. Un método `RemoveTopLevelWindowListener` proporciona un mecanismo para detener las notificaciones que se reciben para que las

86
89

ventanas de alto nivel se puedan abrir o cerrar en el escritorio. Este método utiliza un objeto de visita suplementaria para identificar a este oyente. Por consiguiente, el objeto que se pasa en el método RemoveTopLevelWindowListener es el mismo que se pasa a AddTopLevelWindowListener. Un método OnTopLevelWindowOpened es llamado por el sistema de accesibilidad (200) una vez por cada ventana nueva de alto nivel que esté abierta. Similarmente, un método OnTopLevelWindowClosed puede ser visitado por el sistema de accesibilidad (200) una vez que la ventana de alto nivel se haya cerrado. Para recibir estas notificaciones, el cliente (300) requiere el método AddTopLevelWindowListener.

2. Sucesos de Enfoque

Los clientes (300) con frecuencia requieren un método para rastrear el enfoque. Le es difícil a Microsoft Windows OS hacer esto. Por ejemplo, cuando se cae un menú (v.g. el menú Archivos en Microsoft Word) los ítems del menú obtienen el enfoque a medida que el usuario pasea con el cursor hacia abajo por todos ellos. Cuando el menú se cierra (v.g. el usuario oprime la tecla ESC) en el día de hoy no se envía ningún suceso de enfoque. En vez, el cliente interesado en

los cambios de enfoque debe escuchar una serie de eventos y conjeturar cuáles de esos lógicamente representan un cambio de enfoque. En una incorporación del invento, un método `AddFocusChangeListener` se puede utilizar para notificar al escucha sobre los sucesos de cambio de enfoque. El cliente (300) podrá especificar un objeto de propiedades que se deban devolver. Esta característica de especificar propiedades para devolver con el elemento lógico forma parte del registro para todos los eventos (APIs). El cliente (300) podrá llamar el método `RemoveFocusChangeListener` para detener el recibo de notificaciones para cambios de enfoque. Este método puede utilizar el objeto de la visita adicional para identificar a este escucha y en ese caso los objetos pasados al método `RemoveFocusChangeListener` serán los mismos que aquellos que pasaron al procedimiento `AddFocusChangeListener`. El sistema de accesibilidad (200) llama al método `OnFocusChanged` cuando el enfoque ha cambiado. Un parámetro `FocusEventArgs` utilizado en el método `OnFocusChanged` expone la información relacionada con el cambio de enfoque. Si la información acerca del último elemento lógico enfocado está disponible, entonces ese elemento estará disponible en el parámetro del suceso args en el elemento `OnFocusChanged`. Existen casos cuando ningún

88
91

elemento de la UI tiene enfoque hasta que algo explícitamente coloca enfoque en un elemento.

3. Eventos de Cambio de la Propiedad

Los eventos de cambio de la propiedad de disparan cuando las propiedades de los elementos lógicos cambian. En una incorporación del invento, un cliente (300) visita un método `AddPropertyChangedListener` para recibir notificaciones de cambios de la propiedad. Un método `OnPropertyChanged` es llamado por el sistema de accesibilidad (200) cuando el valor de una propiedad ha cambiado en un elemento lógico dentro del árbol del elemento lógico especificado en el `AddPropertyChangedListener`. Un parámetro de alcance indica para cuáles elementos se debe activar un suceso. Por ejemplo, pasar el elemento lógico de la raíz de una ventana y una solicitud para los descendientes limita las nuevas visitas de cambio de propiedad a esa ventana. Si el parámetro del alcance está fijado para todos los elementos en el árbol, entonces el alcance se ignora y los cambios a las propiedades especificadas que ocurren en el escritorio se despachan. Un cliente (300) puede llamar muchas veces al `AddPropertyChangedListener` con diferentes conjuntos de propiedades y/o un objeto de retrodemanda diferente. La

notificación que proporciona el sistema de accesibilidad (200) indica: la propiedad que cambió; el nuevo valor de la propiedad; y el valor antiguo de la propiedad, si se dispone de él.

Un cliente (300) puede llamar al método `RemovePropertyChangedListener` para no seguir recibiendo notificaciones de cambios a la propiedad. Este método puede utilizar el elemento del alcance y el objeto de retrodemanda para identificar este oyente y en este caso los objetos pasados deben ser los mismos que los pasados al `AddPropertyChangedListener`.

4. Sucesos de los Patrones de Controles

Los sucesos activados desde los patrones de controles necesitan poderse extender y por consiguiente estos sucesos son identificados por un GUID. Cualquier valor GUID es aceptado al registrarse. Para cualquier nuevo patrón de control, los sucesos que éste documenta necesitan ser GUIDs únicos. Se puede necesitar modificar los productos AT para escuchar nuevos sucesos de patrón de control. El oyente también necesita poder observar estos sucesos. Por ejemplo, las pruebas dirigidas pueden desear limitar estos eventos a una aplicación particular o a un control dentro de una

90
93

aplicación. Los patrones de controles definen cuál es la fuente y los consumidores de los sucesos necesitarán referirse a esa parte de la documentación con el fin de saber cómo utilizar un elemento fuente y un objeto argumento de un suceso.

El método `AddEventListener` permitirá que el cliente (300) reciba sucesos para controles. El parámetro del ámbito se puede utilizar para indicar para qué elementos activar sucesos. Por ejemplo, pasar el elemento lógico de la raíz de una ventana y solicitar descendientes limitará los sucesos a esa ventana. Si todos los elementos en un árbol se desean, entonces todos los sucesos del escritorio se enviarán. El método `RemoveEventListener` se puede utilizar para dejar de recibir un suceso para controles. Este método puede utilizar el elemento del ámbito, el objeto de la retrodemanda y un identificador de sucesos para identificar al oyente. En este caso, los objetos pasados deben ser los mismos que los que se pasaron a `AddEventListener`.

Cuando una aplicación ha invocado el método `AddEventListener`, el método `OnEvent` es llamado por el sistema de accesibilidad (200) cuando un suceso específico de control se activa y la fuente del suceso es un elemento lógico dentro

94

del árbol de elementos lógicos identificado en el `AddEventListener`. Cuando los sucesos para controles se activan con frecuencia hay información específica de sucesos disponible.

Un método `RemoveAllListener` puede ser llamado para dejar de recibir cualesquiera sucesos. Esta es una forma rápida de depurar antes de que la aplicación del cliente se cierre. El método de remoción puede utilizarse óptimamente cuando se termina una aplicación.

5. Sucesos de Cambio de la Estructura Lógica

Los sucesos de cambio de la estructura lógica se activan cuando cambia la estructura del árbol del elemento lógico. Un método `AddLogicalStructureChangedListener` se puede implementar para recibir notificaciones para cambios estructurales en el árbol de elementos lógicos. Cuando se añaden, remueven o invalidan elementos lógicos, se llaman los métodos en el objeto de la retrodemanda especificada. El parámetro del ámbito puede limitar los elementos en la forma arriba indicada. Un método `RemoveLogicalStructureChangedListener` se puede llamar para detener el recibo de sucesos para cambios del árbol de elementos lógicos. Este método puede utilizar un objeto de

92
95

retrodemanda y un elemento de ámbito para identificar este oyente y en este caso los objetos pasados deben ser los mismos que aquellos pasados al AddEventListener.

Un método OnChildAdded puede ser llamado por el sistema de accesibilidad (200) cuando un elemento descendiente se añade y el generador es un elemento lógico dentro del árbol de elementos lógicos (222) especificado en AddLogicalStructureChangeListener. Un método OnChildRemoved es llamado por el sistema de accesibilidad (200) cuando el elemento descendiente se remueve y el generador antiguo es un elemento lógico dentro del árbol de elementos lógicos especificado en AddLogicalStructureChangeListener.

Un método OnChildrenBulkAdded es llamado por el sistema de accesibilidad cuando se añade un gran número de descendientes (v.g. superior a 20 descendientes) dentro de un periodo de tiempo corto. Un método OnChildrenBulkRemoved es llamado por el sistema de accesibilidad cuando un gran número de descendientes se remueven dentro de un periodo de tiempo corto. Un método OnChildrenInvalidated es llamado por el sistema de accesibilidad (200) cuando un gran número de descendientes (v.g. más de 20) son tanto añadidos como removidos dentro de un periodo de tiempo corto.

6. Eventos de Multimedia

Otro tipo de suceso es el suceso multimedia. Multimedia puede incluir sonido, video y animación. Los métodos soportarán los sucesos de multimedia y notificarán a un cliente las acciones incluyendo "detenido", "pausa", "avance rápido", "retrobobinado", y "mudo". Se pueden implementar métodos similares a los arriba descritos para añadir y remover oyentes de multimedia.

7. Sucesos Simples de Sonido

Los sucesos simples de sonido se pueden manejar por separado de los sucesos de multimedia. Los sucesos simples de sonido representan sonidos simples de corta duración que existen para traspasarle al usuario algún suceso distinto del sonido mismo. Los eventos simples de sonido pueden incluir: el sonido que se oye cuando ha llegado un nuevo correo; el sonido generado cuando la batería de un computador portátil está baja; o el sonido que se oye cuando se exhibe una casilla de mensajes con el tipo IconExclamation. Un método `AddSoundListener` puede ser llamado para recibir una notificación de los sonidos simples que se están sucediendo y un método `RemoveSoundListener` se puede implementar para dejar de recibir las notificaciones de sucesos de sonido simple.

94
97

Un método OnSound es llamado por el sistema de accesibilidad (200) cuando se ha escuchado un sonido simple. Para recibir esta notificación, la aplicación de escucha llama al AddSoundListener. El método OnSound recupera la siguiente información: el nombre del sonido; la fuente del sonido, y un valor del nivel de alerta que indica la importancia del sonido para el usuario. Los posibles niveles de alerta incluyen: "desconocido", lo que indica que la importancia no se conoce; "de información", lo que indica que la información se ha presentado; "precaución", lo que indica que existe una condición de precaución; "pregunta", lo que indica que se requiere una respuesta del usuario; "exclamación", lo que indica que el suceso no es crítico pero puede ser importante; y "crítico", lo que indica que ha ocurrido un suceso crítico.

8. Suceso de Enfoque Difuso

Los eventos de enfoque difuso aparecen en el escritorio pero se quedan en el trasfondo. Algunos ejemplos de sucesos de enfoque difuso son: ventana de ayuda del círculo de identificación que indica que "Nuevas actualizaciones que están disponibles" en el área de notificaciones; un ícono con destello en la barra de tareas para una aplicación de

95
98

trasfondo que desea cobrar enfoque; y un ícono de impresora que aparece y desaparece de la bandeja de notificaciones cuando se inicia y se acaba la impresión. Puede parecer que estos sucesos se traslapan algo con otras categorías de sucesos (multimedia puede involucrar sucesos de animación como lo hace el enfoque difuso). Sin embargo, el suceso se categorizará basándose en lo que éste conlleva para el usuario más bien que en cómo se transporta.

Un método `AddSoftFocusListener` se puede implementar para recibir notificación de sucesos que tratan de llamar la atención del usuario sin tomar el enfoque de entrada. Un método `RemoveSoftFocusListener` detienen la notificación. Este método puede utilizar el objeto de retrodemanda para identificar este oyente y por consiguiente el objeto pasado debe ser el mismo que el que se pasó a `AddSoftFocusListener`.

Un método `OnSoftFocus` puede ser llamado por el sistema de accesibilidad (200) cuando ha ocurrido un suceso de enfoque difuso. Para recibir esta notificación, la aplicación de escucha o el cliente (300) visita a `AddSoftFocusListener`. Un elemento de fuente, si se dispone de él, se puede utilizar para obtener más información acerca del suceso. Un elemento fuente ejemplar sería el elemento de raíz lógica de una

ventana de ayuda del círculo de identificación utilizada por una de las aplicaciones de notificación en la bandeja del sistema. El método OnSoftFocus devuelve: el nombre del suceso, la fuente del suceso, y el nivel de alerta, indicando la importancia para el usuario.

El siguiente cuadro ilustra las acciones del cliente (300) y del sistema de accesibilidad (200) cuando el cliente (300) utiliza una API AddTopLevelWindowListener para escuchar sucesos de un proceso específico.

Cliente	Sistema de accesibilidad con la ventana de la UI meta.
Visitas al AddTopLevelWindowListener	• El cliente del sistema de accesibilidad inicia una secuencia para observar las ventanas de la aplicación del nivel superior que se están creando y destruyendo. • Aparece la UI nueva y el cliente del sistema de accesibilidad llama al método OnTopLevelWindowOpened del cliente.
Del método OnTopeLevelWindowOpened visita otras APIs para registrarse para sucesos adicionales que ocurran en la ventana blanco de la UI	• El cliente del sistema de accesibilidad comunica la identidad del suceso a la ventana blanco de la UI de manera que ésta pueda ser selectiva en la notificación de sucesos. • El cliente del sistema de accesibilidad comunica la identidad del suceso y filtra información para el servidor del sistema de accesibilidad de manera que pueda además filtrar los sucesos.

9/10
100

	• La ventana blanco de la UI utiliza el API del servidor del sistema de accesibilidad para notificar al costado del servidor de dicho sistema sobre los sucesos de interés.
Maneja los sucesos en los objetos de retrodemanda	• El servidor del sistema de accesibilidad comunica los sucesos de nuevo al cliente del sistema de accesibilidad. • El cliente del sistema de accesibilidad llama de nuevo a los objetos que la aplicación del cliente suministró con el registro.
Los APIs Calls RemoveListener detendrán el recibo de sucesos.	• El cliente del sistema de accesibilidad le comunica al servidor del sistema de accesibilidad y a la ventana de la UI blanco los sucesos que ya no son de interés. • La aplicación . La aplicación blanco deja de notificarle los sucesos al servidor del sistema de accesibilidad.

Notificación de Sucesos

El servidor (400) y el motor de la UI subyacente utilizan los métodos correspondientes para la notificación de sucesos para sustentar los sucesos del sistema de accesibilidad arriba enumerado. Los APIs del servidor de automatización de la UI incluyen métodos que el servidor o el motor de la UI subyacente pueden visitar para lograr esto. Por ejemplo, existe un método NotifyPropertyChanged para que el servidor llame a notificar cuando ha ocurrido un cambio

98
101

particular en la propiedad con respecto al elemento lógico. Es el servidor (400) o el motor subyacente de la UI quienes deben decidir o no si generan los parámetros apropiados y visitan estos métodos de notificación cuando la Interfaz del Usuario cambia.

Métodos del Servidor

Un método `AdviseEventAdded` y un método `AdviseEventRemoved` son llamados por el cliente de automatización de la UI para notificar al servidor (400) cuando los clientes (300) están solicitando sucesos. Esto le permite al servidor (400) no propagar los sucesos al sistema de accesibilidad (200) cuando no haya nadie interesado. Los servidores pueden utilizar estas notificaciones para que el desempeño dependa de si hay o no clientes utilizando los sucesos.

Patrones de Control

El modelo de accesibilidad ofrece un enfoque único para categorizar y exponer la funcionalidad sustentada por un elemento o control particular de la UI. En vez de asociar la funcionalidad con el tipo de control específico, (por ejemplo un botón, una casilla de edición, o una casilla de listas)

94
102

tal como en la técnica anterior, el modelo de accesibilidad define un conjunto de patrones de control comunes que definen cada uno un aspecto del comportamiento de la Interfaz del Usuario. Como estos patrones dependen el uno del otro, se pueden combinar para describir la totalidad del conjunto de funcionalidad sustentado por un elemento particular de la Interfaz del Usuario.

Por ejemplo, en vez de describir un elemento en términos de su nombre de clase o categoría, tal como Botón, el sistema de accesibilidad (200) lo describe como que soporta el patrón de control invocable. Un patrón de control define la estructura, propiedades, sucesos y métodos sustentados por un elemento. Por consiguiente, estos patrones no solamente le permiten al cliente indagar un comportamiento del control; ellos también le permiten manipular programáticamente el control utilizando interfaces diseñadas para un patrón particular. Por ejemplo, un patrón SelectionContainer proporciona métodos para indagar los ítems seleccionados, para seleccionar o desechar un ítem específico o para determinar si el control sustenta modos de selección simples o múltiples.

Los patrones de control actualmente definidos para el sistema de accesibilidad (300) incluyen: (1) Contenedor de Selección; (2) Jerarquía; (3) Invocable; (4) Rejilla Simple; (5) Texto; (6) Valor; (7) Representa Objeto; (8) Rodante; (9) Clasificable; (10) Dibujo; y (11) Otro Contenedor.

Esta técnica permite a los desarrolladores de control implementar un nuevo tipo de control mientras siguen teniendo un enfoque bien definido para exponer su comportamiento a los productos AT y a los programas instructores de prueba (test scripts). Si se introduce un nuevo tipo de comportamiento, un nuevo patrón de control se puede definir para expresar la funcionalidad requerida.

Los productos de tecnología de ayuda o asistiva y los programas instructores de prueba pueden ahora escribirse para entender cómo trabajar con cada patrón en vez de con cada control de la UI. Porque como hay mucho menos patrones de control o clases de control, esta técnica minimiza el código necesario. Este enfoque también fomenta una arquitectura más flexible que pueda interrogar y manipular en forma eficaz los nuevos controles (mientras que ellos soporten patrones de control conocidos).

La siguiente tabla proporciona algunos ejemplos de controles comunes y los patrones que ellos sustentan.

Controles	Patrones de Control Pertinente
Botón	Invocable
Casilla de Chequeo, Radiobotón	Valor
Casilla de Listas	Selección Contenedor, Rodante
Casilla de Combo de Correo	Selección Contenedor, Rodante, valor
Vista del Árbol	Selección Contenedor, Rodante, Jerarquía
Vista de la Lista	Selección Contenedor, Rodante, Clasificable
Casilla de Texto, Edición	Valor, Texto, Rodante

Se utilizarán interfaces más específicas para exponer la funcionalidad asociada con los patrones de control común. Los ejemplos de estos patrones incluyen: (1) contenedores de selección administrativa; (2) contenedores trazados con rejilla; (3) elementos de la UI que contienen valores; (4) íconos que representan objetos (archivos, correos electrónicos, etc.); y (5) elementos de la UI que se pueden invocar. En general, estos patrones no están estrechamente obligados a controles específicos y los controles diferentes pueden implementar los mismos patrones. Por ejemplo, las cajillas de listas, las cajas combo de correo, las vistas del árbol todos implementan el patrón de "recipiente que administra la selección". Algunos controles pueden implementar patrones múltiples si es apropiado: una rejilla de selección implementaría tanto el patrón de "Contenedor con

trazado de rejilla" como el patrón de "Contenedor para el manejo de la selección" (Selection managing container).

No existe un solo "papel" de la propiedad como en las aplicaciones previas. En vez, se utilizan dos mecanismos separados. Los patrones de control determinan la funcionalidad disponible de un control y una propiedad localizable que pueda ser leída por humanos proporciona un nombre de tipo control que el usuario puede entender, tal como "botón", "casilla de listas", etc.

Propiedades

El sistema de accesibilidad (200) representará el método general GetProperty. Las propiedades preferiblemente son representadas por GUIDs, con los métodos de utilidad utilizados para traducir de aquí para allá y de allá para acá de la forma no localizable nemónica (útil para el archivo de instrucciones y para los archivos de configuración) y también para obtener descripciones localizadas. Las dos ventajas clave de un método general GetProperty en vez de los métodos individuales radica en que (a) permite que se añadan nuevas propiedades con el transcurso del tiempo sin cambiar la interfaz, y (b) permite las técnicas de implementación -tales como búsqueda de propiedades en volumen impulsada por series-

103
106

que no son posibles cuando se utilizan métodos separados. Cada propiedad debe tener una intención claramente definida. Si la propiedad tiene la intención de ser para consumo humano o mecánico, o si ésta debe localizarse, etc., es algo que debe definirse claramente.

El presente invento se ha descrito en relación con incorporaciones particulares que en todo aspecto tienen la intención de ser ilustrativas y no restrictivas. Incorporaciones alternativas serán evidentes para quienes conocen la técnica a la que pertenece este invento sin apartarse de su ámbito.

De lo antedicho se desprende que este invento es un documento bien adaptado para lograr todas las finalidades y objetos arriba establecidos, junto con otras ventajas que son obvias e inherentes para el sistema y método. Se entenderá que determinadas características y subcombinaciones son de utilidad y pueden ser empleadas sin referencia a otras características y subcombinaciones. Esto se contempla dentro del alcance de las reivindicaciones.

104
107

MECANISMOS Y MÉTODOS PARA SUCESOS DE UN
SISTEMA DE ACCESIBILIDAD

REIVINDICACIONES

1. Un mecanismo de sucesos dentro de un ambiente de cliente-servidor para mantener a un cliente notificado sobre los cambios en la interfaz del usuario, mecanismo de sucesos éste que comprende: un mecanismo de registro para permitir que un cliente se registre para recibir información interesante de la interfaz del usuario; un cliente de automatización de la interfaz del usuario para recibir información de registro del cliente y para traspasar la información interesante de la interfaz del usuario al cliente cuando se reciba, en donde el cliente de automatización de la interfaz del usuario utiliza la información del registro para notificar al servidor de la interfaz del usuario; y un servidor de automatización de la interfaz del usuario para recibir información de registro del cliente de automatización de la interfaz del usuario y para recibir la información de la interfaz del usuario proveniente de un motor de interfaz del usuario, en donde el servidor de automatización de la interfaz del usuario incluye un dispositivo de filtro para filtrar la información que no es de interés para el cliente y

1521

105
108

un dispositivo de notificación para notificar al cliente de la automatización de interfaz del usuario la información que sea interesante para el cliente.

2. El mecanismo del suceso de la reivindicación 1 en donde el mecanismo de registro acepta la información del registro del cliente, información de registro ésta que incluye: información del objeto; una localización del cliente para la información del objeto; y una lista de propiedades para ser devuelta con el objeto fuente.

15e1

3. El mecanismo de sucesos de la reivindicación 2 en donde el cliente de automatización de la interfaz del usuario comprende medios para entregar la información del objeto a la localización del cliente especificada.

4. El mecanismo de sucesos de la reivindicación 2, en donde el medio de filtrado del servidor de automatización de la interfaz del usuario determina si la información de la interfaz del usuario proveniente del motor de la interfaz del usuario incluye la información del objeto y la lista de propiedades requeridas.

5. El mecanismo de sucesos de la reivindicación 1, en donde el mecanismo de registro recibe una solicitud para un ámbito de información específico.

106
109

6. El mecanismo de sucesos de la reivindicación 5, en donde el servidor de automatización de la interfaz del usuario forma un elemento lógico si la información de la interfaz del usuario es interesante para el cliente y está dentro del ámbito especificado y descarta la información de la interfaz del usuario si ésta no es interesante o no está dentro del ámbito especificado.

7. El mecanismo de sucesos de la reivindicación 1, en donde la notificación del servidor de automatización de la interfaz del usuario proveniente del motor de la interfaz del usuario incluye: un nombre de una propiedad cambiada; un nuevo valor; y cualquier valor antiguo disponible.

8. El mecanismo de sucesos de la reivindicación 1, en donde la información de la interfaz del usuario comprende información relacionada por lo menos con un tipo de suceso relacionado entre el grupo de: sucesos de la ventana de nivel superior; sucesos de enfoque; sucesos de cambio de propiedad; sucesos de patrones de control; sucesos de cambio de estructura lógica; sucesos de multimedia; sucesos de sonido simple; y sucesos de enfoque difuso.

167
110
9. El mecanismo de sucesos de la reivindicación 8, en donde cada suceso recae solamente dentro de uno de los tipos de sucesos.

10. El mecanismo de sucesos de la reivindicación 8, en donde cada tipo de suceso incluye un conjunto de métodos para añadir un oyente, remover un oyente, apuntar el mecanismo de sucesos para escuchar el tipo de suceso, y para que el motor de la interfaz del usuario notifique al servidor de automatización de la UI sus sucesos.

11. Un método implementado por computador para notificar a un cliente sucesos interesantes de la interfaz del usuario, método éste que comprende: recibir una solicitud de registro del cliente para información especificada de la interfaz del usuario; monitorear la información del motor de la interfaz del usuario proveniente de un servidor de automatización de la interfaz del usuario para determinar si la información de la interfaz del usuario especificada está disponible; pasar la información de la interfaz del usuario especificada desde el servidor de automatización de la interfaz del usuario hasta un cliente de automatización de la interfaz del usuario; y entregar la información de la interfaz del usuario especificada desde el cliente de

158

108
111

automatización de la interfaz del usuario hasta una localización adecuada del cliente.

12. El método de la reivindicación 11, en donde recibir una solicitud de registro comprende recibir información de registro que incluye: información del objeto; una localización del cliente para la información del objeto, y una lista de las propiedades requeridas.

13. El método de la reivindicación 12, en donde entregar la información del usuario especificada comprende entregar la información del objeto a la localización del cliente especificada.

14. El método de la reivindicación 12, que comprende además filtrar la información de la interfaz del usuario con el servidor de automatización de la UI para determinar si la información de la interfaz del usuario proveniente del motor de la interfaz del usuario incluye la información del objeto y la lista de propiedades requeridas.

15. El método de la reivindicación 11, en donde recibir una solicitud de registro comprende recibir una solicitud para un ámbito específico de información.

109
D112

16. El método de la reivindicación 15, que comprende además formar un elemento lógico con el servidor de automatización de la interfaz del usuario si la información de la interfaz del usuario es interesante para el cliente y está dentro del ámbito especificado y descartar la información de la interfaz del usuario si ésta no es interesante o no está dentro del ámbito especificado.



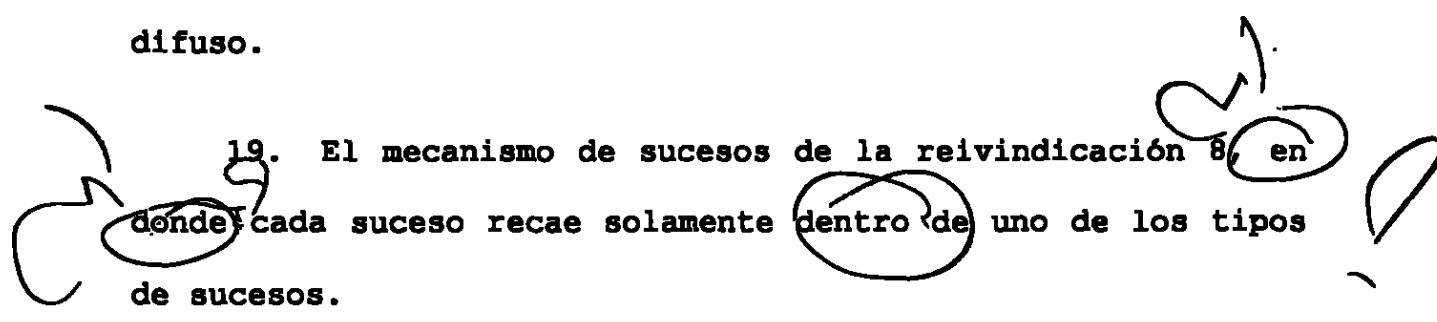
17. El método de la reivindicación 11, que comprende además recibir información del motor de la interfaz del usuario que incluye: un nombre de una propiedad cambiada; un nuevo valor; y cualquier valor antiguo disponible.



18. El método de la reivindicación 11, que comprende además recibir información de la interfaz del usuario relacionada por lo menos con un tipo de suceso seleccionado entre el grupo: sucesos de ventana de alto nivel; sucesos de enfoque; sucesos de cambio de propiedad; sucesos de patrones de control; sucesos de cambio de estructura lógica; sucesos de multimedia; sucesos de sonido simple; y sucesos de enfoque difuso.



19. El mecanismo de sucesos de la reivindicación 8, en dónde cada suceso recae solamente dentro de uno de los tipos de sucesos.



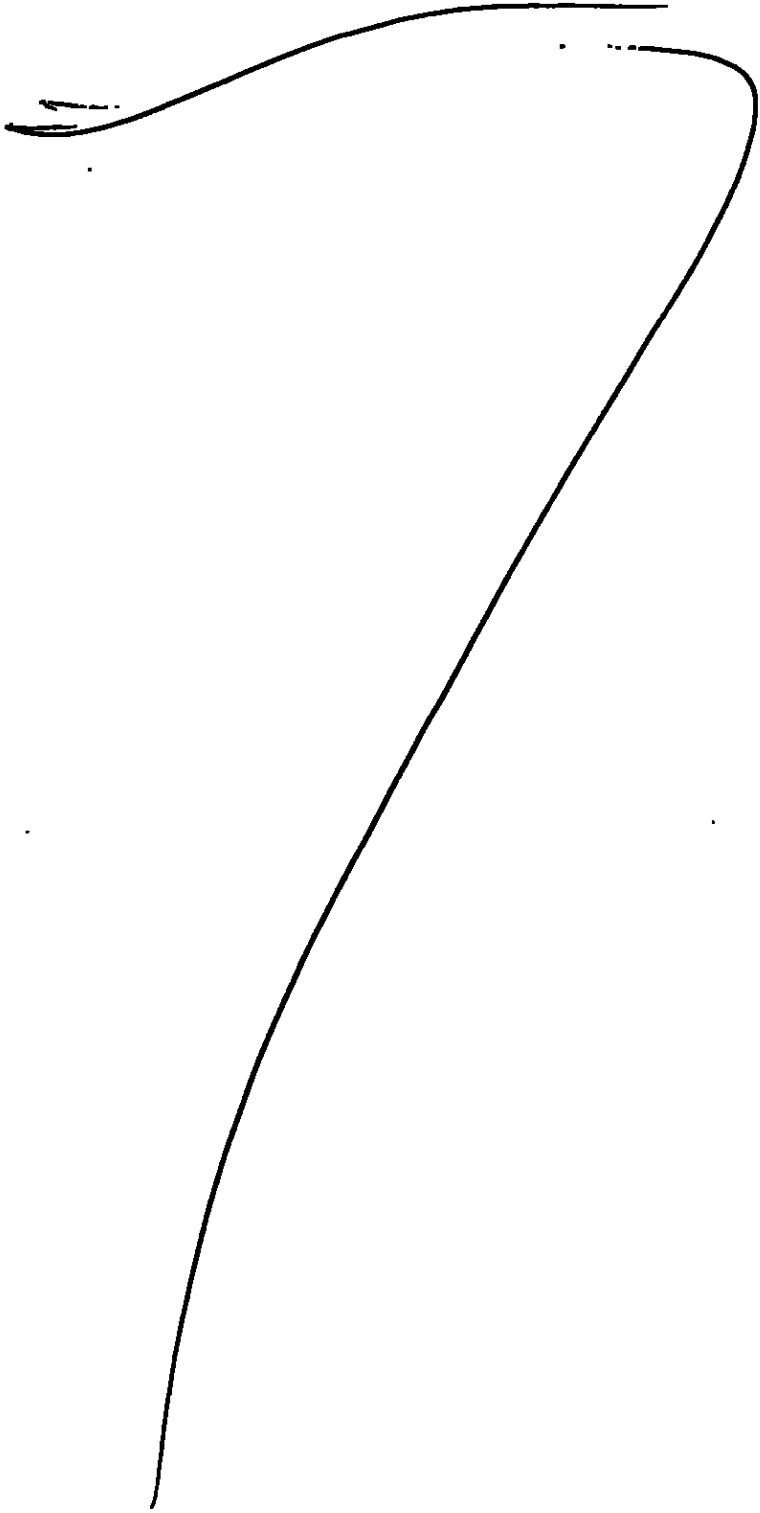
RESUMEN DE LA REVELACIÓN

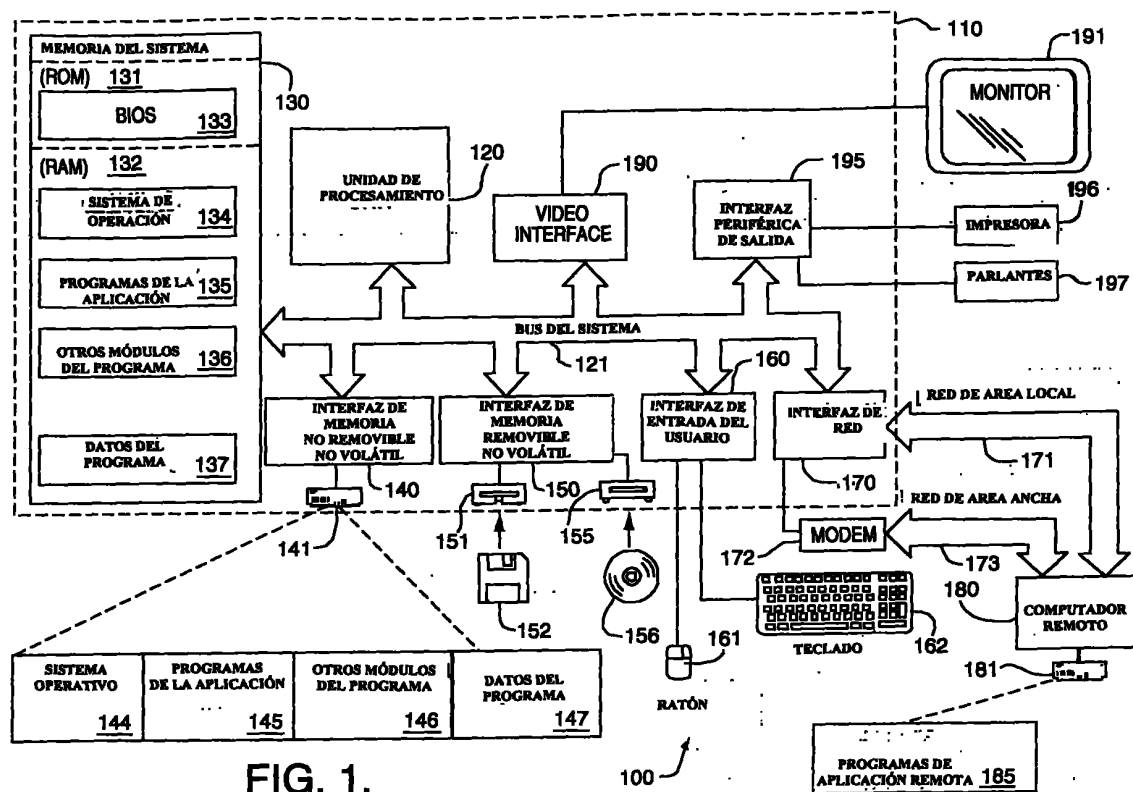
Se describe un método y sistema para proporcionarle información a un cliente con interfaz del usuario. Un mecanismo de sucesos dentro de un ámbito de servidor-cliente para mantener a un cliente notificado de los sucesos de la interfaz del usuario. El mecanismo de sucesos comprende un mecanismo de registro para permitirle al cliente registrarse para recibir información interesante de la interfaz del usuario. Un cliente de automatización de la interfaz del usuario recibe información de registro del cliente y transporta la información interesante de la interfaz del usuario al cliente cuando la recibe. Un servidor de automatización de la interfaz del usuario recibe la información del registro proveniente del cliente de automatización de la interfaz del usuario y notifica a un motor de la interfaz del usuario sobre el registro. El servidor de automatización de la interfaz del usuario adicionalmente recibe la información de la interfaz del usuario proveniente del motor de la interfaz del usuario. El servidor de la automatización de la interfaz del usuario incluye un dispositivo de filtrado para filtrar la información que no sea interesante para el cliente y un dispositivo de notificación para notificarle al cliente de

114

114

automatización de la interfaz del usuario la información que es de interés para el cliente.





116

FIG. 2

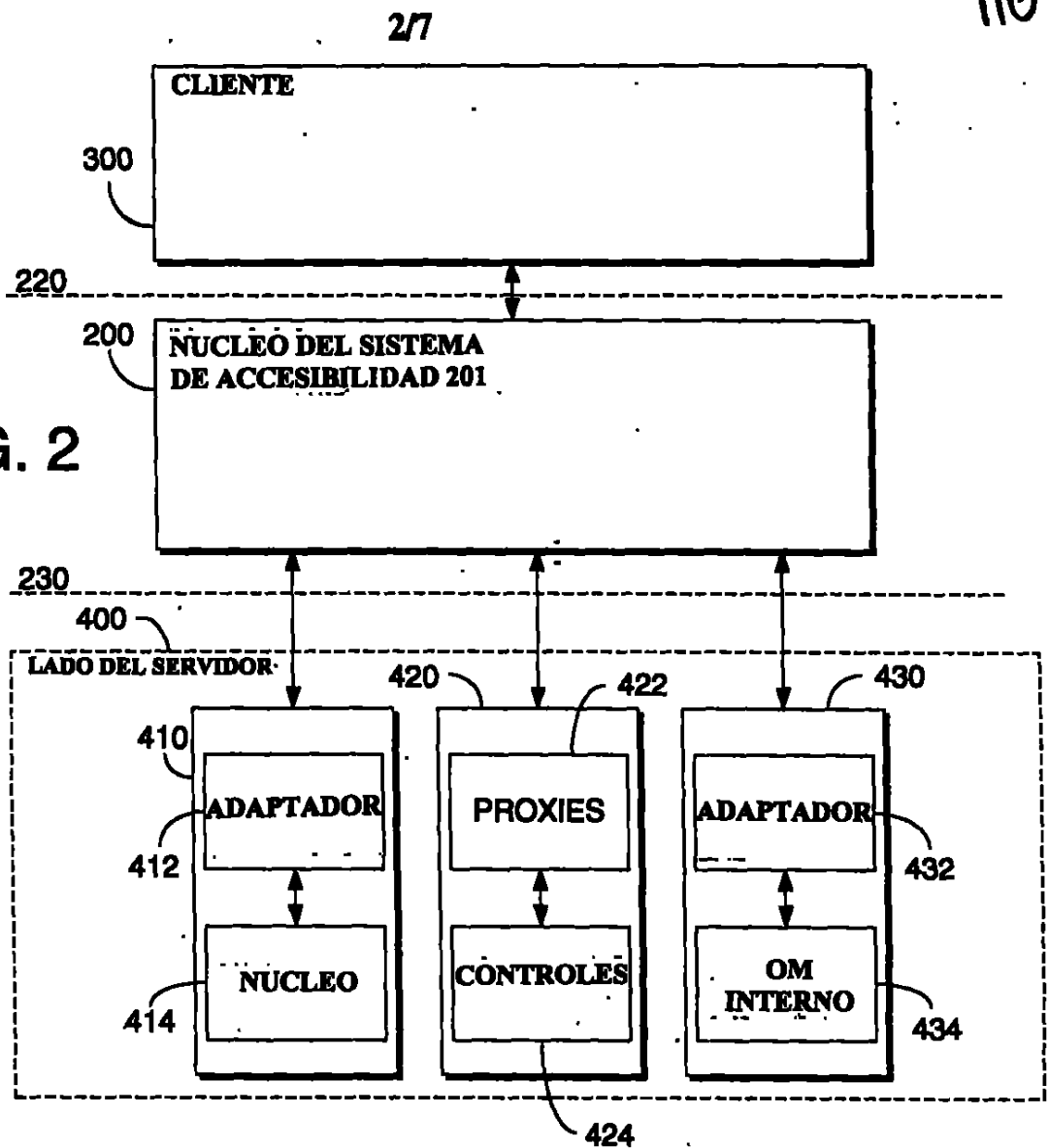
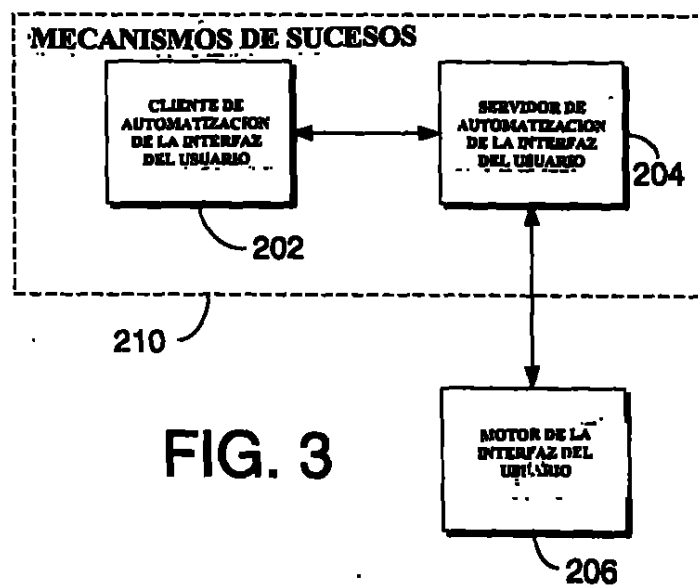


FIG. 3



114
117

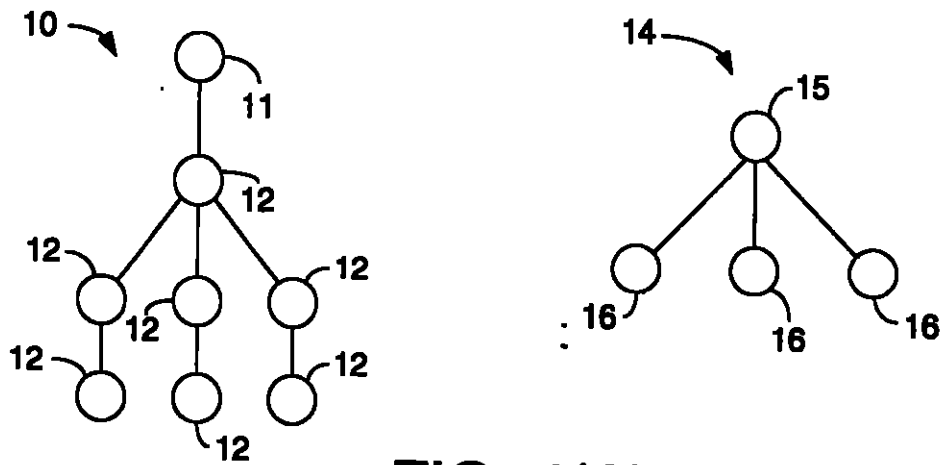


FIG. 4(A)

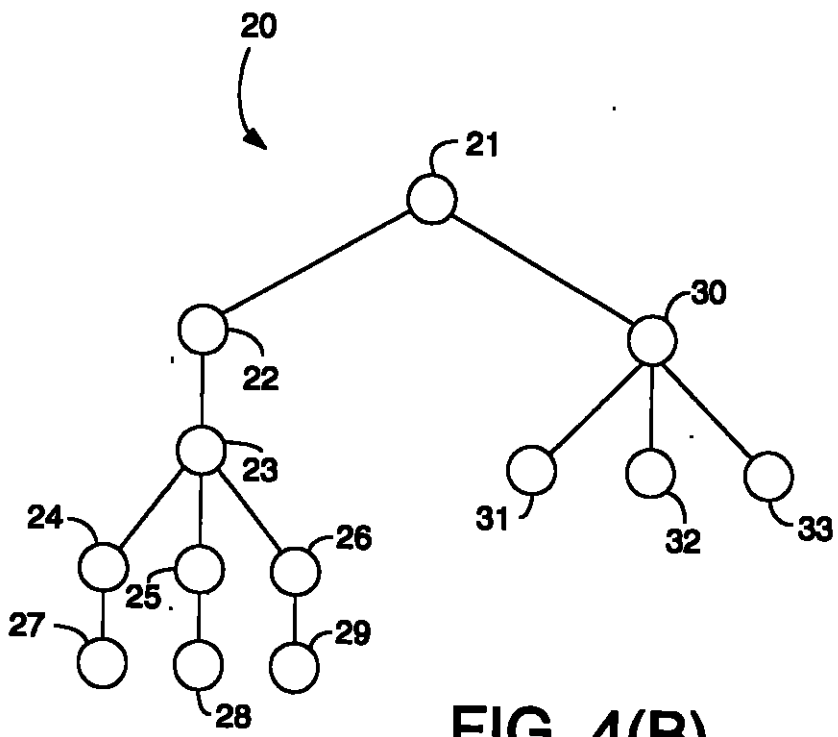
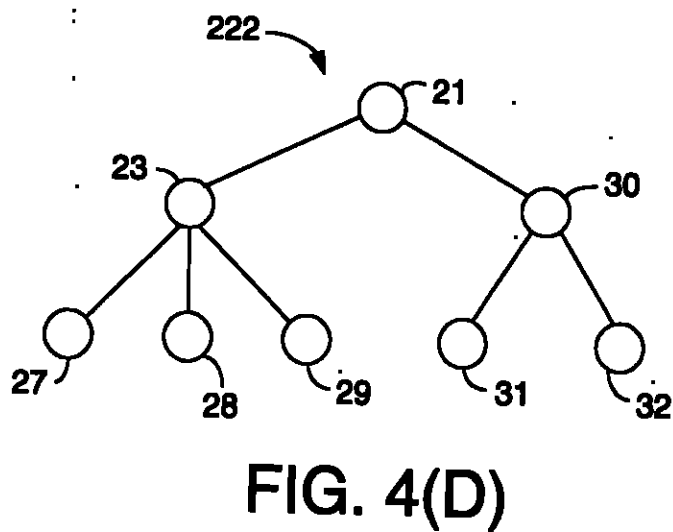
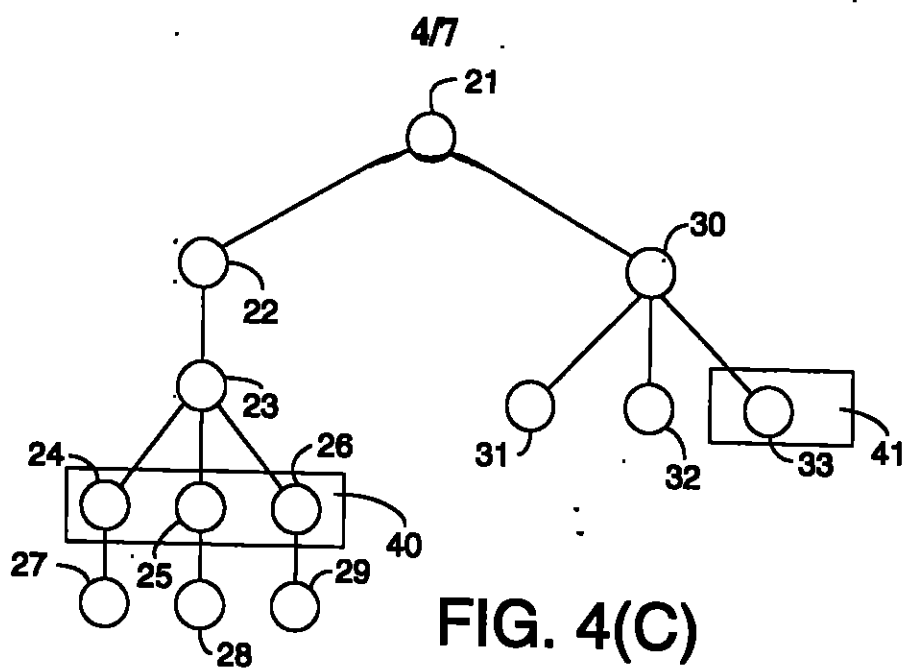


FIG. 4(B)

45
118



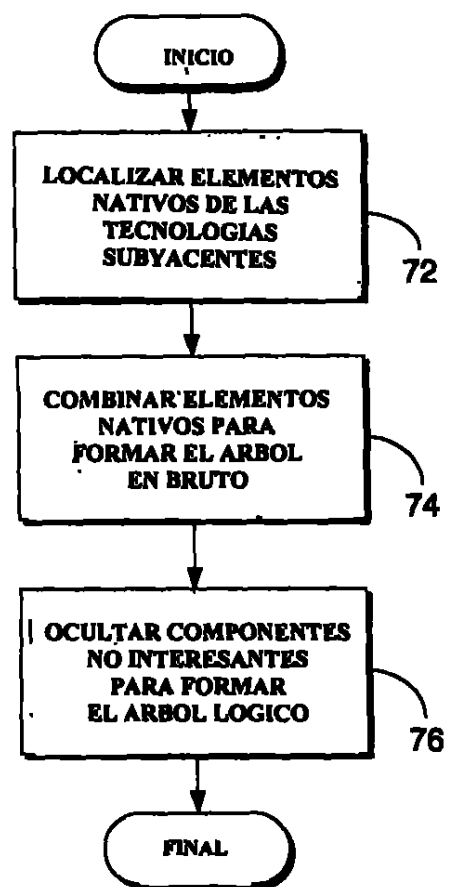


FIG. 5

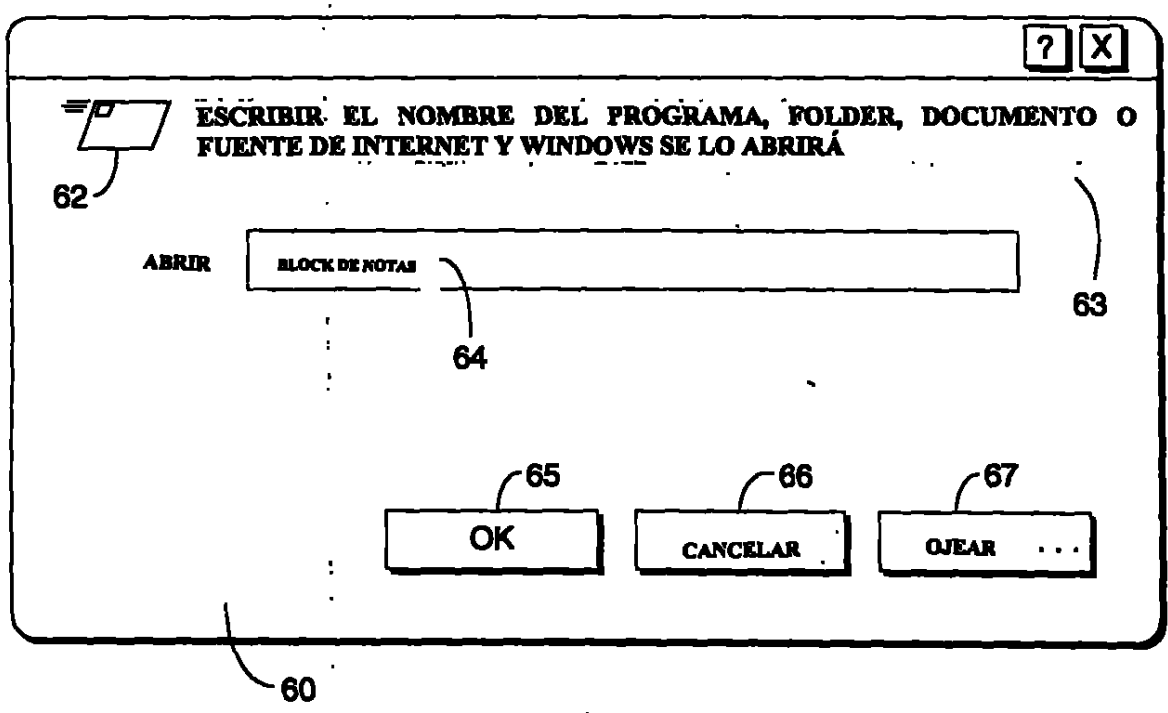


FIG. 6

46
120

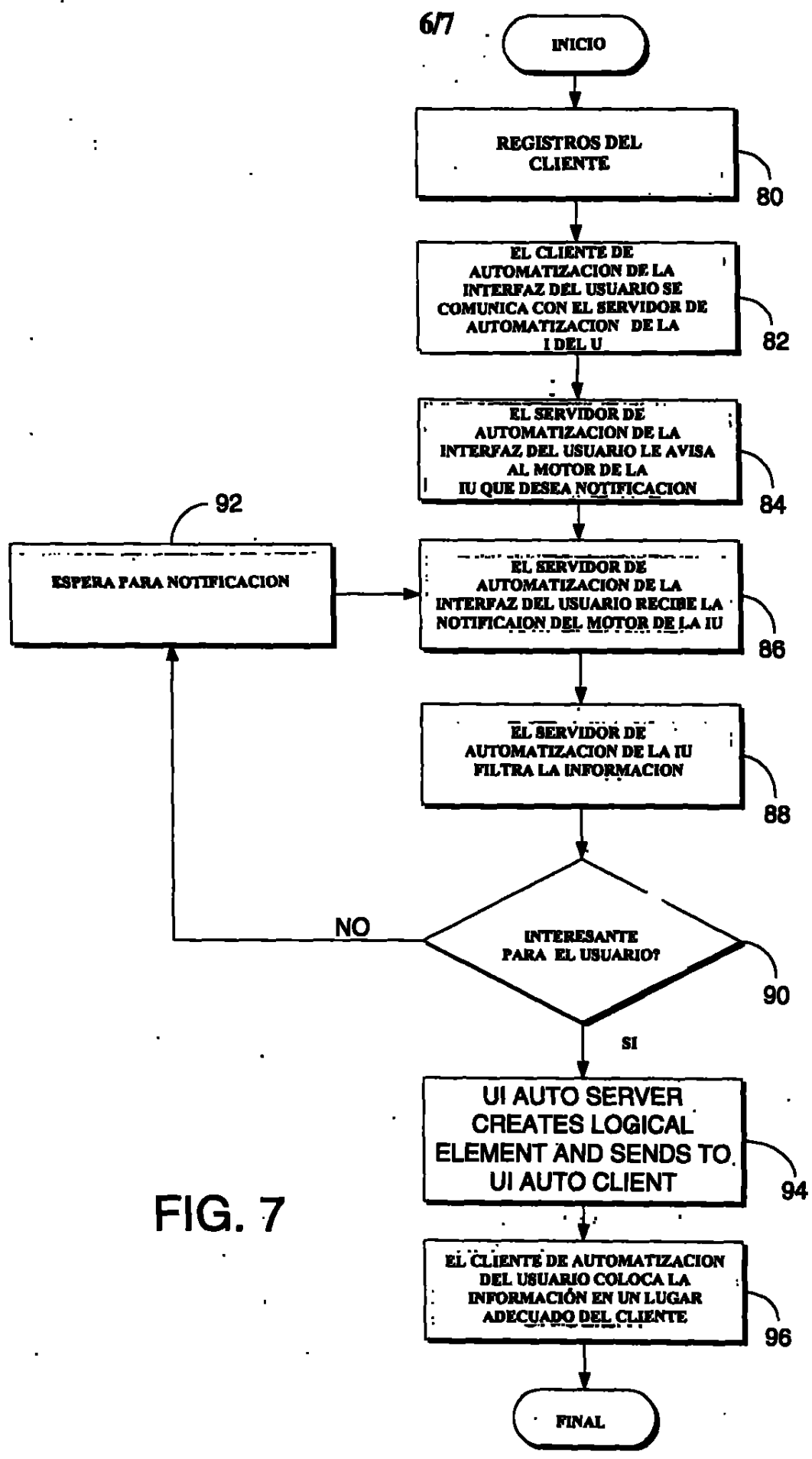
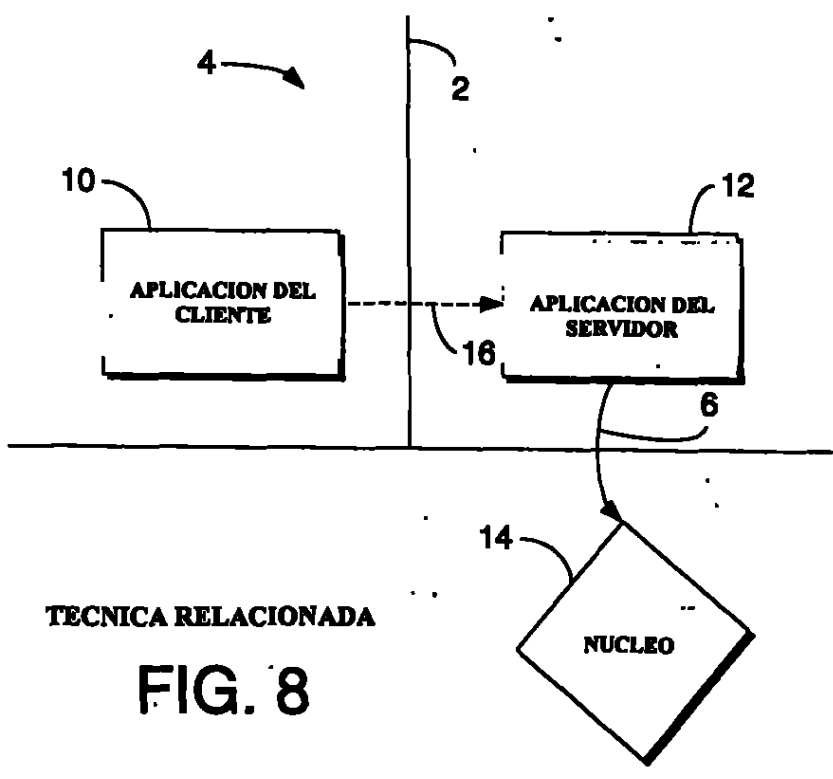


FIG. 7

212
121



TECNICA RELACIONADA

FIG. 8

#18

122

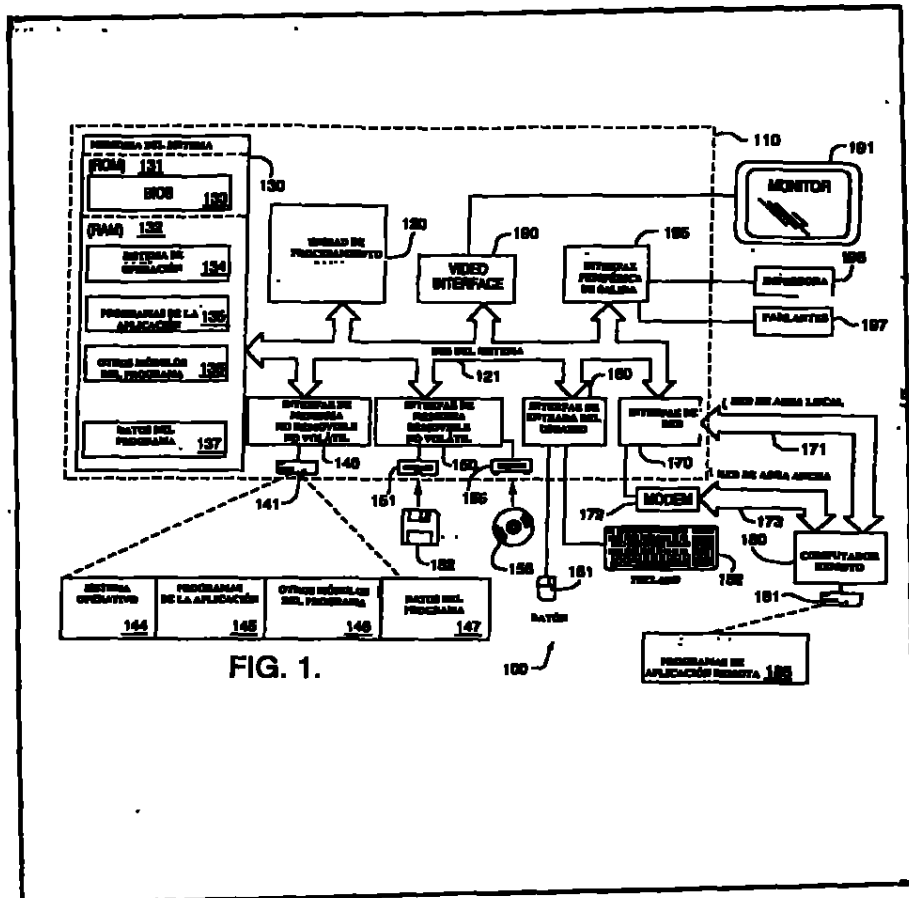


FIG. 1.

~~119~~

123

117E F11116
12x12

122
124

REQUISITOS NECESARIOS PARA PRESENTACION Y AGILIZACION DEL TRAMITE

	USUARIO	RADICADOR
PATENTE DE INVENCION ()	()	()
- Indicación de que se solicita la concesión de una patente.	()	()
- Datos de identificación del solicitante o de la persona que se presenta la solicitud.	()	()
- Descripción de la invención.	()	()
- Dibujos de ser estos pertinentes.	()	()
- Comprobante de Pago de las tasas establecidas.	()	()
COMPLETA ()	INCOMPLETA ()	()
DISEÑO INDUSTRIAL ()		
- Indicación de que se solicita el registro de Diseño Industrial.	()	()
- Datos de identificación del solicitante o de la persona que presenta la solicitud.	()	()
- Representación gráfica y fotográfica del Diseño Industrial o muestra del Material que incorpora el diseño.	()	()
- Comprobante de pago de las tasas establecidas	()	()
COMPLETA ()	INCOMPLETA ()	()
ESQUEMA DE TRAZADO ()		
- Indicación de que solicita el registro de un Esquema de trazado.	()	()
- Datos de identificación del solicitante o de la persona que presenta la solicitud.	()	()
- Representación gráfica del esquema de trazado	()	()
- Comprobante de pago de las tasas establecidas.	()	()
COMPLETA ()	INCOMPLETA ()	()

Firma Responsable:

ANDREA POSE

Fecha:

PROTOCOLO

SUPERINDUSTRIA Y COMERCIO SISTEMA DE REGISTRO

Fecha 29/04/2003

MANTENIMIENTO DE PROTOCOLOS

|Numero : 16069

|Fecha : 04/08/1995

|Conferido: MICROSOFT CORPORATION

|Pais : US ESTADOS UNIDOS DE AMERICA

|Ciudad : 234 REDMOND, WASHINGTON |

|R present: S Sustituir: S Desistir: S

|No. Radicacion: 95 34920 . Clase: 0 Seev: 0 Seac: 0 |

	scncia	Codigo	Ctr	Nombre
1	77	A		CARDENAS NAVAS DARIO
2	3653			CARDENAS CABALLERO EDUARDO
3	5048			CARDENAS MARTINEZ BERNARDO
4	1170			VALDIRI REYES JAMES
5	5851			ADARVE GOMEZ LUZ HELENA

Encontrados (1/1) registro(s)

2020
Bogotá, D. C.

Doctor(a)
DARIO CARDENAS NAVAS
Apoderado(a)

Oficio No. 03461

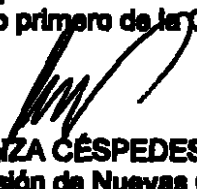
REFERENCIA: Radicación 03 31595
 Trámite 002
 Evento 001
 Actuación 430
 Folios 001

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 el inventor al solicitante o a su causante. (art. 28-k, Dec 486/2000).
- Los folios 46 a 111 se tomarán como solicitud nacional. *d por qué?*
- Debe indicarse el lugar de constitución de la sociedad solicitante (Art. 27-c, D 486/00).
- Se sugiere que para próximas actuaciones ante esta entidad sean utilizados los formatos actualizados establecidos en la circular única.

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 Unica No.10 de 2001.


ALIX CARMENZA CÉSPEDES DE VERGEL
Jefe de la División de Nuevas Creaciones

El anterior requerimiento fue notificado por fijación en lista de fecha

2020 MAYO 26

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 este plazo, acarreará la pérdida de la prioridad invocada.

202027

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